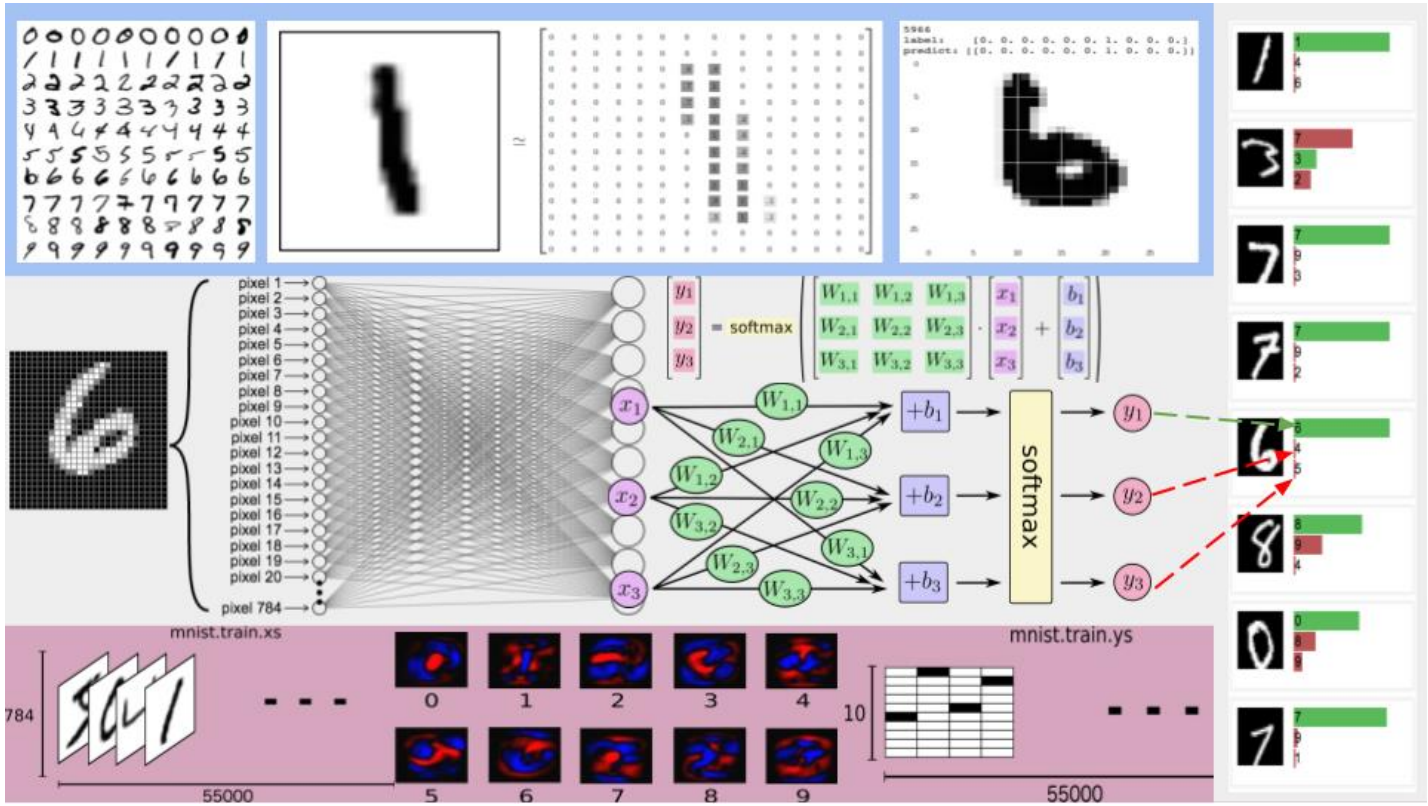
심층신경망

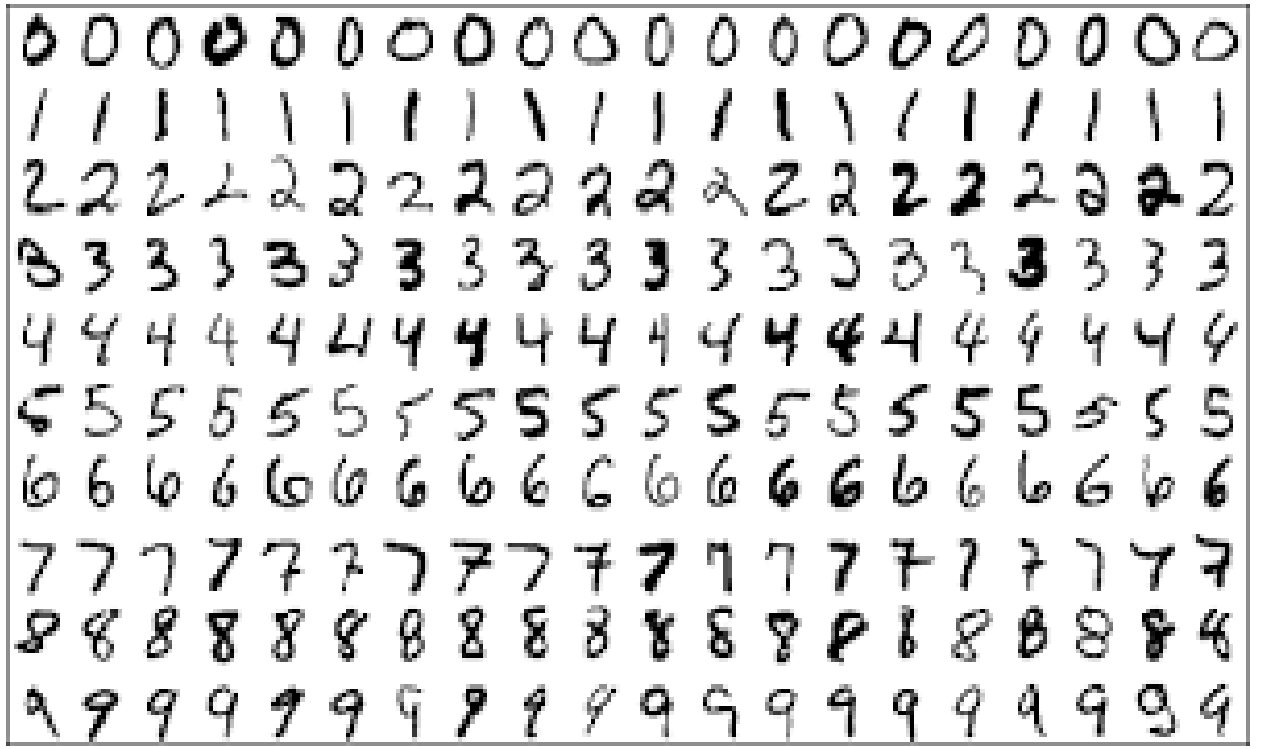

pixel 1
pixel 2
pixel 3
pixel 4
pixel 5
pixel 6
pixel 7
pixel 8
pixel 9
pixel 10
pixel 11
pixel 12
pixel 13
pixel 14
pixel 15
pixel 16
pixel 17
pixel 18
pixel 19
pixel 20
pixel 784
= softmax
softmax
mnist.train.xs
mnist.train.ys
784
55000
10
55000
0
1
2
3
4
5
6
7
8
9

박경규

얕은 신경망
(Shallow Neural Network)

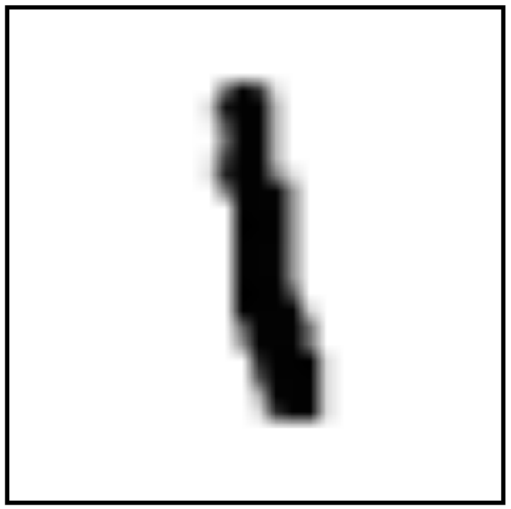
MNIST 데이터셋

- 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

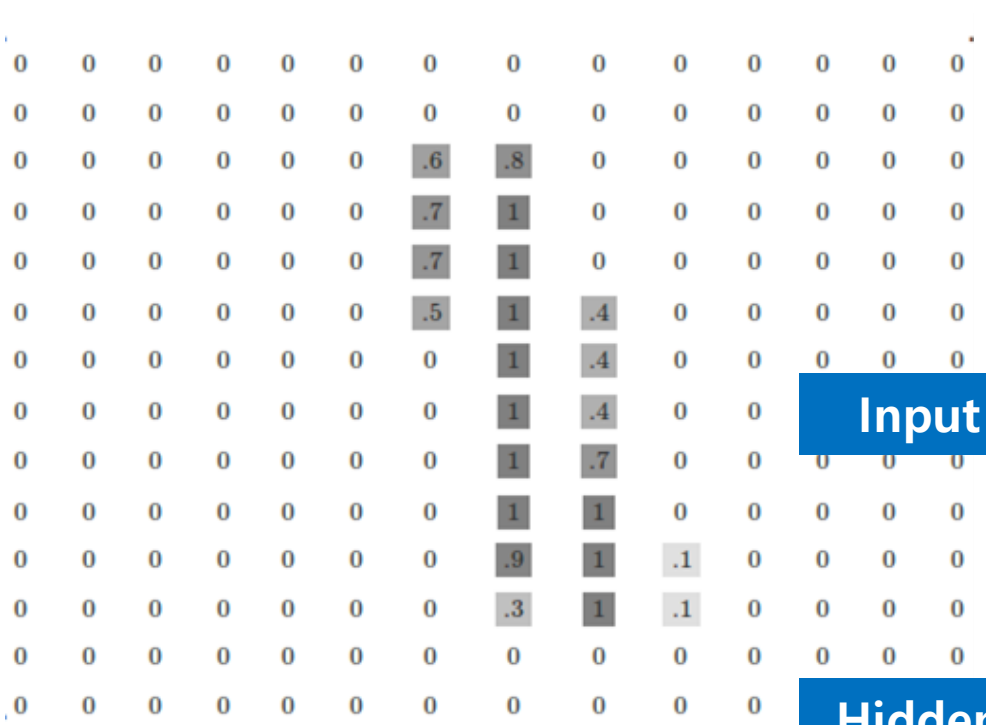
한 자리 숫자를 배열로 표현할 수 있습니다.
컴퓨터에 저장되는 이미지는 픽셀(pixel) 단위로 0~1 사이의 숫자로 표현되어 있습니다.

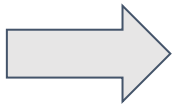


≈

0	0	0	0	0	0	0	0	0	0	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	.7	1	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	.5	1	.4	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	1	.4	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	1	1	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	.3	1	.1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

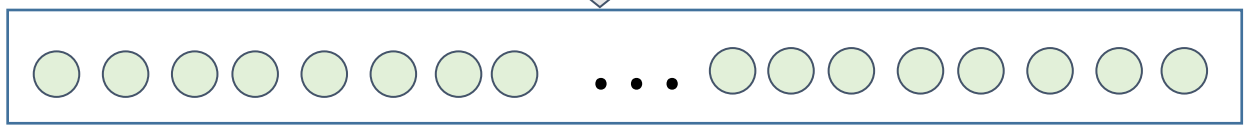
신경망 구조





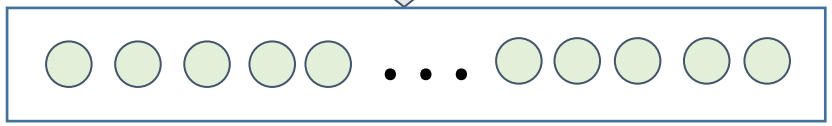
784개 입력

Input Layers



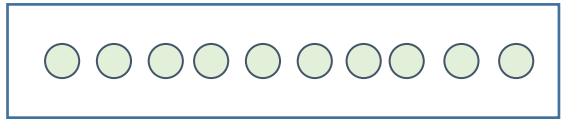
784개 입력뉴런

Hidden Layers



64개 뉴런

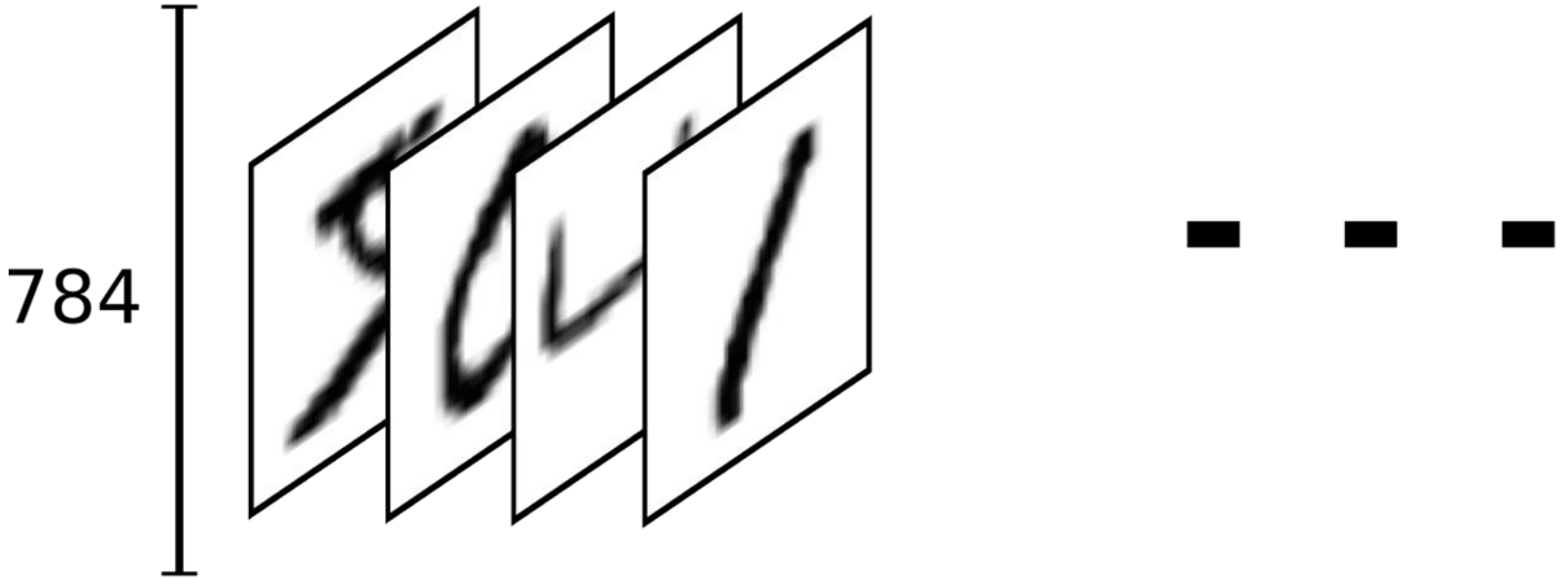
OutputLayers



10개 뉴런

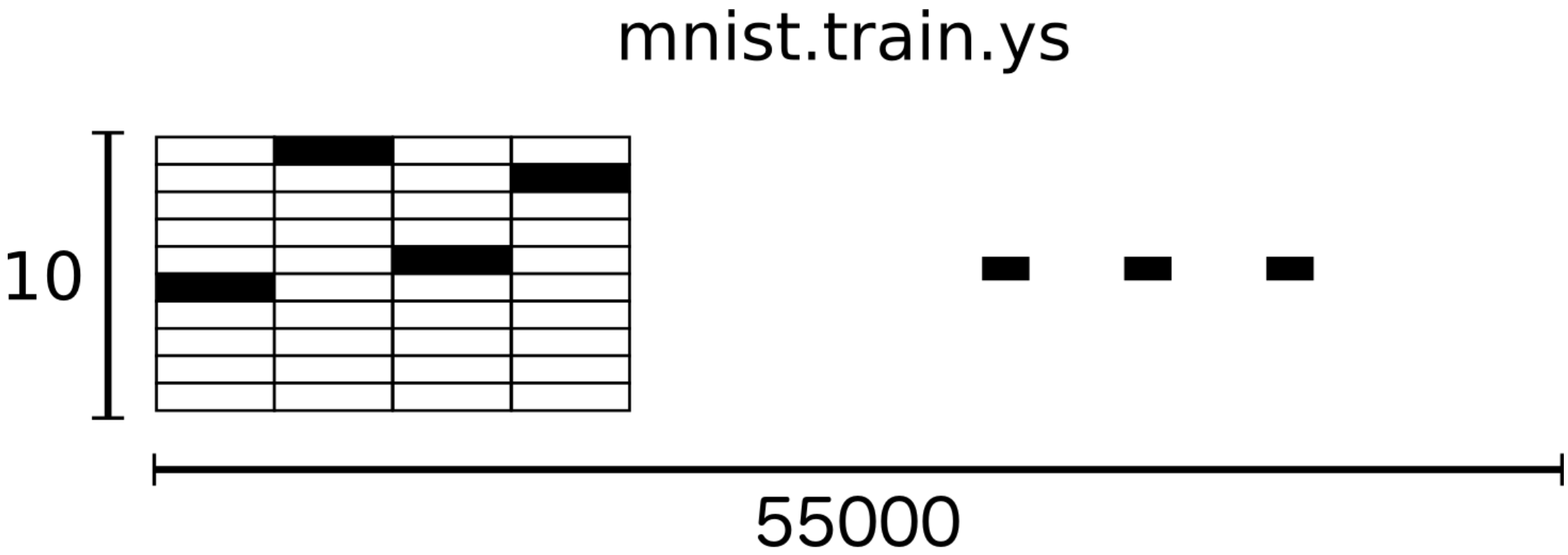
입력 데이터(X)

- 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.



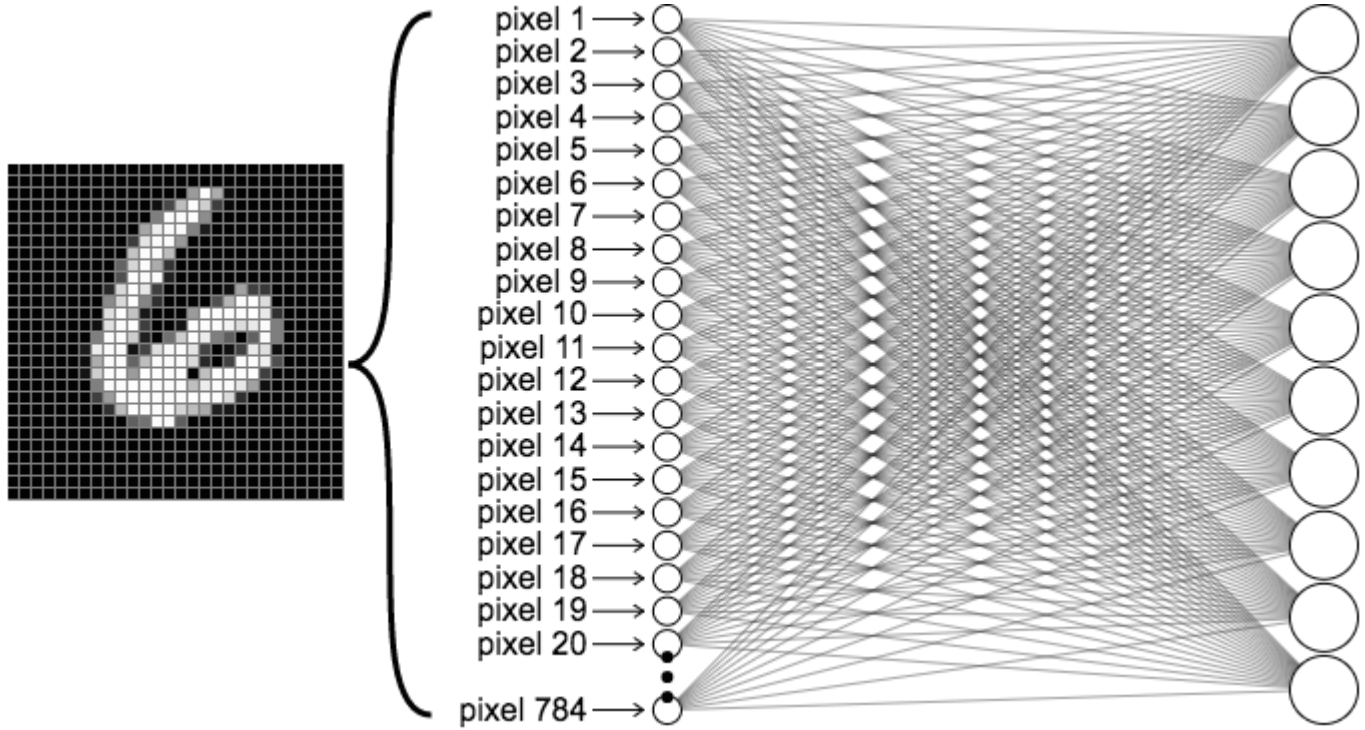
레이블 데이터(y)

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



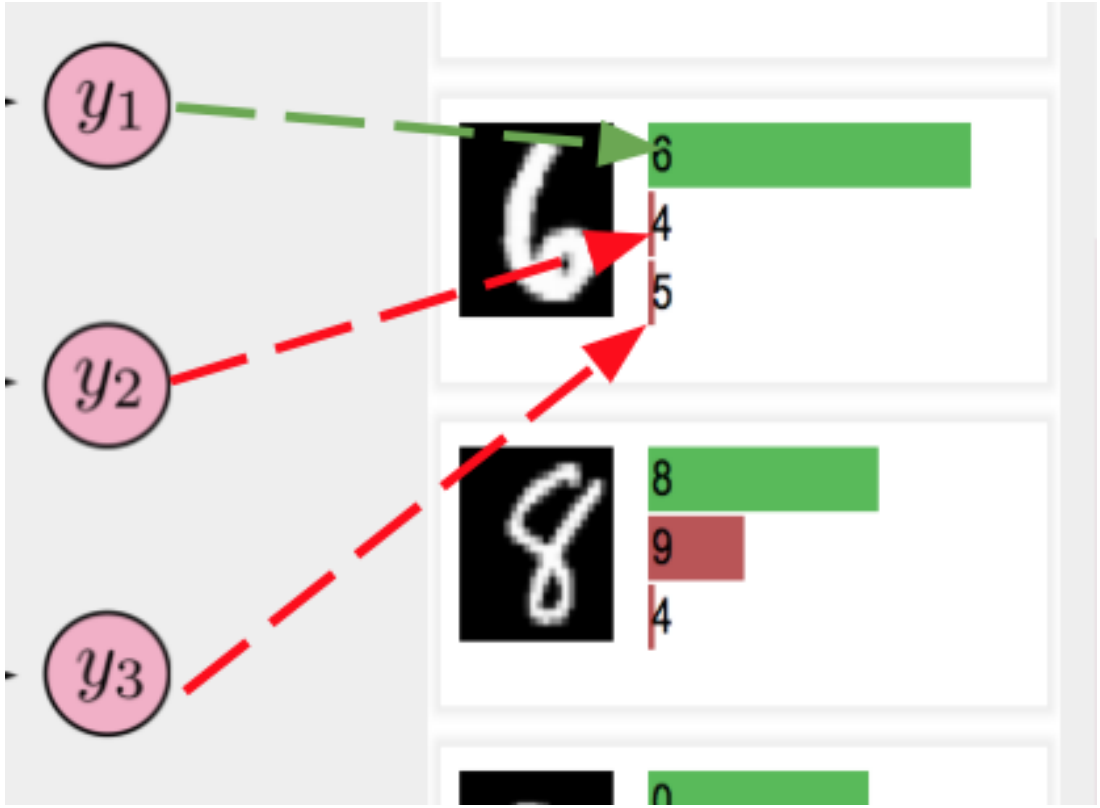
신경망 구성

- 인풋 레이어는 784개(28 x 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 임포트

```
[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]
```

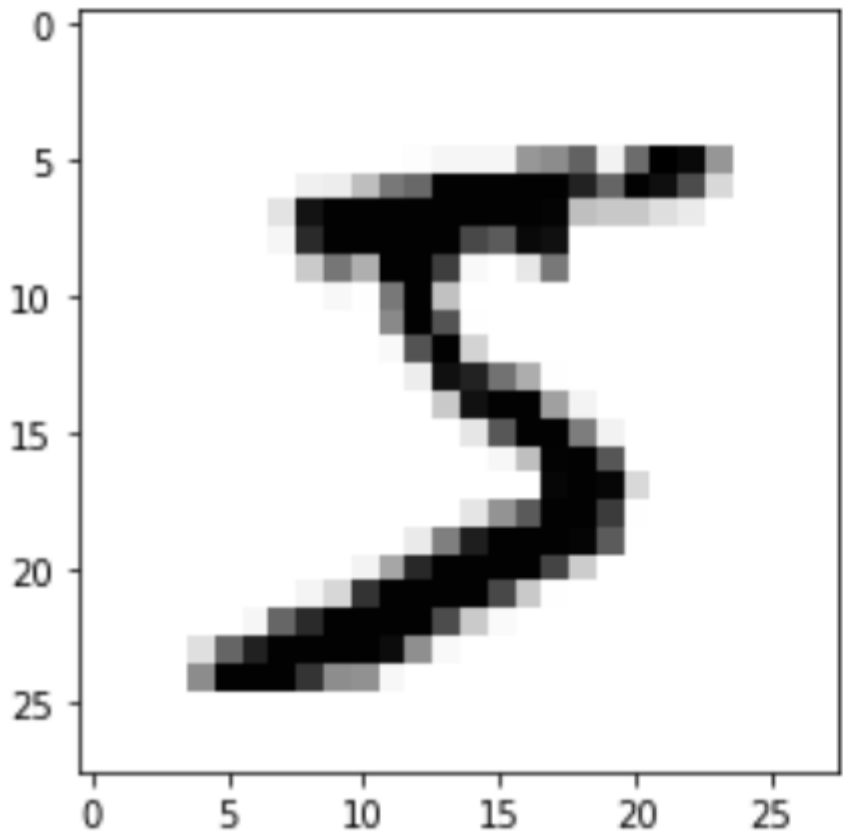
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   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
[0.027395464479923325, 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 y가 $\hat{y}$ 보다 크든지 작은지 상관없이 두 값의 차이를 양수로 만듬
- 제곱하면 y와 $\hat{y}$ 차이가 작은 것보다 큰 것이 불리

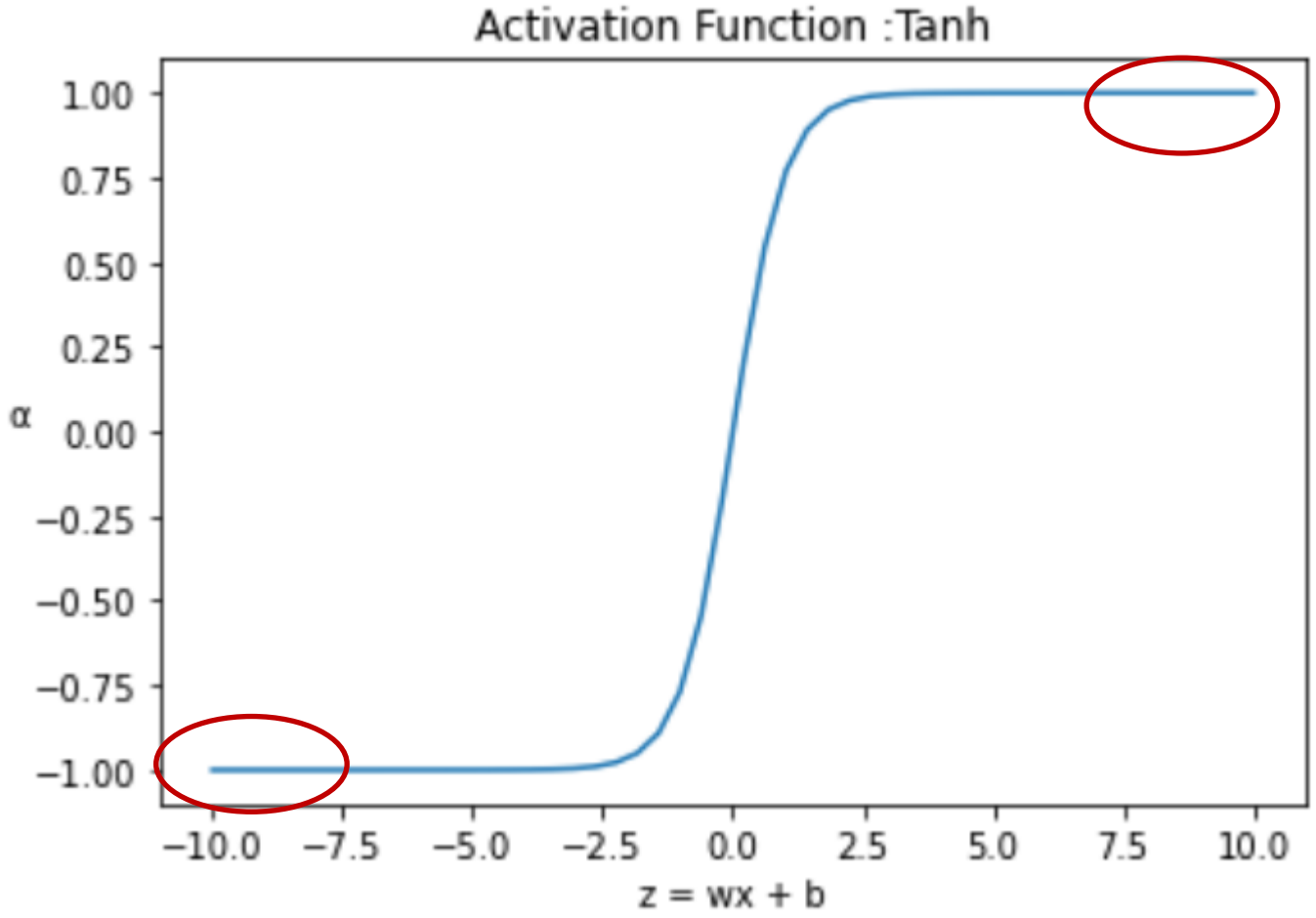
$$Loss = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

y_i: 실제값

$\hat{y}_i$: 모델이 예측한 값

포화된 뉴런(Saturate Neuron)

- 신경망이 뉴런의 w와 b를 튜닝하여 y를 근사하는 것을 학습할 수 있습니다.
- 뉴런의 입력과 파라미터 조합이 극단적인 z를 만들 때 포화되었다고 합니다.
- w와 b를 바꾸어도 α는 아주 조금만 변경되는 포화뉴런에서는 학습이 느려집니다.



Cross Entropy 손실 함수

- MSE 대신 Cross-entropy 손실함수를 사용하여 포화 뉴런이 학습속도에 미치는 영향을 최소화
- y와 $\hat{y}$ 사이의 차이가 클수록 기하급수적으로 비용이 증가됨
- y와 $\hat{y}$ 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)
```

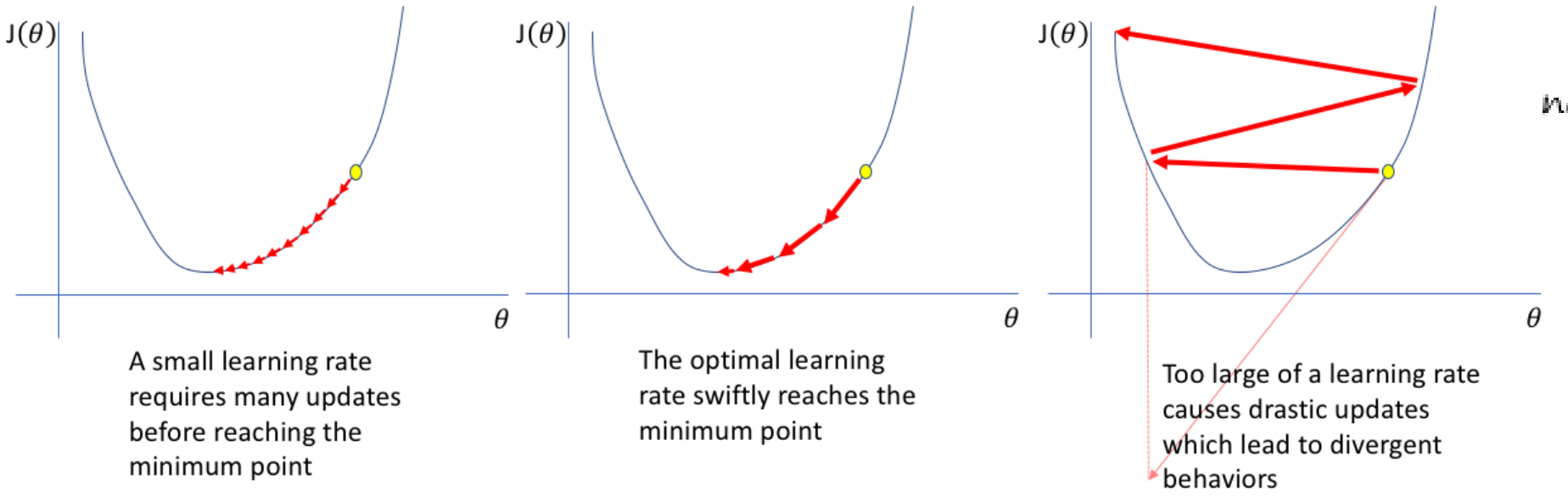
```
0.2231435513142097
```

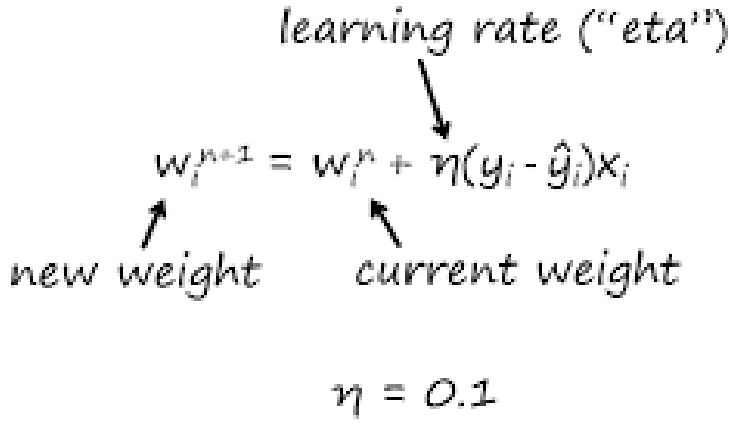
```
[5]  cross_entropy(1, 0.1)
```

```
2.302585092994045
```

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)





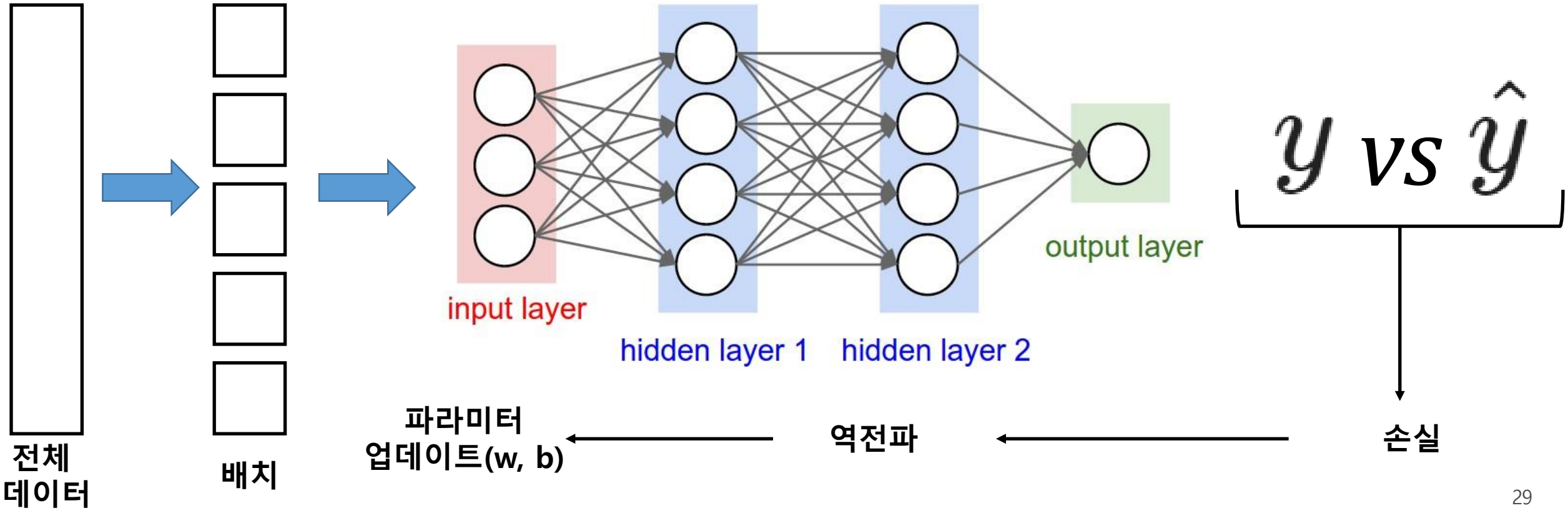
확률적 경사법(SGD, Stochastic Gradient Descent)

- 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.

$$\text{미니배치 개수} = [\frac{\text{훈련 데이터 크기}}{\text{배치 크기}}]$$

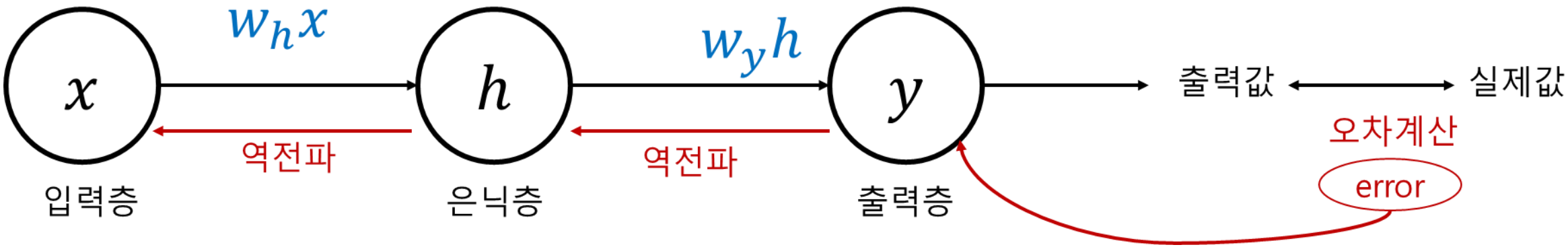
훈련과정

1. 미니배치 x를 샘플링합니다.
2. X를 신경망에 통과시켜 y에 대한 예측 y_hat을 만듭니다.
3. y와 y_hat을 비교하여 손실을 계산합니다.
4. 비용/손실을 기반으로 경사하강법을 적용해 x가 y를 더 잘 예측할 수 있도록 w와 b를 조정합니다.



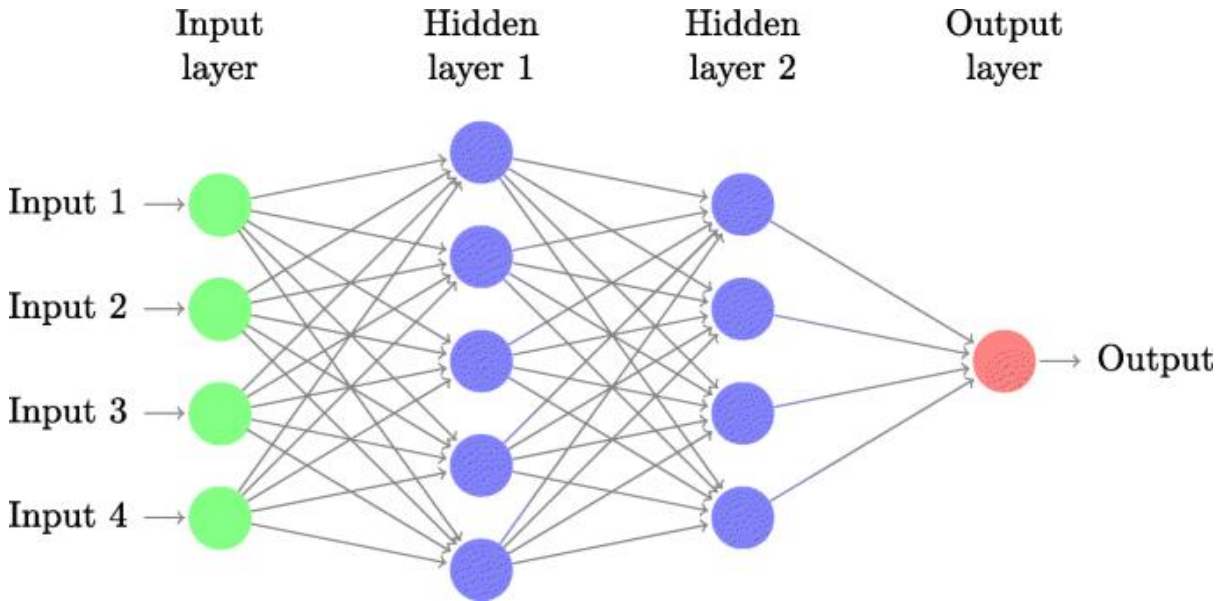
역전파 알고리즘

- 신경망 모델은 파라미터(w, b) 값을 랜덤하게 초기화 합니다.
- 신경망 모델에 x값을 입력하면 신경망은 예측 y_hat을 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 y가 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 정화도가 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

Thank you

kgpark88@gmail.com