밑바닥부터 시작하는 딥러닝

3장 신경망

김태우

CONTENTS

01 신경망 소개

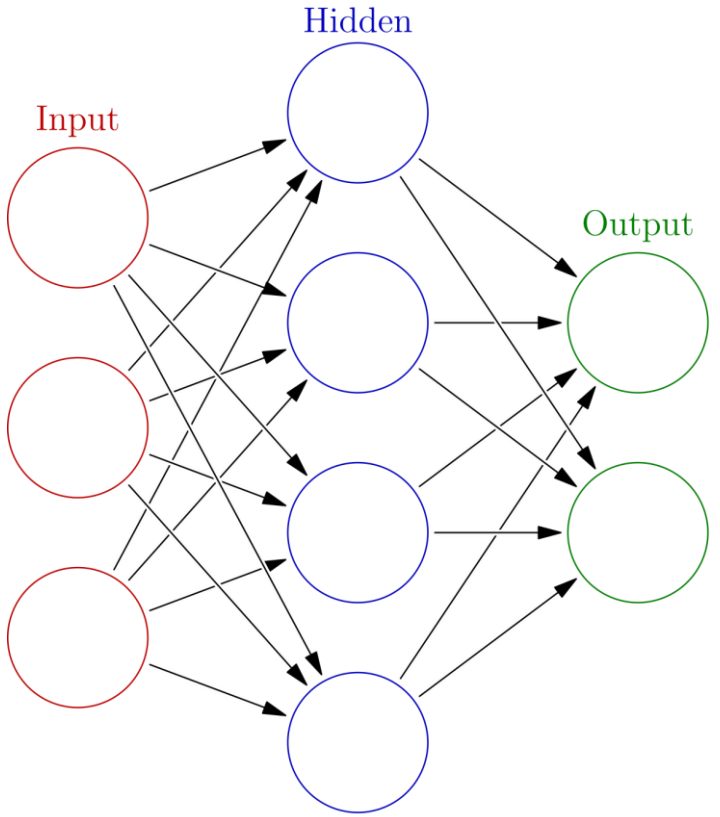
구현 방법 : 퍼셉트론 형태

퍼셉트론은 가중치를 직접 지정해야하지만, 신경망은 스스로 학습

신경망의 가장 왼쪽 층: 입력 층

신경망의 가장 오른쪽 층 : 출력층

입력층과 출력층 사이 층 : 은닉층

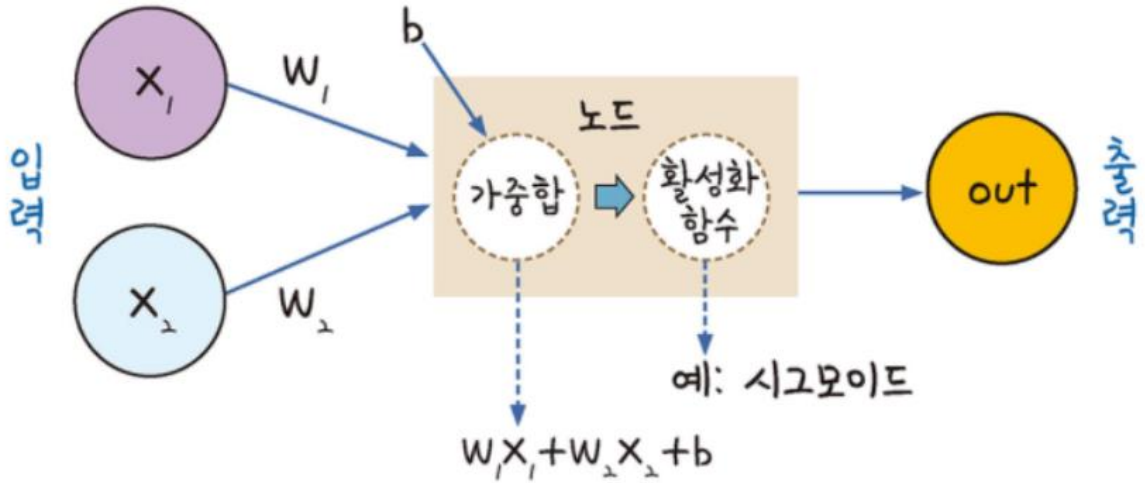


02 활성화 함수

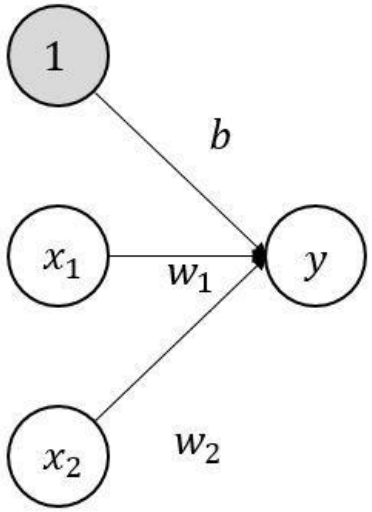
정의 : 입력 신호의 총합을 출력 신호로 변환해주는 함수

필요성 : 출력 값의 범위를 지정할 때 필요

활성화 함수가 적용되는 형태



입력 퍼셉트론



계단 활성화 함수

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

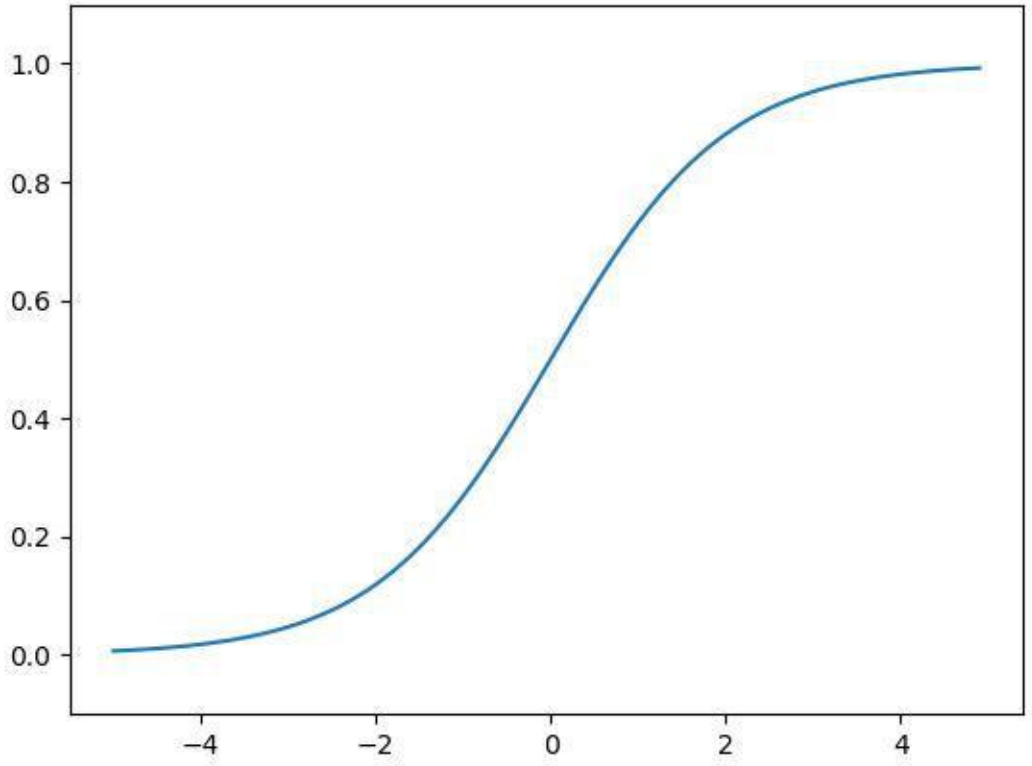
$$h(x) = \begin{cases} 0\ (x \le 0) \\ 1\ (x > 0) \end{cases}$$

02 활성화 함수

시그모이드 함수

함수 식 : $$y = \frac{1}{1 + \exp(-x)}$$

```
1  def sigmoid(x):
2      return 1/(1+np.exp(-x))
3
4  x = np.arange(-5,5,0.1)
5  y = sigmoid(x)
6  plt.plot(x,y)
7  plt.ylim(-0.1,1.1)
8  plt.show()
```

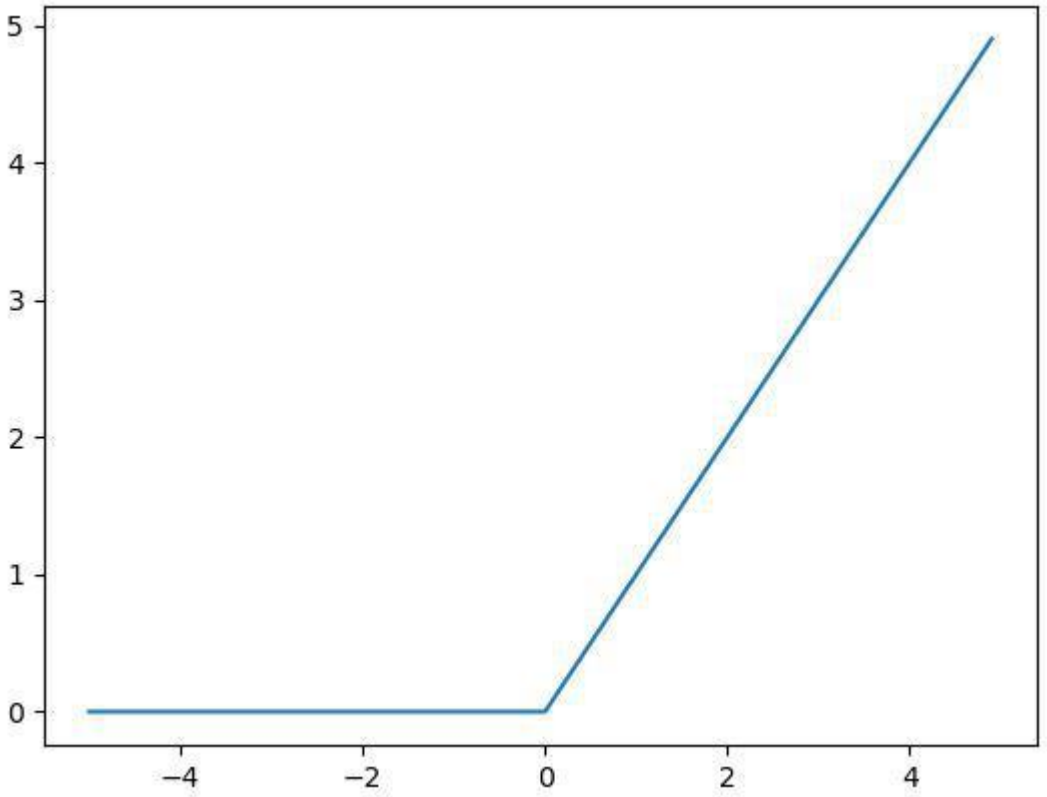


02 활성화 함수

Relu 함수

함수 식 : $$h(x) = \begin{cases} x\ (x > 0) \\ 0\ (x \leq 0) \end{cases}$$

```
def ReLU(x):
    return np.maximum(0,x)

x = np.arange(-5,5,0.1)
y = ReLU(x)
plt.plot(x,y)
plt.show()
```



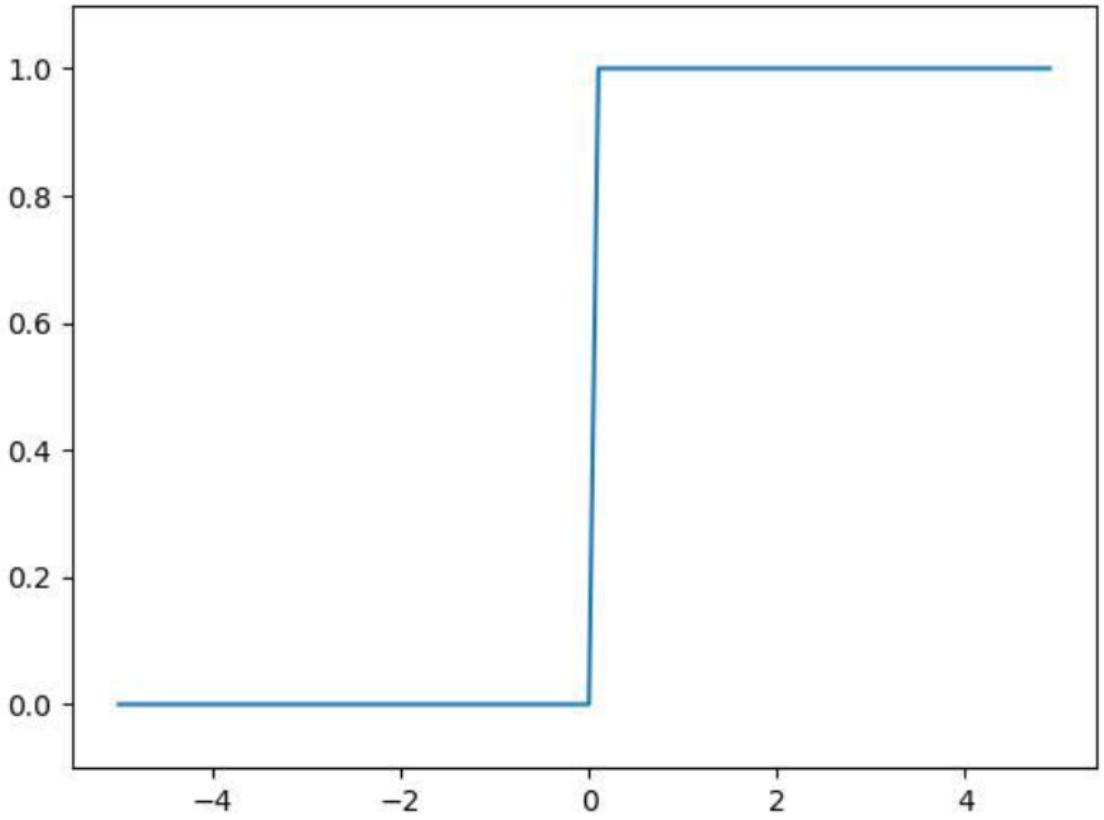
02 활성화 함수

퍼셉트론 구현에 사용된 계단 함수

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

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

```
import matplotlib.pyplot as plt
x = np.arange(-5,5,0.1)
y = step_function(x)
plt.plot(x,y)
plt.ylim(-0.1,1.1)
plt.show()
```

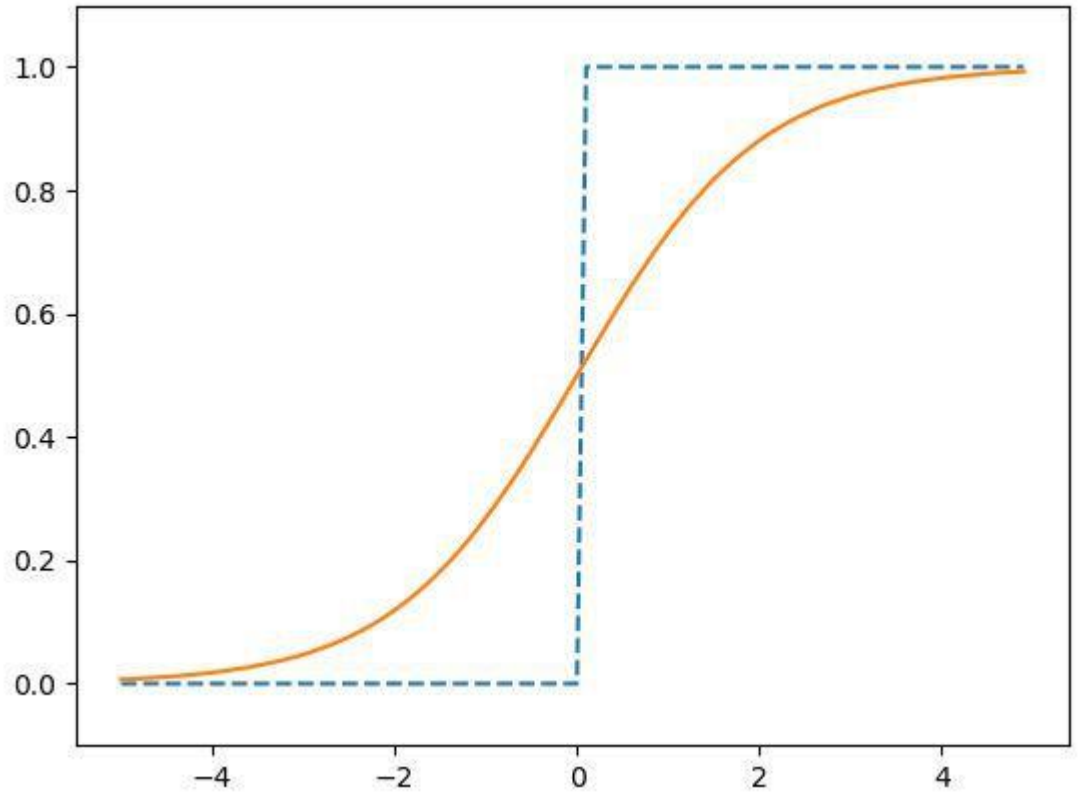


02 활성화 함수

시그모이드와 계단형 함수 비교하기

공통점 : 비선형 함수, 출력이 0과 1사이 범위

차이점 : 시그모이드를 사용할 경우 , 신경망에 실수 값이 흐르게 되어, 입력의 작은 값도 처리



03 신경망에서 행렬 곱

행렬 곱

$$AB=\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f \\ g & h \end{pmatrix}=\begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

넘파이에서 행렬 곱 구현

```
In [22]: A = np.array([[1, 2, 3], [4, 5, 6]])
         print(A.shape)
         B = np.array([[1, 2], [3, 4], [5, 6]])
         print(B.shape)
         np.dot(A, B)

         (2, 3)
         (3, 2)

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

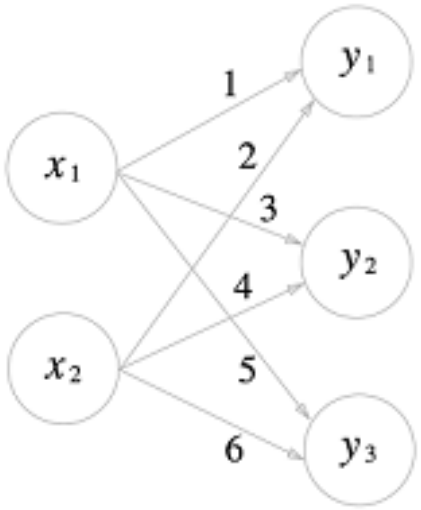
두 행렬 열과 행의 원소 수가 다를 경우

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

```
(2, 2)
(2, 3)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-be6ab816c1cc> in <module>()
      2 print(C.shape)
      3 print(A.shape)
----> 4 np.dot(A, C)

ValueError: shapes (2,3) and (2,2) not aligned: 3 (dim 1) != 2 (dim 0)
```

03 신경망에서 행렬 곱

가중치 행렬의 사용



$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

$$\underset{2}{X} \quad \underset{2\times 3}{W} \quad = \quad \underset{3}{Y}$$

일치

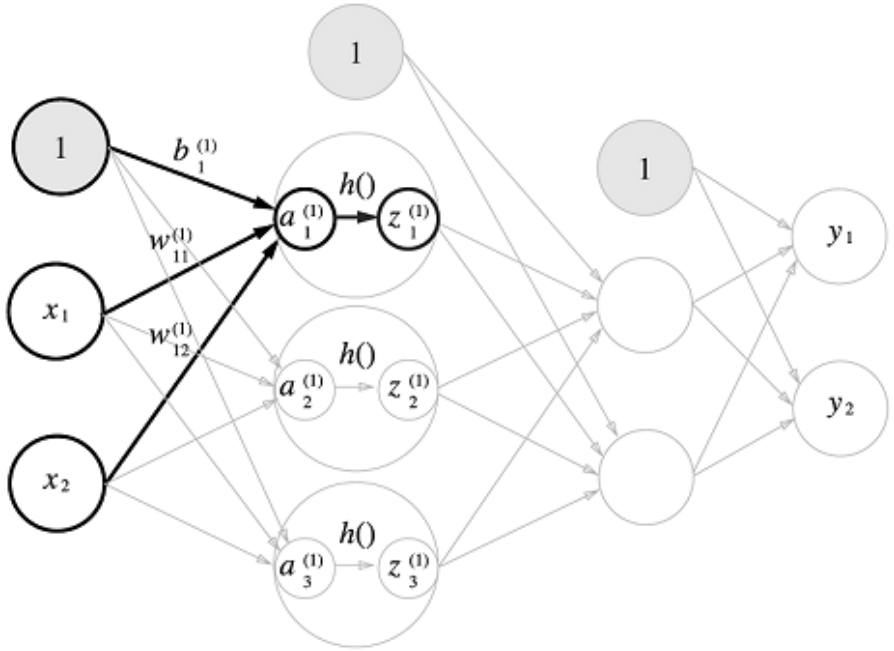
가중치 행렬 dot 연산

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

```
(2,)
[[1 3 5]
 [2 4 6]]
(2, 3)
[ 5 11 17]
```

03 신경망에서 행렬 곱

3층 신경망



$w_{12}^{(1)}$
- 1층의 가중치
- 앞 층의 2번째 뉴런
- 다음 층의 1번째 뉴런

$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

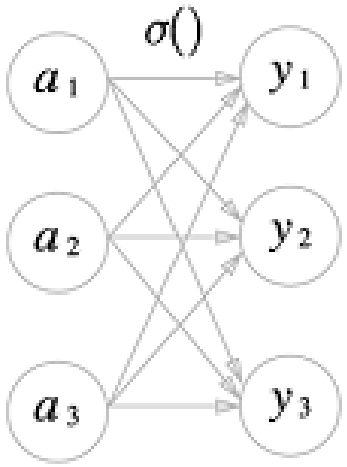
구현 코드

```python
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

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 = np.dot(z2, W3) + b3
    y = identity_function(a3)

    return y

network = init_network()
x = np.array([1.0, 0.5])
y = forward(network, x)
print(y)  [ 0.31682708  0.69627909]
```

03 신경망에서 행렬 곱

항등함수와 소프트맥스 함수

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



소프트맥스 함수 코드

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

    return y
```

03 신경망에서 행렬 곱

소프트맥스 함수 구현 주의사항

오버플로 발생

```
a = np.array([1010, 1000, 990])
np.exp(a) / np.sum(np.exp(a)) # 소프트맥스 함수의 계산
# array([ nan,  nan,  nan])   # 제대로 계산되지 않는다.
```

```
/Users/donglyeolsin/anaconda/lib/python3.5/site-packages/
```

입력신호 최대 값 빼주는 방법

```
c = np.max(a)
a - c
```

```
array([  0, -10, -20])
```

```
np.exp(a - c) / np.sum(np.exp(a - c))
```

```
array([  9.99954600e-01,   4.53978686e-05,   2.06106005e-09])
```

04 숫자 인식, 배치 처리

손글씨 신경망 구성

```
import pickle

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

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
```

04 숫자 인식, 배치 처리

정확도 측정 함수 구현

```
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 얻음
    if p == t[i]:
        accuracy_cnt += 1

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

04 숫자 인식, 배치 처리

배치 처리

```
x, _ = get_data()
network = init_network()
W1, W2, W3 = network['W1'], network['W2'], network['W3']

x.shape
```

```
(10000, 784)
```

```
x[0].shape
```

```
(784,)
```

```
W1.shape
```

```
(784, 50)
```

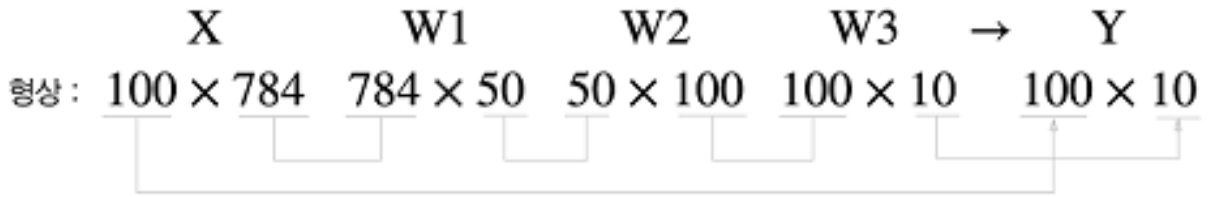
```
W2.shape
```

```
(50, 100)
```

```
W3.shape
```

```
(100, 10)
```

배치 처리
이미지 여러 장을 한꺼번에 입력하는 경우
배치(batch): 하나로 묶은 입력 데이터
(이미지 100개를 묶어 predict() 함수에 한 번에 넘긴 경우)




```
x, t = get_data()
network = init_network()

batch_size = 100 # 배치 크기
accuracy_cnt = 0
for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

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

```
Accuracy:0.9352
```