

# 심층신경망 성능향상



# 얕은 신경망 (Shallow Neural Network)

# MNIST 데이터셋

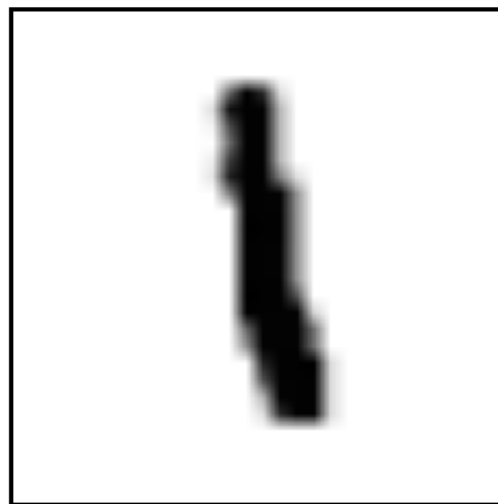
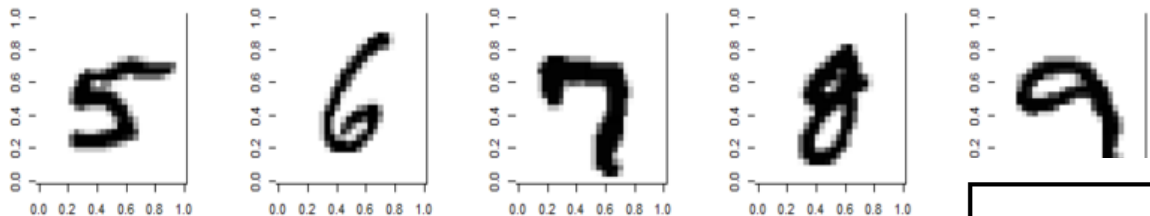
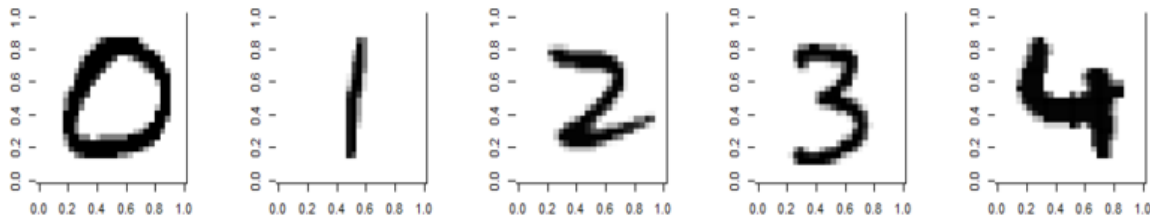
- MNIST : Modified National Institute of Standards and Technology database
- 미국의 고등학생과 인구조사국 직원들이 쓴 손글씨 숫자
- 각 숫자 이미지는 가로 28픽셀, 세로 28픽셀의 정사각형 모양
- Train 데이터셋 : 60,000개
- Test 데이터셋 : 10,000개



# MNIST 데이터셋

한 자리 숫자를 배열로 표현할 수 있습니다.

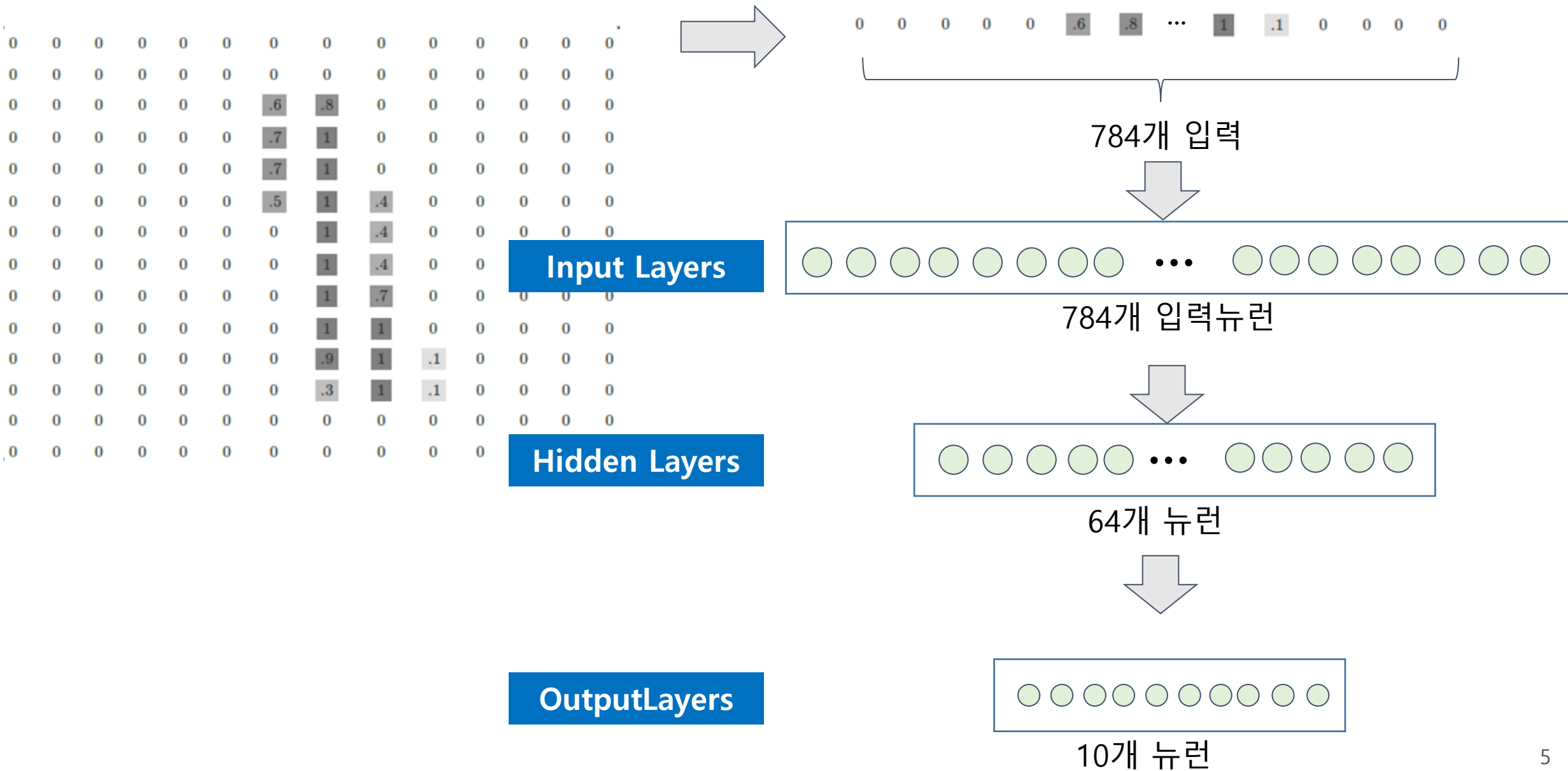
컴퓨터에 저장되는 이미지는 픽셀(pixel) 단위로 0~1 사이의 숫자로 표현되어 있습니다.



28

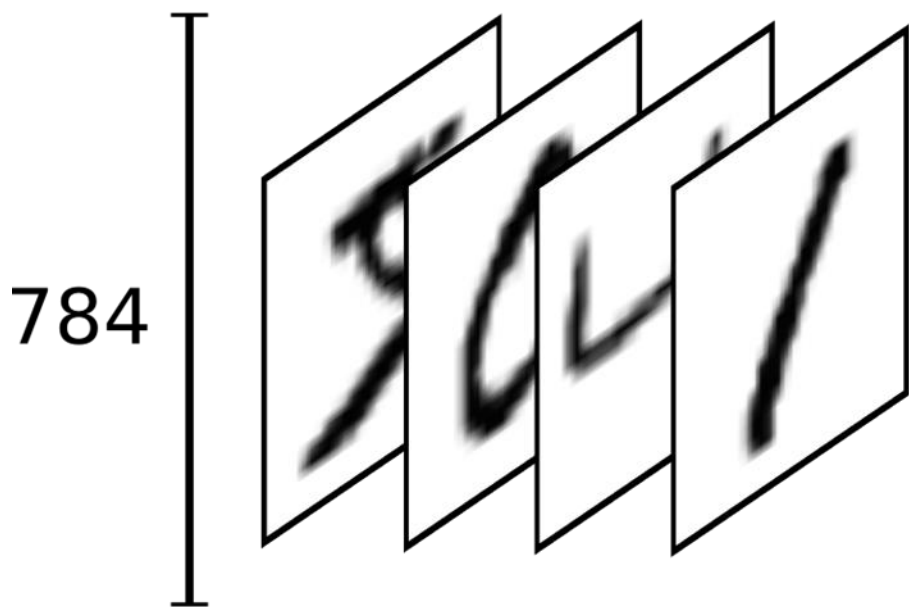
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	.6	.8	0	0	0	0	0	0
0	0	0	0	0	0	0	.7	1	0	0	0	0	0	0
0	0	0	0	0	0	0	.7	1	0	0	0	0	0	0
0	0	0	0	0	0	0	.5	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	0	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	0	1	.4	0	0	0	0	0
0	0	0	0	0	0	0	0	1	.7	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	.9	1	.1	0	0	0	0
0	0	0	0	0	0	0	0	.3	1	.1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

# 신경망 구조



# 입력 데이터(X)

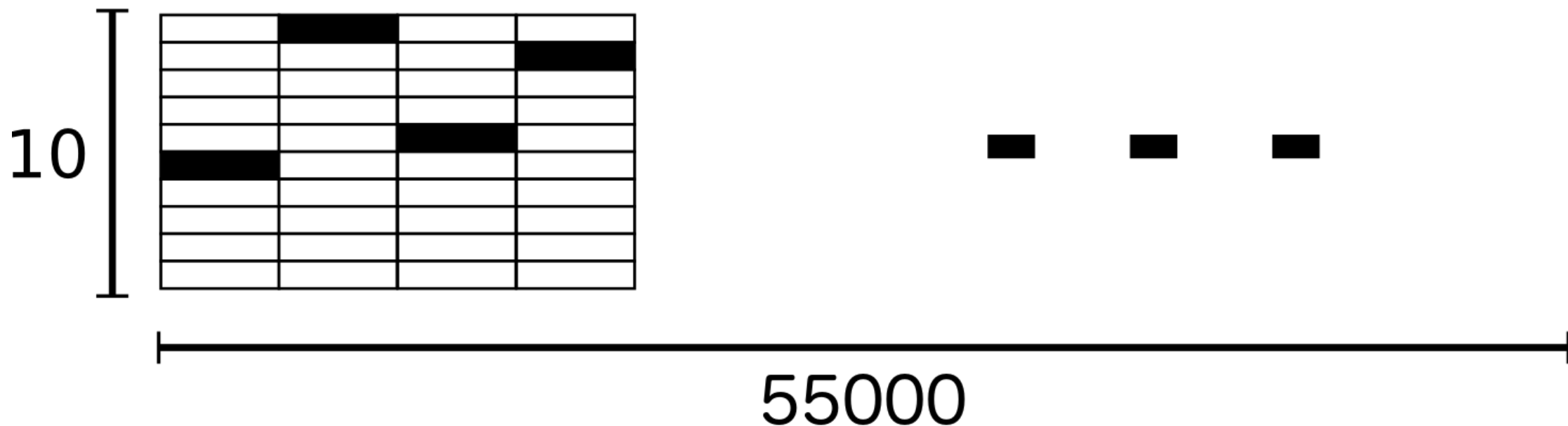
- 0부터 9까지의 10가지 손글씨 이미지를 학습하도록 훈련(training)시킵니다.
- 55,000장의 훈련 이미지 각각을 픽셀(pixel)단위로 읽어서 입력 레이어에 넣어줍니다.
- 이미지 한 장이 가로 28픽셀, 세로 28픽셀이므로 이미지 한 장당 784픽셀의 정보량입니다.



# 레이블 데이터(y)

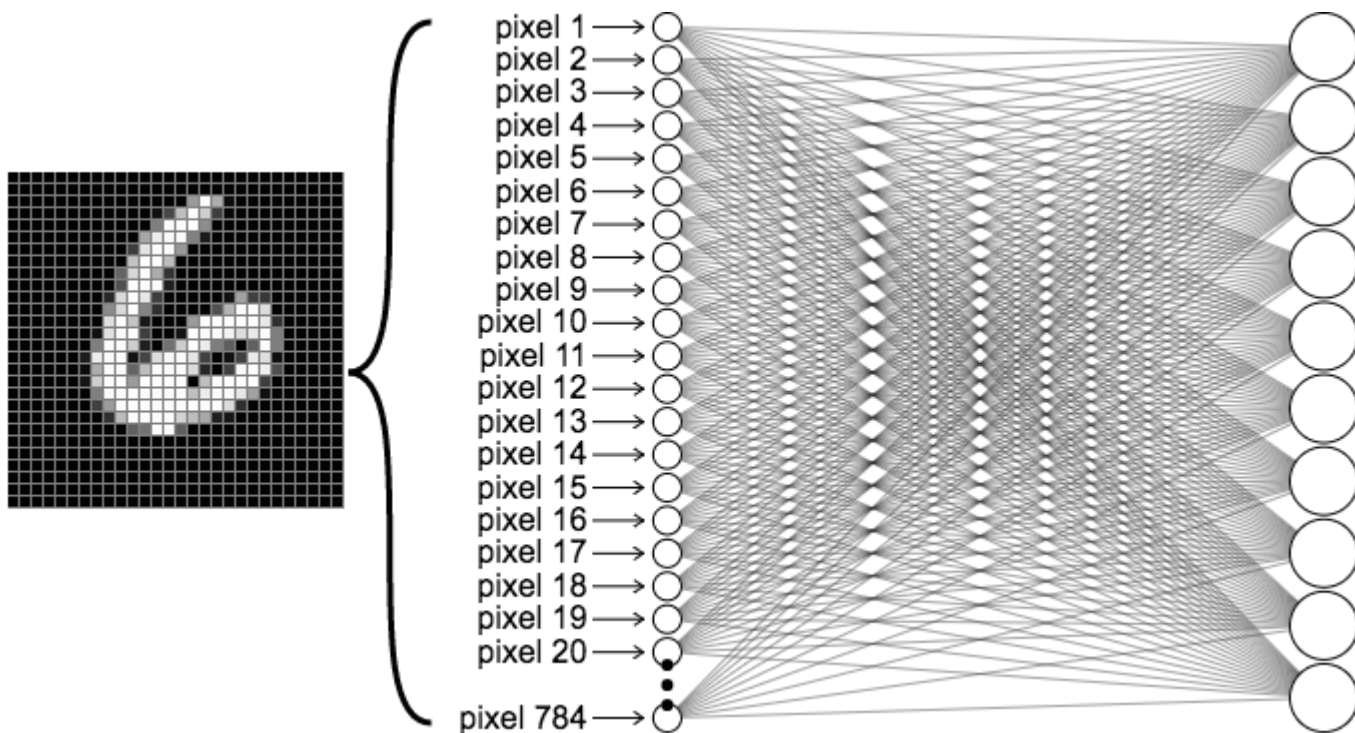
- 원-핫 인코딩(one-hot encoding) 방식으로 출력 레이어에 넣어줍니다.
- 숫자 0은 [1 0 0 0 0 0 0 0 0]
- 숫자 1은 [0 1 0 0 0 0 0 0 0]
- 숫자 2는 [0 0 1 0 0 0 0 0 0]
- 숫자 3은 [0 0 0 1 0 0 0 0 0]

mnist.train.ys



# 신경망 구성

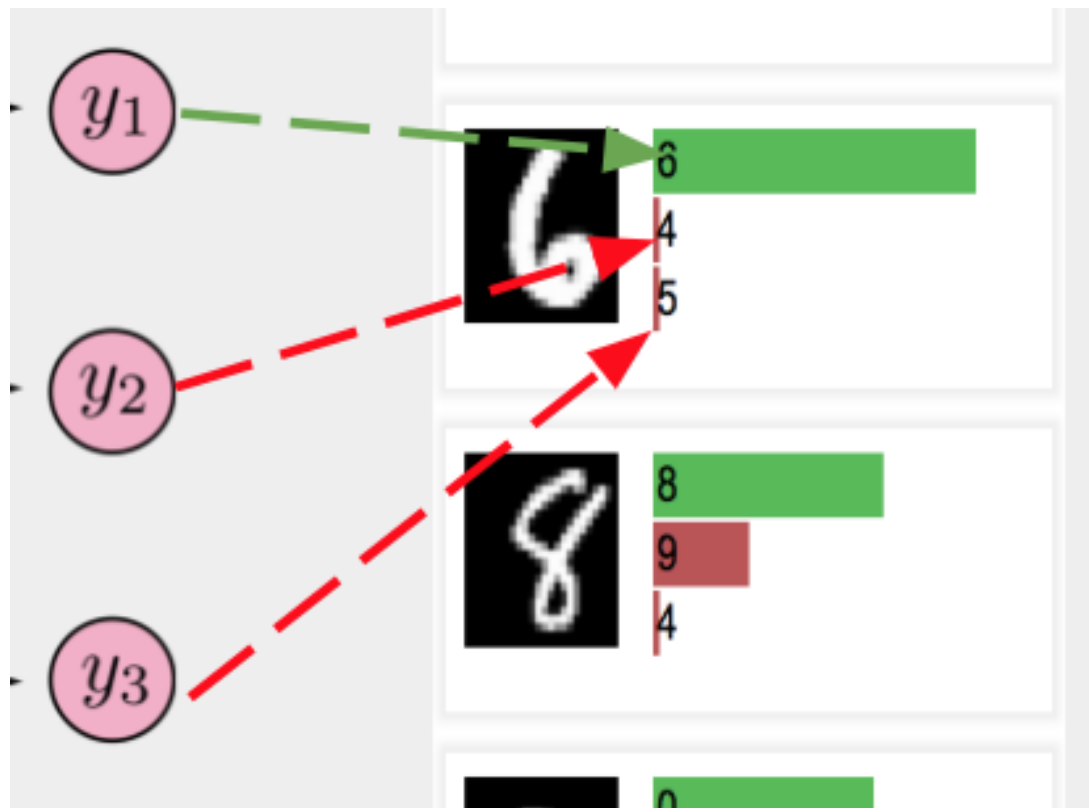
- 인풋 레이어는 784개( $28 \times 28$ )의 노드로 구성되어 있습니다.
- 입력 레이어(784개 노드)와 출력 레이어(10개 노드) 사이에 히든 레이어를 여러 개 구성합니다.





# 결과

- 출력 레이어의 10개 노드가 가진 값의 크기를 비교해 그중에 가장 높은 점수를 정답으로 인정합니다.



# MNIST 분류기 소스코드 - ANN

## 라이브러리 импорт

```
[1] import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
```

```
[2] print(tf.__version__)
```

2.6.0

## MNIST 데이터 로드

```
[3] (X_train, y_train), (X_valid, y_valid) = mnist.load_data()
```

## MNIST 데이터 확인

```
[4] X_train.shape
```

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

```
[5] y_train.shape
```

```
(60000,)
```

[6] X\_train[0]

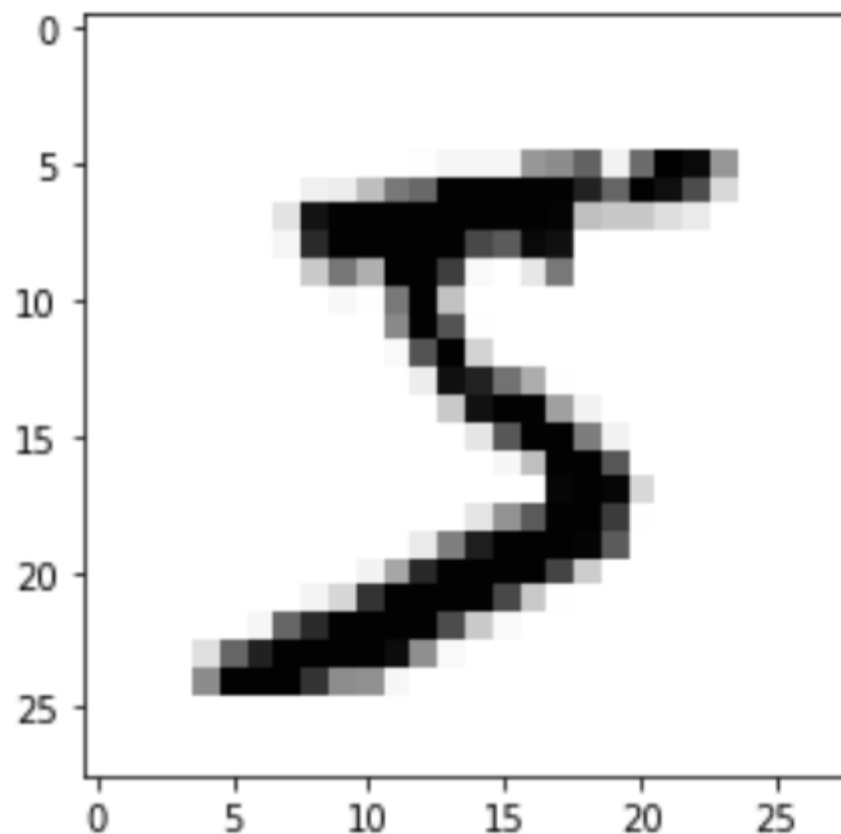
```
array([[ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  3,
        18, 18, 18, 126, 136, 175, 26, 166, 255, 247, 127, 0, 0,
        0,  0],
```

```
[7] y_train[0:10]
```

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4], dtype=uint8)
```

```
[8] plt.imshow(X_train[0], cmap='Greys')
```

<matplotlib.image.AxesImage at 0x7fdabaec9590>



## 데이터 전처리

신경망의 입력층과 출력층의 크기에 맞도록 데이터의 크기를 변경

```
[10] X_train = X_train.reshape(60000, 784).astype('float32')  
      X_valid = X_valid.reshape(10000, 784).astype('float32')
```

데이터 정규화 : 입력값을 0과 1사이의 범위로 변경

```
[11] 1/255.0
```

```
0.00392156862745098
```

```
[12] X_train /= 255  
      X_valid /= 255
```

[13] X\_train[0]

0.9764706	, 0.2509804	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.18039216	, 0.50980395	,
0.7176471	, 0.99215686	, 0.99215686	, 0.8117647	, 0.00784314	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.15294118	,
0.5803922	, 0.8980392	, 0.99215686	, 0.99215686	, 0.99215686	,
0.98039216	, 0.7137255	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.09411765	, 0.44705883	, 0.8666667	, 0.99215686	, 0.99215686	,
0.99215686	, 0.99215686	, 0.7882353	, 0.30588236	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,
0.	, 0.	, 0.	, 0.	, 0.	,

## 레이블을 원핫 인코딩으로 바꾸기

```
[14] n_classes = 10  
     y_train = keras.utils.to_categorical(y_train, n_classes)  
     y_valid = keras.utils.to_categorical(y_valid, n_classes)
```

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



## 신경망 구조 설계

```
[16] model = Sequential()  
      model.add(Dense(64, activation='sigmoid', input_shape=(784,)))  
      model.add(Dense(10, activation='softmax'))
```

```
[17] model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	50240
dense_1 (Dense)	(None, 10)	650

Total params: 50,890

Trainable params: 50,890

Non-trainable params: 0

## 신경망 모델 컴파일

```
[20] model.compile(loss='mean_squared_error',  
                  optimizer=SGD(learning_rate=0.01),  
                  metrics=['accuracy'])
```

## 신경망 모델 훈련

```
[21] model.fit(X_train, y_train,  
              batch_size=128,  
              epochs=200,  
              verbose=1,  
              validation_data=(X_valid, y_valid))
```

Epoch 1/200  
469/469 [=====] - 2s 3ms/step - loss: 0.0923 - accuracy: 0.0986 - val\_loss: 0.0915 - val\_accuracy: 0.0978  
Epoch 2/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0910 - accuracy: 0.1003 - val\_loss: 0.0906 - val\_accuracy: 0.1024  
Epoch 3/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0903 - accuracy: 0.1055 - val\_loss: 0.0901 - val\_accuracy: 0.1128  
Epoch 4/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0899 - accuracy: 0.1160 - val\_loss: 0.0897 - val\_accuracy: 0.1253  
Epoch 5/200  
469/469 [=====] - 1s 2ms/step - loss: 0.0895 - accuracy: 0.1315 - val\_loss: 0.0894 - val\_accuracy: 0.1447  
Epoch 6/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0892 - accuracy: 0.1551 - val\_loss: 0.0891 - val\_accuracy: 0.1704  
Epoch 7/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0890 - accuracy: 0.1839 - val\_loss: 0.0888 - val\_accuracy: 0.2055  
Epoch 8/200  
469/469 [=====] - 1s 2ms/step - loss: 0.0887 - accuracy: 0.2236 - val\_loss: 0.0886 - val\_accuracy: 0.2449  
  
.....  
  
Epoch 197/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0287 - accuracy: 0.8570 - val\_loss: 0.0277 - val\_accuracy: 0.8639  
Epoch 198/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0285 - accuracy: 0.8576 - val\_loss: 0.0276 - val\_accuracy: 0.8646  
Epoch 199/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0284 - accuracy: 0.8579 - val\_loss: 0.0275 - val\_accuracy: 0.8652  
Epoch 200/200  
469/469 [=====] - 1s 3ms/step - loss: 0.0283 - accuracy: 0.8582 - val\_loss: 0.0274 - val\_accuracy: 0.8656

## 신경망 모델 정확도 평가

```
[22] model.evaluate(X_valid, y_valid)
```

```
313/313 [=====] - 0s 1ms/step - loss: 0.0274 - accuracy: 0.8656  
[0.02739546447992325, 0.8655999898910522]
```

# MNIST 분류모델 구현 실습 - ANN



mnist\_ann

# 심층 신경망 (Deep Neural Network)

# 손실함수 (Loss function)

- 경사하강법, 역전파로 인공신경망 파라미터를 학습하고, 손실함수(비용함수)로 출력의 평가 범위를 정량화

## ■ 이차 손실 함수(평균 제곱 오차)

- 제곱하면  $y$ 가  $\hat{y}$  보다 크든지 작든지 상관없이 두 값의 차이를 양수로 만듦
- 제곱하면  $y$ 와  $\hat{y}$  차이가 작은 것보다 큰 것이 불리

$$Loss = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

↓
↓  
실제값
모델이 예측한 값

# Cross Entropy 손실 함수

- MSE 대신 Cross-entropy 손실함수를 사용하여 포화 뉴런이 학습속도에 미치는 영향을 최소화
- $y$ 와  $\hat{y}$  사이의 차이가 클수록 기하급수적으로 비용이 증가됨
- $y$ 와  $\hat{y}$  사이의 차이가 클수록 뉴런이 빨리 학습함
- 비용이 클수록 크로스 엔트로피를 사용한 신경망이 더 빠르게 학습합니다.

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$



# Cross Entropy 손실 함수

```
[1] from numpy import log
```

```
[2] def cross_entropy(y, a):  
    return -1*(y*log(a) + (1-y)*log(1-a))
```

```
[3] cross_entropy(1, 0.999)
```

```
↳ 0.0010005003335835344
```

```
[4] cross_entropy(1, 0.8)
```

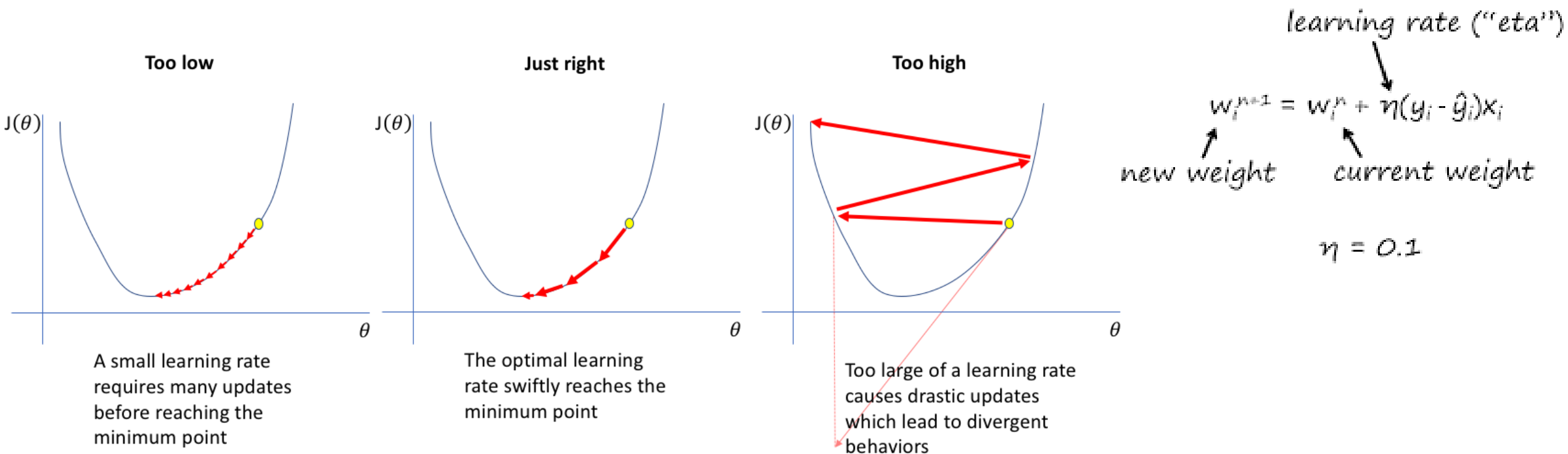
```
↳ 0.2231435513142097
```

```
[5] cross_entropy(1, 0.1)
```

```
↳ 2.3025850929940455
```

# 학습률 (Learning Rate)

- 학습률은 0.01 또는 0.001 로 설정
- 훈련이 매우 느리다면(손실이 조금만 감소한다면) 학습률의 소수점 자리수를 줄임 (예: 0.001 → 0.01)
- 모델 학습이 안 되면 학습률 소수점 자리를 증가 (예: 0.001 → 0.0001)



# 확률적 경사법 (SGD, Stochastic Gradient Descent)

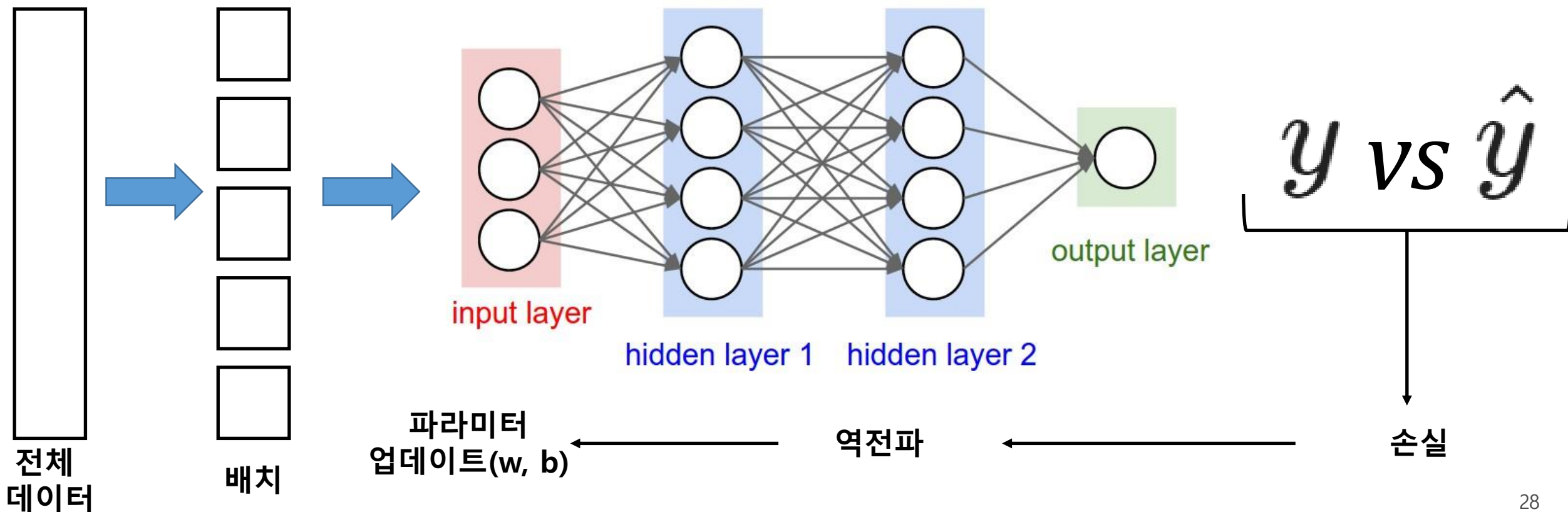
- 대규모 데이터셋은 전체 데이터를 메모리에 적재하기 어렵고, 수백만개의 파라미터를 가진 신경망 훈련의 계산복잡도로 기본 경사하강법은 비효율적입니다.
- 메모리와 계산의 제약에 대한 해결책으로, 훈련 데이터를 미니배치로 나누어 경사하강법을 수행하는 확률적 경사하강법이 있습니다.



$$\text{미니배치 개수} = \left\lceil \frac{\text{훈련 데이터 크기}}{\text{배치 크기}} \right\rceil$$

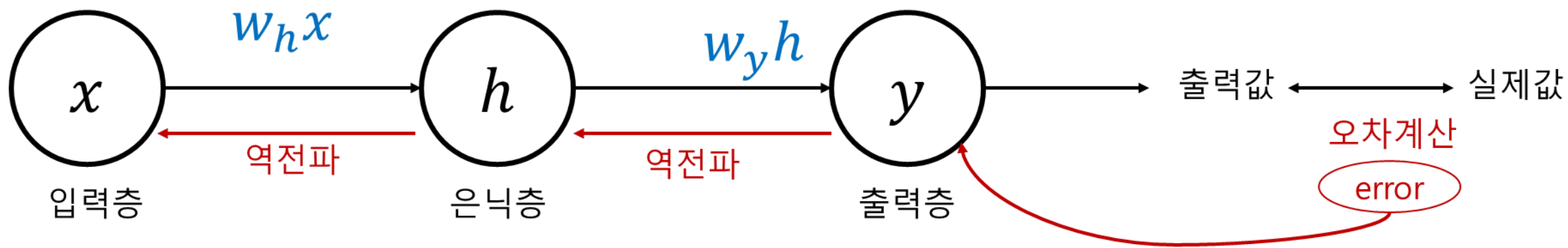
# 훈련과정

1. 미니배치  $x$ 를 샘플링합니다.
2.  $x$ 를 신경망에 통과시켜  $y$ 에 대한 예측  $\hat{y}$ 을 만듭니다.
3.  $y$ 와  $\hat{y}$ 을 비교하여 손실을 계산합니다.
4. 비용/손실을 기반으로 경사하강법을 적용해  $x$ 가  $y$ 를 더 잘 예측할 수 있도록  $w$ 와  $b$ 를 조정합니다.



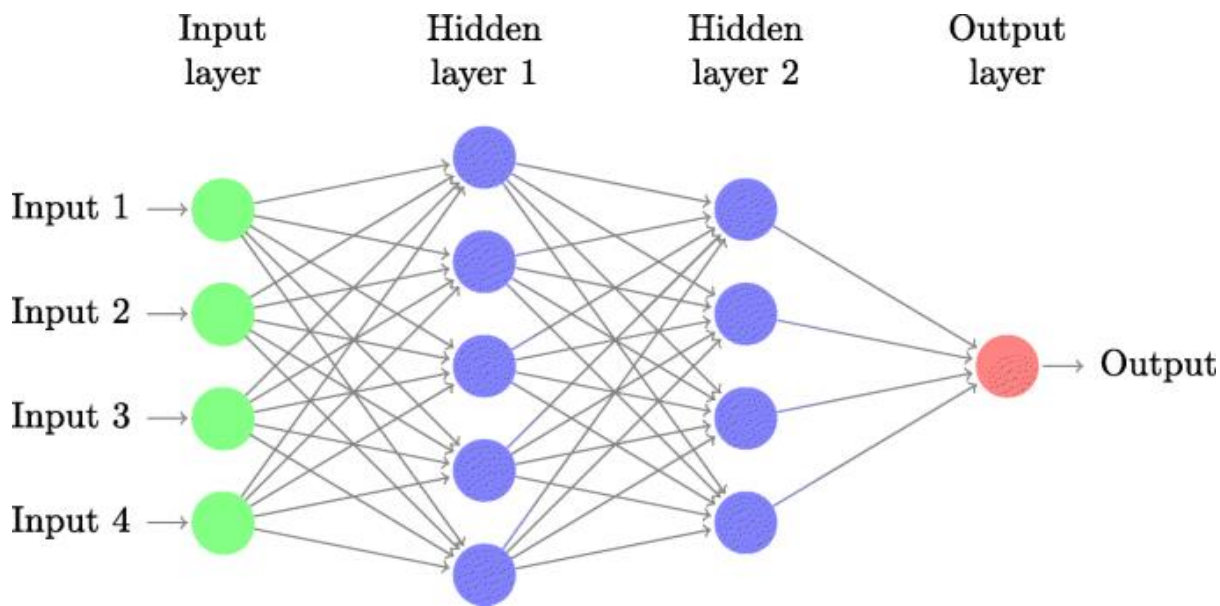
# 역전파 알고리즘

- 신경망 모델은 파라미터( $w$ ,  $b$ ) 값을 랜덤하게 초기화 합니다.
- 신경망 모델에  $x$ 값을 입력하면 신경망은 예측  $y_{\text{hat}}$ 을 출력합니다.
- 신경망에서 손실 최소화를 위해 가중치를 업데이트 해야 합니다.
- 역전파를 사용해 신경망에 있는 모든 가중치에 대한 비용함수의 Gradient를 계산합니다.
- 가중치에 대한 비용함수의 Gradient에 비례하여 가중치를 조정함으로써 역전파는 비용을 감소하는 방향으로 가중치를 변경할 수 있습니다.



# 은닉층과 뉴런의 개수

- 학습률, 배치 크기, 은닉층 개수, 뉴런 개수는 모델을 훈련하기 전에 지정하는 모델의 설정값입니다.
- 신경망이 예측할 정답  $y$ 가 추상적일 수록 은닉층을 추가합니다.
- 은닉층을 줄여도 비용이 증가하지 않으면 층을 줄입니다. 더 빠르게 훈련되고 컴퓨팅 자원이 적게 소요됩니다.
- 뉴런 개수가 너무 많으면 신경망의 계산복잡도가 증가하고, 너무 적으면 신경망의 정확도가 억제됩니다.
- 모델에 사용하는 데이터에 따라 표현할 저수준 특성이 많으면 신경망의 앞쪽 층에 많은 뉴런을 둡니다.
- 표현할 고수준 특성이 많다면 뒤쪽 층에 뉴런을 추가하는 것이 좋습니다.
- 많은 딥러닝 모델을 만들고 훈련하면, 적절한 은닉층 개수와 뉴런 개수에 대한 감을 얻게 될 것입니다.



# MNIST 분류기 소스코드 - DNN

```
[6] model = Sequential()  
    model.add(Dense(64, activation='relu', input_shape=(784,)))  
    model.add(Dense(64, activation='relu'))  
    model.add(Dense(10, activation='softmax'))
```

```
[7] model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	50240
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 10)	650

Total params: 55,050  
Trainable params: 55,050  
Non-trainable params: 0

```
[8] model.compile(loss='categorical_crossentropy',  
                  optimizer=SGD(learning_rate=0.1),  
                  metrics=['accuracy'])
```

## 모델 훈련하기

```
[9] model.fit(X_train, y_train,  
              batch_size=128,  
              epochs=20,  
              verbose=1,  
              validation_data=(X_valid, y_valid))
```

Epoch 1/20

469/469 [=====] - 2s 4ms/step - loss: 0.4619 - accuracy: 0.8685

Epoch 2/20

469/469 [=====] - 1s 3ms/step - loss: 0.2201 - accuracy: 0.9355

-----



# MNIST 분류모델 구현 실습 - DNN



mnist\_dnn

# 심층 신경망 성능향상 방법

# 가중치 초기화

- 가중치 초기화 기법은 뉴런 포화가 일어날 가능성을 감소시키고, 딥러닝 성능을 크게 끌어 올립니다.
- 기본적으로 텐서플로는 모델을 만들 때 합리적인 값으로  $w$ 와  $b$ 를 초기화 합니다.
- 그리고, 문제에 맞도록 기본값을 바꿀 수 있습니다.

## Bias 초기화

```
[3] b_init = Zeros()
```

## 가중치 초기화 - 표준 정규 분포

```
[4] w_init = RandomNormal(stddev=1.0)
```

## 가중치 초기화 - 세이버이어 글로럿 분포

```
[5] # w_init = glorot_normal()
```

## 가중치 초기화 - 세이버이어 글로럿 균등 분포

```
[6] # w_init = glorot_uniform()
```

# 가중치 초기화

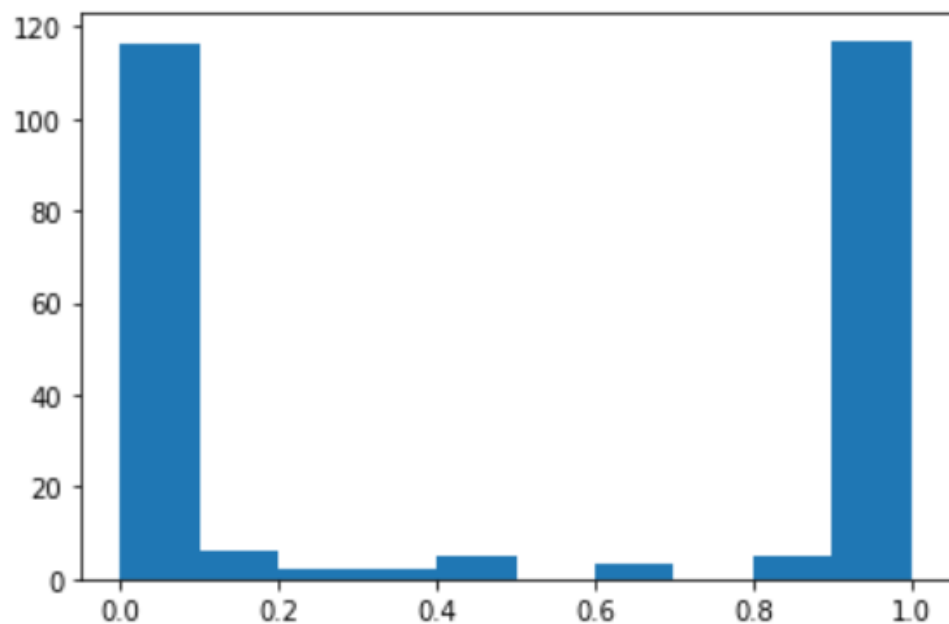
## 신경망 모델

```
[7] model = Sequential()  
    model.add(Dense(n_dense,  
                    input_dim=n_input,  
                    kernel_initializer=w_init,  
                    bias_initializer=b_init))  
  
    # model.add(Activation('sigmoid'))  
    # model.add(Activation('tanh'))  
    model.add(Activation('relu'))
```

# 가중치 초기화

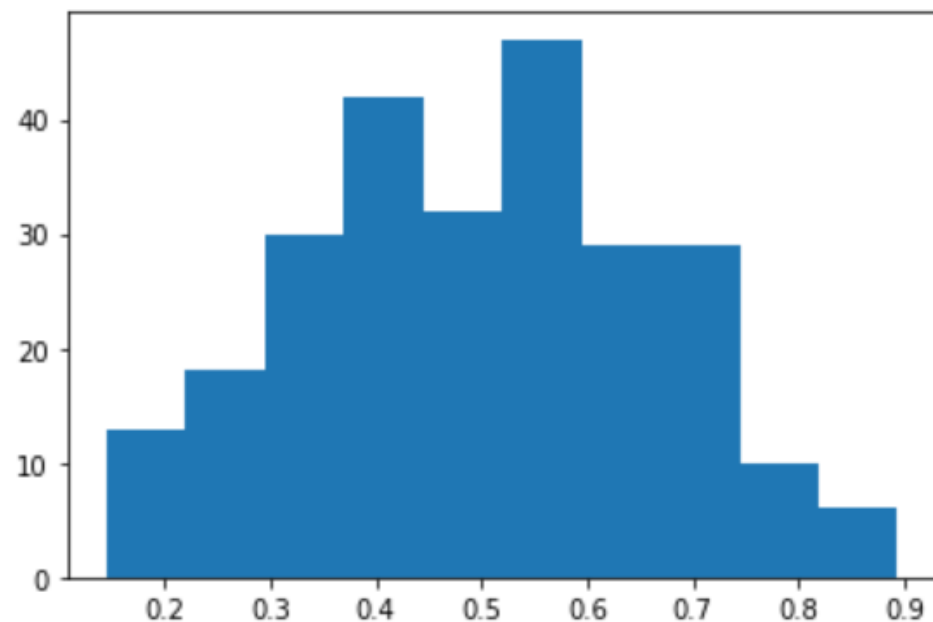
## 가중치 초기화 - 표준 정규 분포

```
[4] w_init = RandomNormal(stddev=1.0)
```



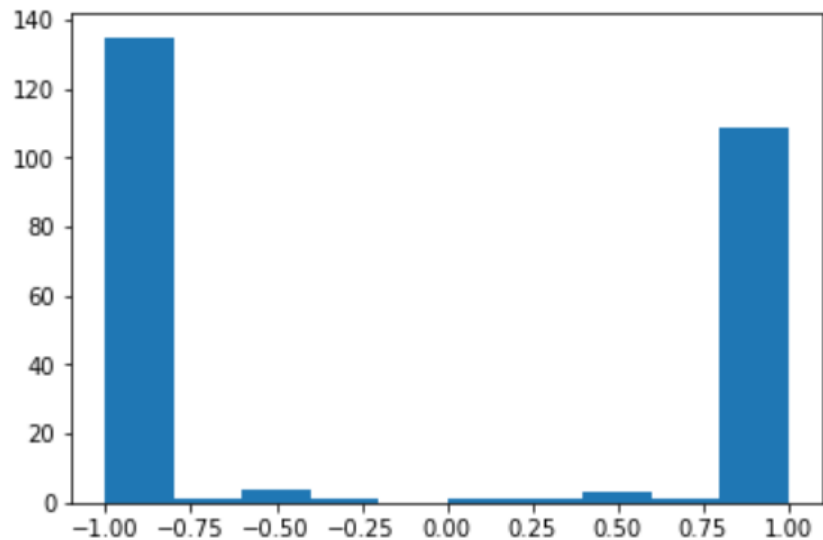
## 가중치 초기화 - 세이버 글로트 분포

```
[5] w_init = glorot_normal()
```

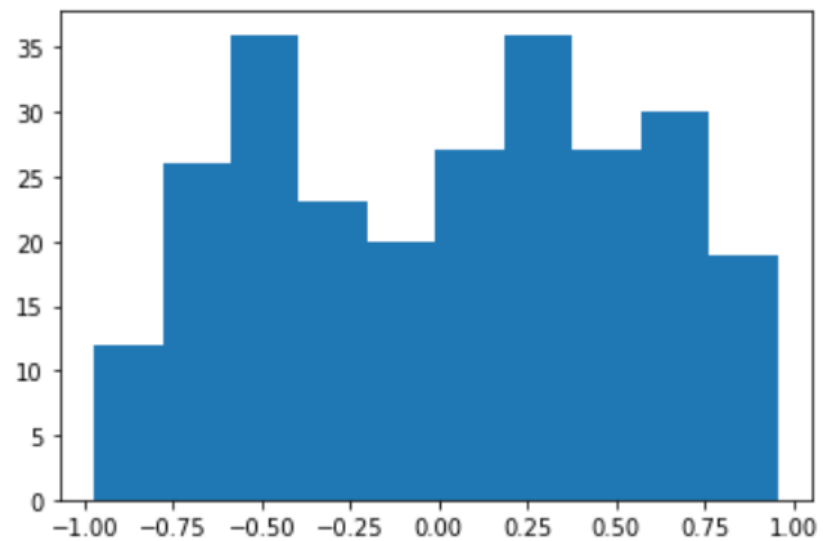


# 가중치 초기화

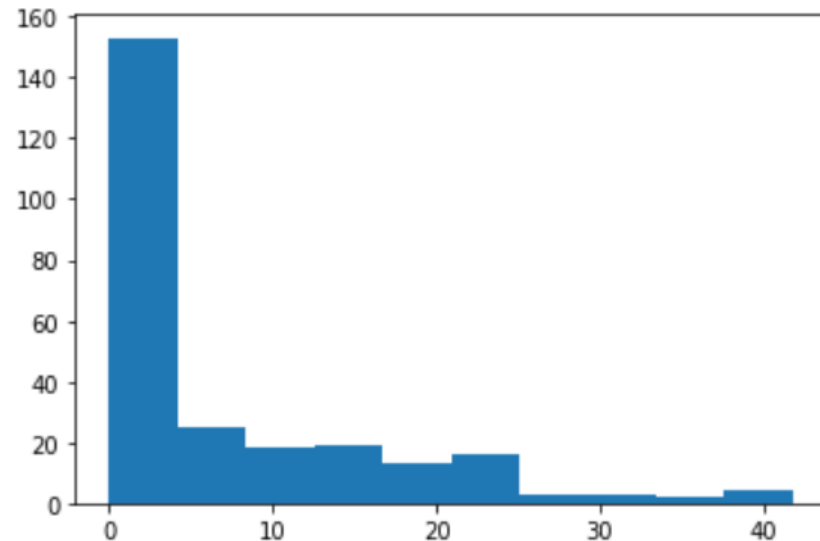
**tanh + 표준 정규분포 초기화**



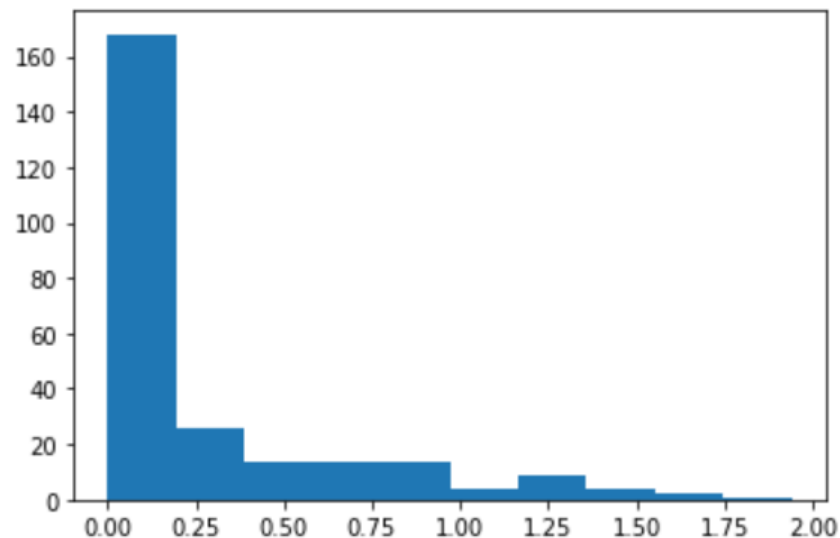
**tanh + 글로렛 초기화**



**ReLU+ 표준 정규분포 초기화**

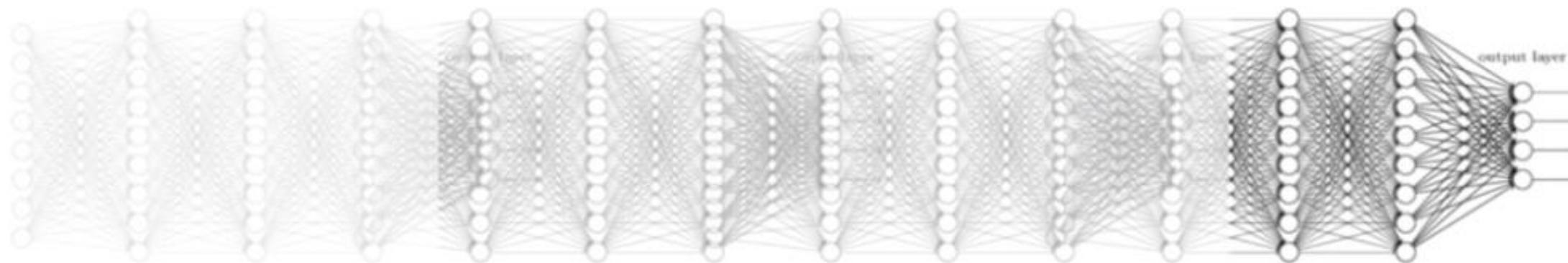


**ReLU+ 글로렛 초기화**



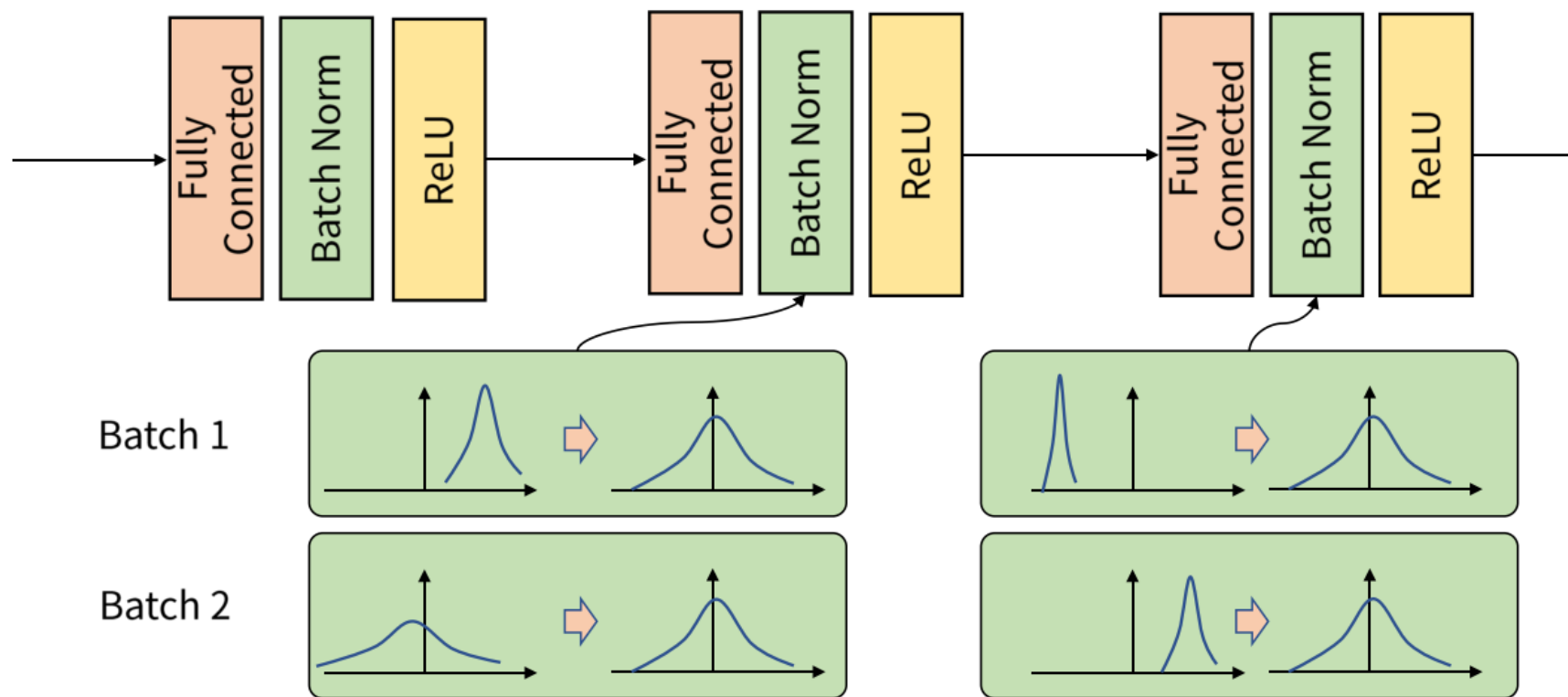
# 그레이디언트 소실 (Gradient Vanishing)

- 역전파가 출력층에서 입력층으로 작동하면서 손실이 최소화 되도록 신경망의 파라미터를 조정합니다.
- 각 파라미터는 손실에 대한 그레이디언트에 비례하여 조정 됩니다.
- 마지막 은닉층에서 첫 번째 은닉층으로 갈수록 손실에 비례한 파라미터의 그레이디언트가 소멸됩니다.
- 많은 은닉층을 신경망을 추가하면, 그레이디언트 소실 문제 때문에 멀리 떨어진 은닉층이 일정량 만큼 학습할 수 없습니다.
- X에서 y로 근사하는 신경망의 전체 능력이 훼손 됩니다.



# 배치 정규화(Batch Normalization)

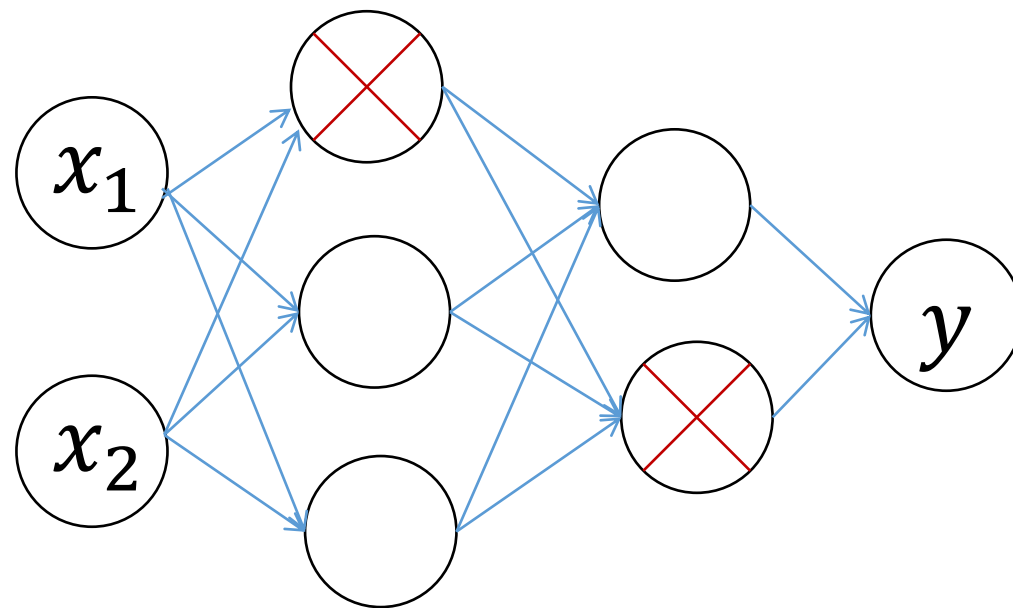
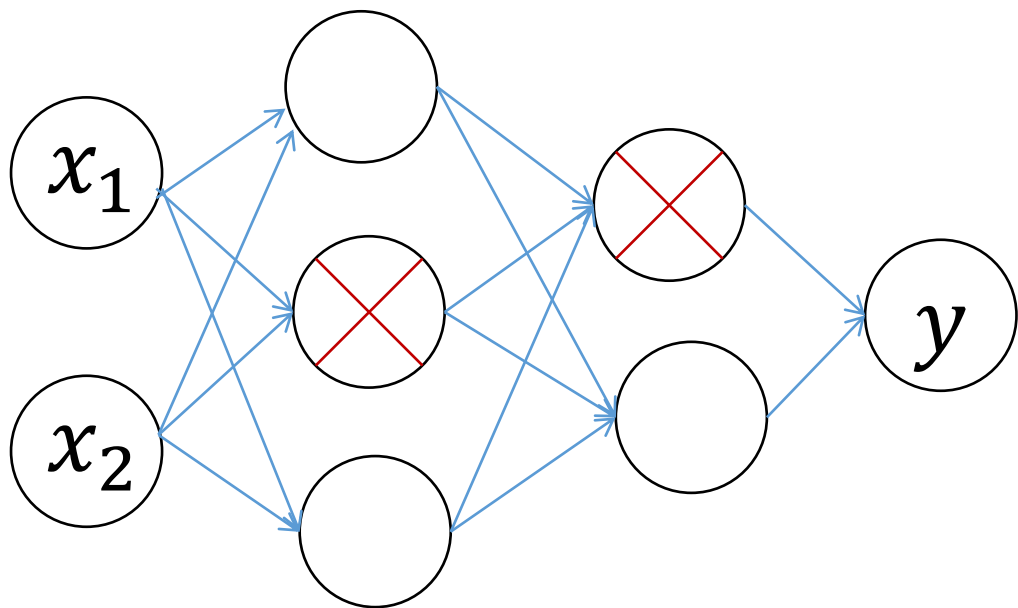
- 신경망이 훈련하는 동안 층의 파라미터 분포는 점진적으로 이동합니다.
- 이를 내부 공변량 변화(Internal Covariate Shift)라고 합니다.
- 배치 정규화를 통해 값의 분포를 평균이 0이고 표준 편차가 1인 분포로 다시 조정됩니다.
- 배치 정규화는 층간 독립적 학습, 더 높은 학습률 선택, 일반화가 잘 되게 하는 규제와 같은 긍정적인 효과가 있습니다.





# 과대적합(Overfitting) 방지 - Dropout

- 훈련 손실은 계속 내려가지만 검증 손실이 상승하는 상황을 과대적합(Overfitting) 이라고 합니다.
- 머신러닝 분야에서는 과대적합을 줄이기 위해 L1규제(Lasso), L2규제(Ridge)를 사용합니다.
- 딥러닝에서는 신경망에 특화된 드롭아웃 기법을 사용합니다.
- 드롭아웃(Dropout)은 각 층에 있는 랜덤한 일부 뉴런이 훈련하는 동안 없는 것처럼 동작하는 것입니다.
- 드롭아웃은 훈련 데이터셋의 일부 특성이 과도하게 신경망의 특정 정방향 계산을 주도하지 못하도록 만듭니다. 데이터의 특정 특성에 과도하게 의존하지 않게 됩니다.



# 과대적합(Overfitting) 방지 - Data Augmentation

- 훈련 데이터셋이 많을수록 모델의 일반화 성능이 높아집니다.
- Image Augmentation은 딥러닝 모델을 훈련하기 위해 새로운 이미지를 생성하는 프로세스입니다.
- 이미지 회전, 블러 처리, 이미지 이동, 노이즈 추가 등의 여러가지 변환으로 훈련 데이터를 생성합니다.



# 고급 옵티마이저

## ■ Momentum

- local minimum 에 빠지게 않도록 이전 gradient들을 계산에 포함해서 현재 파라미터를 업데이트 합니다.
- 이전 gradient들을 모두 동일한 비율로 포함시키지는 않고 비율을 감소시켜 줍니다.

## ■ Adagrad

- 가중치 기울기의 제곱을 통해 학습률을 점차 줄여나가면서 학습이 진행될수록 세밀한 학습이 가능하도록 하는 Optimizer입니다.

## ■ RMSProp

- RMSProp은 Hyper Parameter  $\rho$ 를 추가하여 누적 가중치 기울기는  $(0,1)$  사이의 값이므로 시간이 지날 수록 점차 작아지고, 최신 가중치 기울기를 더욱 반영하여 AdamGrad에서 발생하는  $G(t)$  값이 무한히 커져 minimum과 같은 극점 근처에서 학습 속도가 느려지고, local minimum에 수렴하는 문제를 해결하였다.

## ■ Adam (Adaptive moment estimation)

- Adam은 Momentum과 RMSProp을 병합한 Optimizer입니다.
- RMSProp의 특징인 gradient의 제곱을 지수 평균한 값을 사용하며 Momentum의 특징으로 gradient를 제공하지 않은 값을 사용하여 지수 평균을 구하고 수식에 활용합니다.

# MNIST 분류기 소스코드 - DNN

```
[31] model = Sequential()

model.add(Dense(64, activation='relu', input_shape=(784,)))
model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(10, activation='softmax'))
```

## TensorBoard 로깅 디렉토리 설정

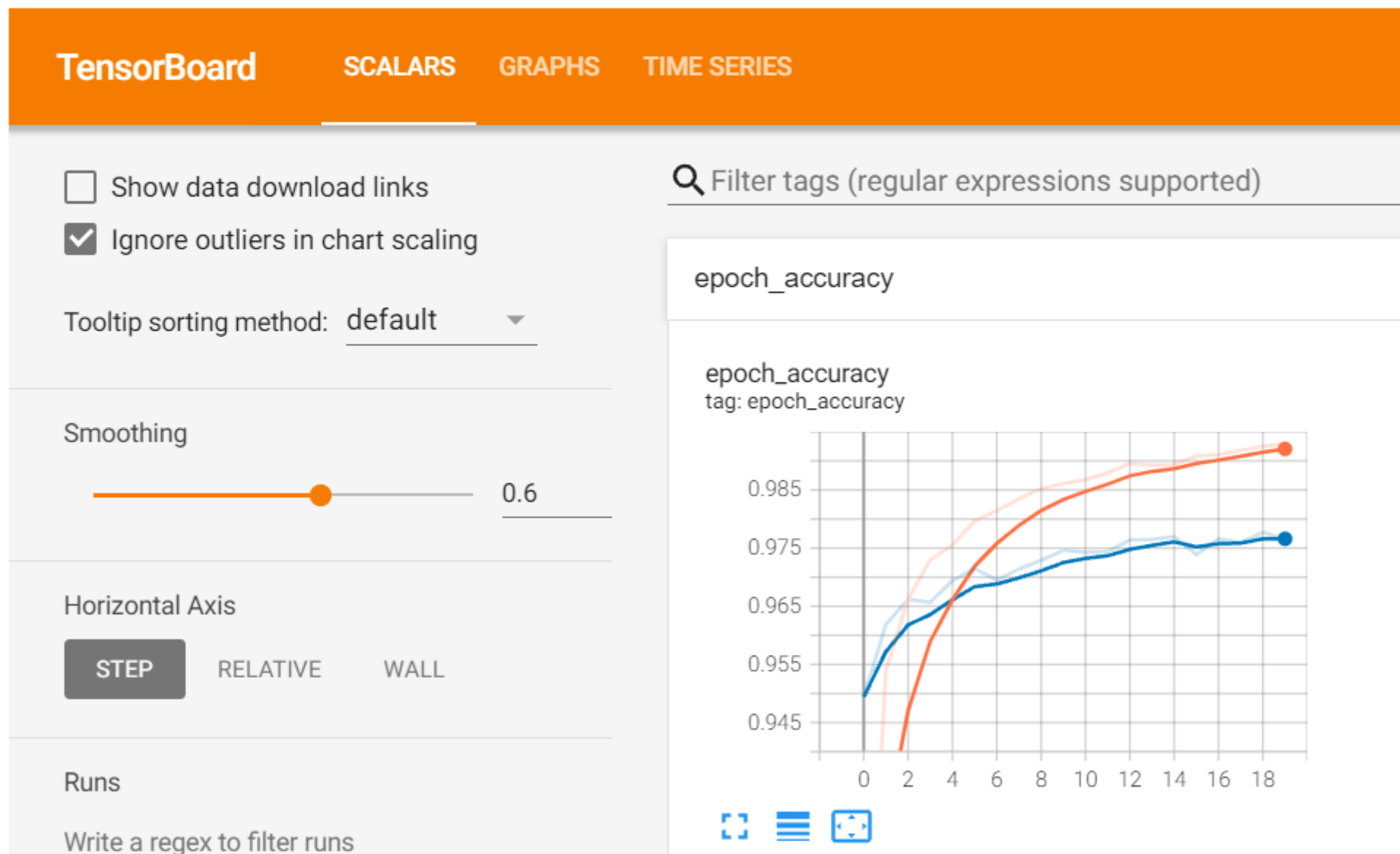
```
[34] tensorboard = TensorBoard('logs/deep-net')
```

## 모델 훈련

```
[35] model.fit(X_train, y_train,  
               batch_size=128,  
               epochs=20, verbose=1,  
               validation_data=(X_valid, y_valid),  
               callbacks=[tensorboard])
```

```
[36] %load_ext tensorboard
```

```
[37] %tensorboard --logdir logs/deep-net
```



# MNIST 분류모델 구현 실습 - DNN



`weight_initialization.ipynb`

`mnist_dnn_tensorboard_ann`

# Fashion MNIST



# Fashion MNIST 데이터셋

- 10개의 범주(category)와 70,000개의 흑백 이미지(28x28 픽셀)로 개별 옷 품목을 나타냅니다.



# Fashion MNIST 분류 모델

```
fashion_mnist = keras.datasets.fashion_mnist
```

```
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

```
model = keras.Sequential([  
    keras.layers.Flatten(input_shape=(28, 28)),  
    keras.layers.Dense(128, activation='relu'),  
    keras.layers.Dense(10, activation='softmax')  
])
```

```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy',  
              metrics=['accuracy'])
```

```
model.fit(train_images, train_labels, epochs=5)
```

# Fashion MNIST 분류모델 구현 실습



<https://www.tensorflow.org/tutorials/keras/classification>

THANK YOU

[kgpark88@gmail.com](mailto:kgpark88@gmail.com)