파이썬으로 배우는 딥러닝(Deep Learning)

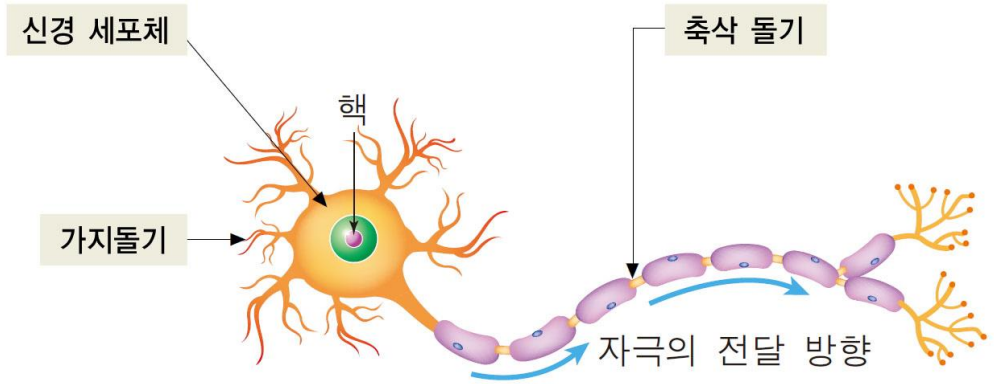
2회차 수업
신경망은 어떻게 작동할까?

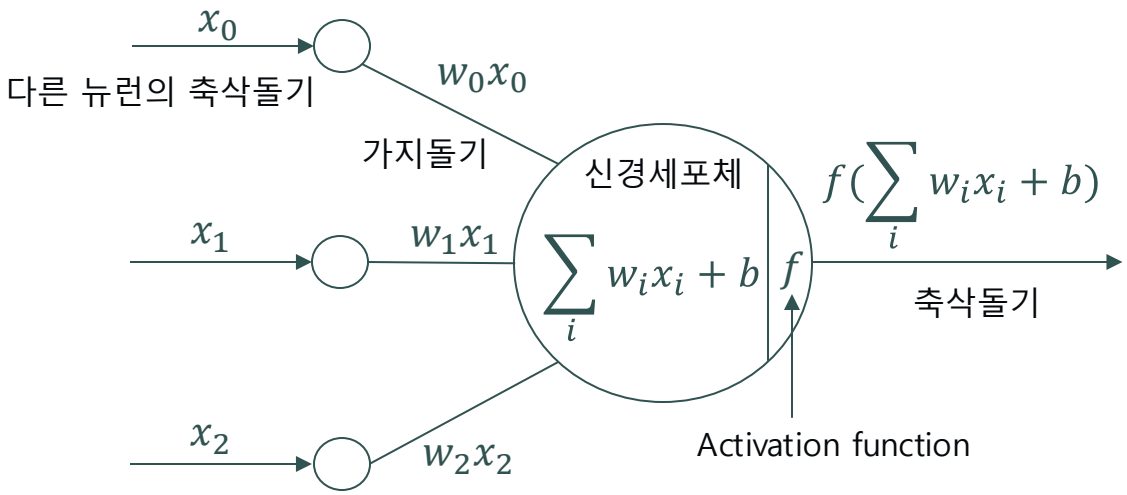
목 차

인공신경망에서 다양한 Activation function 함수 적용

활성화 함수 : 입력값을 **일정 기준에 따라 변화시켜 출력하는 비선형 함수**

입력값과 가중치(weight)를 곱하고 편향(bias)을 더한 후 활성화 함수를 적용함
활성화 함수는 **주어진 값이 특정 조건을 만족하게 되면 해당 값 출력**.





- 퍼셉트론의 활성화 함수는 계단함수(Step function)의 형태를 갖음
- **다양한 활성화 함수가 적용되면서 인공신경망으로 발전 : Sigmoid, ReLU, Softmax 등**

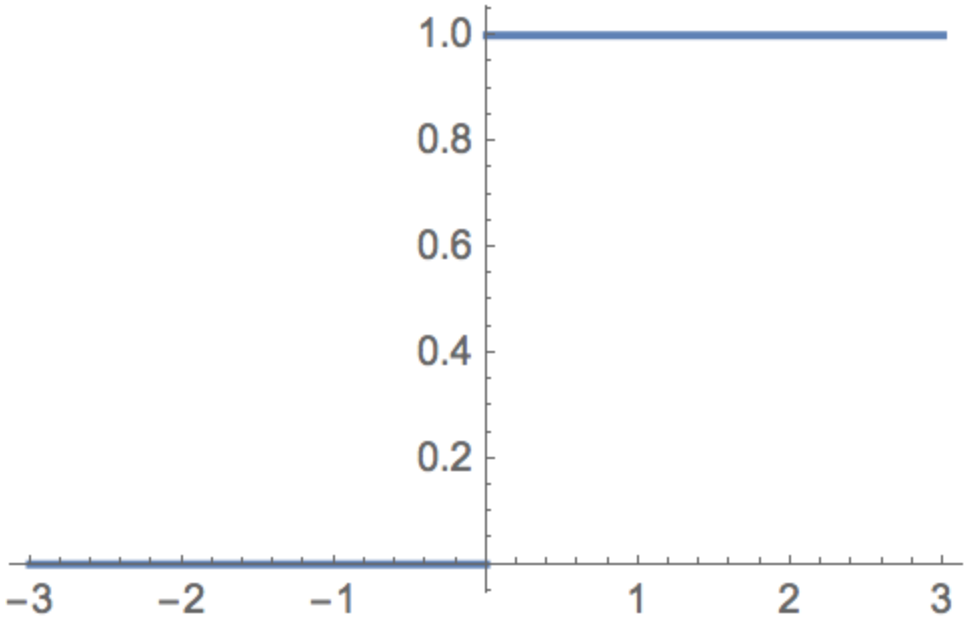
https://saintbinary.tistory.com/8

활성화 함수(Activation function)(1)

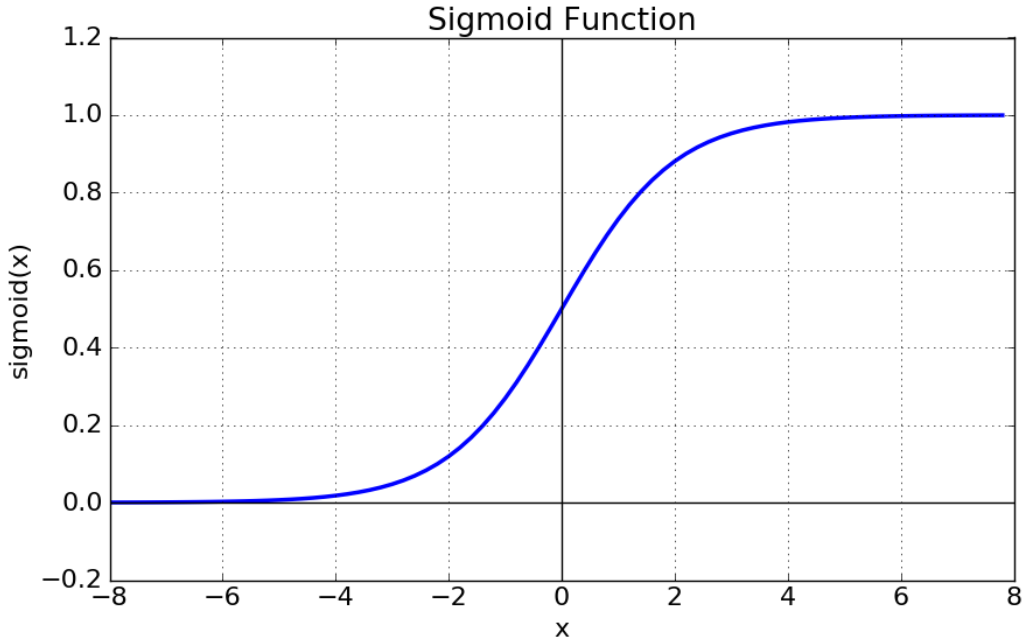
활성화 함수 : 입력값을 **일정 기준에 따라 변화시켜 출력하는 비선형 함수**

입력값과 가중치(weight)를 곱하고 편향(bias)을 더한 후 활성화 함수를 적용함
활성화 함수는 **주어진 값이 특정 조건을 만족하게 되면 해당 값 출력**.

계단 함수



Sigmoid 함수 $f(x) = \frac{1}{1+e^{-x}}$



https://saintbinary.tistory.com/8

활성화 함수(Activation function)(2)

◆ 활성화 함수 : 입력값을 **일정 기준에 따라 변화시켜 출력하는 비선형 함수**

◆ 입력값과 가중치(weight)를 곱하고 편향(bias)을 더한 후 활성화 함수를 적용함

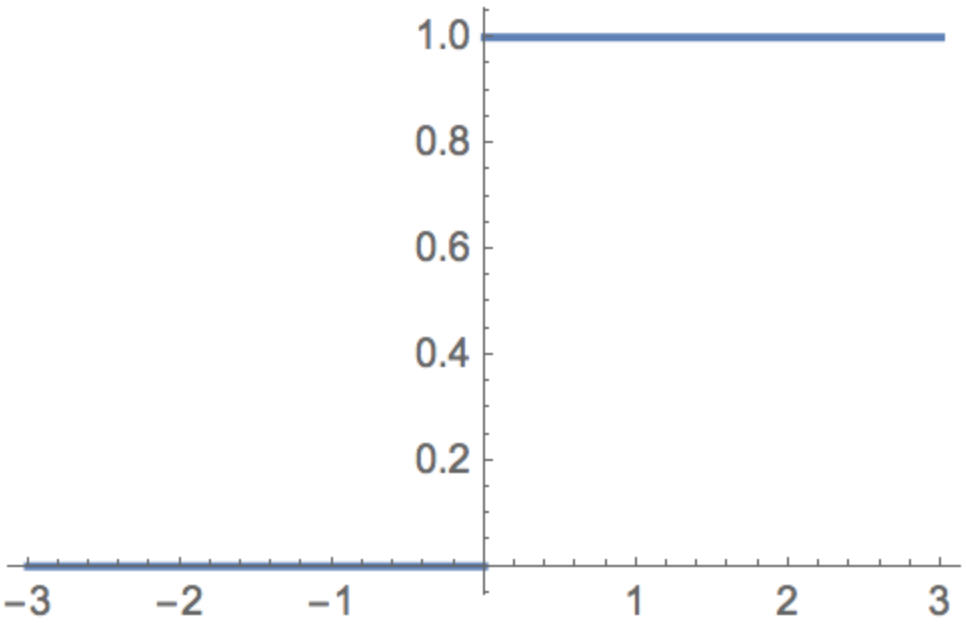
활성화 함수는 **주어진 값이 특정 조건을 만족하게 되면 해당 값 출력**.

◆ 활성화 함수 사용 이유

- 비선형성(Non-linearity) 추가 : 복잡한 데이터 패턴 학습 및 다양한 함수 근사
- 출력 범위 제한 : 다음 층 입력 값 크기 관리. 수렴 속도 향상 예. Singmoid [0, 1], tanh[-1, 1]
- Gradient 계산의 용이성 : 역전파로 가중치 조정시 활성화 함수의 도함수 필요
- 계산 효율성 : 계산이 단순하고 빠르게 수행돼 전체 학습 및 추론 속도 향상

활성화 함수(Activation function)의 종류(1)

계단 함수



<실습 과제>

계단 함수를

구현해보자

참고>

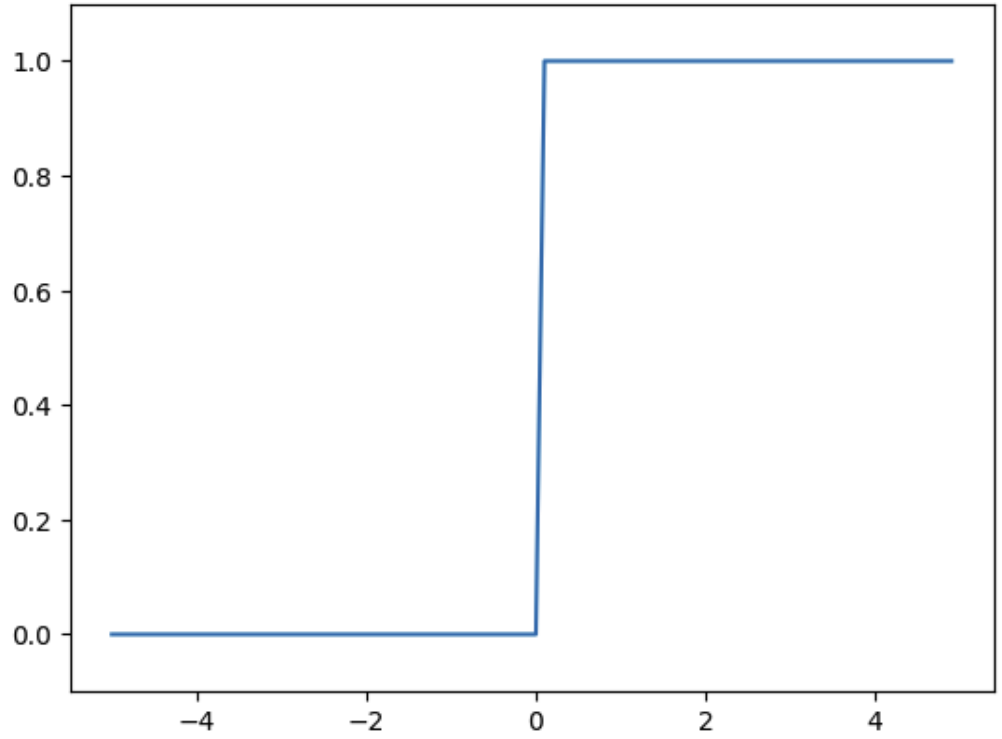
ch02/step_function.ipynb

에 구현

활성화 함수(Activation function)의 종류(2)

실습: ch02/step_function.ipynb

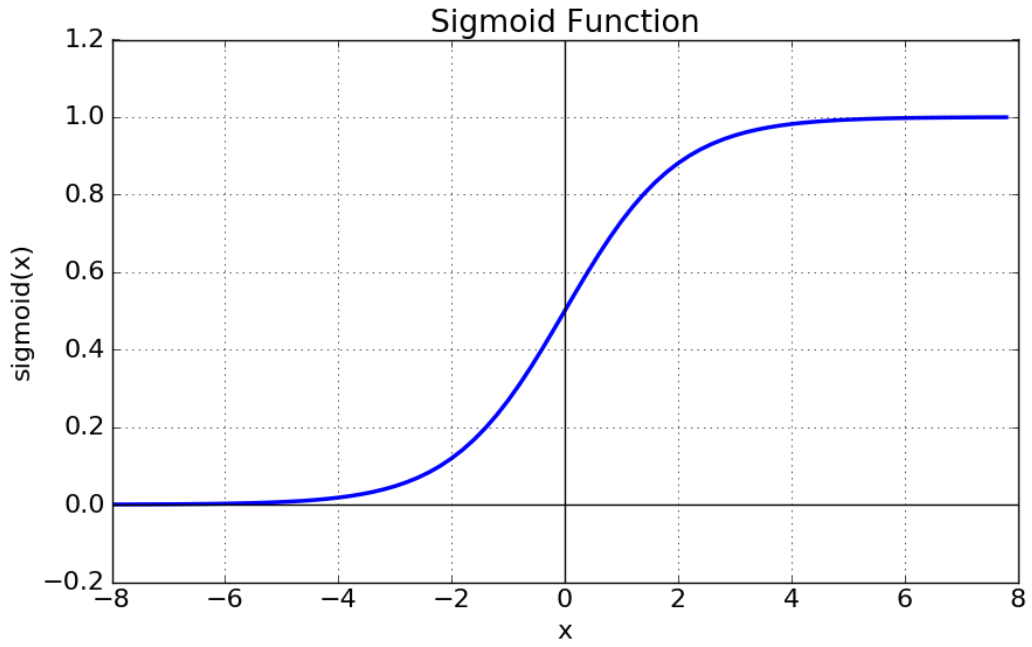
```python
# coding: utf-8
import numpy as np
import matplotlib.pylab as plt


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

```



x가 0보다 큰 경우
1을 출력

활성화 함수(Activation function)의 종류(3)

Sigmoid 함수 $f(x) = \frac{1}{1+e^{-x}}$



<실습 과제>

Sigmoid 함수를 구현해보자

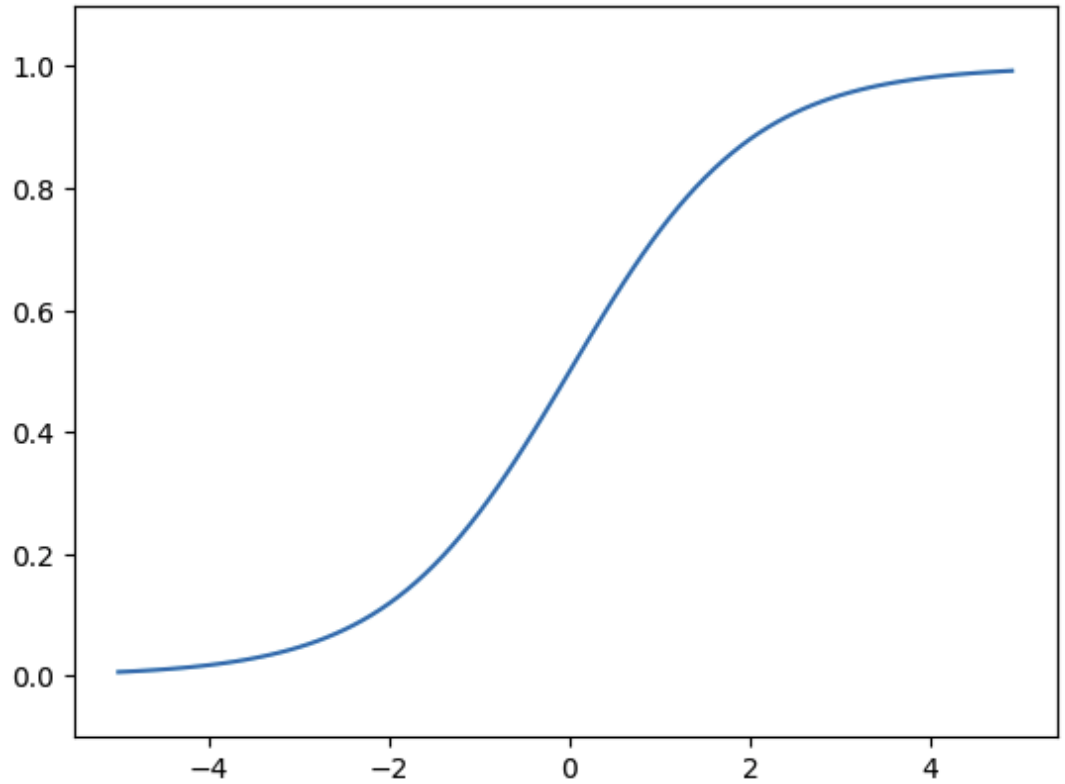
참고> ch02/sigmoid.ipynb 에 구현

활성화 함수(Activation function)의 종류(4)

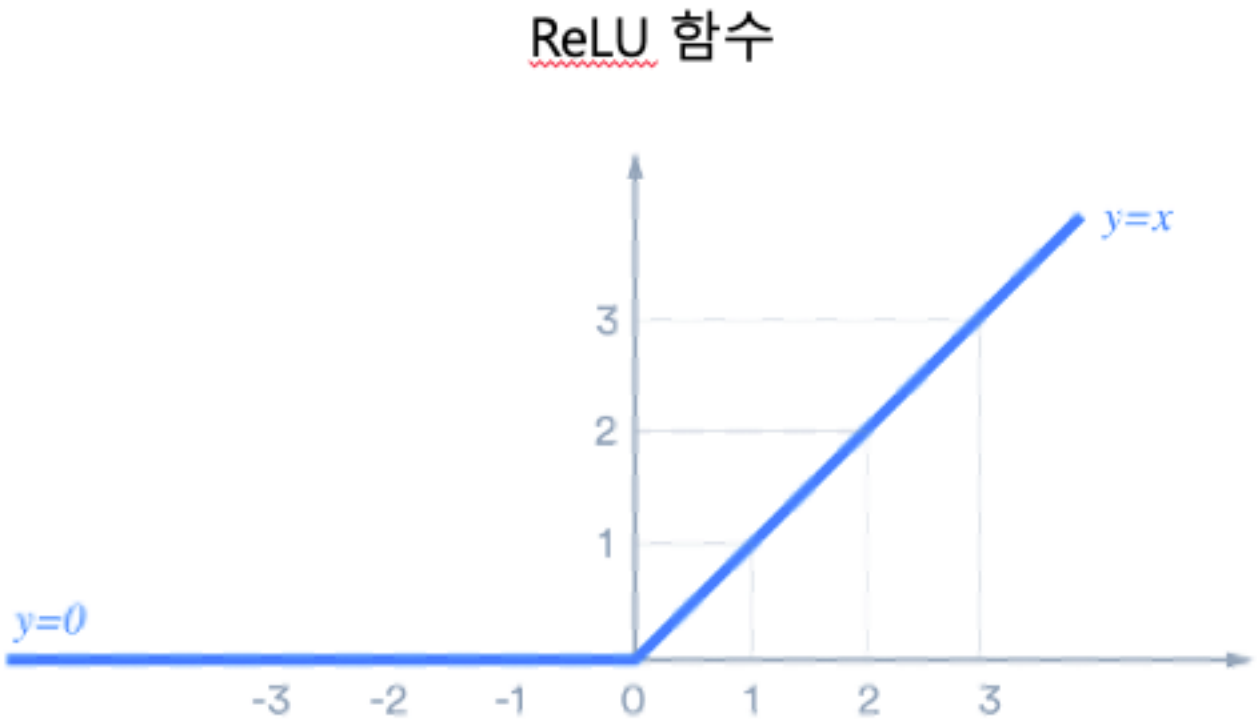
실습: ch02/sigmoid.ipynb

```python
# coding: utf-8
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()

```

Sigmoid 공식을 구현한 함수



활성화 함수(Activation function)의 종류(5)



<실습 과제>

Relu 함수를 구현해보자

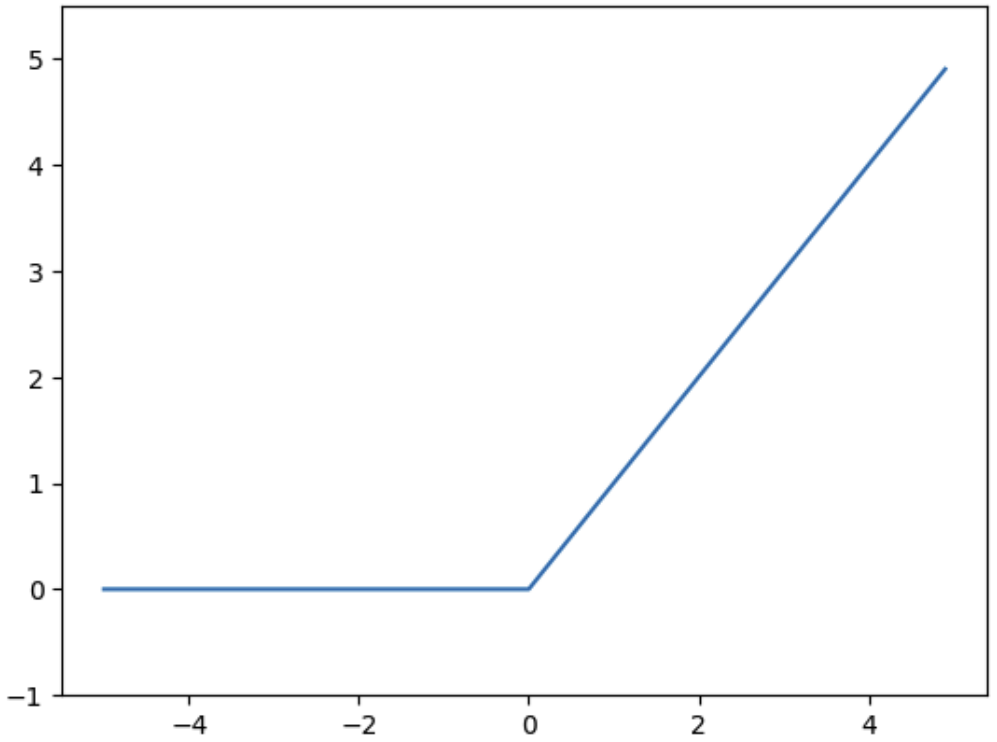
참고> ch02/relu.ipynb에 구현

활성화 함수(Activation function)의 종류(6)

실습: ch02/relu.ipynb

```python
# coding: utf-8
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()

```

x가 0보다 큰 경우에 만 x를 그대로 출력



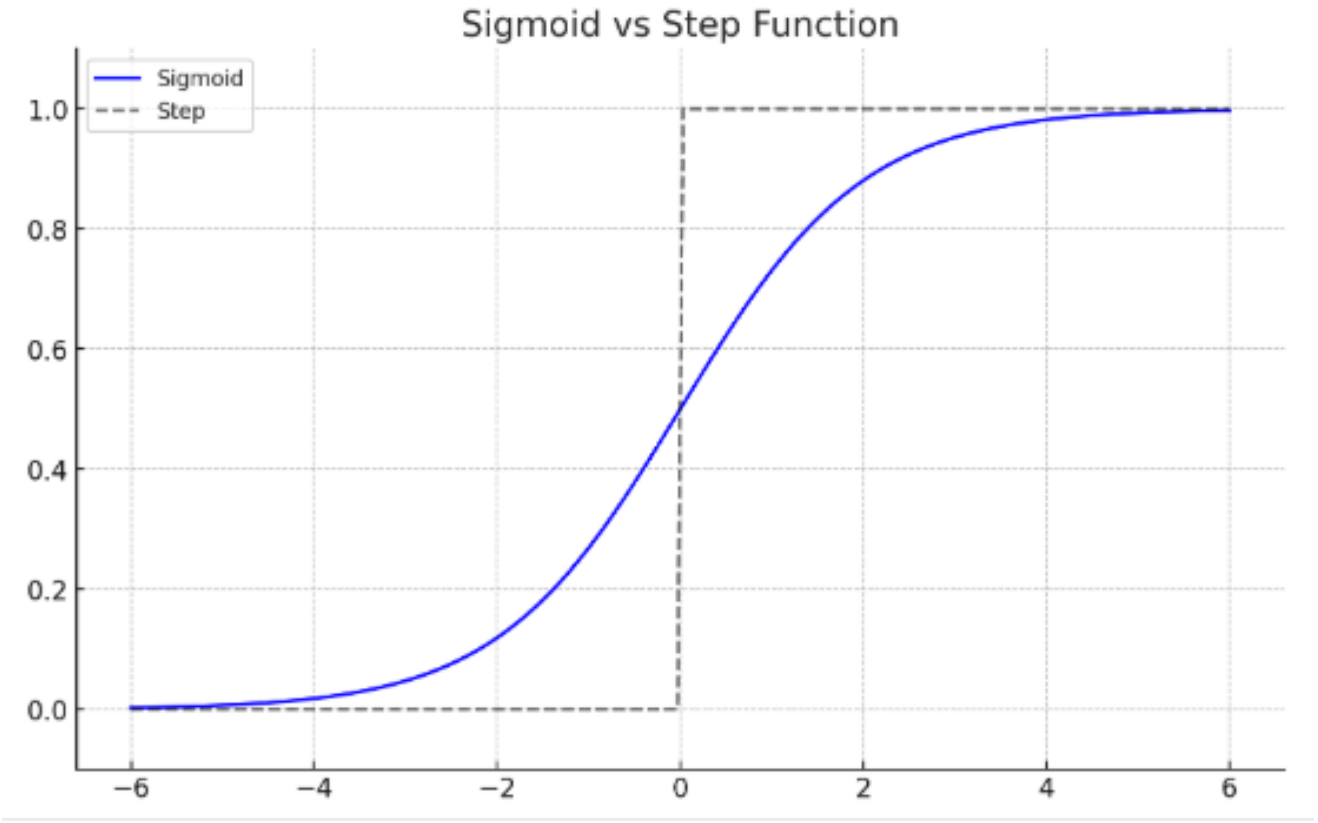
Sigmoid 함수와 계단함수 비교

◆ 계단 함수

- 0을 경계로 출력이 갑자기 변화

◆ Sigmoid 함수

- 부드러운 곡선
- 입력에 따라 출력이 연속적으로 변화
- 신경망에서 중요한 역할



목 차

<실습 과제>

행렬 연산을 연습해보자

ch02/Mat_manipulation.ipynb 에 작성

다차원 배열 : 행렬

실습: ch02/Mat_manipulation.ipynb

◆ Numpy library 활용

- 차원수는 ndim으로 확인
- 형상은 shape으로 확인
- 처음 차원은 0, 다음은 1번째 차원
- 특히 2차원 배열을 matrix라고 부름
- 가로 방향을 row
- 세로 방향을 column

```
import numpy as np
```
✓ 0.0s

```
A = np.array([1,2,3,4])
print(A)
print(np.ndim(A))
print(A.shape)
```
✓ 0.0s

```
[1 2 3 4]
1
(4,)
```

```
B = np.array([[1, 2], [3, 4], [5, 6]])
print(B)
print(np.ndim(B))
print(B.shape)
```
✓ 0.0s

```
[[1 2]
 [3 4]
 [5 6]]
2
(3, 2)
```

행렬의 연산(1)

◆ 덧셈

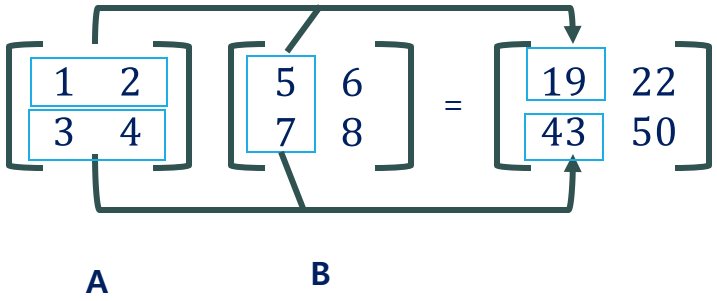
예 : $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}$

◆ 상수배

예 : $2\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 & 2 \cdot 2 \\ 2 \cdot 3 & 2 \cdot 4 \end{pmatrix}$

◆ 곱셈1

예 :



실습: ch02/Mat_manipulation.ipynb

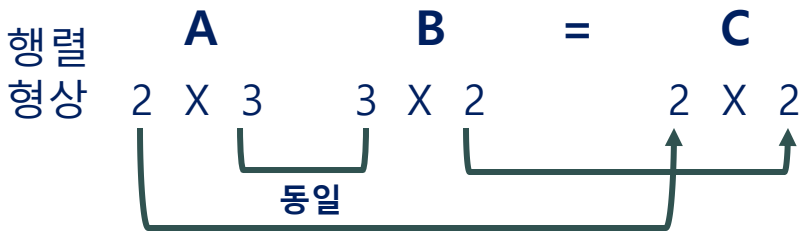
```
A = np.array([[1, 2], [3, 4]])
print(A.shape)
B = np.array([[5, 6], [7, 8]])
print(B.shape)
np.dot(A, B)
```

✓ 0.0s

```
(2, 2)
(2, 2)

array([[19, 22],
       [43, 50]])
```

행렬의 연산(3)

◆ 곱셈2



실습: ch02/Mat_manipulation.ipynb

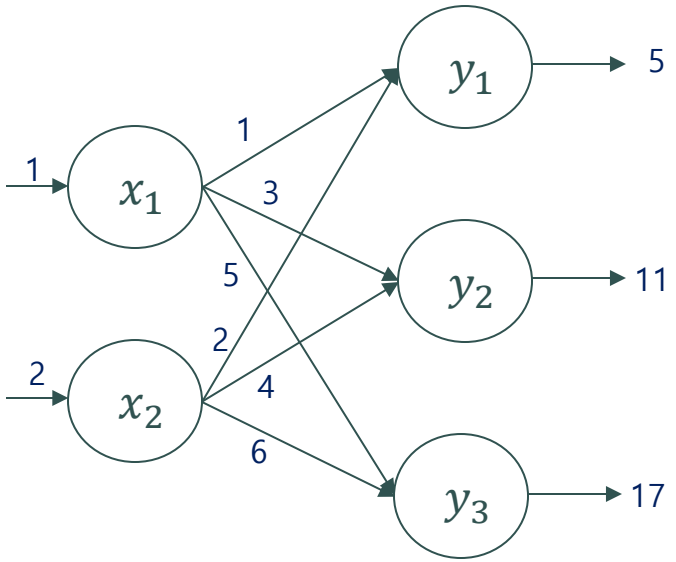
```
A = np.array([[1, 2, 3], [4, 5, 6]])
print(A.shape)
print(A)

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

np.dot(A, B)
```

✓ 0.0s

```
(2, 3)
[[1 2 3]
 [4 5 6]]
(3, 2)
[[1 2]
 [3 4]
 [5 6]]

array([[22, 28],
       [49, 64]])
```

신경망에서의 행렬의 곱

◆ 행렬의 곱셈과 작동방식이 동일함



$$\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 5 & 11 & 17 \end{bmatrix}$$

X **W**

실습: ch02/Mat_manipulation.ipynb

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

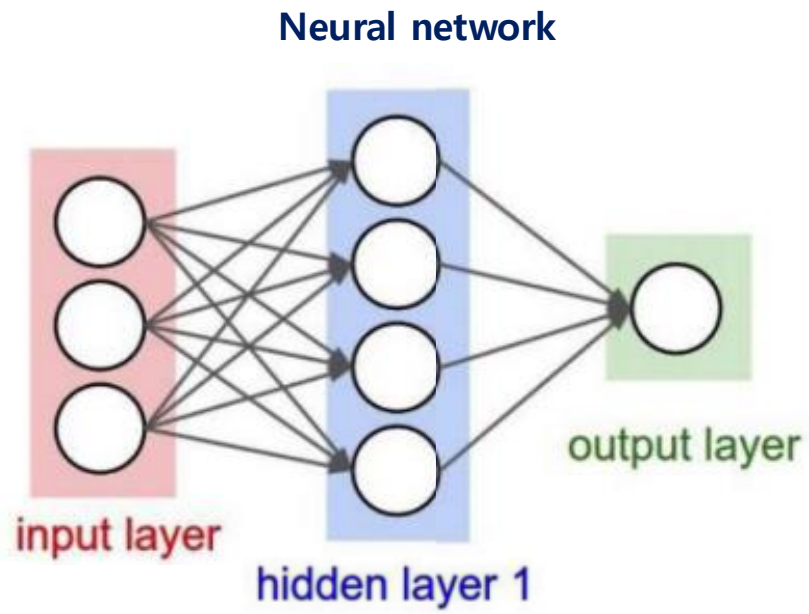
```
[1 2]
[[1 3 5]
 [2 4 6]]
[ 5 11 17]
```

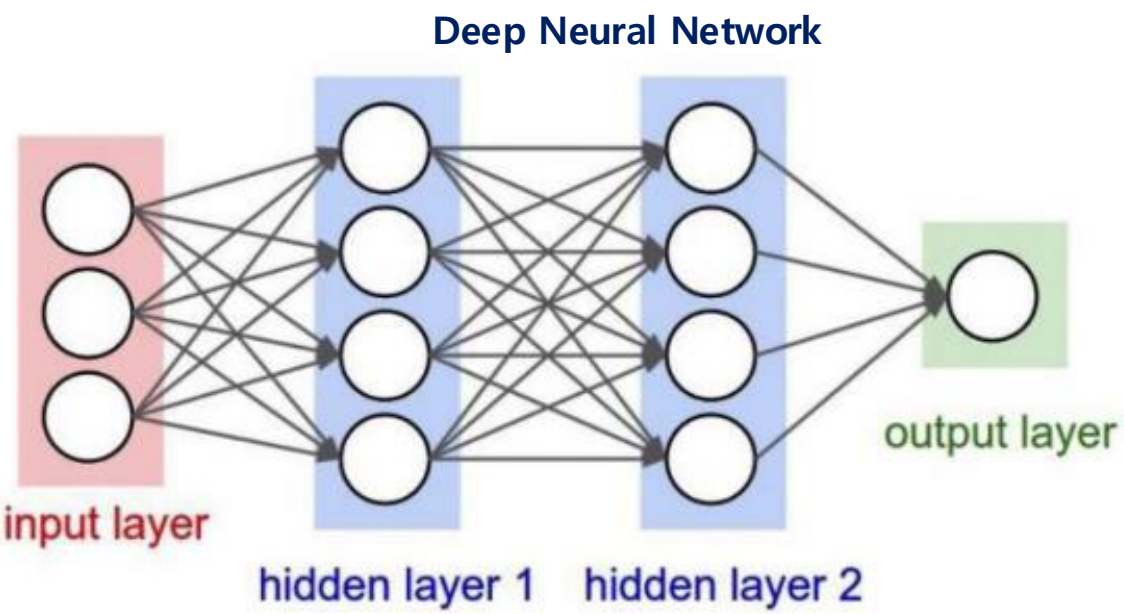
목 차

다층 퍼셉트론 : MLP(Multi Layer Perceptron)

◆ MLP : 인공신경망의 초기 모델 - 선형 모델인 퍼셉트론을 보완한 모델.

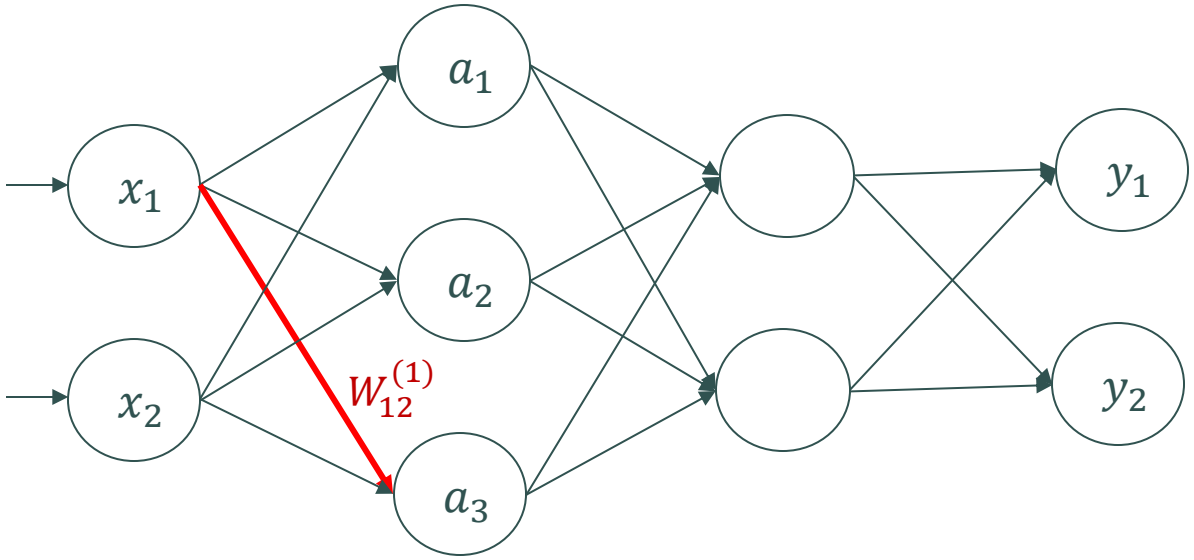
- 입력층(Input layer) : input 데이터
- 은닉층(Hidden layer) : 여러 퍼셉트론의 조합 (비선형 활성화 함수 사용)
- 출력층(Output layer) : 모델의 최종 결과값

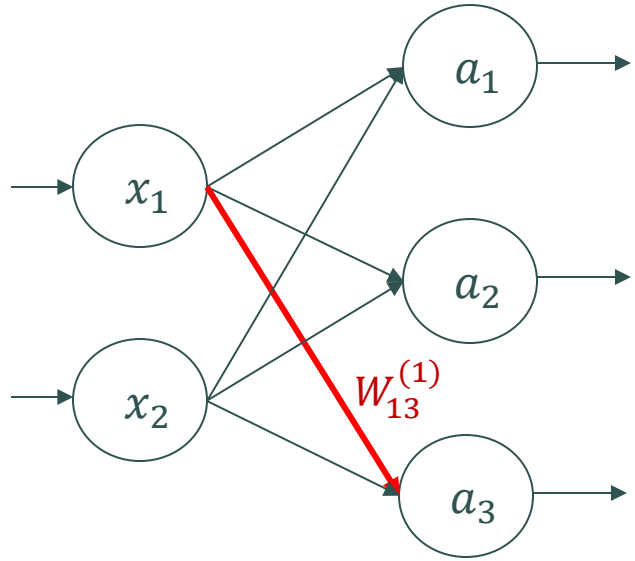




3층 신경망 구현하기

◆ 입력 2개, 출력 2개, 은닉층 2개인 3층 신경망 기본 구조와 표기법





$W_{13}^{(1)}$

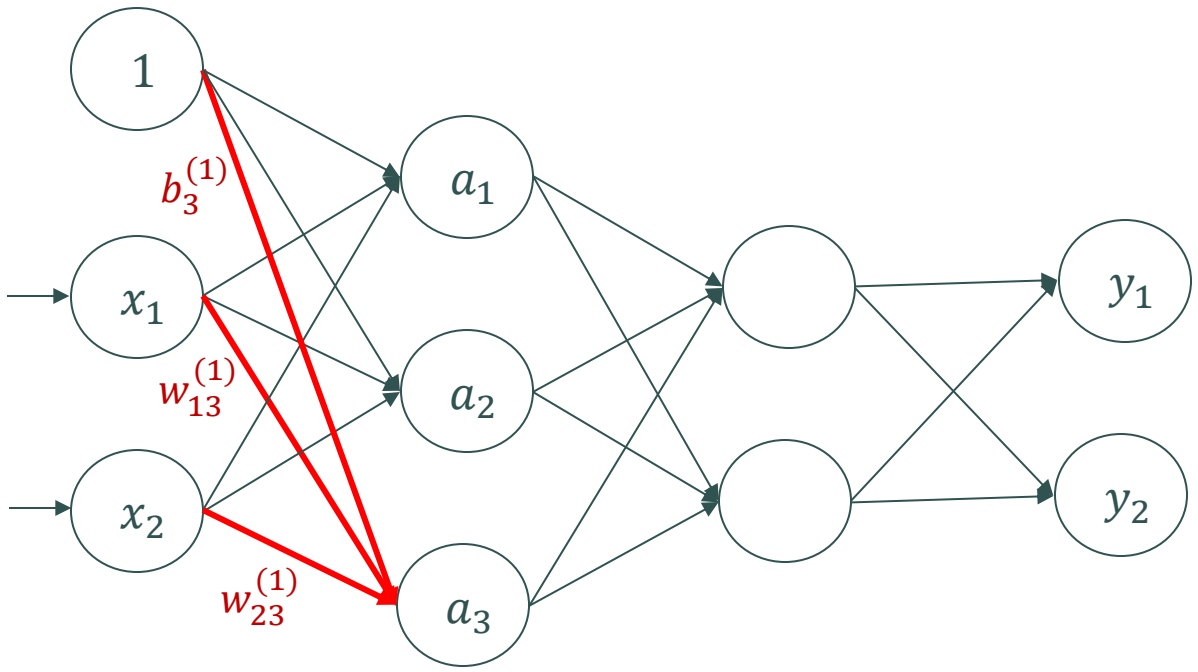
1층 가중치

앞 층 1번째 뉴런

다음 층 3번째 뉴런

각 층의 신호전달 구현하기(1)

◆ 입력층에서 1층으로 신호 전달



입력층 : $X = (x_1, x_2)$

편향값 : $B^{(1)} = (b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$

출력층 : $A^{(1)} = (a_1^{(1)}, a_2^{(1)}, a_3^{(1)})$

가중치 : $w^{(1)} = \begin{bmatrix} w_{12}^{(1)}, w_{12}^{(1)}, w_{13}^{(1)} \\ w_{21}^{(1)}, w_{22}^{(1)}, w_{23}^{(1)} \end{bmatrix}$

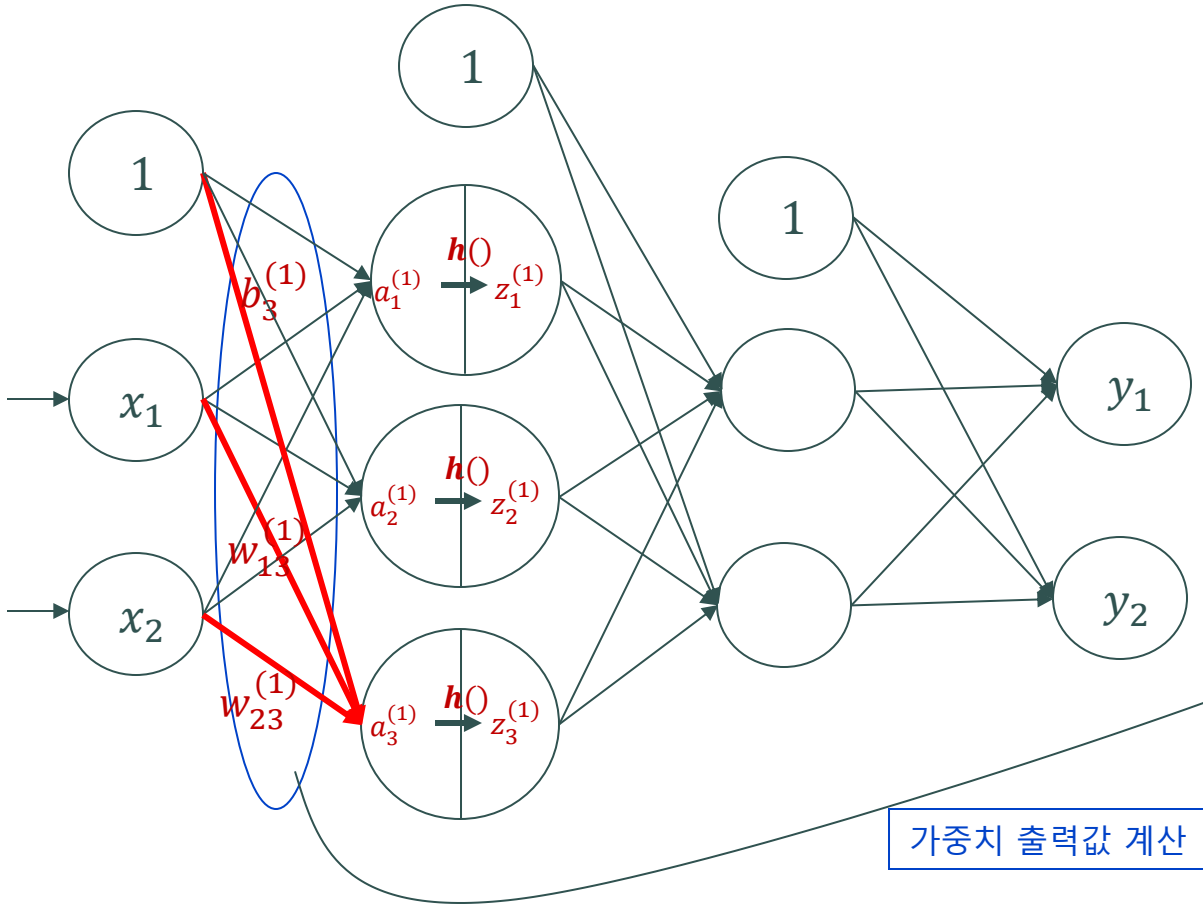
$$a_3^{(1)} = w_{13}^{(1)} * x_1^{(1)} + w_{23}^{(1)} * x_2^{(1)} + b_3^{(1)}$$

$$A^{(1)} = X^{(1)} * W^{(1)} + B^{(1)}$$

각 층의 신호전달 구현하기(2)

◆ 입력층에서 1층으로의 신호 전달

실습: ch02/SimpleNeuralnet.ipynb



가중치 출력값 계산

```
import numpy as np
import os, sys

current_dir = os.getcwd()
parent_dir = os.path.dirname(current_dir)
sys.path.append(parent_dir)
from common.functions import *
```

```
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])

print('X = ', X)
print('B1 = ', B1)
print('W1 = ', W1)

A1 = np.dot(X, W1) + B1
print('X * W1 + B1 =', A1)
```

✓ 0.0s

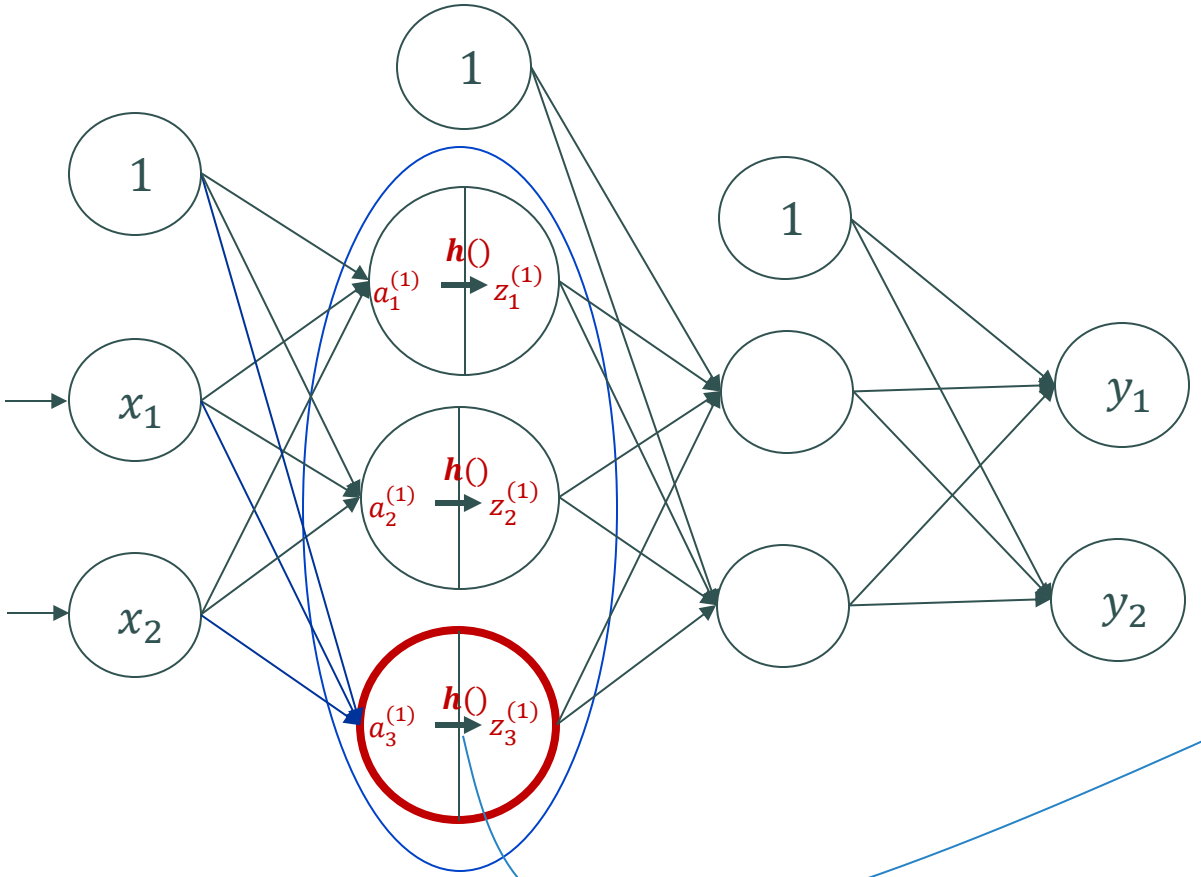
```
X =  [1.  0.5]
B1 =  [0.1 0.2 0.3]
W1 =  [[0.1 0.3 0.5]
 [0.2 0.4 0.6]]
X * W1 + B1 = [0.3 0.7 1.1]
```

실행결과

각 층의 신호전달 구현하기(3)

실습: ch02/SimpleNeuralnet.ipynb

◆ 활성화 함수의 적용



활성화 함수 적용

```
Z1 = sigmoid(A1)
print('A1 = ', A1)
print('Z1 = ', Z1)
```

✓ 0.0s

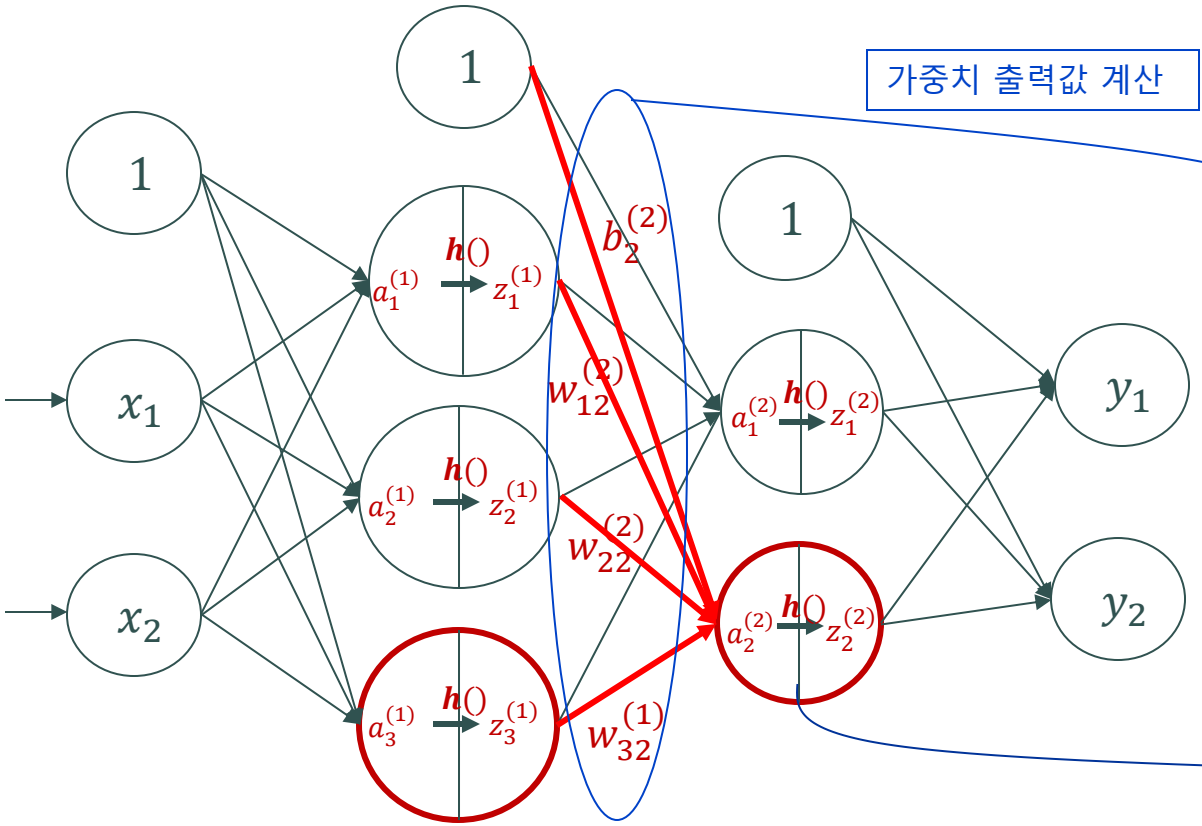
실행결과

```
A1 =  [0.3 0.7 1.1]
Z1 =  [0.57444252 0.66818777 0.75026011]
```

각 층의 신호전달 구현하기(4)

실습: ch02/SimpleNeuralnet.ipynb

◆ 1층에서 2층으로 신호 전달



가중치 출력값 계산

```
W2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
B2 = np.array([0.1, 0.2])

print('B2 = ', B2)
print('W2 = ', W2)

A2 = np.dot(Z1, W2) + B2
print('Z1 * W2 + B2 =', A2)
```

✓ 0.0s

실행결과

```
B2 =  [0.1 0.2]
W2 =  [[0.1 0.4]
 [0.2 0.5]
 [0.3 0.6]]
Z1 * W2 + B2 = [0.51615984 1.21402696]
```

```
print('A2 = ', A2)

Z2 = sigmoid(A2)

print('Z2 = ', Z2)
```

✓ 0.0s

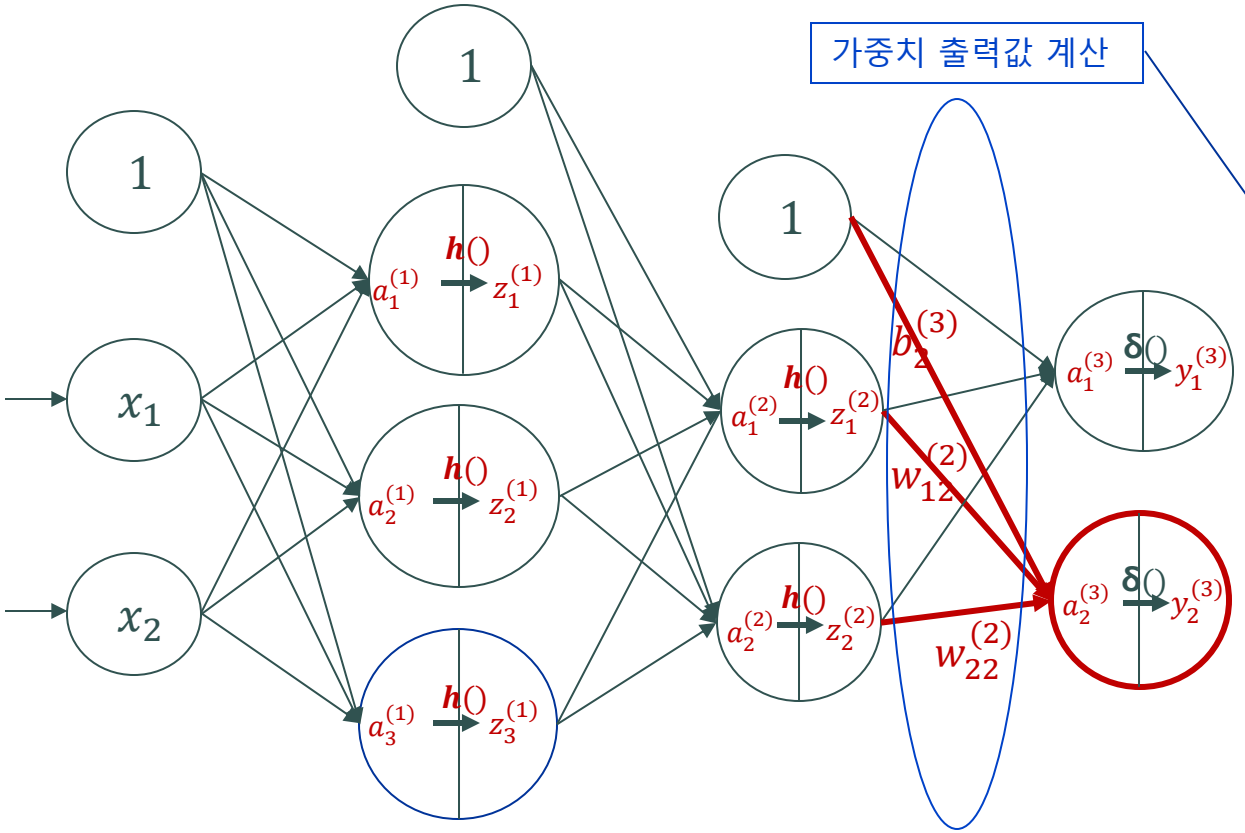
```
A2 =  [0.51615984 1.21402696]
Z2 =  [0.62624937 0.7710107 ]
```

실행결과

각 층의 신호전달 구현하기(5)

실습: ch02/SimpleNeuralnet.ipynb

◆ 2층에서 3층으로 신호 전달 구현




```
W3 = np.array([[0.2, 0.5], [0.3, 0.6]])
B3 = np.array([0.1, 0.2])

print('B3 = ', B3)
print('W3 = ', W3)

A3 = np.dot(Z2, W3) + B3
print('Z2 * W3 + B3 =', A3)
```

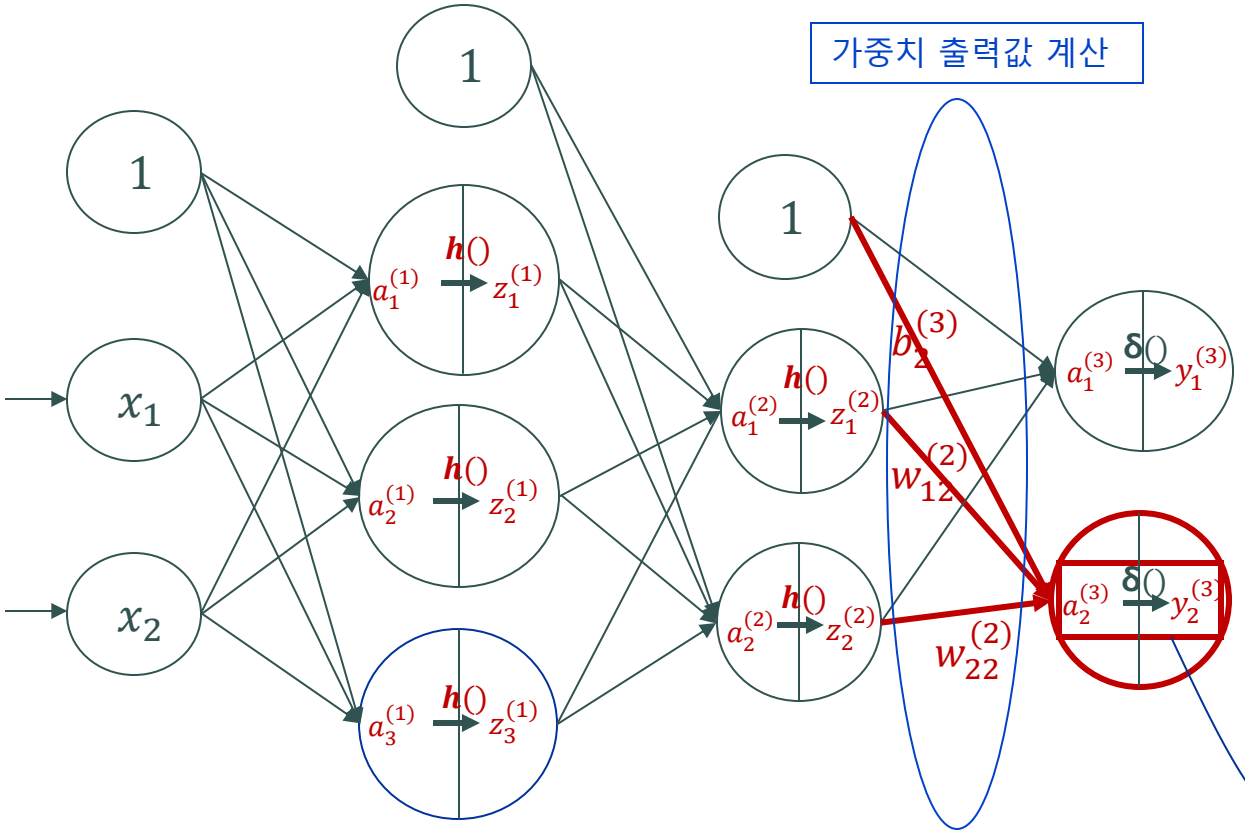
✓ 0.0s

```
B3 =  [0.1 0.2]
W3 =  [[0.2 0.5]
 [0.3 0.6]]
Z2 * W3 + B3 = [0.45655308 0.9757311 ]
```

목 차

Softmax 변환(1)

◆ 최종 출력단에서 Softmax 변환을 적용한다.



$(a_1, a_2, a_3, \cdots, a_n)$

e^x 을 적용해서 양수로

$(e^{a_1}, e^{a_2}, e^{a_3}, \cdots, e^{a_n})$

normalize

$$\left(\frac{e^{a_1}}{e^{a_1}+e^{a_2}+\cdots+e^{a_n}}, \frac{e^{a_2}}{e^{a_1}+e^{a_2}+\cdots+e^{a_n}}, \cdots, \frac{e^{a_n}}{e^{a_1}+e^{a_2}+\cdots+e^{a_n}}\right)$$

Softmax 변환(2)

◆ 정의

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- z_i는 입력 벡터 z의 i번째 원소
- K는 입력 벡터 z의 원소수
- e 는 자연상수(약 2.718)

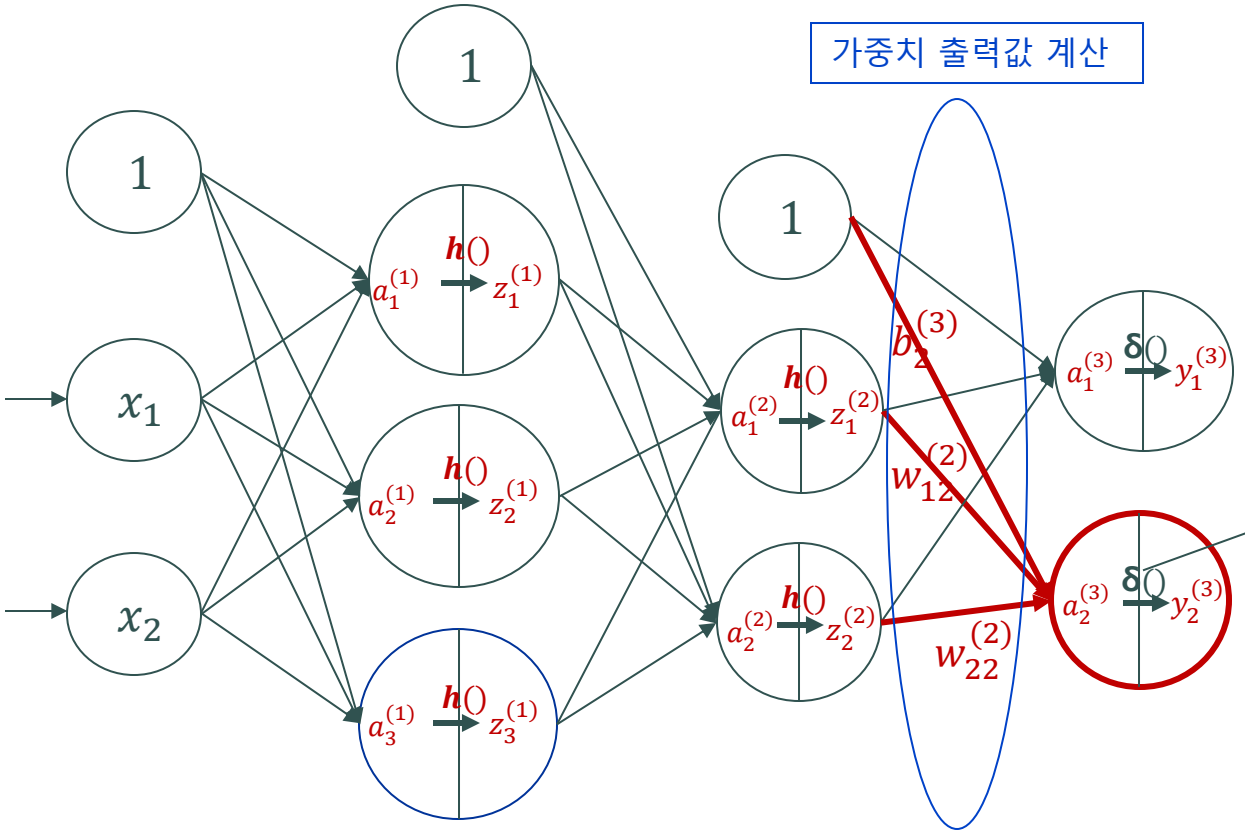
◆ 성질

- 확률분포 : Softmax 함수의 출력값들은 항상 0과 1사이의 값이며, 모든 출력의 합은 1. 따라서 출력벡터를 확률분포로 해석할 수 있음.
- 미분 가능성 : Softmax 함수는 미분 가능하므로 역전파 학습과정에서 사용 가능
- 입력값의 상대적 크기 반영 : 입력값의 상대적 크기에 따라 출력을 조정하며, 큰 입력값일수록 더 큰 확률을 가짐
- 순서보존 : 입력값의 순서는 출력 값의 순서에 영향을 미침.

Softmax 변환(3)

실습: ch02/SimpleNeuralnet.ipynb

◆ 출력값의 해석




```
print('A3 = ', A3)
Y = softmax(A3)
print('Y = ', Y)
```

✓ 0.0s

```
A3 =  [0.45655308 0.9757311 ]
Y =  [0.37304446 0.62695554]
```

두번째 원소가 63%의 확률로 정답일 가능성이 제일 높다.

Softmax 변환(4)

◆ Softmax 함수 구현시 주의점

- 너무 높은 출력값을 사용하여 Softmax를 적용하면 NaN이 나오는 현상 발생함.
- 입력 신호 중 최대값(eg. c)를 빼준 후 Softmax를 적용하여 에러 방지

실습과제

<실습 과제>

Simple neural net을 구현해 보자

참고>ch02/SimpleNeuralnet.ipynb에 작성함

실습

필요한 라이브러리 Import

```
[1]
import numpy as np
import os, sys

current_dir = os.getcwd()
parent_dir = os.path.dirname(current_dir)
sys.path.append(parent_dir)
from common.functions import *
```

입력층에서 1층으로 신호 전달

```
[2]
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])

print('X = ', X)
print('B1 = ', B1)
print('W1 = ', W1)

A1 = np.dot(X, W1) + B1
print('X * W1 + B1 =', A1)
```

```
X =  [1.  0.5]
B1 =  [0.1 0.2 0.3]
W1 =  [[0.1 0.3 0.5]
 [0.2 0.4 0.6]]
X * W1 + B1 = [0.3 0.7 1.1]
```

활성화 함수의 적용

```
[3]
print('A1 = ', A1)

Z1 = sigmoid(A1)

print('Z1 = ', Z1)
```

```
A1 =  [0.3 0.7 1.1]
Z1 =  [0.57444252 0.66818777 0.75026011]
```

1층에서 2층으로 신호 전달

```
[4]
W2 = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
B2 = np.array([0.1, 0.2])

print('B2 = ', B2)
print('W2 = ', W2)

A2 = np.dot(Z1, W2) + B2
print('Z1 * W2 + B2 =', A2)
```

```
B2 =  [0.1 0.2]
W2 =  [[0.1 0.4]
 [0.2 0.5]
 [0.3 0.6]]
Z1 * W2 + B2 = [0.51615984 1.21402696]
```

실습

활성화 함수의 적용

```
[5]
1  print('A2 = ', A2)
2
3  Z2 = sigmoid(A2)
4
5  print('Z2 = ', Z2)
```

```
A2 =  [0.51615984 1.21402696]
Z2 =  [0.62624937 0.7710107 ]
```

2층에서 3층으로 신호 전달

```
[6]
1  W3 = np.array([[0.2, 0.5], [0.3, 0.6]])
2  B3 = np.array([0.1, 0.2])
3
4  print('B3 = ', B3)
5  print('W3 = ', W3)
6
7  A3 = np.dot(Z2, W3) + B3
8  print('Z2 * W3 + B3 =', A3)
```

```
B3 =  [0.1 0.2]
W3 =  [[0.2 0.5]
 [0.3 0.6]]
Z2 * W3 + B3 = [0.45655308 0.9757311 ]
```

최종 출력층에 활성화 함수로 Softmax 적용

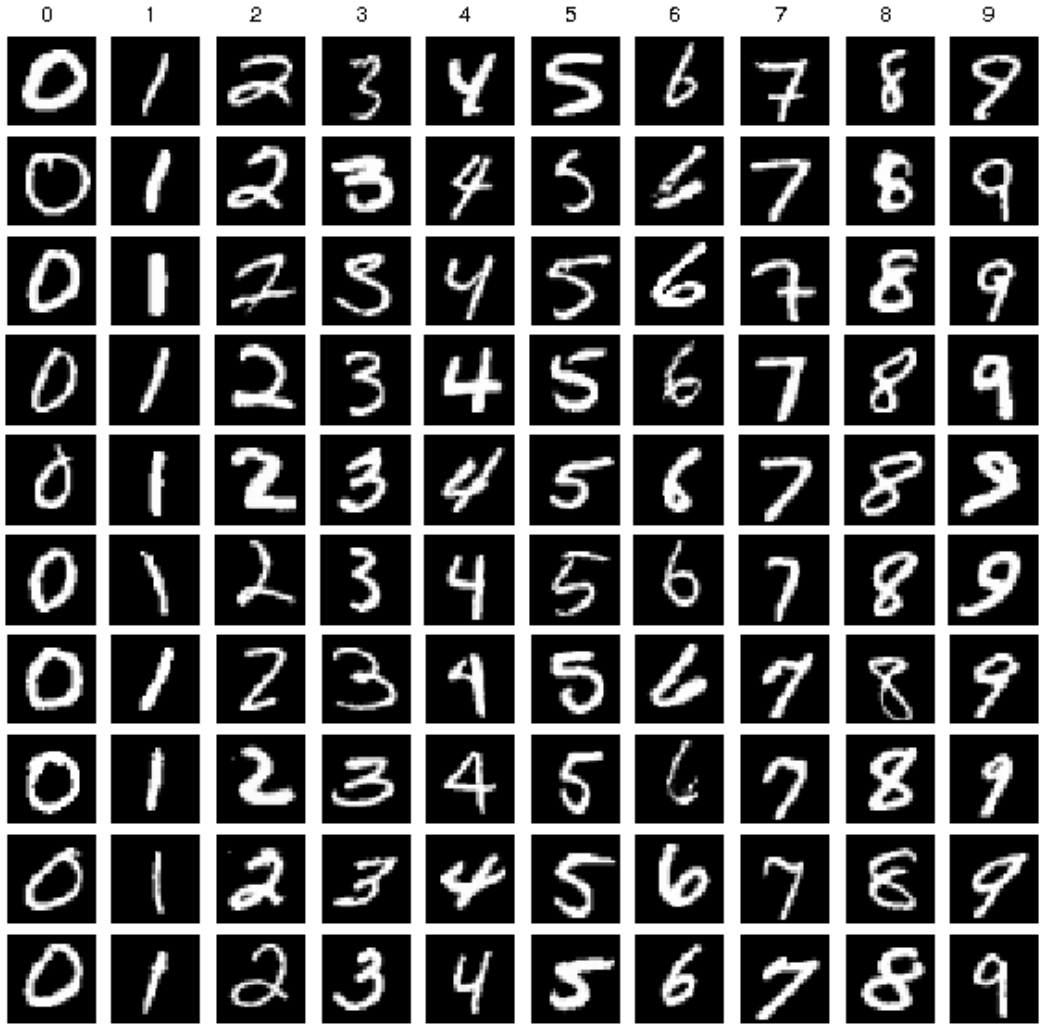
```
[7]
1  print('A3 = ', A3)
2  Y = softmax(A3)
3  print('Y = ', Y)
```

```
A3 =  [0.45655308 0.9757311 ]
Y =  [0.37304446 0.62695554]
```

목 차

숫자인식을 위한 MNIST 데이터 셋

- MNIST 데이터셋은 0부터 9까지의 손글씨 이미지로 구성
- 훈련 데이터가 6만장, 테스트 데이터가 1만장
- 각 데이터는 이미지와 라벨로 이루어짐
- 각 이미지는 28×28 해상도의 흑백 사진
- 각 픽셀은 0에서 255로 밝기 표현



<실습 과제>

MNIST 데이터를 로드하여

화면에 그려보자

ch02/mnist_show.ipynb

실습 과제

실습: ch02/mnist_show.ipynb

```
1   # coding: utf-8
2   import sys, os
3   print(os.getcwd())
4   current_dir = os.path.dirname(os.getcwd())
5   parent_dir = os.path.dirname(current_dir)
6   print(current_dir)
7   sys.path.append(current_dir)
8
9   import numpy as np
10  from dataset.mnist import load_mnist
11  from PIL import Image
12  import matplotlib.pyplot as plt
13
14
15  def img_show(img):
16      plt.imshow(img, cmap='gray')
17      plt.axis('off')  # 축 숨기기
18      plt.show()
19      #pil_img = Image.fromarray(np.uint8(img))
20      #pil_img.show()
21
22  (x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)
23
24  img = x_train[0]
25  label = t_train[0]
26  print(label)  # 5
27
28  print(img.shape)  # (784,)
29  img = img.reshape(28, 28)  # 형상을 원래 이미지의 크기로 변형
30  print(img.shape)  # (28, 28)
31
32  img_show(img)
```

추가 Library를 위한 PATH 추가

추가 Library import

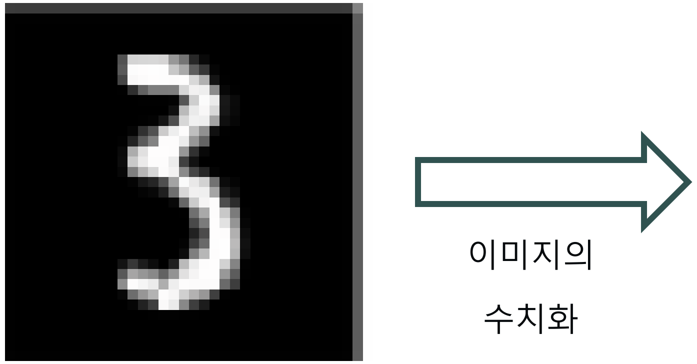
Image drawing 부분

mnist data loading from interrnet

1차원 이미지를 2차원 이미지로 변경

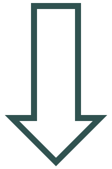
Image matrix

- 28x28 행렬, 0에서 255로 각 픽셀의 밝기를 표현하도록 구성됨.



이미지의 수치화

```
[[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0  38  43 105 255 253 253 253 253 253 174   6   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0  43 139 224 226 252 253 252 252 252 252 252 252 158  14   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0 178 252 252 252 252 253 252 252 252 252 252 252 252  59   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0 109 252 252 230 132 133 132 132 189 252 252 252 252  59   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   4  29  29  24   0   0   0   0  14 226 252 252 172   7   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  85 243 252 252 144   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  88 189 252 252 252  14   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  91 212 247 252 252 252 204   9   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0  32 125 193 193 193 253 252 252 252 238 102  28   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0  45 222 252 252 252 252 253 252 252 252 177   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0  45 223 253 253 253 253 255 253 253 253 253  74   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0  31 123  52  44  44  44  44 143 252 252  74   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  15 252 252  74   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  86 252 252  74   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   5  75   9   0   0   0   0   0   0  98 242 252 252  74   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0  61 183 252  29   0   0   0   0  18  92 239 252 252 243  65   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0 208 252 252 147 134 134 134 134 203 253 252 252 188  83   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0 208 252 252 252 252 252 252 252 252 253 230 153   8   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0  49 157 252 252 252 252 252 217 207 146  45   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   7 103 235 252 172 103  24   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
  [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]]
```



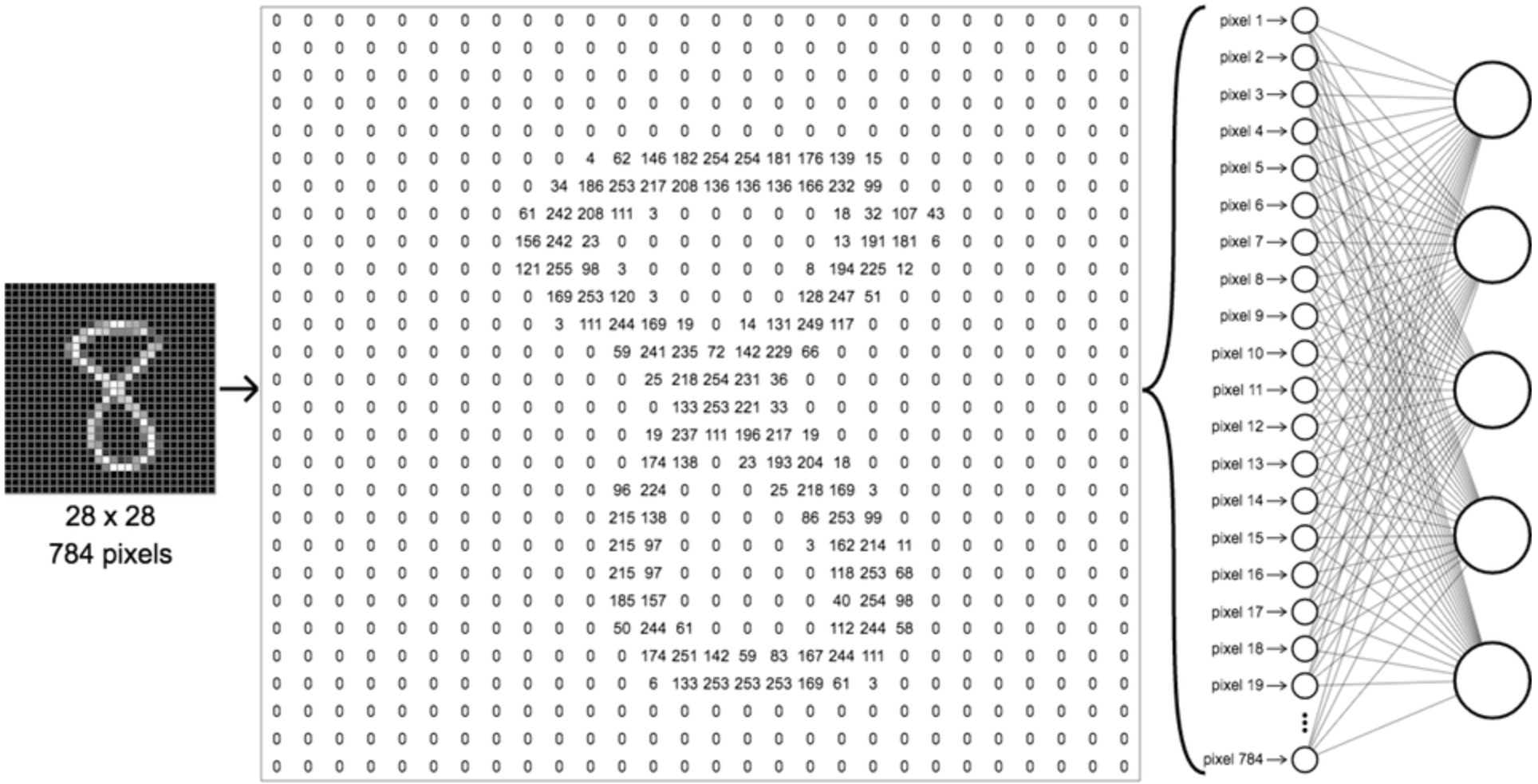
평탄화

(Flatten)

(0, 0, 0, 0, ..., ..., ..., ..., ..., ..., 38, 43, 205, ..., ..., ..., 108, 24, ..., ..., 0, 0, 0)

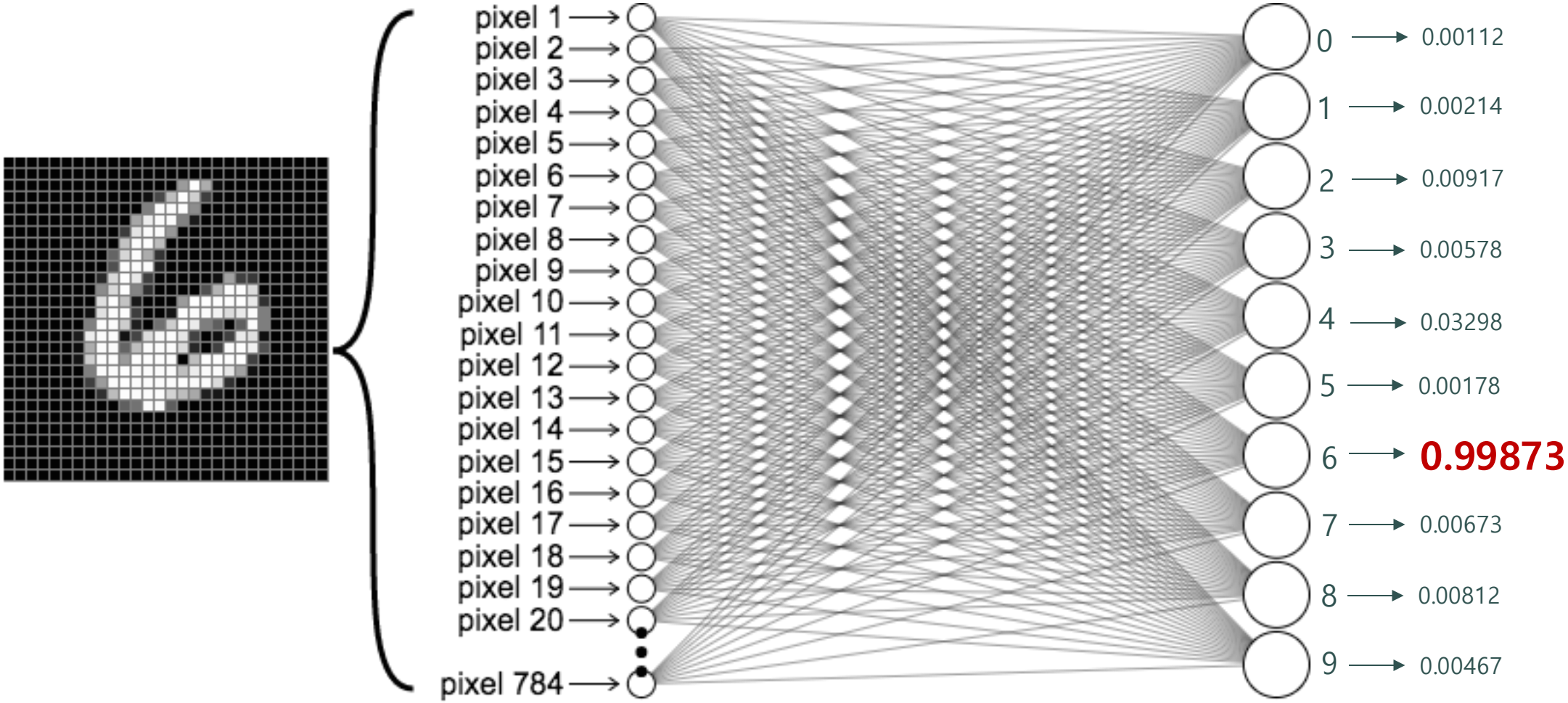
MNIST 데이터의 입력

- MNIST 데이터셋은 0부터 9까지의 손글씨 이미지로 구성



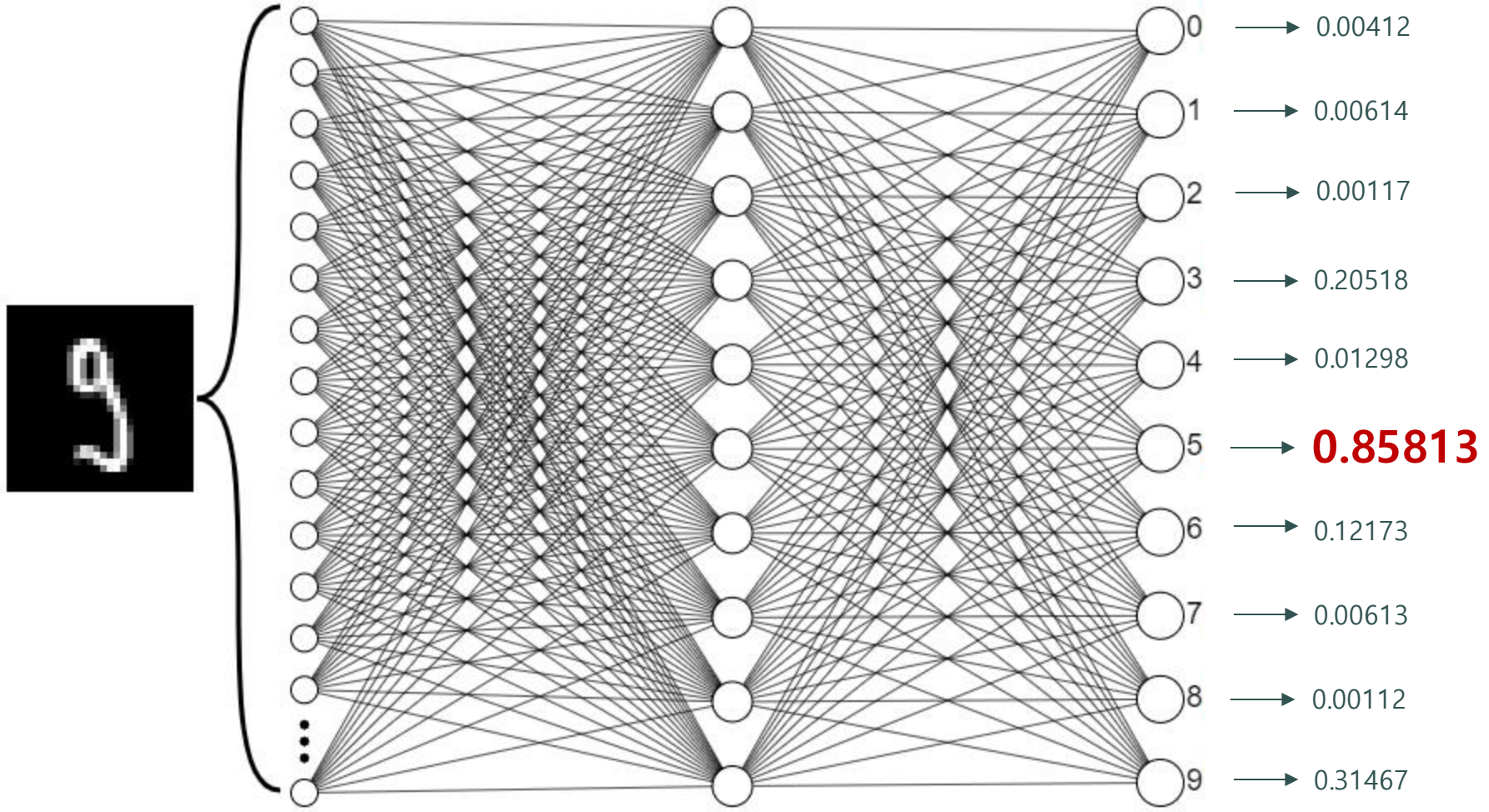
MNIST 숫자 데이터의 인식 결과

- 숫자인식에 성공한 경우



MNIST 숫자 데이터의 인식 결과

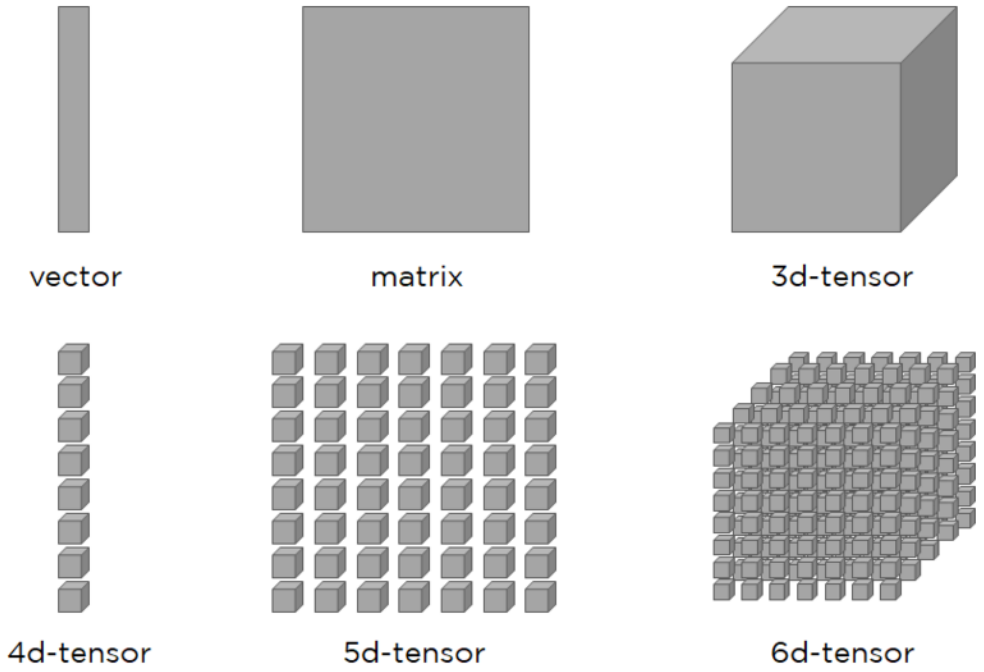
- 숫자인식에 실패한 경우



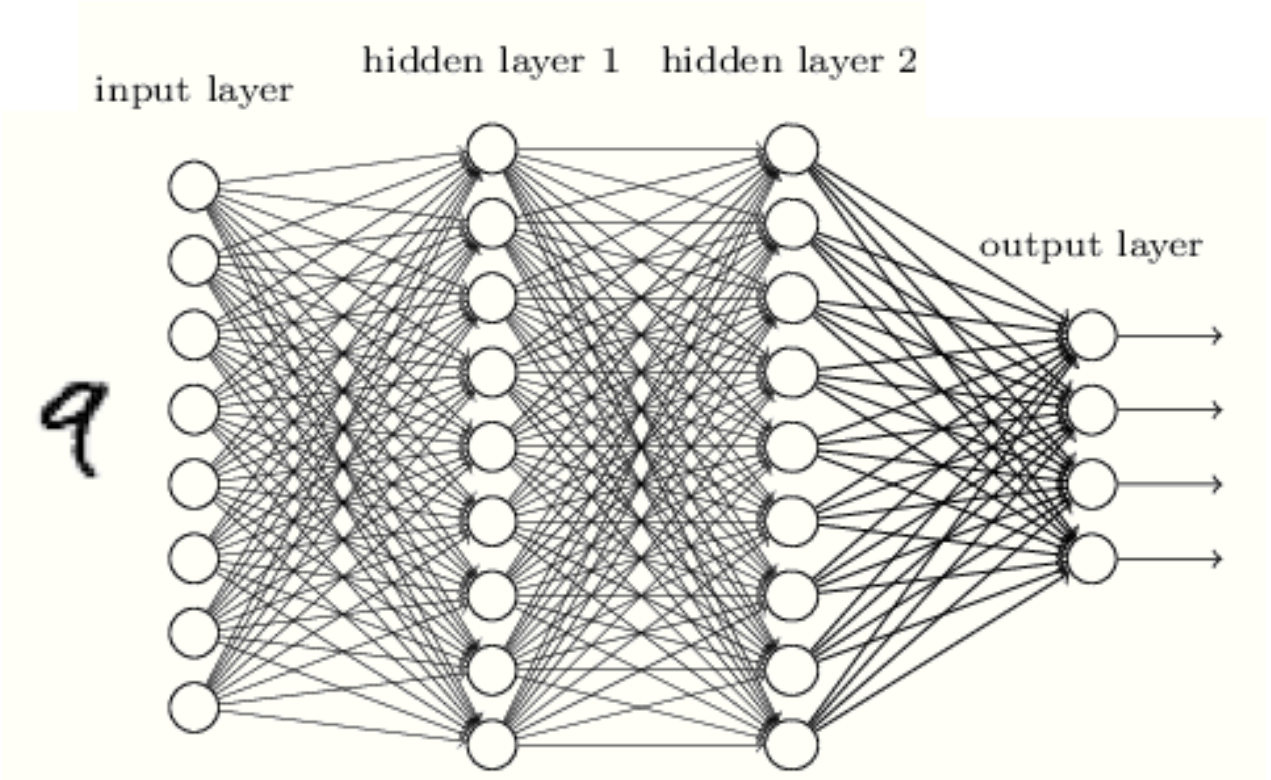
용어 정리

- Scalar, Vector, Mactrix, 3-Tensor, n-Tensor

RANK	TYPE	EXAMPLE
0	scalar	[1]
1	vector	[1,1]
2	matrix	[[1,1],[1,1]]
3	3-tensor	[[[1,1],[1,1]],[[1,1],[1,1]],[[1,2],[2,1]]]
n	n-tensor	



sample_weight.pkl



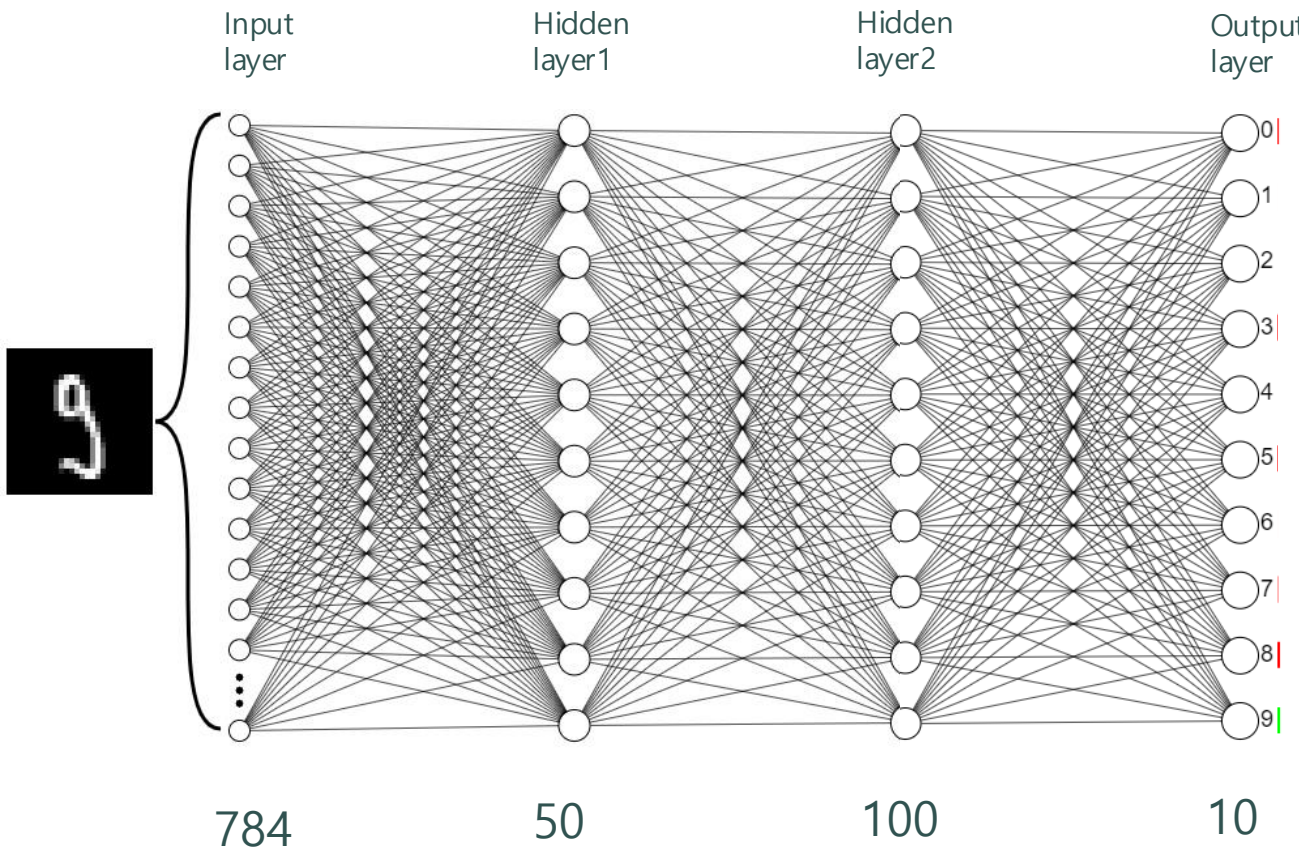
784 50 100 10

- 첫번째 가중치 행렬 : 784×50
- 첫번째 편향 벡터 : 50
- 두번째 가중치 행렬 : 50×100
- 두번째 편향 벡터 : 100
- 세번째 가중치 행렬 : 100×10
- 세번째 편향 벡터 : 10

- parameter 개수 :

784×50+50+50×100+100+100×10+10
= 45360

sample_weight.pkl



- 첫번째 가중치 행렬 : 784×50
- 첫번째 편향 벡터 : 50
- 두번째 가중치 행렬 : 50×100
- 두번째 편향 벡터 : 100
- 세번째 가중치 행렬 : 100×10
- 세번째 편향 벡터 : 10

- parameter 개수 :

784×50+50+50×100+100+100×10+10
= 45360

<실습 과제>

MNIST 데이터로

숫자인식하는

Python code를 작성해보자

ch02/neuralnet_mnist.ipynb

실습 과제

실습: ch02/neuralnet_mnist.ipynb

```
1   # coding: utf-8
2   import os, sys, time
3   print(os.getcwd())
4   current_dir = os.path.dirname(os.getcwd())
5   print(current_dir)
6   os.chdir(current_dir)
7
8   import numpy as np
9   import pickle
10  from dataset.mnist import load_mnist
11  from common.functions import sigmoid, softmax
12
13
14  def get_data():
15      (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, flatten=True, one_hot_label=False)
16      return x_test, t_test
17
18
19  def init_network():
20      print(os.getcwd())
21      with open("ch02/sample_weight.pkl", 'rb') as f:
22          network = pickle.load(f)
23      return network
```

추가 Library import를 위한 Current dir 변경

추가 Library import

숫자 데이터 다운로드 from internet

신경망 가중치값 파일 로딩

실습 과제

실습: ch02/neuralnet_mnist.ipynb

```
26  def predict(network, x):
27      W1, W2, W3 = network['W1'], network['W2'], network['W3']
28      b1, b2, b3 = network['b1'], network['b2'], network['b3']
29
30      a1 = np.dot(x, W1) + b1
31      z1 = sigmoid(a1)
32      a2 = np.dot(z1, W2) + b2
33      z2 = sigmoid(a2)
34      a3 = np.dot(z2, W3) + b3
35      y = softmax(a3)
36
37      return y
38
39
40  x, t = get_data()
41  network = init_network()
42  accuracy_cnt = 0
43  for i in range(len(x)):
44      y = predict(network, x[i])
45      p= np.argmax(y) # 확률이 가장 높은 원소의 인덱스를 얻는다.
46      if p == t[i]:
47          print('Predicted num=', p, '  Original num=', t[i])
48          time.sleep(0.5)
49          accuracy_cnt += 1
50
51  print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
52
```

신경망 출력값 계산

신경망 출력값 계산

예측값과 정답이 맞는지 비교

정답을 맞춘경우 1점 추가

최종 정확도 출력

배치(묶음 처리)

- 숫자 데이터를 개별로 처리하면 I/O에서 병목현상이 발생함.
- 입력 데이터 묶음 처리를 통해 I/O에 주는 부하를 줄임.

$$a_1^{(k)} = w_{11}^{(k)} x_1 + w_{21}^{(k)} x_2 + b_1^{(k)} \qquad a'_1{}^{(k)} = w_{11}^{(k)} x'_1 + w_{21}^{(k)} x'_2 + b_1^{(k)}$$

$$a_2^{(k)} = w_{12}^{(k)} x_1 + w_{22}^{(k)} x_2 + b_2^{(k)} \qquad a'_2{}^{(k)} = w_{12}^{(k)} x'_1 + w_{22}^{(k)} x'_2 + b_2^{(k)}$$

$$a_3^{(k)} = w_{13}^{(k)} x_1 + w_{23}^{(k)} x_2 + b_3^{(k)} \qquad a'_3{}^{(k)} = w_{13}^{(k)} x'_1 + w_{23}^{(k)} x'_2 + b_3^{(k)}$$

$$\Updownarrow$$

$$\begin{pmatrix} a_1^{(k)} & a_2^{(k)} & a_3^{(k)} \\ a'_1{}^{(k)} & a'_2{}^{(k)} & a'_3{}^{(k)} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \\ x'_1 & x'_2 \end{pmatrix} \begin{pmatrix} w_{11}^{(k)} & w_{12}^{(k)} & w_{13}^{(k)} \\ w_{21}^{(k)} & w_{22}^{(k)} & w_{23}^{(k)} \end{pmatrix} + \begin{pmatrix} b_1^{(k)} & b_2^{(k)} & b_3^{(k)} \\ b_1^{(k)} & b_2^{(k)} & b_3^{(k)} \end{pmatrix}$$

<실습 과제>

MNIST 데이터로

배치로 숫자인식하는

Python code를 작성해보자

ch02/neuralnet_mnist_batch.ipynb

실습 과제

실습: ch02/neuralnet_mnist_batch.ipynb

```python
# coding: utf-8
import sys, os
from pathlib import Path

print(os.getcwd())
current_dir = os.path.dirname(os.getcwd())
print(current_dir)
os.chdir(current_dir)

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("ch02/sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network

```

Library import를 위한 현재 디렉터리 변경

Library import

숫자 데이터 다운로드 from internet

신경망 가중치 값 로딩

실습 과제

실습: ch02/neuralnet_mnist_batch.ipynb

```
26
27  def predict(network, x):
28      w1, w2, w3 = network['W1'], network['W2'], network['W3']
29      b1, b2, b3 = network['b1'], network['b2'], network['b3']
30
31      a1 = np.dot(x, w1) + b1
32      z1 = sigmoid(a1)
33      a2 = np.dot(z1, w2) + b2
34      z2 = sigmoid(a2)
35      a3 = np.dot(z2, w3) + b3
36      y = softmax(a3)
37
38      return y
39
40
41  x, t = get_data()
42  network = init_network()
43
44  batch_size = 100 # 배치 크기
45  accuracy_cnt = 0
46
47  for i in range(0, len(x), batch_size):
48      x_batch = x[i:i+batch_size]
49      y_batch = predict(network, x_batch)
50      p = np.argmax(y_batch, axis=1)
51      accuracy_cnt += np.sum(p == t[i:i+batch_size])
52
53  print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

신경망 출력값 계산

배치(묶음)의 크기 지정

배치(묶음) 단위로 예측 작업 수행