Deep Learning

Keras

# *Application & Tips*
# *Learning, data preparation, overfitting*

**Learning,**
**data preparation,**
**overfitting,**
**Optimizer**

**Batch Normalization**
**examples**

*Yoon Joong Kim*
*Department of Computer Engineering, Hanbat National University*
*yjkim@hanbat.ac.kr*

# Content

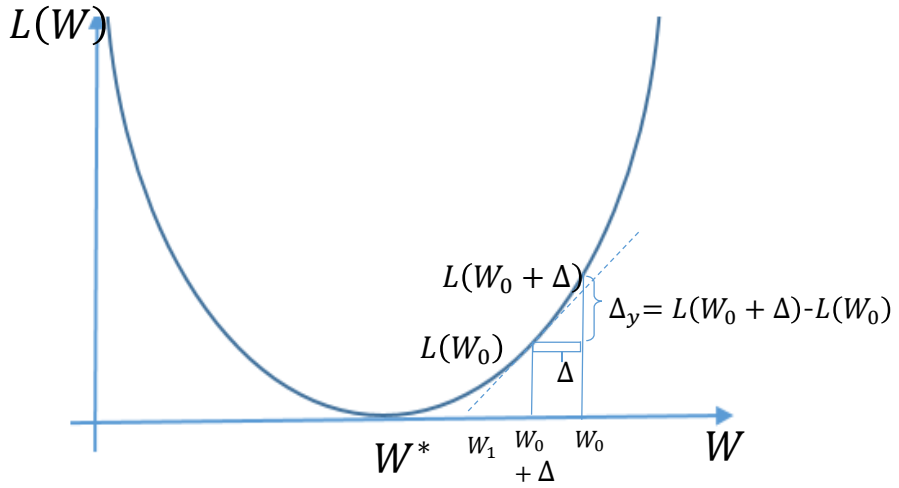
# 1. Learning rate

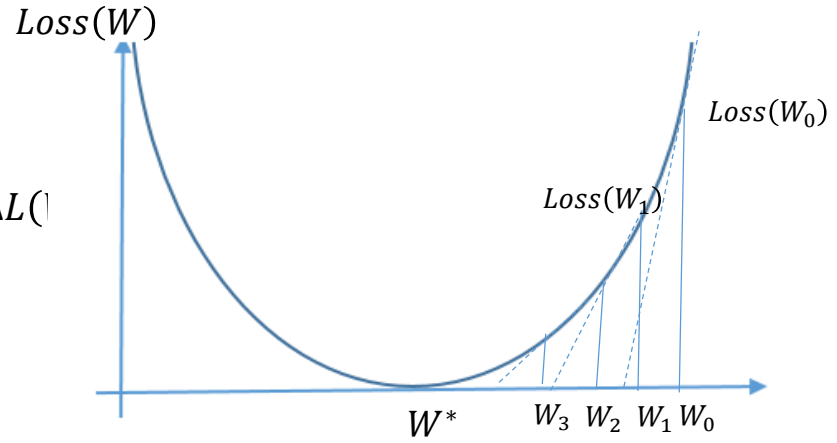
- Gradient descent algorithm
  - $W^* = \underset{W}{ArgMin} \frac{\partial}{\partial w} Loss(W)$
    - $W_{n+1} = W_n - \alpha \Delta L(W_n)$
    - $loss(W|X,Y) = dist(H(X|W,b),Y)$
  - $W_1 = W_0 - 0.5\Delta L(W_0)$
    - $\Delta L(W_0) = \left|\frac{\partial}{\partial w} L(W)\right|_{W=W_0}$
      $= \lim_{\Delta \to 0} \frac{L(W_0+\Delta)-L(W_0)}{\Delta} = \lim_{\Delta \to 0} \frac{\Delta_y}{\Delta}$
  - 훈련과정
    - $initialize\ W_0$
    - $W_1 = W_0 - 0.5\Delta L(W_0)$

      ……

$W_2 = W_1 - 0.5\Delta L($

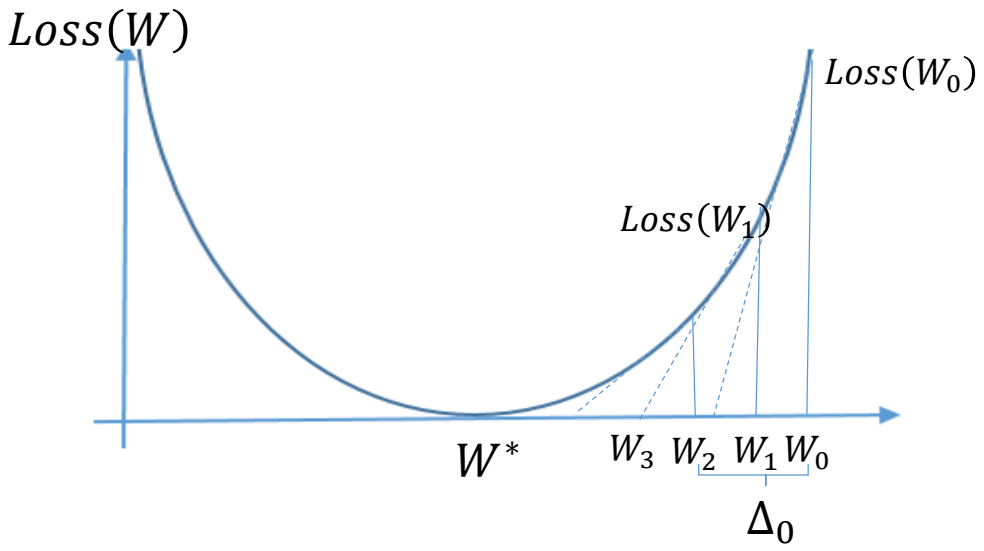
$W^*$



# 1. Learning rate(cont.)

- 1.1 Gradient descent algorithm




$initialize\ W_0$

$W_1 = W_0 - 0.001\Delta L(W_0)$

$W_2 = W_1 - 0.001\Delta L(W_1)$

.....

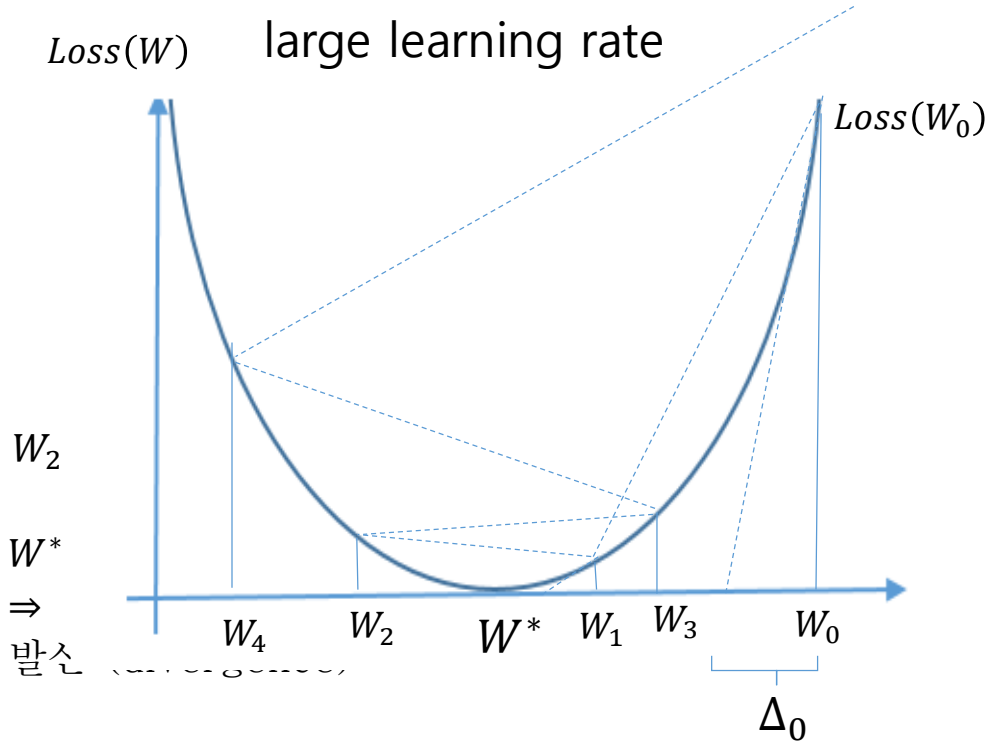
$W^*$

```
model=sequential()
model.add(Dense(1,input_dim=3))
model.compile(loss='mse',optimizer='sgd')
model.fit(X,Y,epochs=1000)
```

# 1. Learning rate(cont.)

- 1.2 Large learning rate :overshooting(발산)
  - Learning
    - $W_{n+1} = W_n - \alpha \Delta \mathrm{Loss}(W_n)$
    - W0 => W* ?
      W* :global minimum
  - $\alpha = 10, large\ value$
  - $initialize\ W_0$
    - $W_1 = W_0 - 10\Delta L(W_0)$

    ……

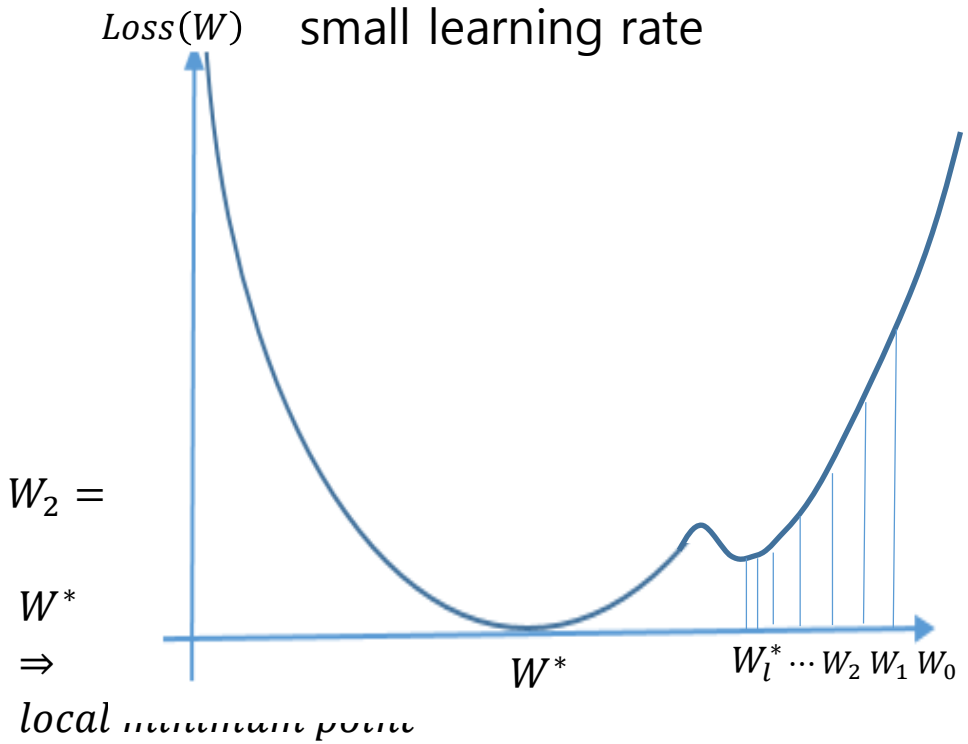



  - How to converge to $W^*$?
    수렴 (convergence) ?

# 1. Learning rate(cont.)

- 1.3 Small learning rate : local minimum point
  - Learning
    - $W_{n+1} = W_n - \alpha \Delta Loss(W_n)$
    - W0 => W* ?
      W* :global minimum
  - $\alpha = 0.001, small\ value$
  - $initialize\ W_0$
    - $W_1 = W_0 - 0.001 \Delta L(W_0)$

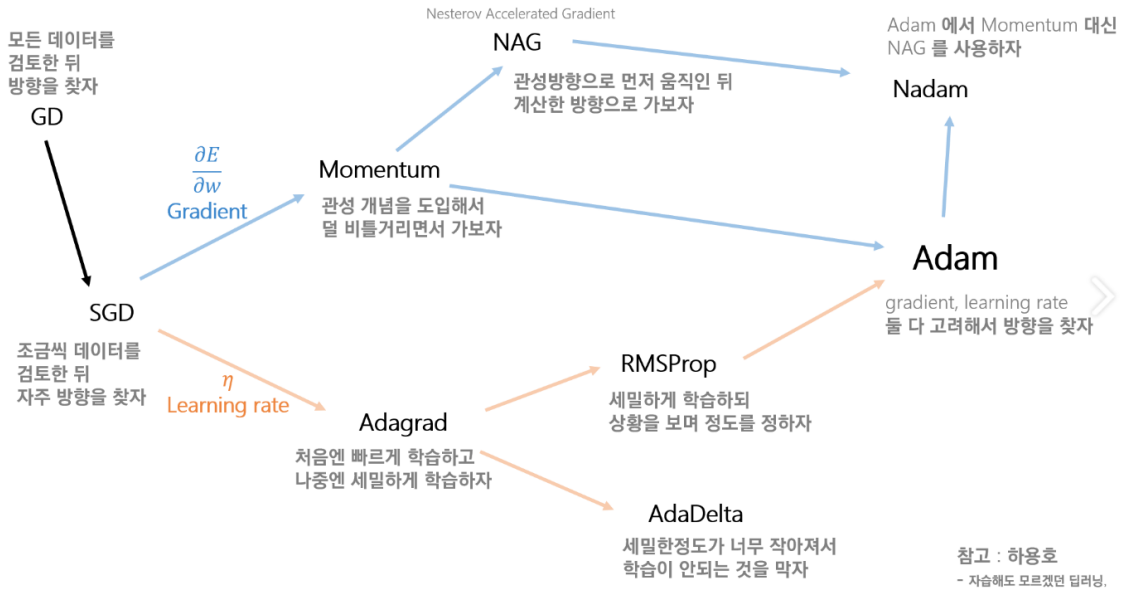
    ……
  - How to converge to $W^*$?
    수렴 (convergence) ?




Local minimum point
$W_l^*$로 수렴 (convergence)

# 1. Learning rate

- 1.4 Optimizer 종류 [link] [linlk]
  - Momentum 방식은
    - 말 그대로 Gradient Descent를 통해 이동하는 과정에 일종의 '관성'을 주는 것이다. 현재 Gradient를 통해 이동하는 방향과는 별개로, 과거에 이동했던 방식을 기억하면서 그 방향으로 일정 정도를 추가적으로 이동하는 방식이다.
  - Adagrad(Adaptive Gradient)는
    - 변수들을 update할 때 각각의 변수마다 step size를 다르게 설정해서 이동하는 방식이다. 이 알고리즘의 기본적인 아이디어는 '지금까지 많이 변화하지 않은 변수들은 step size를 크게 하고, 지금까지 많이 변화했던 변수들은 step size를 작게 하자' 라는 것이다.
  - RMSProp은
    - 딥러닝의 대가 제프리 힌톤이 제안한 방법으로서, Adagrad의 단점을 해결하기 위한 방법이다. Adagrad의 식에서 gradient의 제곱값을 더해나가면서 구한 Gt 부분을 합이 아니라 지수평균으로 바꾸어서 대체한 방법이다.
  - Adam (Adaptive Moment Estimation)은
    - RMSProp과 Momentum 방식을 합친 것 같은 알고리즘이다

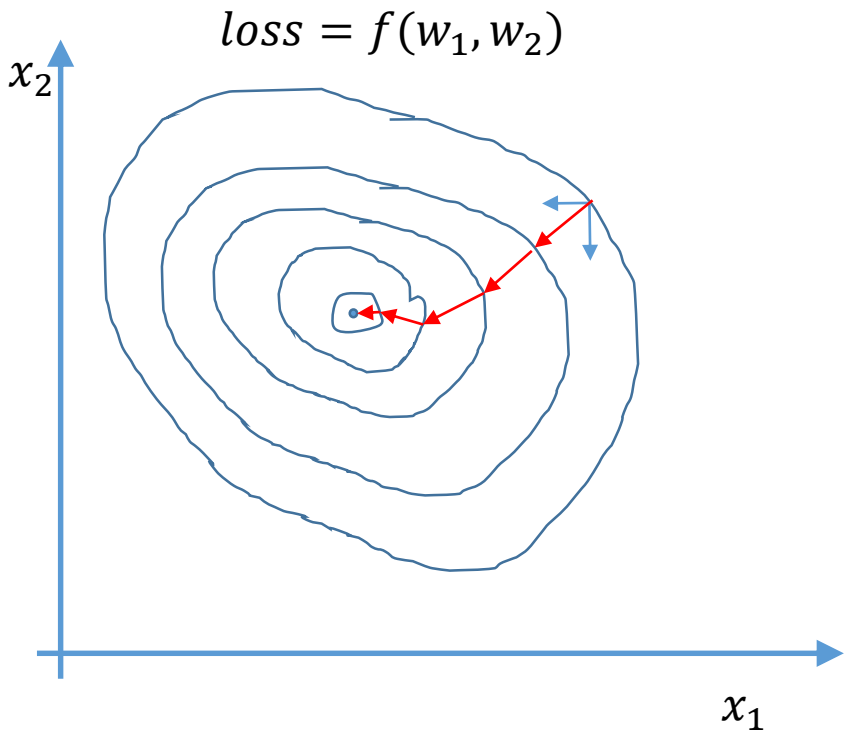


# 1. Learning rate

- How to choose the learning rate?
  - Try several learning rates
    - Observe the cost function
    - Check it goes down in a reasonable rate
    - Check it converges to global minimum point $W^*$
- Optimizer 의 선정
  - Momentum
  - SGD(Stochastic Gradient Descent)
  - Adagrad(Adaptive Gradient)
  - RMSProp
  - Adam (Adaptive Moment Estimation)

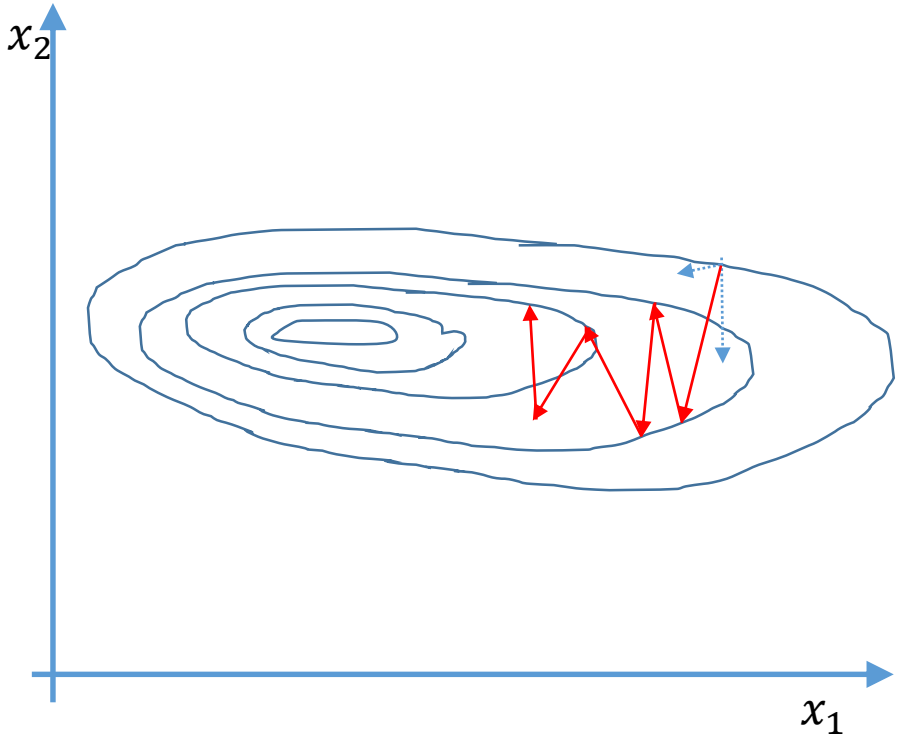
# 2. Data preprocessing for thr gradient descent algorithm

- Training process of Gradient descent algorithm



# 2. Data preprocessing for a gradient descent(cont.)

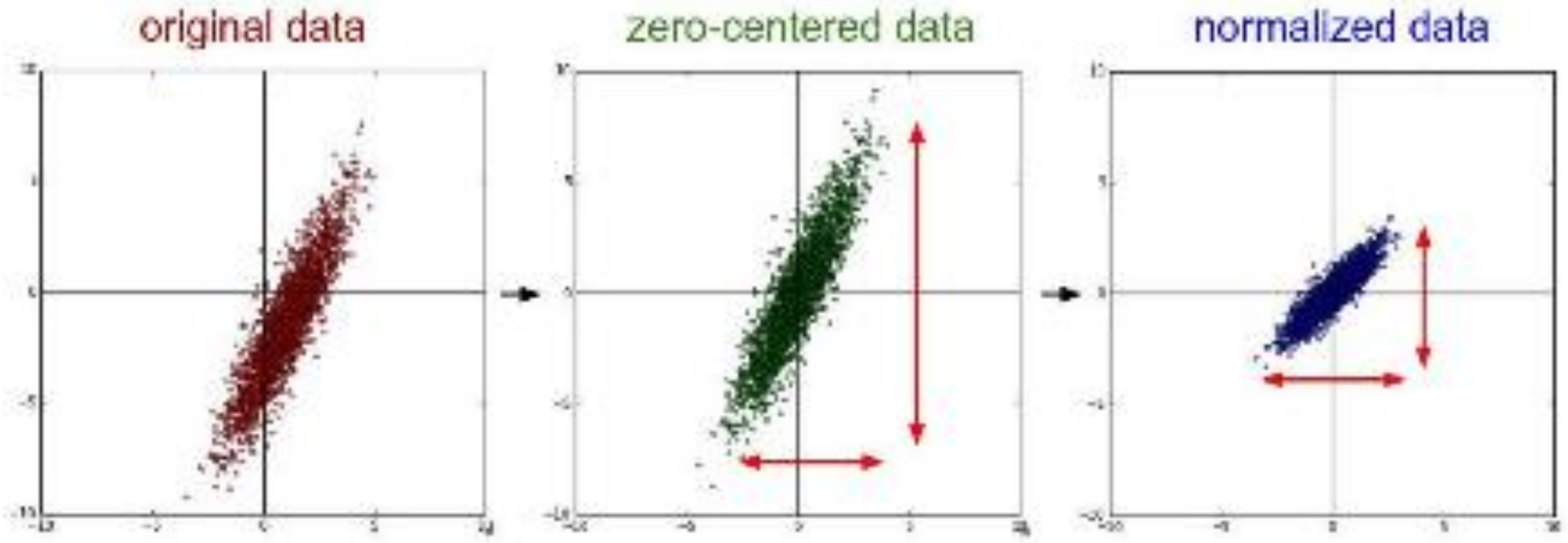
- 2.1 Too big difference in value among elements of dataset




| x1 | x2 | y |
|---|---|---|
| 1 | 9000 | A |
| 2 | -5000 | A |
| 4 | -2000 | B |
| 6 | 8000 | B |
| 9 | 9000 | C |

# 2. Data preprocessing for a gradient descent(cont.)

- 2.2 데이터셋 원소 분포의 형태



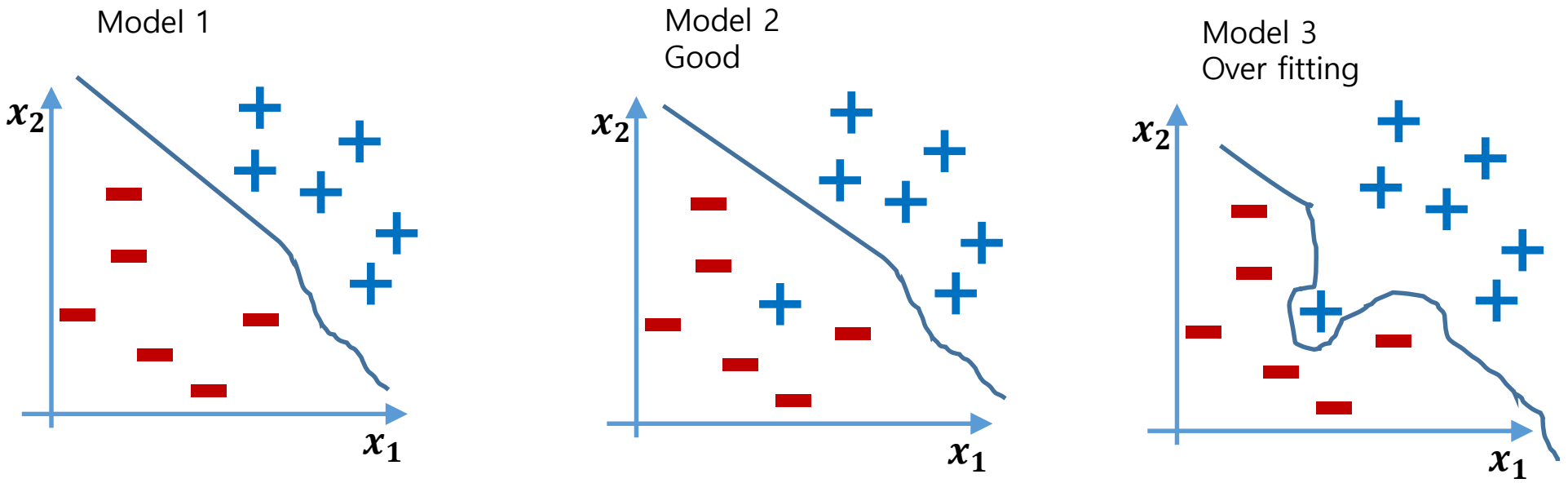
# 2. Data preprocessing for a gradient descent(cont.)

- 2.3 Mean, std normalization
  - $\boldsymbol{X' = \frac{X - X.mean}{X.std}}$

- 2.4 Min-max normalization
  - $\boldsymbol{X' = \frac{X - X.min}{X.max - X.min}}$

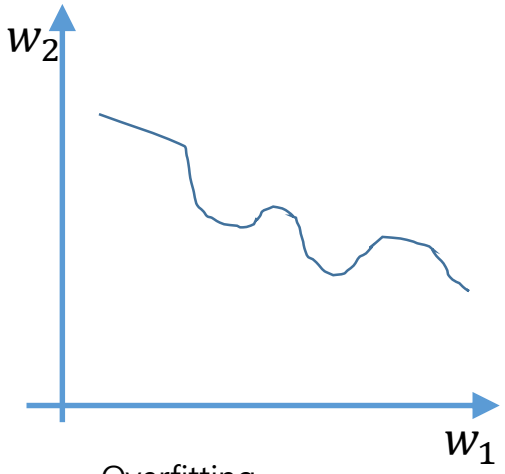
# 3. Overfitting

- Overfitting
  - 모델을 반복 훈련하는 과정에서
    훈련데이터 및 평가데이터에 대하여 학습이 잘되지만
    어느정도 반복 훈련한 후에 훈련데이터에 대해서는 학습이 계속 잘되지만
    평가데이터에 대해서는 학습이 더 이상 안되는 현상을 의미한다.

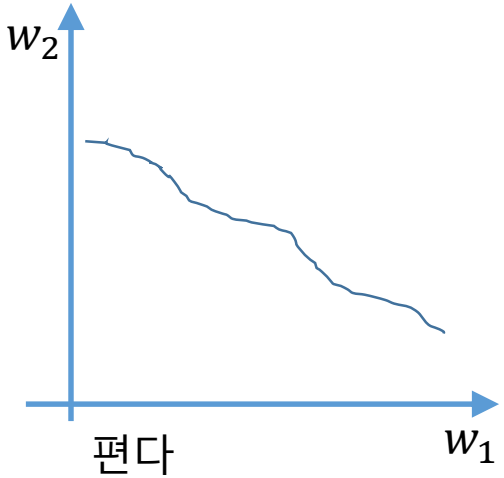


# 3. Overfitting (cont.)

- Solutions for overfitting
  - More training data!
  - Reduce the number of features
  - Regularization(일반화)
  - Dropout
- **Regularization(일반화)**
  - Let’s not have too big numbers in the weight




Overfitting
구부러짐
W 특정 값이 너무 크다




편다
구부러짐을 편다
W가 적은 값을 갖게 만든다.

# 4. Overfitting

- **Regularization(일반화)**
- Let's not have too big numbers in the weight

$$Loss(X,Y) = D(H(X|W,B),Y)$$

$$Loss(X,Y) = D(H(X|W,B),Y) + \lambda \sum W^2$$

$$W^* = \underset{W}{ArgMin} \frac{\partial}{\partial W} Loss(W|X,Y)$$

Regularization strength

W becomes small value

$\lambda$:0
$\lambda$:1
$\lambda$:0.001

```
fromkeras.regularizersimportl1_l2
reg = l1_l2(l1=0.01, l2=0.01)
model.add(Dense(1,input_dim=x.shape[1],W_regularizer=reg))
```

# 4. Optimizers

- Ussage of optimizers

```
from keras import optimizers
model = Sequential()
model.add(Dense(64, kernel_initializer='uniform', input_shape=(10,)))
model.add(Activation('softmax'))
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)
```

- Parameters common to all Keras optimizers
  - The parameters clipnorm and clipvalue can be used with all optimizers to control gradie nt clipping:

```
from keras import optimizers
# All parameter gradients will be clipped to # a maximum norm of 1.
sgd = optimizers.SGD(lr=0.01, clipnorm=1.)
```

```
from keras import optimizers
# All parameter gradients will be clipped to
# a maximum value of 0.5 and
# a minimum value of -0.5.
sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)
```

# 4. Optimizers (cont.)

- Stochastic gradient descent optimizer.

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

- RMSProp optimizer

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

- Adagrad optimizer

```
keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)
```

- Adadelta optimizer.

```
keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)
```

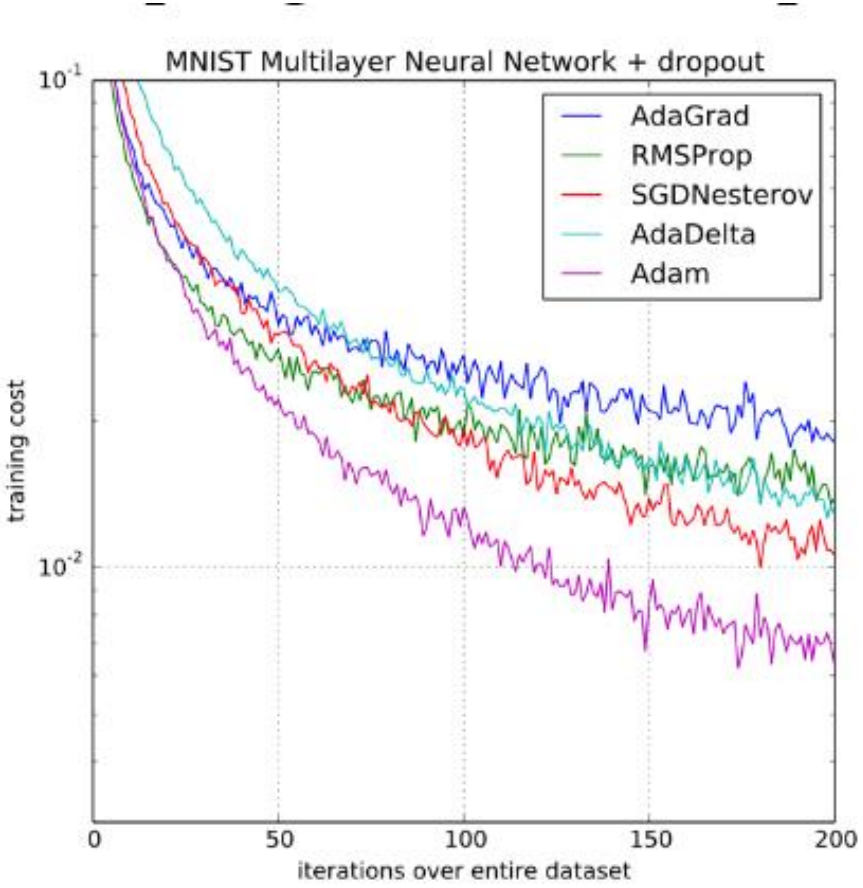
- Adam optimizer

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

https://keras.io/optimizers/

# 4. Optimizers (cont.)

- ADAM: a method for stochastic optimization [Kingma et al. 2015]



# 4. Optimizers (cont.)

- Use Optimizer

```
from keras import optimizers

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'))

adam = optimizers.Adam(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='softmax', optimizer=adam)
```

# 5. Batch Normalization

- Batch Normalization의 장점
  - ‘기존 Deep Network에서는 learning rate를 너무 높게 잡을 경우 gradient가 explode/vanish 하거나, 나쁜 local minima에 빠지는 문제가 있었다. 이는 parameter들의 scale 때문인데, Batch Normalization을 사용할 경우 propagation 할 때 parameter의 scale에 영향을 받지 않게 된다. 따라서, learning rate를 크게 잡을 수 있게 되고 이는 빠른 학습을 가능케 한다.
  - Batch Normalization의 경우 자체적인 regularization 효과가 있다. 이는 기존에 사용하던 weight regularization term 등을 제외할 수 있게 하며, 나아가 Dropout을 제외할 수 있게 한다 (Dropout의 효과와 Batch Normalization의 효과가 같기 때문.) . Dropout의 경우 효과는 좋지만 학습 속도가 다소 느려진다는 단점이 있는데, 이를 제거함으로서 학습 속도도 향상된다.
- 참고자료

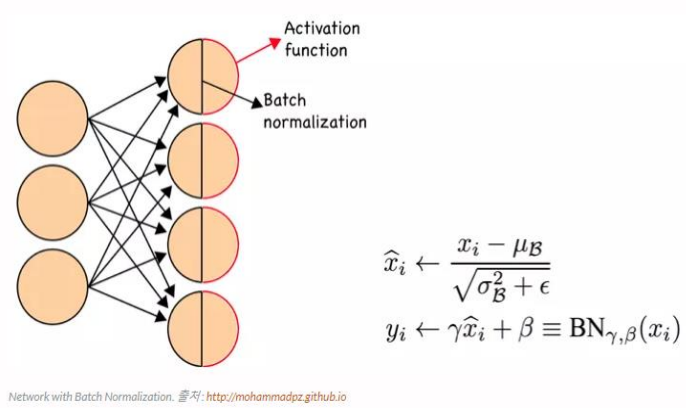
https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EA%B5%AC%ED%98%84/
https://github.com/hunkim/DeepLearningZeroToAll/blob/master/lab-10-6-mnist_nn_batchnorm.ipynb
https://arxiv.org/abs/1502.03167

# 5. Batch Normalization (cont.)

- **‘10 Deep Learning Trends’** at NIPS 2015 [link]
  - ‘Brad Neuberg’ , NIPS (Neural Information Processing Systems) ,Rusia, 12,2015.
  1. Neural network architectures are getting more complex and sophisticated
  2. All the cool kids are using LSTMs
  3. Attention models are showing up
  4. Neural Turing Machines remain interesting but aren't being leveraged yet for real work
  5. Computer vision and NLP aren't separate silos anymore — deep learning for computer vision and NLP are cross-hybridizing each other
  6. Symbolic differentiation is becoming even more important
  7. Surprising results are happening with neural network model compression
  8. The intersection of deep and reinforcement learning continues
  9. If you aren't using batch normalization you should
     - Batch normalization is now considered a standard part of the neural network toolkit and was referenced throughout work at the conference.
  10. Neural network research and productionisation go hand in hand
- Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariance Shift)
  - ICML 2015 [link]

# 5. Batch Normalization (cont.)

- Batch Normalization
  - Batch Normalization은 기본적으로 Gradient Vanishing / Gradient Exploding 이 일어나지 않도록 하는 아이디어 중의 하나이다. 지금까지는 이 문제를 Activation 함수의 변화 (ReLU 등), Careful Initialization, small learning rate 등으로 해결하였지만, 이 논문에서는 이러한 간접적인 방법보다는 training 하는 과정 자체를 전체적으로 안정화하여 학습 속도를 가속시킬 수 있는 근본적인 방법을 찾고 싶어 했다.
  - 이들은 이러한 불안정화가 일어나는 이유가 'Internal Covariance Shift' 라고 주장하고 있다. Internal Covariance Shift라는 현상은 Network의 각 층이나 Activation 마다 input의 distribution이 달라지는 현상을 의미한다. 이 현상을 막기 위해서 간단하게 각 층의 input의 distribution을 평균 0, 표준편차 1인 input으로 normalize 시키는 방법을 생각해볼 수 있고, 이는 whitening이라는 방법으로 해결할 수 있다. Whitening은 기본적으로 들어오는 input의 feature들을 uncorrelated 하게 만들어주고, 각각의 variance를 1로 만들어주는 작업이다.



# 5. Batch Normalization (cont.)

**Batch Normalization Example**[link]

- MLP Batch Normalization

```
from keras.layers import Dense
from keras.layers import BatchNormalization
...
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1))
```

- RNN Batch Normalization

```
# example of batch normalization for a lstm
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import BatchNormalization
...
model.add(LSTM(32))
model.add(BatchNormalization())
model.add(Dense(1))
...
```

- CNN Batch Normalization

```
# example of batch normalization for an cnn
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import BatchNormalization
...
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dense(1))
...
```

# Summary

- Learning rate
  - Learning rate
  - Too large : Over shooting
  - Too small : Local minimum
- Data preprocessing
  - Zero centered data
  - Normalized data
  - Regularization
  - dropout
- Overfitting
  - More training data
  - Reduce the number of features
  - Regularization
- Optimizers
- Batch Normalization

- Examples
  1. Learning rate
  2. Dataset normalization
  3. MNIST 10 digit image classification
  4. MNIST 10 digit image classification(2)

# Example 1. Learning rate

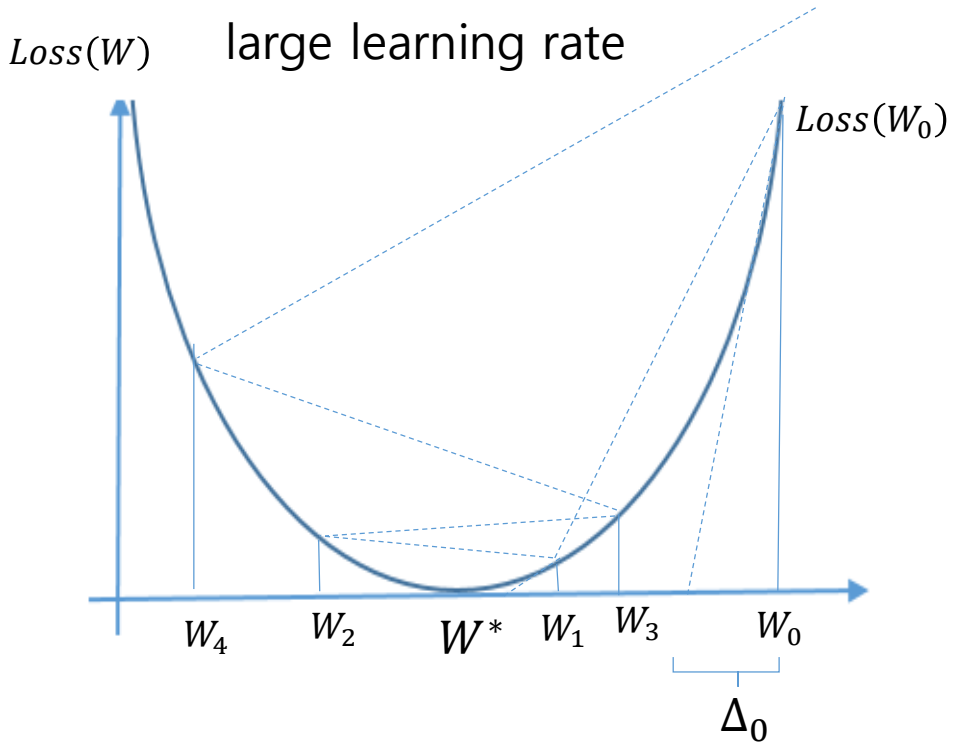
- Learning rate

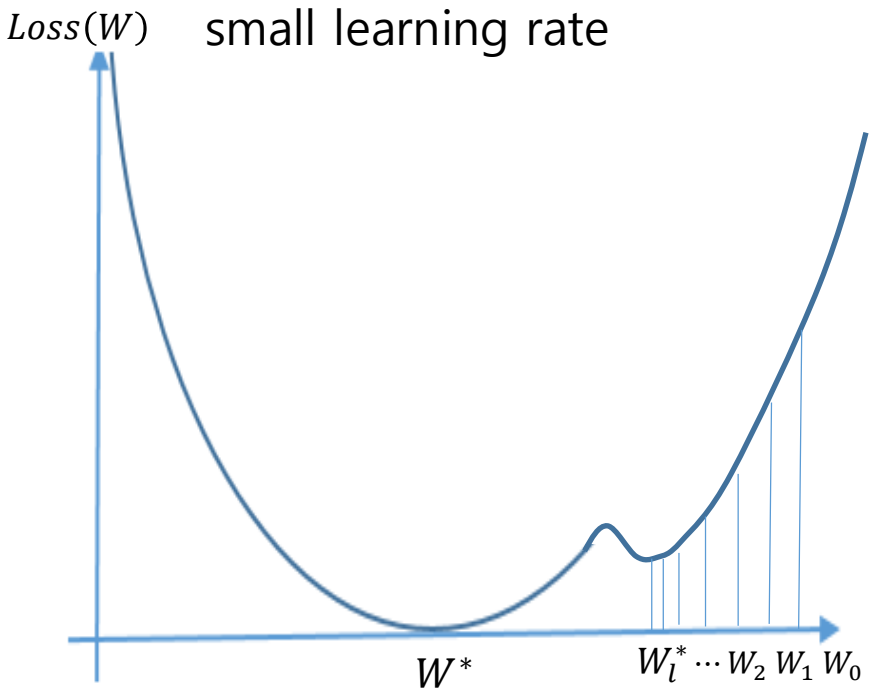
```
#데이터셋 생성
X          = np.array([[1, 2, 1], [1, 3, 2], [1, 3, 4], [1, 5, 5], [1, 7, 5], [1, 2, 5], [1, 6, 6], [1, 7, 7]])
Y_ohe      = np.array([[0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 1, 0], [0, 1, 0], [0, 1, 0], [1, 0, 0], [1, 0, 0]])
X_ev       = np.array([[2, 1, 1], [3, 1, 2], [3, 3, 4]])
Y_ev_ohe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

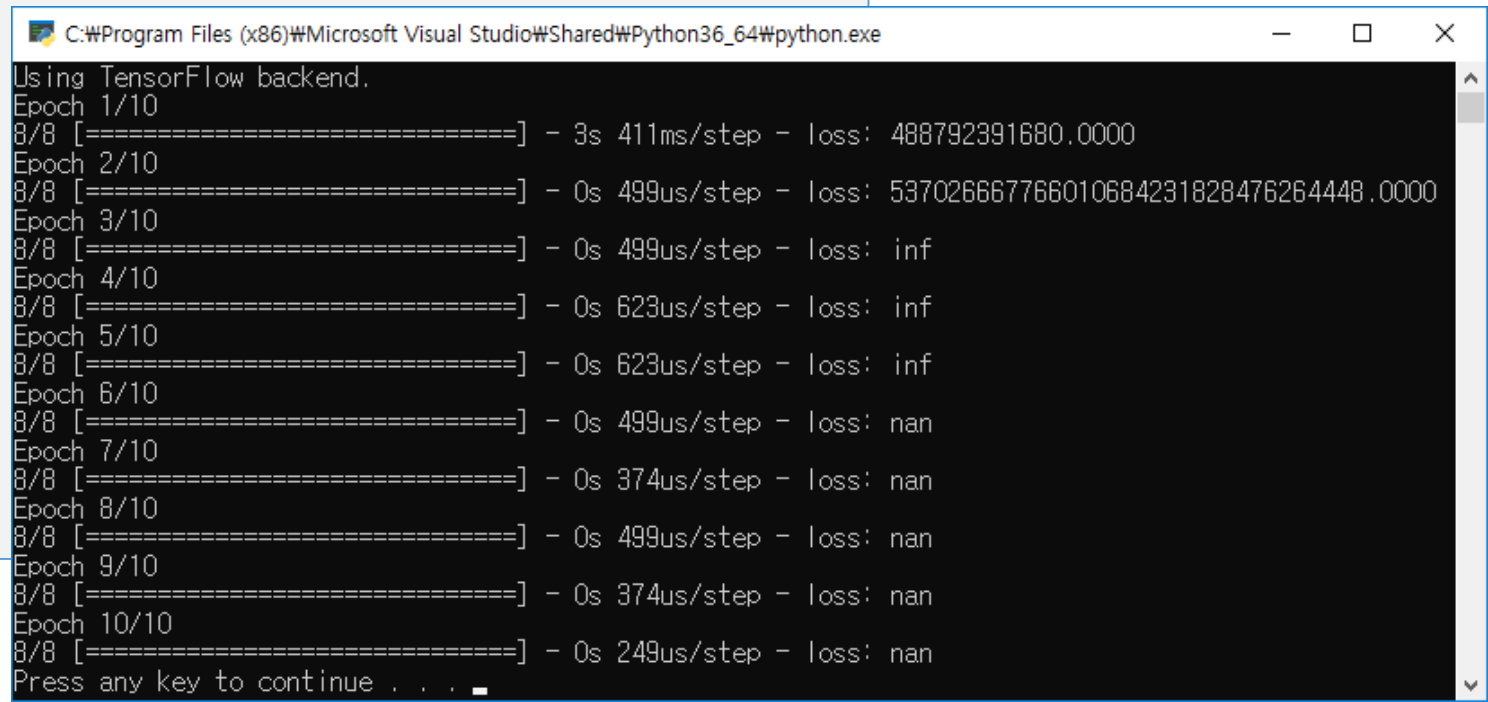
- Learning rate에 따른  학습 현상



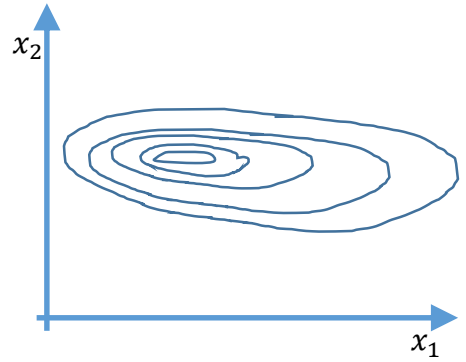
# Example 2. Dataset normalization

```
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
```



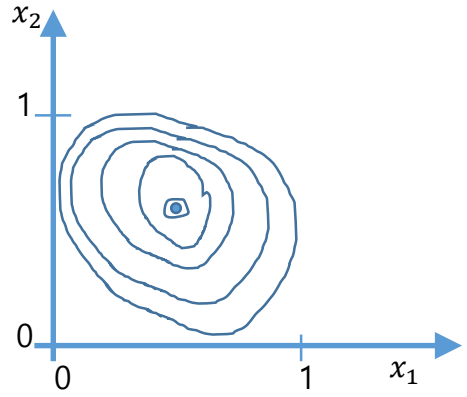
# Example 2. Dataset normalization

```
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]])
```



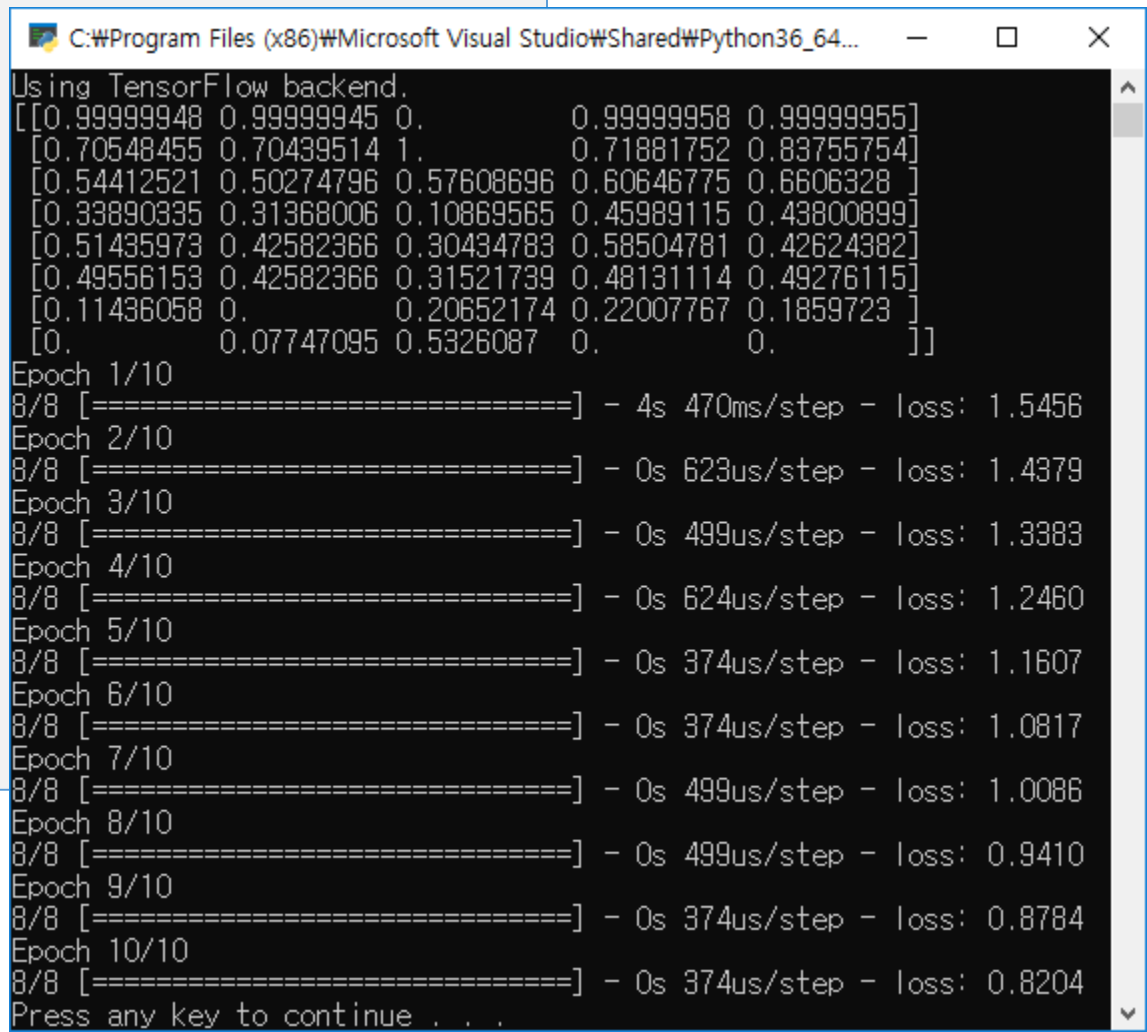

```
XY = MinMaxScaler(XY)
print(XY)
```

```
[[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

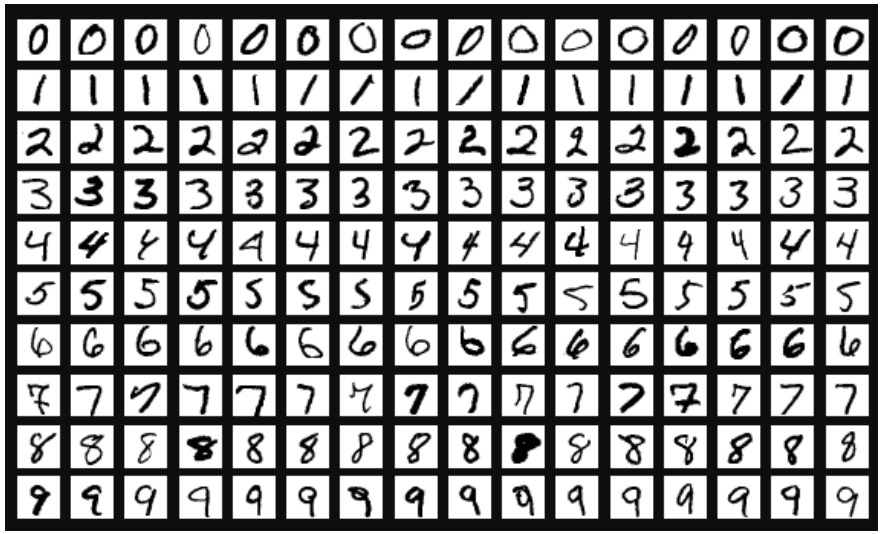
```
XY = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
  [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
[819.929993, 824.400024, 1438100, 818.97998, 824.159973],
[816, 820.958984, 1008100, 815.48999, 819.23999],
[819.359985, 823, 1188100, 818.469971, 818.97998],
[819, 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)
```



# Example 3. MNIST 10 digit image classifier

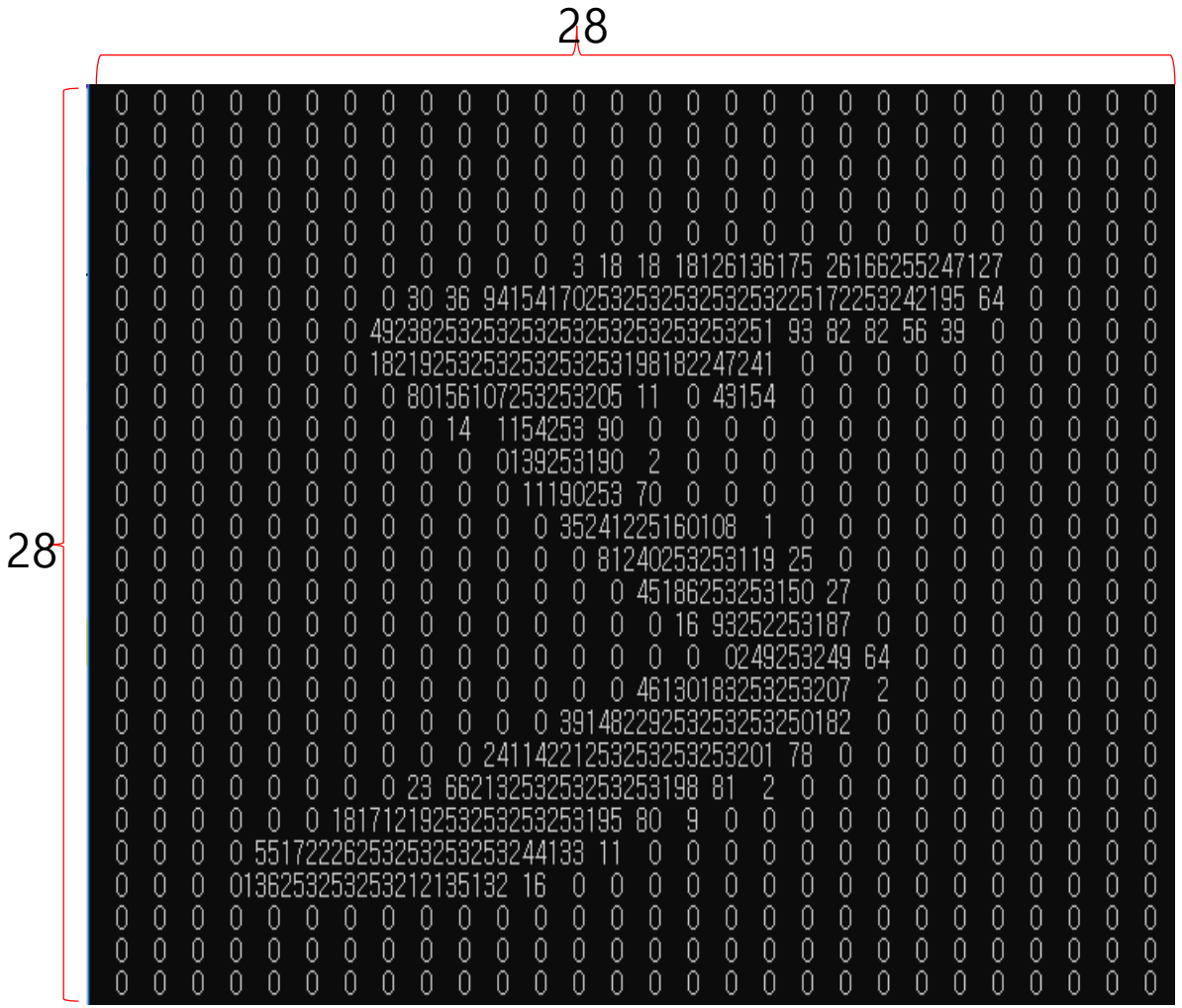
- MNIST data
  - for 10 digit image recognizer with softmax hypothesis and cross-entropy cost function
  - The **MNIST database** (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems
  - https://en.wikipedia.org/wiki/MNIST_database

- mnist 데이터 분석
  - 0부터 9까지 필기체 이미지, 이미지 크기 : 28x28 픽셀, 0~256
  - 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,)
  - Reshape및 정규화
    - 60000x28x28 => 60000x784
    - 60000장의 28x28이미지를 1차원의 784열로 변환하고 실수로 변환한 후 0~1.0으로 정규화 한다.
  - one_hot_encoding
    - 모든 이미지는 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()          #로드 이미지 dataset
X_train  = X_train.reshape(X_train.shape[0], 784).astype('float64') / 255    #0~1로 정규화
X_val    = X_validation.reshape(X_validation.shape[0], 784).astype('float64') / 255 #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,)
#X_validation.shape  :(10000, 28, 28)    Y_validation.shpe  : (10000,)
```

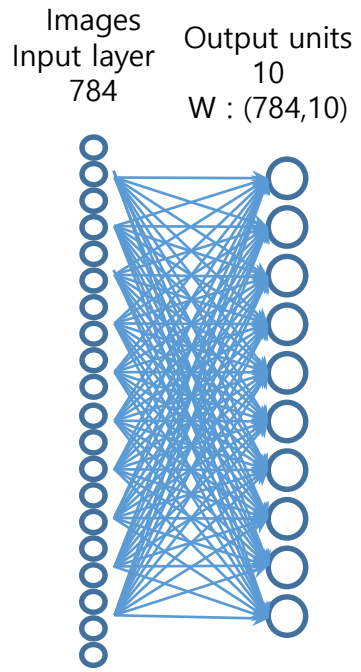
# Example 3. MNIST 10 digit image classifier (cont.)

- 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 = 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 classes
        optimizer='adam',       #adam opimizer
        metrics=['accuracy'])   #metrics

#모델 학습
#학습시 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 좋아지 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=32,              #학습의 최대 반복 회수
        batch_size=200,         #매학습시 학습할 데이터셋
        verbose=0,              #학습과정 출력 량(모드)
        callbacks=[cb_checkpoint, cb_early_stopping]) #모니터린 이벤트 개체 리스트
print(pd.DataFrame(history.history))
```

```
Epoch 00001: val_loss improved from inf to 0.43524, saving model to ./model/mnist-01-0.4352.hdf5
Epoch 00002: val_loss improved from 0.43524 to 0.34579, saving model to ./model/mnist-02-0.3458.hdf5
Epoch 00003: val_loss improved from 0.34579 to 0.31477, saving model to ./model/mnist-03-0.3148.hdf5
Epoch 00004: val_loss improved from 0.31477 to 0.29872, saving model to ./model/mnist-04-0.2987.hdf5
```

```
Epoch 00025: val_loss improved from 0.26160 to 0.26120, saving model to ./model/mnist-25-0.2612.hdf5
Epoch 00026: val_loss did not improve from 0.26120
Epoch 00027: val_loss did not improve from 0.26120
Epoch 00028: val_loss did not improve from 0.26120
Epoch 00029: val_loss did not improve from 0.26120
Epoch 00030: val_loss did not improve from 0.26120
Epoch 00031: val_loss did not improve from 0.26120
Epoch 00032: val_loss did not improve from 0.26120
```

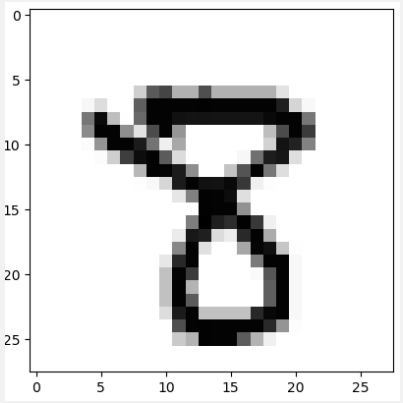
| | val_loss | val_accuracy | loss | accuracy |
|---|---|---|---|---|
| 0 | 0.435241 | 0.8915 | 0.804401 | 0.803000 |
| 1 | 0.345785 | 0.9103 | 0.401706 | 0.894583 |
| 2 | 0.314768 | 0.9161 | 0.344418 | 0.907183 |
| 3 | 0.298723 | 0.9168 | 0.318097 | 0.913117 |
| 4 | 0.288094 | 0.9185 | 0.302631 | 0.916400 |
| 5 | 0.280990 | 0.9229 | 0.291981 | 0.919100 |
| 6 | 0.276642 | 0.9225 | 0.284668 | 0.921033 |
| 7 | 0.273216 | 0.9236 | 0.278510 | 0.922883 |
| 8 | 0.272397 | 0.9245 | 0.273808 | 0.923733 |
| 9 | 0.268702 | 0.9245 | 0.269880 | 0.924967 |
| 10 | 0.267866 | 0.9251 | 0.266578 | 0.925717 |
| 11 | 0.265853 | 0.9265 | 0.263722 | 0.927000 |
| 12 | 0.267327 | 0.9246 | 0.261532 | 0.927267 |
| 13 | 0.263356 | 0.9271 | 0.259301 | 0.927817 |
| 14 | 0.264646 | 0.9261 | 0.257375 | 0.928850 |
| 15 | 0.266565 | 0.9254 | 0.255930 | 0.929367 |
| 16 | 0.263036 | 0.9267 | 0.253820 | 0.929783 |
| 17 | 0.264356 | 0.9269 | 0.252680 | 0.930217 |
| 18 | 0.262571 | 0.9274 | 0.251175 | 0.931233 |
| 19 | 0.263560 | 0.9280 | 0.250213 | 0.930950 |
| 20 | 0.266105 | 0.9254 | 0.249160 | 0.931133 |
| 21 | 0.261597 | 0.9279 | 0.248019 | 0.931617 |
| 22 | 0.261873 | 0.9272 | 0.247178 | 0.931467 |
| 23 | 0.261984 | 0.9275 | 0.246444 | 0.932217 |
| 24 | 0.261196 | 0.9279 | 0.245306 | 0.932550 |
| 25 | 0.262838 | 0.9274 | 0.244615 | 0.932933 |
| 26 | 0.262251 | 0.9276 | 0.243934 | 0.933133 |
| 27 | 0.261790 | 0.9277 | 0.243081 | 0.933317 |
| 28 | 0.262959 | 0.9268 | 0.242550 | 0.933333 |
| 29 | 0.261441 | 0.9285 | 0.241952 | 0.933717 |
| 30 | 0.262244 | 0.9273 | 0.241305 | 0.933667 |
| 31 | 0.263750 | 0.9266 | 0.240600 | 0.934117 |

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

- 모델평가

```
#최고로 학습된 모델의 성능 (분류의 정밀도 acc)을 평가셋으로  평가
model=load_model('model/mnist-25-0.2610.hdf5')          #저장된 최고의 모델을 로드
score=odel.evaluate(X_validation, Y_validation_ohe)      #평가용 셋으로 평가
print('score :',score)                          #[0.2637501769691706, 0.9265999794006348]로스와 인식률

#평가용 데이터셋으로 모델을 평가하여 정밀도(acc)를 출력한다.
print('\nAccuracy: {:.4f}'.format(score[1]))      #Accuracy: 0.9266

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

#model로 이미지를 예측(인식)하고 argmax하여 인식결과를 계산한다.
preds=model.predict(X1)             # 추출된 이미지 X1의 예측(인식) 10개의 벡터
print('preds : ' ,preds)                        # 예측결과 출력
#[[5.4910076e-10 1.3713418e-11 2.1246641e-03 4.2107690e-02 1.9038754e-06
9.0653886e-04 6.7351021e-11 3.7085364e-04 9.5318204e-01 1.3063980e-03]]
Y1_hat= np.argmax(preds)            # 10개중 최대 값의 인덱스 예측결과 계산
#이미지 번호(ground truth) Y1 와 인식결과 Y1-hat를 출력하여 비교한다.
print('\nlabel : {}     y_hat:{} '.format(Y1,Y1_hat))    #label : [8]     y_hat : 8

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

```
10000/10000 [==============================] - 10s 989us/step
score : [0.2637501769691706, 0.9265999794006348]

Accuracy: 0.9266

poreds : [[5.4910076e-10 1.3713418e-11 2.1246641e-03 4.2107690e-02 1.9038754e-06
9.0653886e-04 6.7351021e-11 3.7085364e-04 9.5318204e-01 1.3063980e-03]]

label : [8]      y_hat : 8
```



# 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 NN

```
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