Deep Learning
Deep Neural Network

Yoon Joong Kim,
Hanbat National University

Deep Learning

Keras

Application & Tips(2)

Learning Rate
Dataset normalization
MNIST digit Classification

Yoonjoong Kim

Department of Computer Engineering, Hanbat National University

yjkim@hanbat.ac.kr

Content

Example 1. Learning rate

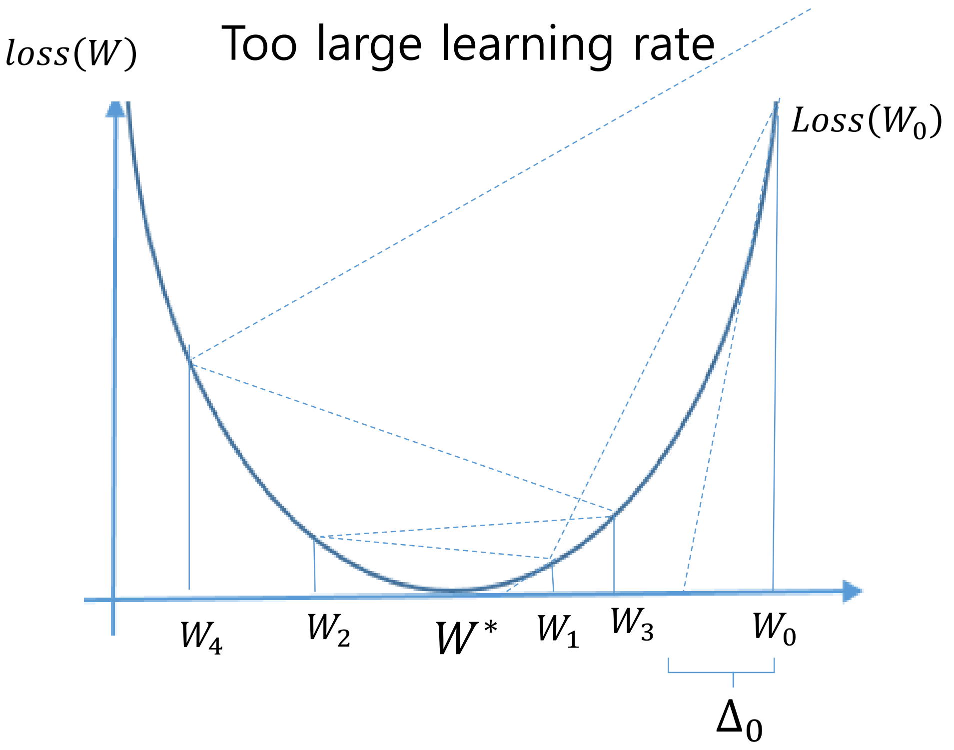
- Learning rate

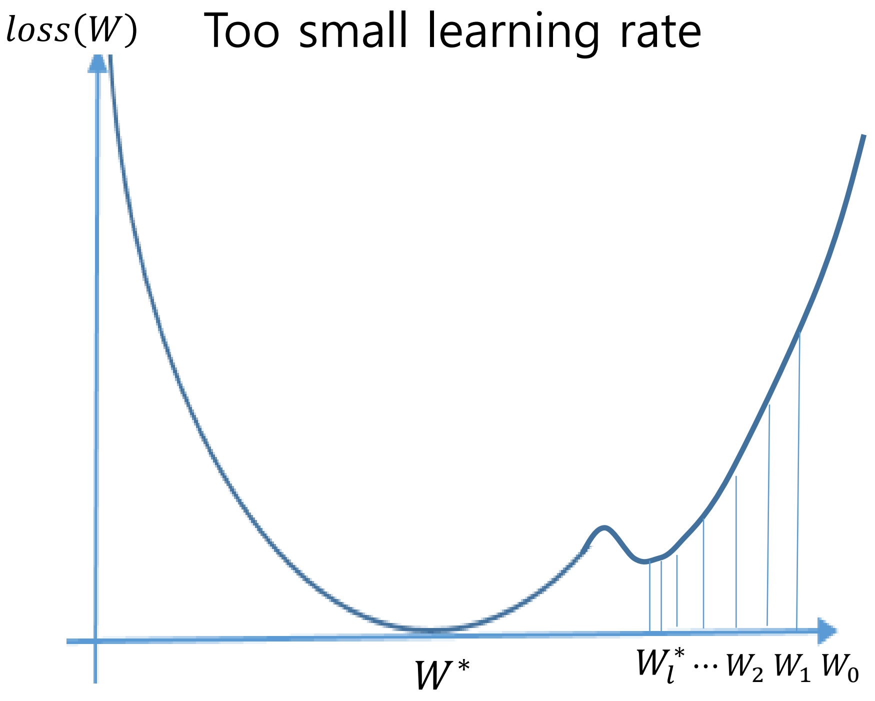
```
#데이터셋 생성
X          = np.array([[1, 2, 1], [1, 3, 2], [1, 3, 4], [1, 5, 5], [1, 7, 5], [1, 2, 5], [1, 6, 6], [1, 7, 7]])
Y_ohe      = np.array([[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]])
X_ev       = np.array([[2, 1, 1], [3, 1, 2], [3, 3, 4]])
Y_ev_ohe = np.array([[0, 0, 1], [0, 0, 1], [0, 0, 1]])

#모델 구조 정의,
model=Sequential()
model.add(Dense(units=3,input_dim=X.shape[1],activation='softmax'))
#모델학습방법설정
sgd=optimizers.SGD(lr=0.1)
model.compile(loss='categorical_crossentropy',optimizer=sgd,metrics=['accuracy'])
#모델학습
hist=model.fit(X,Y_ohe,epochs=200,verbose=1,validation_data=(X_ev, Y_ev_ohe))
#모델평가
loss_and_metrics = model.evaluate(X_ev, Y_ev_ohe, verbose=0)
print('## Evaluation Accuracy :', loss_and_metrics[1])
```

```
Epoch 197/200
8/8 [==============================] - 0s 374us/step - loss: 0.6118 - acc: 0.7500 - val_loss: 0.0300 - val_acc: 1.0000
Epoch 198/200
8/8 [==============================] - 0s 499us/step - loss: 0.6110 - acc: 0.7500 - val_loss: 0.0296 - val_acc: 1.0000
Epoch 199/200
8/8 [==============================] - 0s 374us/step - loss: 0.6102 - acc: 0.7500 - val_loss: 0.0293 - val_acc: 1.0000
Epoch 200/200
8/8 [==============================] - 0s 499us/step - loss: 0.6094 - acc: 0.7500 - val_loss: 0.0289 - val_acc: 1.0000
## Evaluation ##
## Accuracy : 1.0
```

Example 1. Learning rate

- Learning rate에 따른 학습 현상





Example 2. Dataset normalization

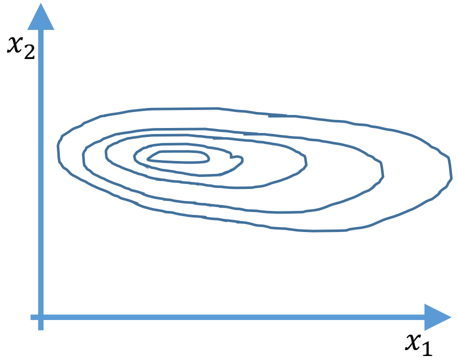
```
XY = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
X = XY[:, 0:-1]
Y = XY[:, [-1]]

#모델 구조 정의,
model=Sequential()
model.add(Dense(units=1,input_dim=X.shape[1],
               activation='linear’))

#모델학습방법설정
sgd=optimizers.SGD(lr=0.01)
model.compile(loss='mse',optimizer=sgd)

#모델학습
hist=model.fit(X,Y,epochs=10,verbose=1
```

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64\python.exe

```
Using TensorFlow backend.
Epoch 1/10
8/8 [==============================] - 3s 411ms/step - loss: 488792391680.0000
Epoch 2/10
8/8 [==============================] - 0s 499us/step - loss: 537026667766010684231828476264448.0000
Epoch 3/10
8/8 [==============================] - 0s 499us/step - loss: inf
Epoch 4/10
8/8 [==============================] - 0s 623us/step - loss: inf
Epoch 5/10
8/8 [==============================] - 0s 623us/step - loss: inf
Epoch 6/10
8/8 [==============================] - 0s 499us/step - loss: nan
Epoch 7/10
8/8 [==============================] - 0s 374us/step - loss: nan
Epoch 8/10
8/8 [==============================] - 0s 499us/step - loss: nan
Epoch 9/10
8/8 [==============================] - 0s 374us/step - loss: nan
Epoch 10/10
8/8 [==============================] - 0s 249us/step - loss: nan
Press any key to continue . . .
```

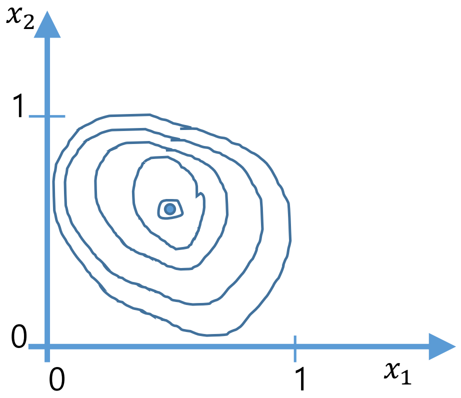
Example 2. Dataset normalization

```
XY = np.array(
[[828.659973, 833.450012, 908100, 828.349976, 831.659973],
 [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
 [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
 [816, 820.958984, 1008100, 815.48999, 819.23999],
 [819.359985, 823, 1188100, 818.469971, 818.97998],
 [819, 823, 1198100, 816, 820.450012],
 [811.700012, 815.25, 1098100, 809.780029, 813.669983],
 [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
```




```
XY = MinMaxScaler(XY)
print(XY)
```

```
[[0.99999948 0.99999945 0.         0.99999958 0.99999955]
 [0.70548455 0.70439514 1.         0.71881752 0.83755754]
 [0.54412521 0.50274796 0.57608696 0.60646775 0.6606328 ]
 [0.33890335 0.31368006 0.10869565 0.45989115 0.43800899]
 [0.51435973 0.42582366 0.30434783 0.58504781 0.42624382]
 [0.49556153 0.42582366 0.31521739 0.48131114 0.49276115]
 [0.11436058 0.         0.20652174 0.22007767 0.1859723 ]
 [0.         0.07747095 0.5326087  0.         0.        ]]
```



Example 2. Dataset normalization

```python
XY = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
   [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
[819.929993, 824.400024, 1438100, 818.97998, 824.159973],
[816, 820.958984, 1008100, 815.48999, 819.23999],
[819.359985, 823, 1188100, 818.469971, 818.97998],
[819, 823, 1198100, 816, 820.450012],
[811.700012, 815.25, 1098100, 809.780029, 813.669983],
[809.51001, 816.659973, 1398100, 804.539978, 809.559998]])
def MinMaxScaler(data):
        numerator = data - np.min(data, 0)
        denominator = np.max(data, 0) - np.min(data, 0)
        return numerator / (denominator + 1e-5)

XY   = MinMaxScaler(XY)
print(XY)
X      = XY[:, 0:-1]
Y      = XY[:, [-1]]

model=Sequential()
model.add(Dense(units=1,input_dim=X.shape[1],activation='linear'))
sgd=optimizers.SGD(lr=0.01)
model.compile(loss='mse',optimizer=sgd)
model.fit(X,Y,epochs=10,verbose=1)
```

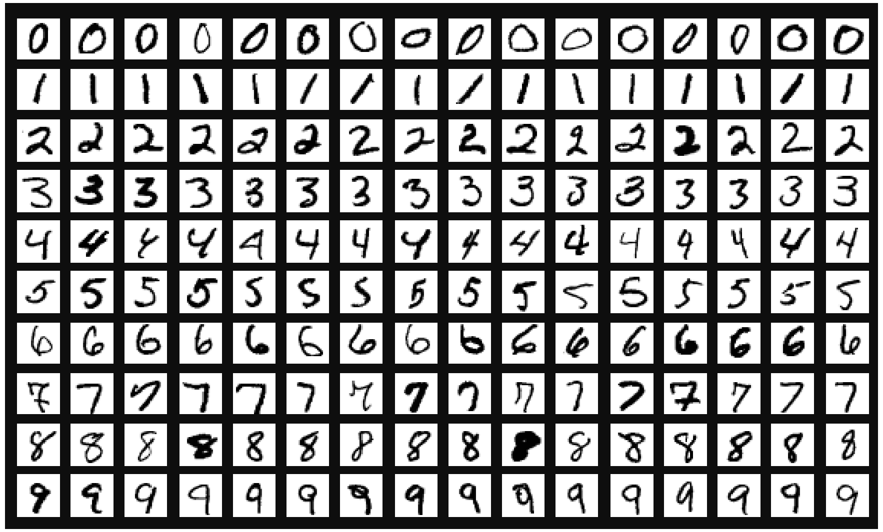
```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64...
Using TensorFlow backend.
[[0.99999948 0.99999945 0.         0.99999958 0.99999955]
 [0.70548455 0.70439514 1.         0.71881752 0.83755754]
 [0.54412521 0.50274796 0.57608696 0.60646775 0.6606328 ]
 [0.33890335 0.31368006 0.10869565 0.45989115 0.43800899]
 [0.51435973 0.42582366 0.30434783 0.58504781 0.42624382]
 [0.49556153 0.42582366 0.31521739 0.48131114 0.49276115]
 [0.11436058 0.         0.20652174 0.22007767 0.1859723 ]
 [0.         0.07747095 0.5326087  0.         0.        ]]
Epoch 1/10
8/8 [==============================] - 4s 470ms/step - loss: 1.5456
Epoch 2/10
8/8 [==============================] - 0s 623us/step - loss: 1.4379
Epoch 3/10
8/8 [==============================] - 0s 499us/step - loss: 1.3383
Epoch 4/10
8/8 [==============================] - 0s 624us/step - loss: 1.2460
Epoch 5/10
8/8 [==============================] - 0s 374us/step - loss: 1.1607
Epoch 6/10
8/8 [==============================] - 0s 374us/step - loss: 1.0817
Epoch 7/10
8/8 [==============================] - 0s 499us/step - loss: 1.0086
Epoch 8/10
8/8 [==============================] - 0s 499us/step - loss: 0.9410
Epoch 9/10
8/8 [==============================] - 0s 374us/step - loss: 0.8784
Epoch 10/10
8/8 [==============================] - 0s 374us/step - loss: 0.8204
Press any key to continue . . .
```

Example 3. MNIST 10 digit image classifier

- MNIST data
  - for 10 digit image recognizer with softmax hypothesis(model) and cross-entropy cost function(loss function)

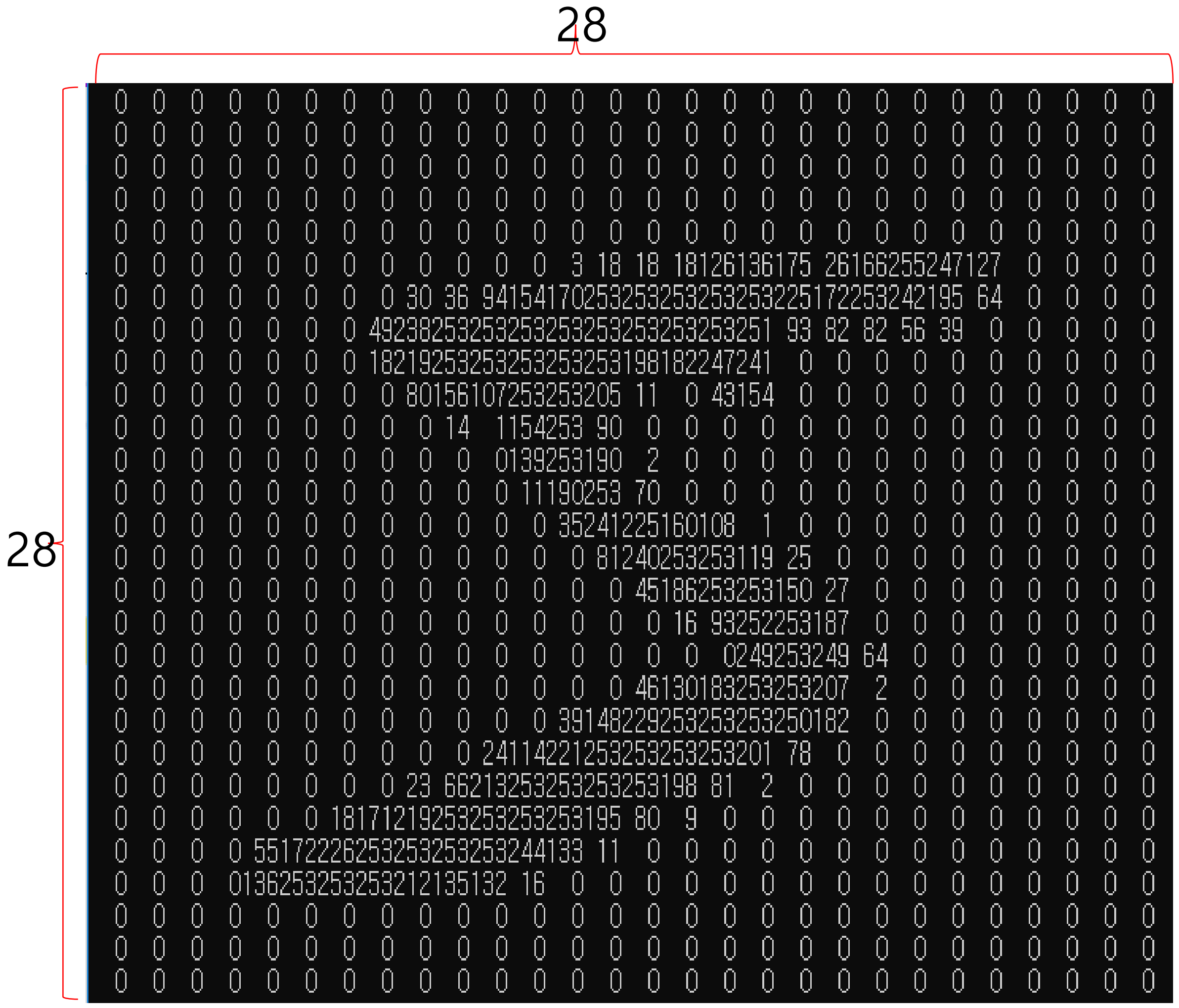
  - The **MNIST database** (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems
  - https://en.wikipedia.org/wiki/MNIST_database

- mnist 데이터 분석
  - 0부터 9까지 필기체 이미지, 이미지 크기 : 28x28 픽셀, 0~256 dpeth
  - 학습용 60000이미지
  - 평가용 10000이미지
  - 이미지 shape
    - X_train.shape : (60000, 28, 28) Y_train.shape: (60000,)
    - X_validation.shape:(10000, 28, 28) Y_validation.shpe : (10000,)



Example 3. MNIST 10 digit image classifier (cont.)

- Mnist 데이터 분석
  - 28x28x1 image
  - X_train[0]의 이미지
    - X_train[0][0][0]=0
      X_train[0][5][12]=3 X_train[0][5][13]=18
      X_train[0][6][7]=0 X_train[0][6][8]= 30
  - Y_train[0] : 5



Example 3. MNIST 10 digit image classifier (cont.)

- 데이터 셋 생성
  - 로드 된 이미지 shape
    - X_train.shape : (60000, 28, 28)   Y_train.shape : (60000,)
    - X_val.shape : (10000, 28, 28)   Y_val.shpe : (10000,)
  - Reshape 및 정규화
    - 60000x28x28 => 60000x784
    - 60000장의 28x28이미지를 1차원의 784열로 변환하고 실수로 변환한 후 0~1.0으로 정규화 한다.
  - one_hot_encoding
    - 모든 이미지의 타켓(label)은 는 10개의 숫자중의 하나이므로 Y의 값을 one_hot_encoding(10)
    - Y_train[0]=5 => Y_train_ohe[0]=[0,0,0,0,0,1,0,0,0,0]

```
from keras.datasets import mnist                                             # mnist 이미지 패키지

(X_train,Y_train), (X_validation, Y_validation) = mnist.load_data()          # 이미지 dataset 로드
X_train  = X_train.reshape(X_train.shape[0], 784).astype(‘float64’) / 255    #선형으로 그리고 0~1로 정규화
X_val    = X_validation.reshape(X_validation.shape[0], 784).astype('float64') / 255  #선형으로 그리고 0~1로 정규화

Y_train_ohe        = np_utils.to_categorical(Y_train, 10)                    #one_hot_encoding(Y,10)
Y_validation_ohe = np_utils.to_categorical(Y_validation, 10)                 #one_hot_encoding(Y_va)
#X_train.shape       : (60000, 28, 28)      Y_train.shape    : (60000,)      #데이터셋 shape
#X_validation.shape  :(10000, 28, 28)       Y_validation.shpe  : (10000,)
```

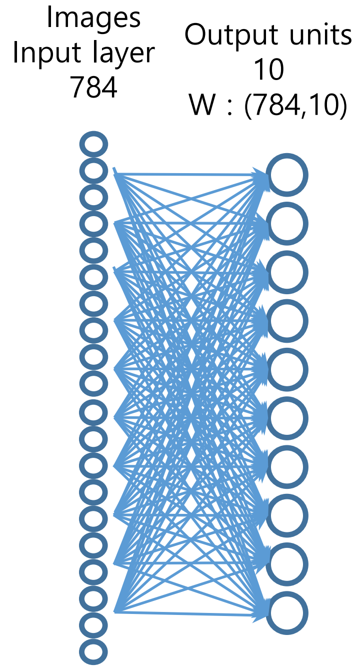
Example 3. MNIST 10 digit image classifier (cont.)

- 모델생성

```
from keras.models import load_model, Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping
from  keras import optimizers
import numpy as np
import os
from keras.datasets import mnist
from keras.utils import np_utils
```

```
#저장할 모델의 이름및 패쓰를 설정
MODEL_SAVE_FOLDER_PATH = './model/'
if not os.path.exists(MODEL_SAVE_FOLDER_PATH):
    os.mkdir(MODEL_SAVE_FOLDER_PATH)
model_path = MODEL_SAVE_FOLDER_PATH + 'mnist-' + '{epoch:02d}-{val_loss:.4f}.hdf5
model_path = 'mnist-best-mode.hdf5'
```

```
model = Sequential()

#모델생성
model.add( Dense(            #fully connected feed forward neural network
        units     =10,       #number of output nodes
        input_dim=784,       #number of input nodes
        activation='softmax’))   #activation function

#학습 방법 설정
model.compile(
        loss       ='categorical_crossentropy’,   #loss function to classify 10 classes
        optimizer ='adam’,                        #adam opimizer
         metrics   =['accuracy’])                 #metrics
```



Example 3. MNIST 10 digit image classifier (cont.)

- 모델 학습

```
#학습 방법 설정
model.compile(loss='categorical_crossentropy', #loss function to classify 10 cla
       optimizer='adam',       #adam opimizer
       metrics=['accuracy'])   #metrics

#모델 학습
#학습 시 조건인 만족될 때 호출되는 인스턴스 생성
#평가 loss가 개선되는 이벤트 및 모델을 저장하기위한 개체정의
cb_checkpoint = ModelCheckpoint(
       filepath=model_path,          #모델을 저장할 주소
       monitor='val_loss',           #모니터링 개체 – 평가용 loss
       verbose=1,                    #출력 모드
       save_best_only=True)          #모니터링개체의 조건 설정
#학습종료를 위한  이벤트 정의
cb_early_stopping = EarlyStopping(
       monitor='val_loss',           #모니터링 개체 – 평가용 loss
       patience=10)                  #loss가 10번 이상 좋아지지 않으면 종료
#모델 학습
history=model.fit(
       X_train, Y_train_ohe,         #학습용 데이터 셋
       validation_data=(X_validation, Y_validation_ohe),  #평가용 데이터셋
       epochs=128,                   #학습의 최대 반복 회수
       batch_size=10000,                  #매학습시 학습할 데이터셋의 크기 (샘플의 수)
       verbose=0,                    #학습과정 출력 량(모드)
       callbacks=[cb_checkpoint, cb_early_stopping]) #모니터용 이벤트 개체 리스트
print(pd.DataFrame(history.history))
```

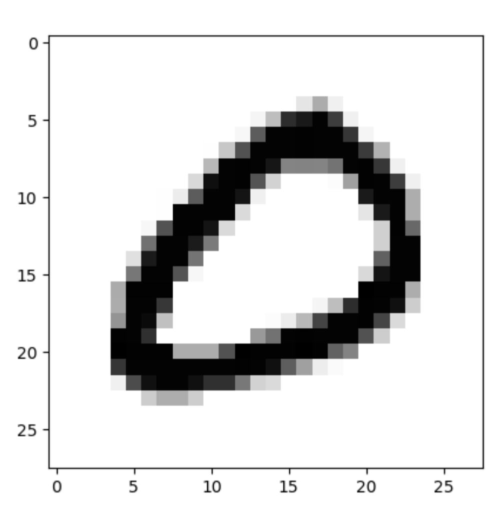
```
Epoch 00001: val_loss improved from inf to 2.09325, saving model to mnist_best_model.hdf5

Epoch 00002: val_loss improved from 2.09325 to 1.83853, saving model to mnist_best_model.hdf5

Epoch 00003: val_loss improved from 1.83853 to 1.62151, saving model to mnist_best_model.hdf5

Epoch 00004: val_loss improved from 1.62151 to 1.43904, saving model to mnist_best_model.hdf5
Epoch 00123: val_loss improved from 0.30111 to 0.30057, saving model to mnist_best_model.hdf5

Epoch 00124: val_loss improved from 0.30057 to 0.30014, saving model to mnist_best_model.hdf5

Epoch 00125: val_loss improved from 0.30014 to 0.29980, saving model to mnist_best_model.hdf5

Epoch 00126: val_loss improved from 0.29980 to 0.29925, saving model to mnist_best_model.hdf5

Epoch 00127: val_loss improved from 0.29925 to 0.29874, saving model to mnist_best_model.hdf5

Epoch 00128: val_loss improved from 0.29874 to 0.29834, saving model to mnist_best_model.hdf5
     val_loss  val_accuracy      loss  accuracy
0    1.985495        0.3502  2.154175  0.228483
1    1.747847        0.5176  1.889576  0.430333
2    1.544695        0.6405  1.670425  0.572950
3    1.374505        0.7186  1.483914  0.669367
4    1.234919        0.7521  1.329626  0.724217
..        ...           ...       ...       ...
123  0.300143        0.9193  0.305743  0.917700
124  0.299804        0.9190  0.305134  0.917567
125  0.299245        0.9194  0.304564  0.917850
126  0.298738        0.9194  0.303975  0.917983
127  0.298341        0.9194  0.303406  0.918200

[128 rows x 4 columns]

Accuracy best: 0.9194

 preds=predict(X_validation[8992]) : [[9.9728870e-01 2.0276344e-08 5.2090338e-04 2.3325051e-04 2.1251537e-08
  1.9167213e-03 3.2738666e-05 2.0543259e-06 4.1525323e-06 1.4797728e-06]]

label:[0]  argmax(preds):0
```

Example 3. MNIST 10 digit image classifier (cont.)

- 모델평가

```
# best 학습된 모델로 평가데이터셋의 성능 평가
Model =load_model(model_path )                          # best 모델 로드
score  =model.evaluate(X_validation, Y_validation_ohe) #평가용 셋으로 성능계산

print('\nAccuracy: {:.4f}'.format(score[1]))            # 성능 정확도 출력

# 평가용 데이터셋에서 임의의 이미지를 추출하여 평가한다.
r=np.random.randint(0,X_validation.shape[0]-1)  # 임의의 이미지의 난수 생성
X1=X_validation[r:r+1]                          # r번 평가 이미지 로드
Y1=Y_validation[r:r+1]                          # r번 평가 이미지의 번호 로드

preds=model.predict(X1)                         # 로드 된 이미지 X1의 예측 10개의 실수 벡터
print('\n preds=predict(X_validation[{}]) : {} '.format(r, preds))     # 예측된 벡터출력
print('\nlabel:{}  argmax(preds):{} '.format(Y1,np.argmax(preds))) # 수와 예측된 수 출력

# 로드 된 이미지 X1을 그림으로 출력하여 인식결과와 비교한다.
import matplotlib.pyplot as plt
plt.imshow(X1.reshape(28,28),cmap='Greys',interpolation='nearest')
plt.show()
```

```
Accuracy best: 0.9194

preds=predict(X_validation[8992]) :
 [[9.9728870e-01 2.0276344e-08 5.2090338e-04 2.3325051e-04 2.1251537e-08
  1.9167213e-03 3.2738666e-05 2.0543259e-06 4.1525323e-06 1.4797728e-06]]

label:[0]   argmax(preds):0
```



Example 4. Tips of NN for the Mnist digit classifier

- 1 hidden layer NN for the Mnist digit classifier

```
#데이터 셋 생성
(X_train,Y_train), (X_val, Y_val) = mnist.load_data()
X_train     = X_train.reshape(X_train.shape[0], 784).astype('float64') / 255#flatten and normalize
X_val       = X_val.reshape(X_val.shape[0], 784).astype('float64') / 255     #flatten and normalize
Y_train_ohe = np_utils.to_categorical(Y_train, 10)         #one_hot_encoding(Y)
Y_val_ohe   = np_utils.to_categorical(Y_val, 10)           #one_hot_encoding(Y_val)

#모델의 구조 설정
model = Sequential()
model.add(Dense(units=10, input_dim=784,
        kernel_initializer='random_uniform', activation='softmax'))

#모델의 학습방법 설정
model.compile(
        loss='categorical_crossentropy’,          #크로스엔트로피로 손실함수 계산
        optimizer='adam’,                         #adam optimizer
        metrics=['accuracy’])                     # loss와 acc계산

#모델의 학습
model.fit(X_train, Y_train_ohe,                   #학습용데이터 셋
        validation_data=(X_val, Y_val_ohe),#평가용 데이터 셋
        epochs=20, batch_size=512, verbose=1) #학습과정 출력제한

#평가용 데이터셋으로 모델을 평가하여 정밀도(acc)를 출력한다.
print('\nAccuracy: {:.4f}'.format(model.evaluate(X_val, Y_val_ohe)[1]))
```

```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import os
from keras.datasets import mnist
from keras.utils import np_utils
```

```
Epoch 19/20
60000/60000 [==================================] - 1s 18us
/step - loss: 0.4730 - acc: 0.8838 - val_loss: 0.4485 - val_acc: 0.8940
Epoch 20/20
60000/60000 [==================================] - 1s 19us
/step - loss: 0.4613 - acc: 0.8857 - val_loss: 0.4376 - val_acc: 0.8947
10000/10000 [==================================] - 0s 32us
/step

Accuracy: 0.8947
```

Example 4. Application Tips for the Mnist digit classifier (cont.)

- 2 hidden layers

```
model = Sequential()
model.add(Dense(units=256, input_dim=784,
                kernel_initializer='random_uniform', activation='linear'))
model.add(Dense(units=256, kernel_initializer='random_uniform', activation='linear'))
model.add(Dense(units=10,  kernel_initializer='random_uniform', activation='softmax'))
```

Accuracy: 0.9267

- 2 hidden NN with relu

```
model = Sequential()
model.add(Dense(units=256, input_dim=784,
                kernel_initializer='random_uniform', activation='relu'))
model.add(Dense(units=256, kernel_initializer='random_uniform', activation='relu'))
model.add(Dense(units=10,   kernel_initializer='random_uniform', activation='softmax'))
```

Accuracy: 0.9806

- 2 hidden NN with relu activation and Xavier initialization

```
model = Sequential()
model.add(Dense(units=256, input_dim=784,
                kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dense(units=10,   kernel_initializer='glorot_uniform', activation='softmax'))
```

Accuracy: 0.9817

Example 4. Application Tips for the Mnist digit classifier (cont.)

- 5 hidden (Deep) NN with relu activation and Xavier initialization

```
model = Sequential()
model.add(Dense(units=256, input_dim=784,
                kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dense(units=10, kernel_initializer='glorot_uniform', activation='softmax'))
```

Accuracy: 0.9798

- 5 hidden (Deep) NN with relu activation, Xavier initialization and dropout

```
model = Sequential()
model.add(Dense(units=256, input_dim=784,
kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=10, kernel_initializer='glorot_uniform', activation='softmax'))
```

```
#model.add(Dropout(0.1))
```

Accuracy: 0.9803

```
#model.add(Dropout(0.2))
```

Accuracy: 0.9840

```
#model.add(Dropout(0.3))
```

Accuracy: 0.9819

Example 4. Application Tips the Mnist digit classifier (cont.)

- 5 hidden (Deep) NN with relu activation , Xavier initialization and activity regularizer

```
from keras.regularizers import l1
reg = l1(0.000001)
model = Sequential()
model.add(Dense(units=256, input_dim=784,
kernel_initializer='glorot_uniform', activation='relu' ,activity_regularizer=reg))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu',activity_regularizer=reg))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu',activity_regularizer=reg))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu',activity_regularizer=reg))
model.add(Dropout(0.2))
model.add(Dense(units=256, kernel_initializer='glorot_uniform', activation='relu',activity_regularizer=reg))
model.add(Dropout(0.2))
model.add(Dense(units=10, kernel_initializer='glorot_uniform', activation='softmax' ,activity_regularizer=reg))
```

```
Epoch 16/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0585 - acc: 0.9881 - val_loss: 0.0920 - val_acc: 0.9815
Epoch 17/20
60000/60000 [==============================] - 2s 40us/step - loss: 0.0558 - acc: 0.9887 - val_loss: 0.0934 - val_acc: 0.9801
Epoch 18/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0542 - acc: 0.9889 - val_loss: 0.0992 - val_acc: 0.9791
Epoch 19/20
60000/60000 [==============================] - 2s 38us/step - loss: 0.0520 - acc: 0.9896 - val_loss: 0.0908 - val_acc: 0.9801
Epoch 20/20
60000/60000 [==============================] - 3s 42us/step - loss: 0.0510 - acc: 0.9900 - val_loss: 0.0962 - val_acc: 0.9804
10000/10000 [==============================] - 1s 76us/step

Accuracy: 0.9815
```

불필요 할 수도, 적절한 값의 선택이 필요

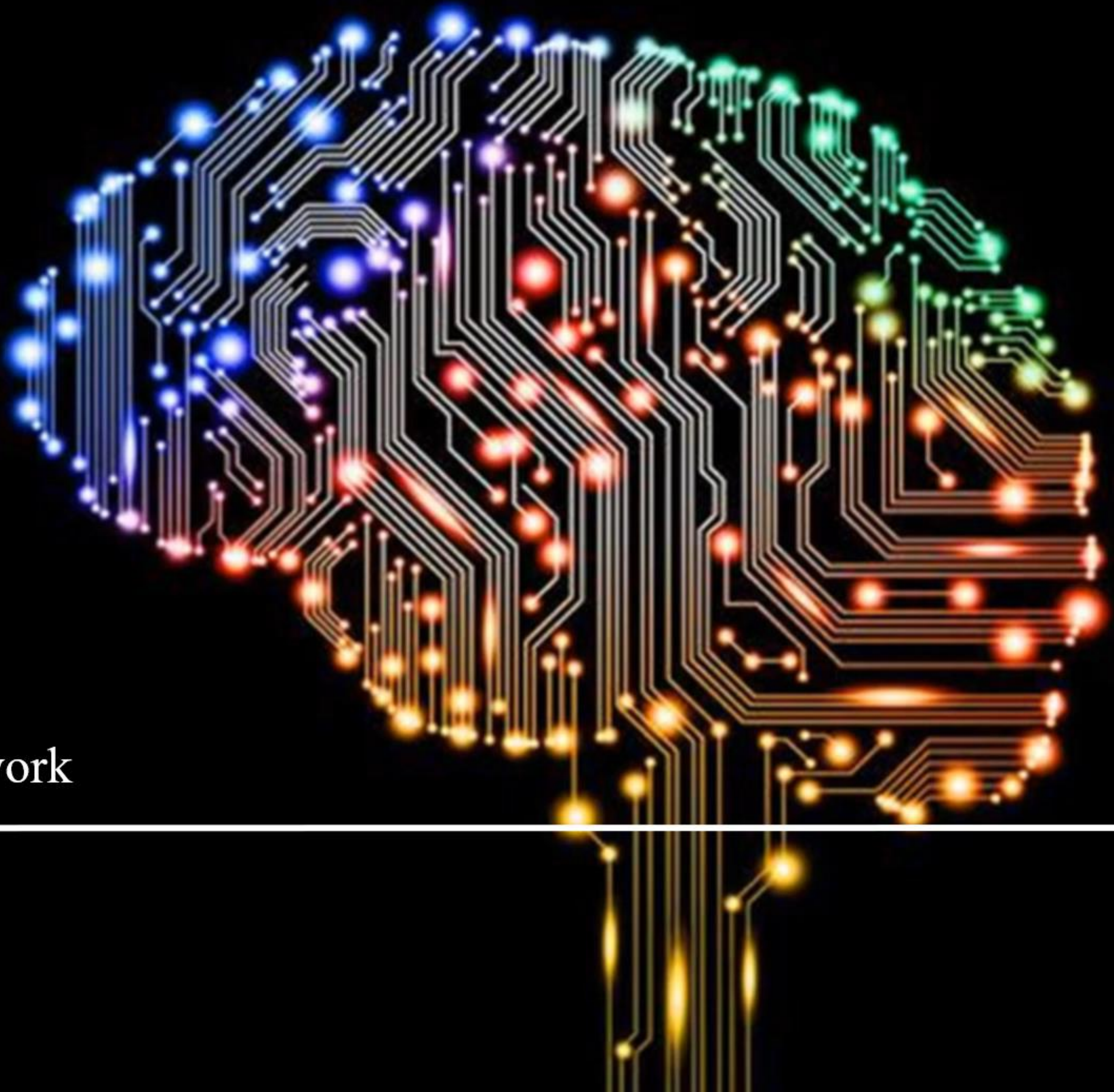
Summary

1. Learning rate
   1. Gradient descent algorithm
   2. Large rate
   3. Small rate
2. Data preprocessing for the gradient decent algorithm
   1. Mean, std normalization
   2. Min-max normalization
3. Overfitting
   1. More training data or Reduce the number of features
   2. Regularization
   3. Dropout
4. 최적화기(Optimizer)의 종류

5. Batch Normalization
6. Examples
   1. Learning rate
   2. Dataset normalization
   3. Mnist digit classifier
      1. Mnist 이미지 데이터 분석
      2. Model 개발 및 평가
      3. 이미지 인식 및 출력방법
   4. Application Tips for the Mnist digit classifier

Summary

1. Learning rate
   1. Gradient descent algorithm
   2. Large rate
   3. Small rate
2. Data preprocessing for the gradient decent algorithm
   1. Mean, std normalization
   2. Min-max normalization
3. Overfitting
   1. More training data or Reduce the number of features
   2. Regularization
   3. Dropout
4. 최적화기(Optimizer)의 종류
5. Batch Normalization
6. Examples
   1. Learning rate
   2. Dataset normalization
   3. Mnist digit classifier
      1. Mnist 이미지 데이터 분석
      2. Model 개발 및 평가
      3. 이미지 인식 및 출력방법
   4. Application Tips for the Mnist digit classifier



Deep Learning
Deep Neural Network

Yoon Joong Kim,
Hanbat National University