Machine Learning

Neural Network

❖ Neural Network

- **활성화 함수.**
  - **입력 신호의 총합을 출력 신호로 변환하는 함수.**

$$a = b + w_1 x_1 + w_2 x_2$$

$$y = h(a)$$

❖ Neural Network

- **활성화 함수.**
  - **퍼셉트론의 활성화 함수 계단 함수**(Step function).

$$y = h(b + w_1x_1 + w_2x_2)$$

$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

❖ Neural Network

- **활성화 함수.**
  - **계단 함수(Step function) 구현.**

```
def step_function(x):
    if x > 0:
        return 1
    else:
        return 0
```

```
def step_function(x):
    y =  x > 0
    return y.astype(np.int)
```

❖ Neural Network

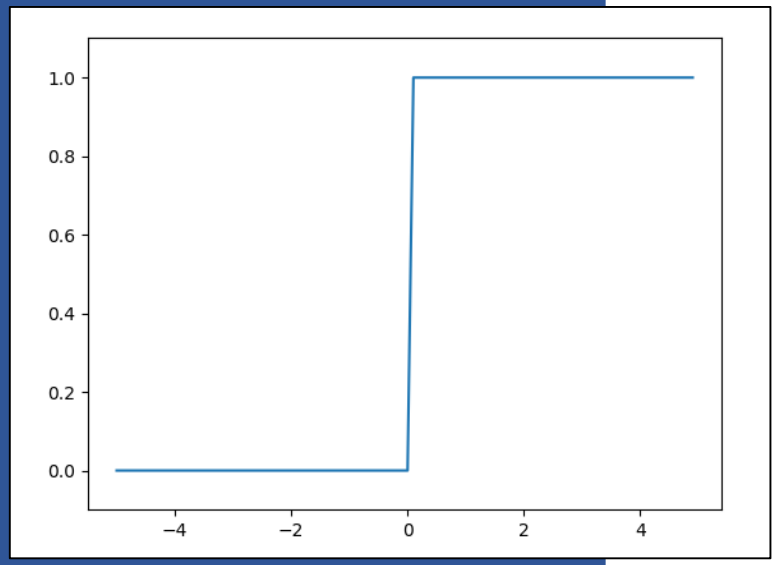
- **활성화 함수**.
  - **계단 함수**(Step function) **구현**.

```
def step_function(x):
    return np.array(x > 0, dtype=np.int)
```

❖ Neural Network

- **활성화 함수.**
  - **계단 함수(Step function) 구현.**

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


def step_function(x):
    return np.array(x > 0, dtype=np.int)

X = np.arange(-5.0, 5.0, 0.1)
Y = step_function(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)  # y축의 범위 지정
plt.show()
```



❖ Neural Network

- **활성화 함수**.
  - **시그모이드 함수**(Sigmoid function).

$$h(x) = \frac{1}{1 + \exp(-x)}$$

❖ Neural Network

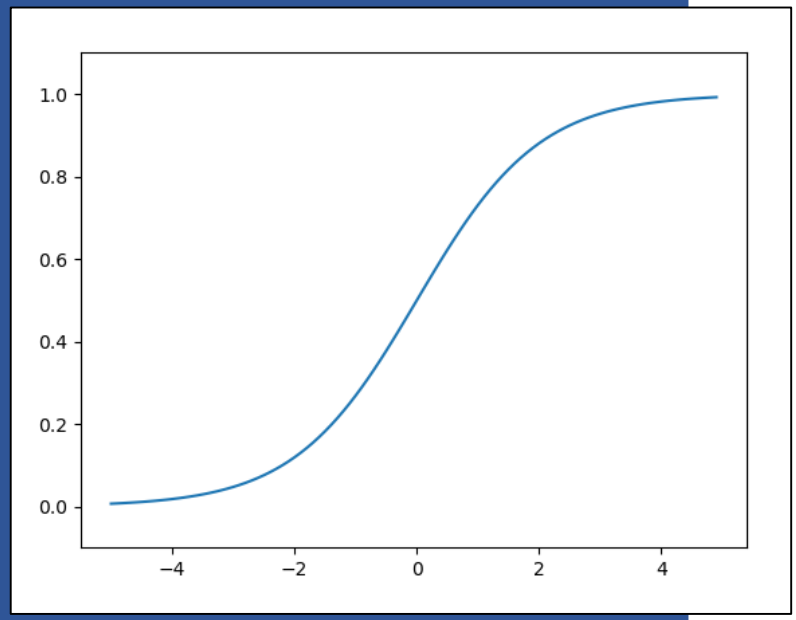
- **활성화 함수**.
  - **시그모이드 함수**(Sigmoid function)**의 구현**.

```
def sigmoid(x):
    return 1/(1 + np.exp(-x))
```

❖ Neural Network

- **활성화 함수.**
  - **시그모이드 함수(Sigmoid function)의 구현.**

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


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```

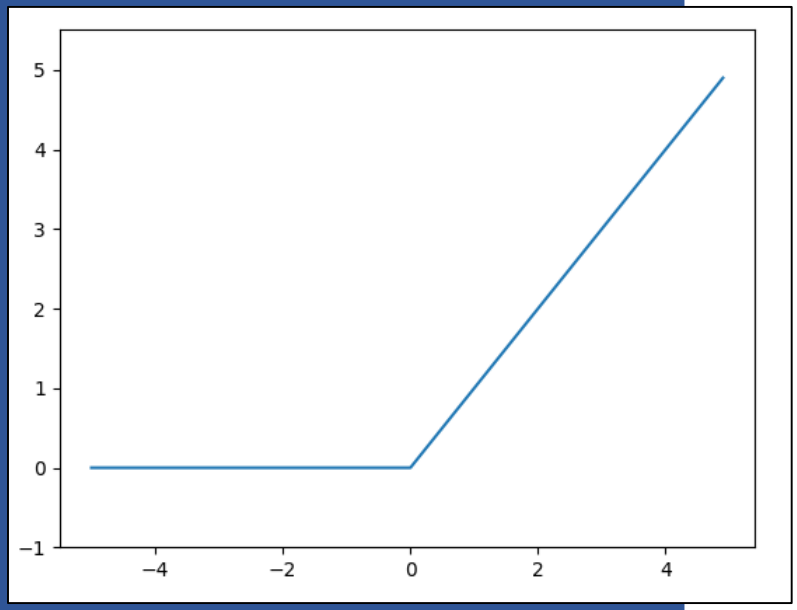


❖ Neural Network

- **활성화 함수.**
  - ReLu(Rectified Linear Unit) **함수.**

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

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


def relu(x):
    return np.maximum(0, x)

x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.ylim(-1.0, 5.5)
plt.show()
```



❖ Neural Network

- **다차원 배열의 계산.**
  - **다차원 배열.**

```
# 1차원 배열
import numpy as np
A = np.array([1, 2, 3, 4])
print(A)
np.ndim(a)
A.shape
```

```
# 다차원 배열
import numpy as np
A = np.array([[1, 2],  [3, 4], [5, 6]])
print(A)
np.ndim(a)
A.shape
```

❖ Neural Network

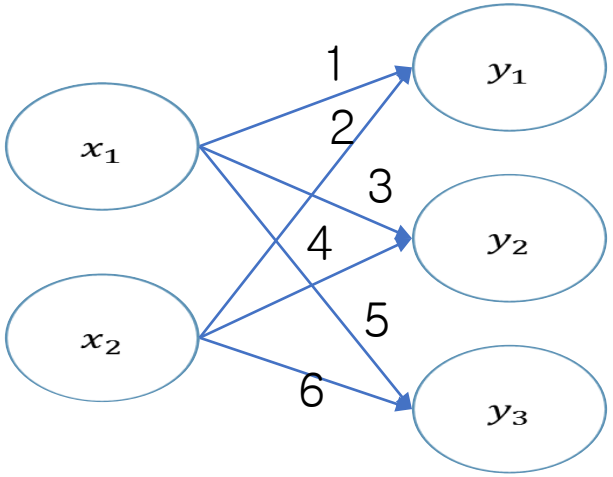
- 다차원 배열의 계산.
  - 행렬의 곱.

```
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

np.dot(A, B)
```

```
A = np.array([[1, 2, 3], [4, 5, 6]])
B = np.array([[1, 2], [3, 4], [5, 6])

np.dot(A, B)
```

❖ Neural Network

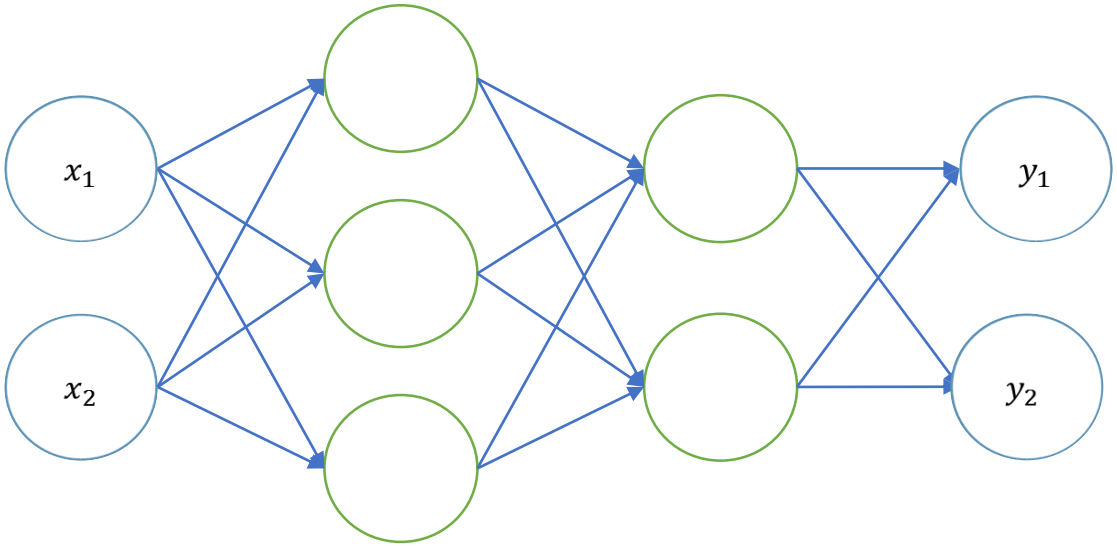
- **신경망의 내적.**




```
X = np.array([1, 2])
W = np.array([[1, 3, 5], [2, 4, 6]])
Y = np.dot(X, W)
```

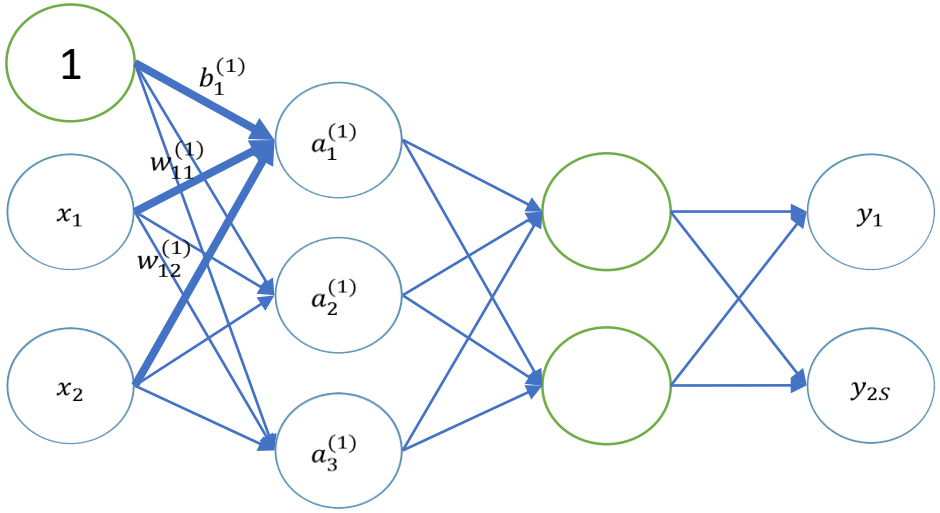
❖ Neural Network

- 3층 신경망 구현하기.



❖ Neural Network

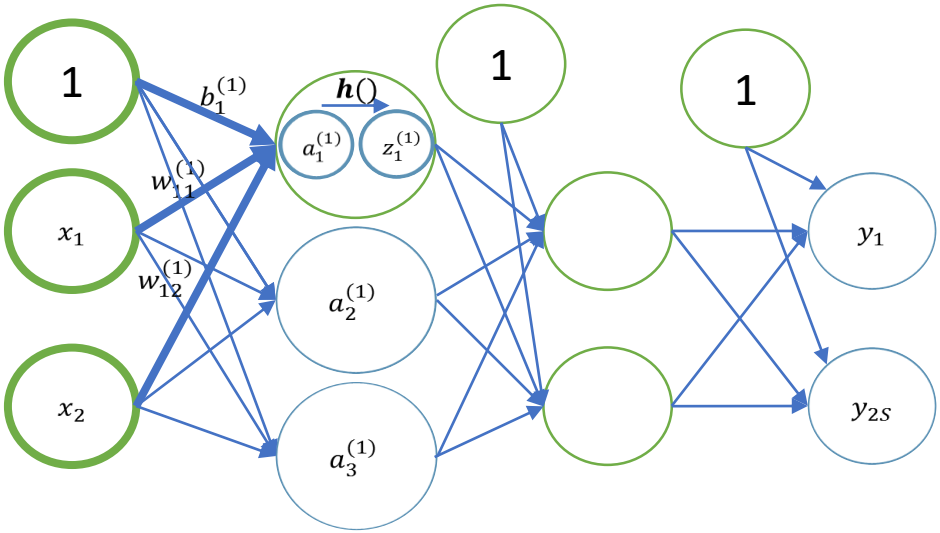
- 3층 신경망 구현하기.




```
X = np.array([1.0 , 0.5])
W1 = np.array([[0.1 , 0.3 , 0.5], [0.2,  0.4 , 0.6 ]])
B1 = np.array([0.1, 0.2, 0.3])
A1 = np.dot(X, W1) + B1
```

❖ Neural Network

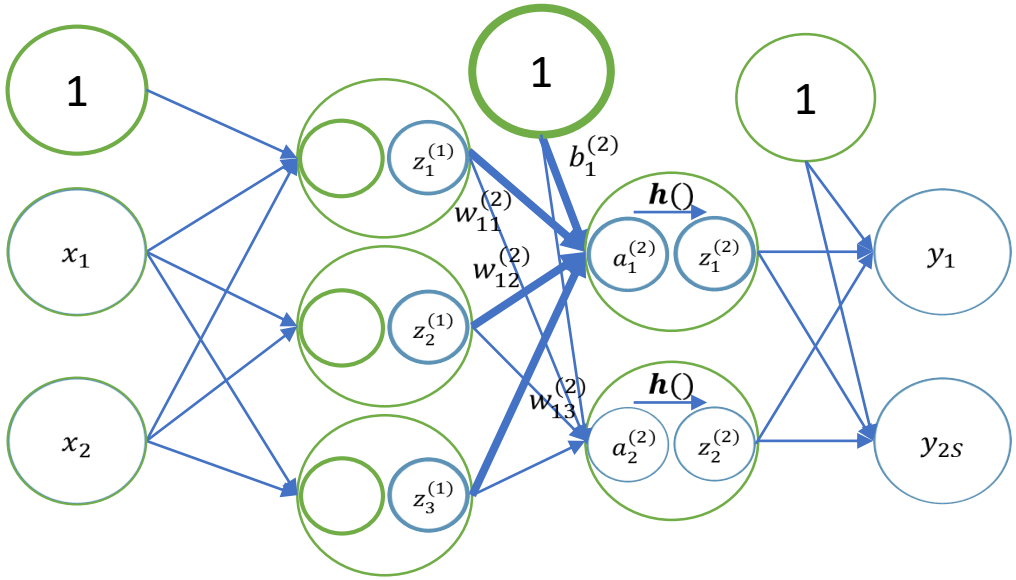
- 3층 신경망 구현하기.




```
Z1 = sigmoid(A)
print(A1)
print(Z1)
```

❖ Neural Network

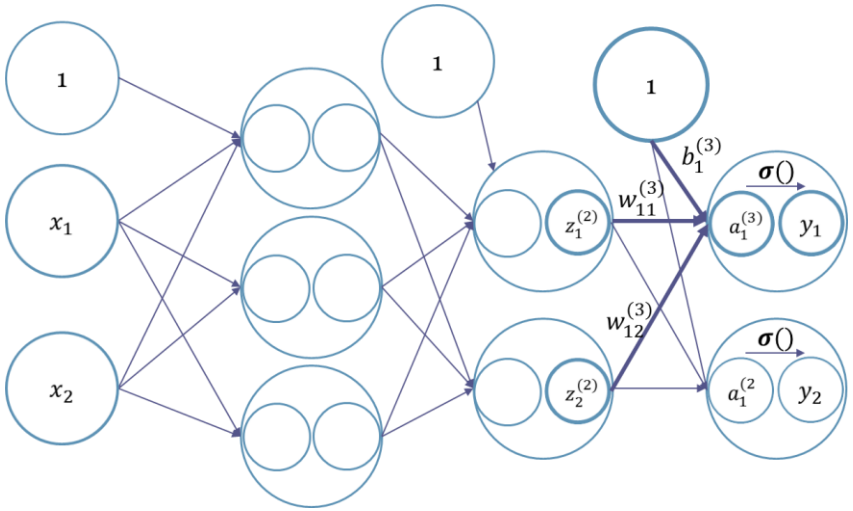
- 3층 신경망 구현하기.




```
W2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
B2 = np.array([0.1, 0.2])
A2 = np.dot(Z1, W2) + B2
Z2 = sigmoid(A2)
```

❖ Neural Network

- 3층 신경망 구현하기.




```
def identity_function(x):
        return x

W3 = np.array([[0.1, 0.3], [0.2, 0.4]])
B3 = np.array([0.1, 0.2])
A3 = np.dot(Z2, W3) + B3
Y = identity_function(A3)
```

❖ Neural Network

- 3층 신경망 구현하기.(구현 정리)

```
def init_network():
        network = {}
        network['W1'] = np.array([[0.1 , 0.3 , 0.5], [0.2,  0.4 , 0.6 ]])
        network['b1'] = np.array([0.1, 0.2, 0.3])
        network['W2'] = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
        network['b2'] = np.array([0.1, 0.2])
        network['W3'] = np.array([[0.1, 0.3], [0.2, 0.4]])
        network['b3'] = np.array([0.1, 0.2])

        return network
```

❖ Neural Network

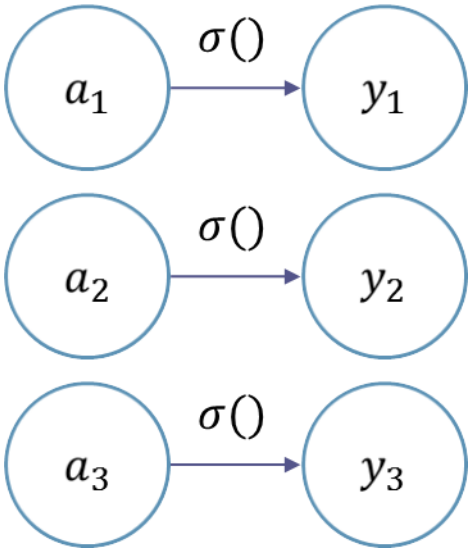
- 3층 신경망 구현하기.(구현 정리)

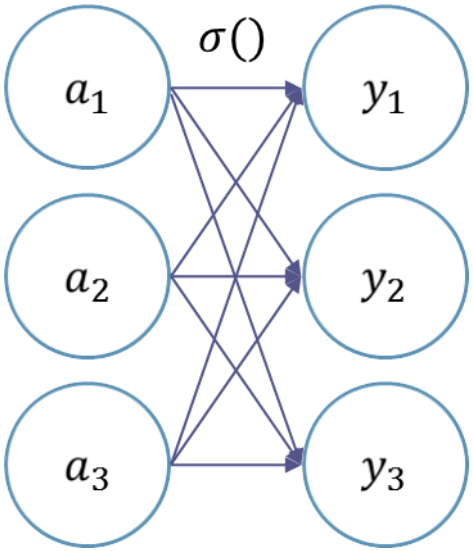
```
def forward(network, x):
        W1, W2, W3 = network['W1'], network['W2'], network['W3']
        b1, b2, b3 = network['b1'], network['b2'], network['b3']

        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        z2 = sigmoid(a2)
        a3 = bp.dot(z2, W2) + b3
        y = identity_function(a3)

        return y
```

```
network = init_network()
x = np.array([1.0, 0.5])
y = forward(network, x)
print(y)
```

❖ Neural Network

- 출력층 설계하기
  - 항등 함수 & 소프트 맥스 함수





❖ Neural Network

- 출력층 설계하기
  - 소프트 맥스 함수
    - 출력은 0 ~ 1 사이의 실수
    - 출력의 총합은 1

$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

❖ Neural Network

- 출력층 설계하기
  - 소프트 맥스 함수

```
a = np.array([0.3, 2.9, 4.0])

exp_a = np.exp(a)
print(exp_a)

sum_exp_a = np.sum(exp_a)
print(sum_exp_a)

y = exp_a / sum_exp_a
print(y)
```

```
def softmax(a):
        exp_a = np.exp(a)
        sum_exp_a = np.sum(exp_a)
        y = exp_a / sum_exp_a

        return y
```

❖ Neural Network

- 출력층 설계하기
  - 소프트 맥스 함수

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} = \frac{\mathrm{C}\exp(a_k)}{C\sum_{i=1}^{n} \exp(a_i)}$$

$$= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n} \exp(a_i + \log C)}$$

$$= \frac{\exp(a_k + C')}{\sum_{i=1}^{n} \exp(a_i + C')}$$

❖ Neural Network

- 출력층 설계하기
  - 소프트 맥스 함수

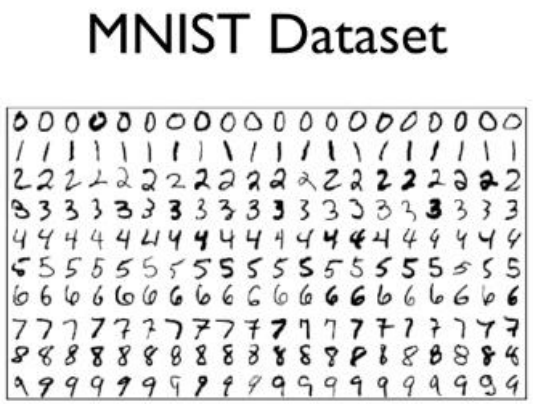
```
a = np.array([1010, 1000, 990])
np.exp(a) # 제대로 계산되지 않음.

c = np.max(a)

np.exp(a-c) / np.sum(np.exp(a – c)) # 계산결과가 출력됨.
```

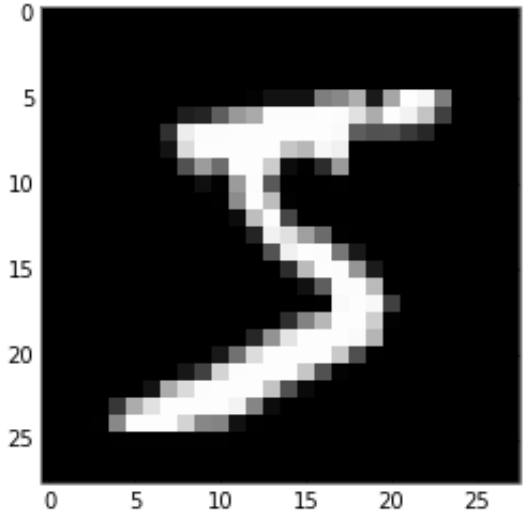
```
def softmax(a):
        c = np.max(a)
        exp_a = np.exp(a - c)
        sum_exp_a = np.sum(exp_a)
        y = exp_a / sum_exp_a

        return y
```

❖ Neural Network

▪ 손글씨 숫자 인식

- MNIST 데이터셋
  - mnist?
    - 손글씨 숫자 이미지 집합
    - 0 ~ 9까지의 숫자 이미지로 구성
    - 훈련 이미지 60,000장(학습), 시험 이미지 10,000장(분류)





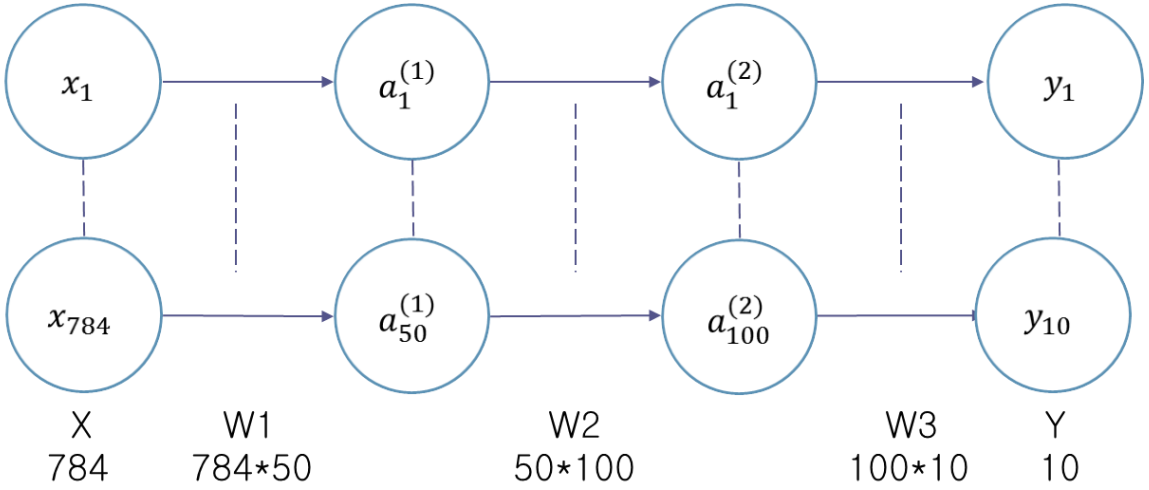
❖ Neural Network

- **손글씨 숫자 인식**

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist

def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)

img = x_train[0]
label = t_train[0]
print(label)  # 5

print(img.shape)  # (784,)
img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
print(img.shape)  # (28, 28)

img_show(img)
```

❖ Neural Network

- 손글씨 숫자 인식
  - 신경망 구성
    - 입력층 뉴런 784개(28*28), 출력층 뉴런 10개(0~9)
    - 은닉층 2개(첫 번째 - 50개 뉴런, 두 번째 - 100개 뉴런)
    - 은닉층 뉴런의 개수는 임의지정 값



❖ Neural Network

- **손글씨 숫자 인식**

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
import pickle
from dataset.mnist import load_mnist
from common.functions import sigmoid, softmax


def get_data():
    (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True,
one_hot_label=False)
    return x_test, t_test


def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network
```

❖ Neural Network

- 손글씨

```
def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = softmax(a3)

    return y

x, t = get_data()
network = init_network()
accuracy_cnt = 0
for i in range(len(x)):
    y = predict(network, x[i])
    p= np.argmax(y) # 확률이 가장 높은 원소의 인덱스를 얻는다.
    if p == t[i]:
        accuracy_cnt += 1

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```