Chapter

2

Tensorflow를 이용한 다층신경망 구현

1. Neural Network
2. Activation Function
3. Array
4. 신경망 구현하기
5. 출력층 구현하기
6. MNIST 맛보기

학습 목표

✓ 퍼셉트론에서 신경망까지 발전 과정을 이해한다.
✓ 딥러닝의 주요 활성화 함수를 이해한다.
✓ 신경망을 직접 구현한다.

주요 내용

✓ 딥러닝의 기본인 신경망에 대한 기초적인 이해
✓ 딥러닝에 사용되는 활성화 함수 이해
✓ 신경망을 이해하는데 기본적인 계산 이해

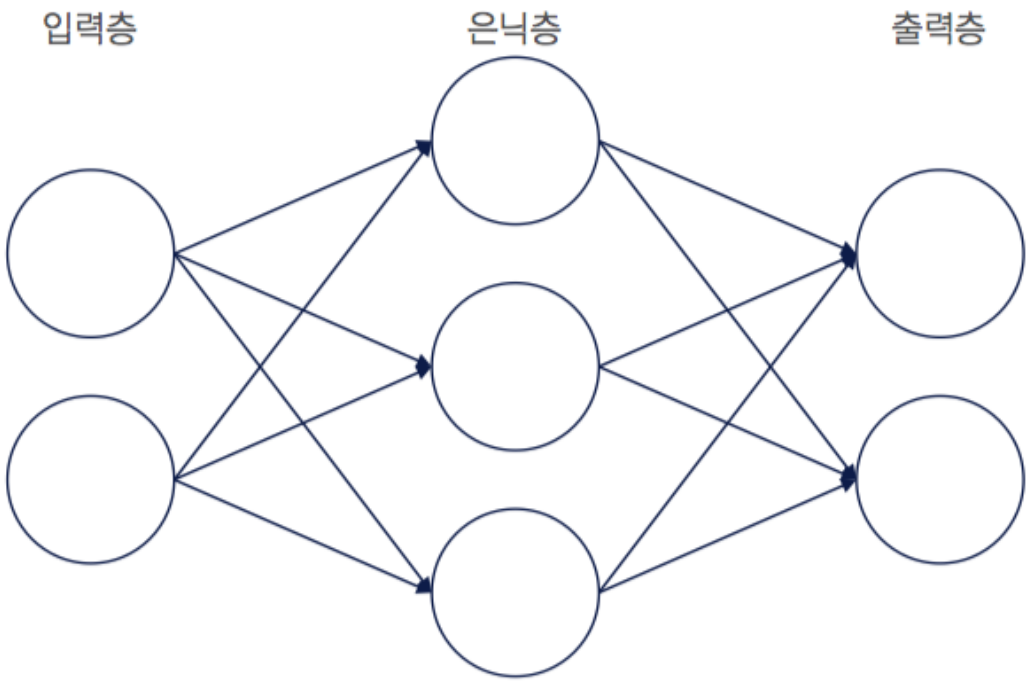
2

1. Neural Network

Neural Network

❖ 퍼셉트론에서 신경망으로

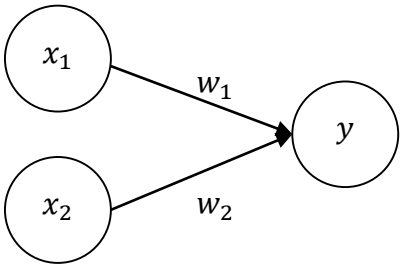
- 신경망 ?
  - 가중치 매개변수의 적절한 값을 데이터로부터 자동으로 학습
  - 신경망은 신호를 어떻게 전달할까?



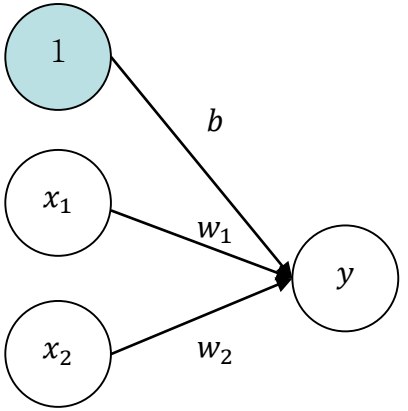
Neural Network

❖ 퍼셉트론에서 신경망으로

- Perceptron을 잠시 복습해봅시다.
- $y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$



- 가중치가 b이고 Input이 1인 뉴런 추가
  - x_1, x_2, 1 세 개의 input이 뉴런에 입력되고, 입력된 뉴런에 가중치를 곱한 후 그 합이 0을 넘으면 1을 출력하고 그렇지 않으면 0을 출력하는 퍼셉트론
  - $y = h(b + w_1x_1 + w_2x_2)$
  - $h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$

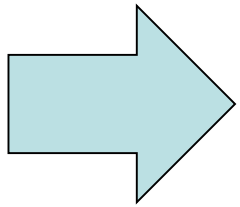


Neural Network

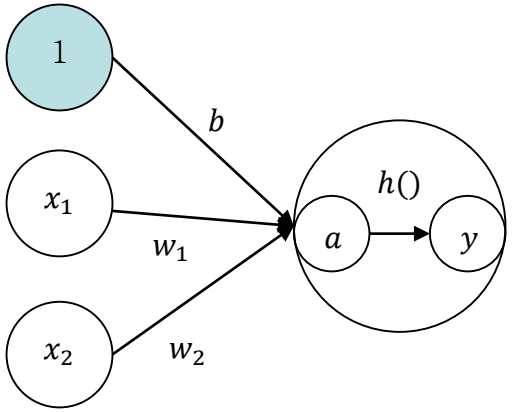
❖ 활성화 함수?

- Activation Function(활성화 함수)
  - 입력 신호의 총합을 출력 신호로 변환하는 함수($h(x)$)

- $y = h(b + w_1x_1 + w_2x_2)$
- $h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$



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



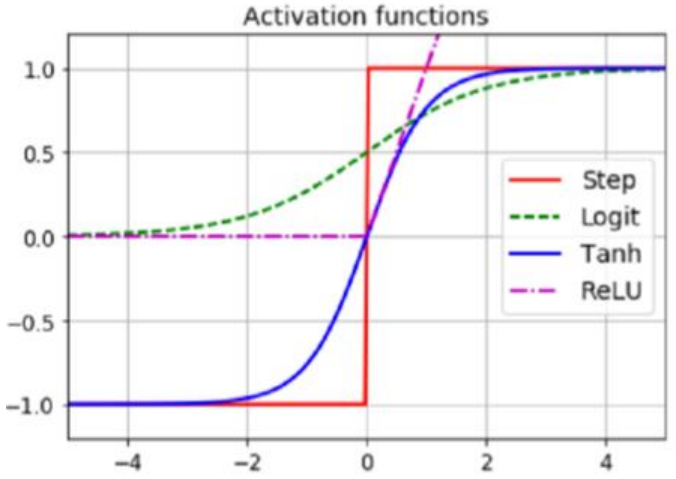
2

2. Activation Function

Activation Function

❖ 활성화 함수?

- 퍼셉트론에서는 어떤 활성화 함수를 썼는가 ?
  - 계단 함수(Step Function) 사용
  - $h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$



- 그렇다면 신경망에서는 어떤 활성화 함수들을 사용하는가 ?
  - Sigmoid Function 함수를 많이 사용
  - $h(x) = \frac{1}{1+exp^{(-x)}}$
  - $exp^{(-x)}$ 는 $e^{(-x)}$ 이며 e는 자영 상수로 2.7182….
    - 로그와 유사
  - Sigmoid Function, Tanh Function, ReLU etc..

- 신경망이란?
  - 활성화 함수로 Sigmoid 함수를 이용하여 입력 값을 변환하고, 이 값을 다음 뉴런에 전달 !

Activation Function

❖ Step Function Code

- 계단 함수(Step Function)는 입력이 0을 넘으면 1을 출력하고 그 외에는 0을 출력하는 함수
- 배열도 지원하는 코드를 작성하고 싶은 경우?

```
In [2]: import numpy as np
        import matplotlib.pylab as plt
```

```
In [3]: x = np.array([-1.0, 1.0, 2.0])
        y = x > 0
        print("x = ", x)
        print("y = ", y)
```

```
x =  [-1.  1.  2.]
y =  [False  True  True]
```

```
In [4]: y = y.astype(np.int)
        print("y = ", y)
```

```
y =  [0 1 1]
```

- 넘파이 배열의 부등호 연산 수행 -> bool 배열 생성
- Bool자료형 변환 -> astype() 메소드 사용

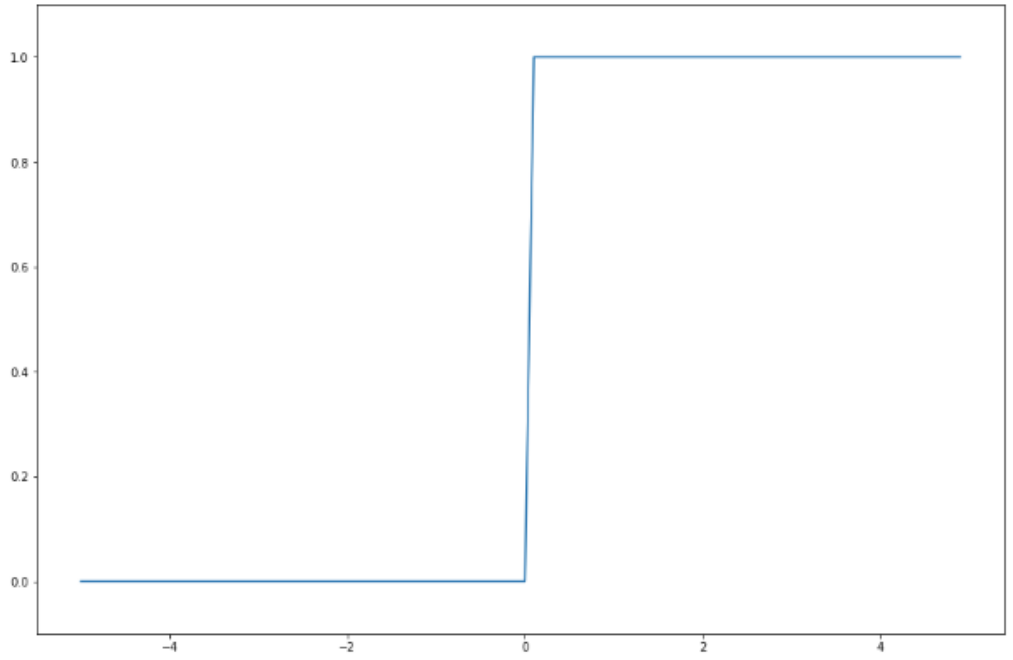
Activation Function

❖ Step Function 그래프

- matplotlib 사용
- np.arange(-5.0, 5.0, 0.1) 은 넘파이 배열 생성
- step_function()은 인수로 받은 넘파이 배열의 원소 각각을 인수로 계단 함수를 실행하여 결과 배열로 돌려줌

```
In [5]: def step_function(x):
            return np.array(x > 0, dtype=np.int)
```

```
In [6]: X = np.arange(-5.0, 5.0, 0.1)
        Y = step_function(X)
        plt.figure(figsize = (15, 10))
        plt.plot(X, Y)
        plt.ylim(-0.1, 1.1)  # y축 범위 지정
        plt.show()
```



Activation Function

❖ Sigmoid Function

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

```
In [8]: def sigmoid(x):
            return 1 / (1 + np.exp(-x))
```

Activation Function

❖ (참고)Sigmoid Function

```
In [9]: x = np.array([-1, 1.0, 2.0])
        y = sigmoid(x)
        print("y =", y)

        y = [0.26894142 0.73105858 0.88079708]
```

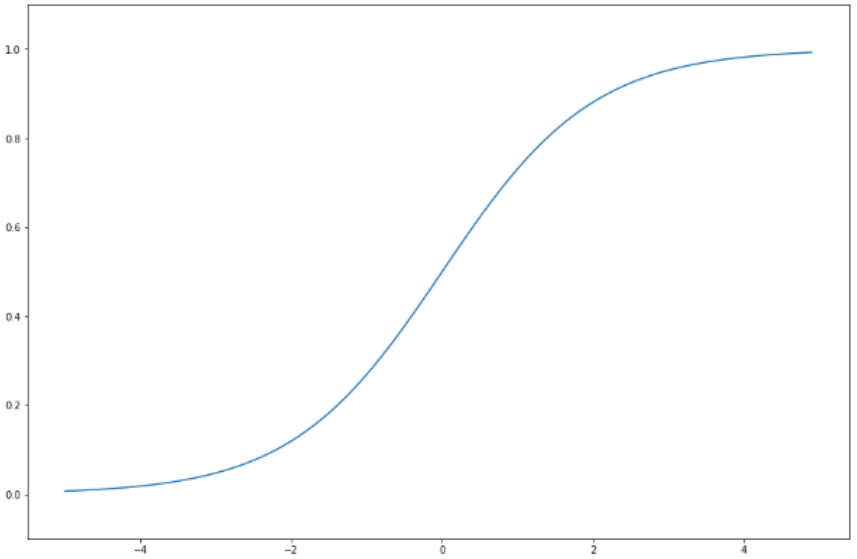
```
In [10]: t = np.array([1.0, 2.0, 3.0])
         print("t + 1 =", t + 1)
         print("1 / t =", 1/t)

         t + 1 = [2. 3. 4.]
         1 / t = [1.         0.5        0.33333333]
```

Activation Function

❖ Sigmoid Function 그래프

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

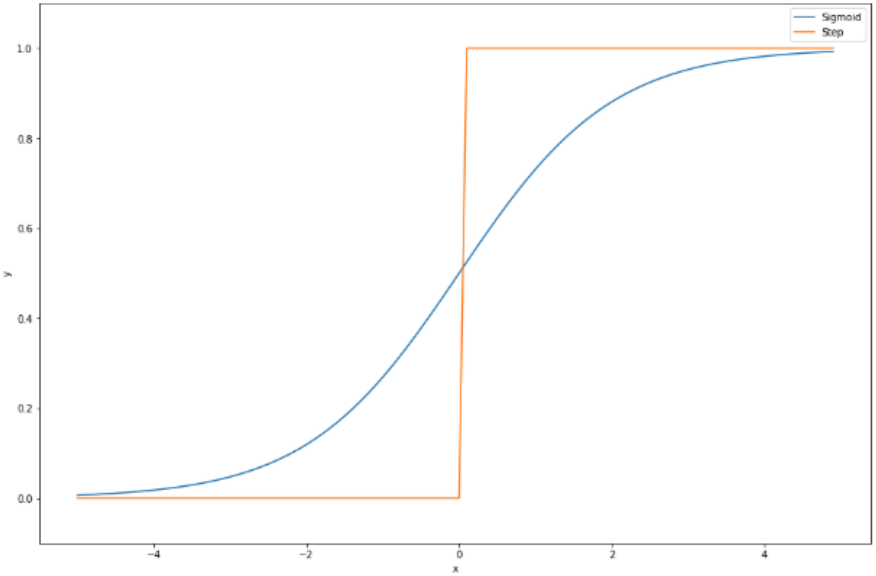
```
In [1]: import numpy as np
        import matplotlib.pylab as plt

In [2]: def sigmoid(x):
            return 1 / (1 + np.exp(-x))

In [3]: X = np.arange(-5.0, 5.0, 0.1)
        Y = sigmoid(X)
        plt.figure(figsize = (15, 10))
        plt.plot(X, Y)
        plt.ylim(-0.1, 1.1)
        plt.show()
```



Activation Function

❖ Sigmoid Function vs. Step Function

- Singmoid 와 step function의 차이?
  - 매끄러움
  - 계단 함수: 0을 기점으로 출력이 급격하게 변화
  - 시그모이드 함수: 입력에 따라 연속적 변화



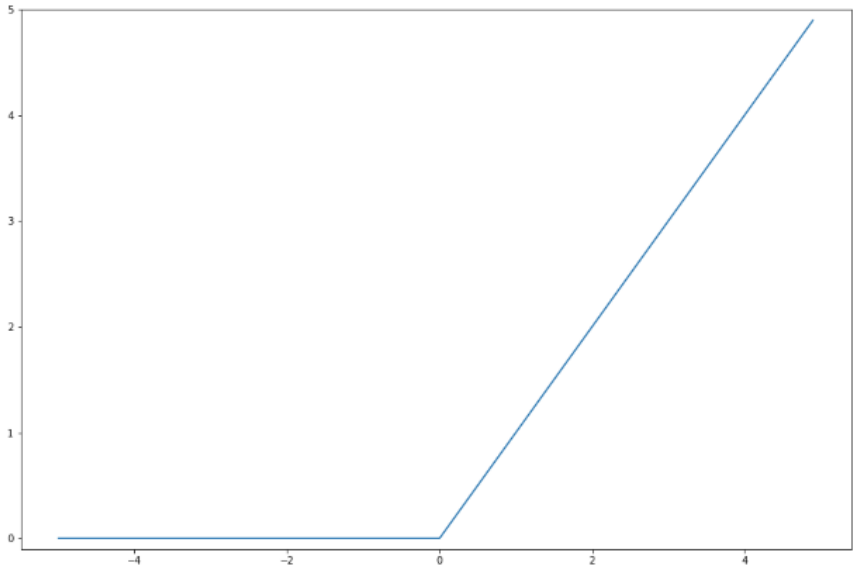
- Singmoid 와 step function의 공통점?
  - 입력이 작은 경우의 출력은 0에 가깝고 입력이 커질수록 1에 가까워지는 구조
  - 즉, 입력이 중요하면 큰 값 출력, 중요하지 않으면 작은 값 출력
  - 출력은 0에서 1 사이
  - **비선형 함수**
    - 왜 신경망에서는 비선형 함수를 사용해야 할까?

Activation Function

❖ ReLU(Rectified Linear Unit)

- ReLU는 입력이 0을 넘으면 그 입력을 그대로 출력하고, 0 이하이면 0을 출력하는 함수
- $h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$

```
In [13]: import numpy as np
         import matplotlib.pylab as plt

In [14]: def relu(x):
             return np.maximum(0, x)

In [15]: X = np.arange(-5.0, 5.0, 0.1)
         Y = relu(X)
         plt.figure(figsize = (15, 10))
         plt.plot(X, Y)
         plt.ylim(-0.1, 5)
         plt.show()
```



2

3. Array

Array

❖ 다차원 배열

- 다차원 배열?
  - N 차원으로 수를 나열하는 것

- 먼저 1차원 배열을 만들어 봅시다.
  - 배열의 차원 수 확인: np.ndim()
  - 배열의 형상 확인: shape

```
In [16]: A = np.array([1, 2, 3, 4])
         print("A =", A)
         print("A.ndim =",np.ndim(A))
         print("A.shape =", A.shape)
```

```
A = [1 2 3 4]
A.ndim = 1
A.shape = (4,)
```

Array

❖ 다차원 배열

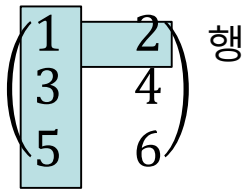
- 다차원 배열?
  - N 차원으로 수를 나열하는 것

- 이번엔 2차원 배열을 만들어 봅시다.
  - 3*2 배열을 만들어 보세요
  - 배열의 차원 수 확인: np.ndim()
  - 배열의 형상 확인: shape

```
In [18]: B = np.array([[1,2], [3,4], [5,6]])
         print("B =", B)
         print("B.ndim = ", np.ndim(B))
         print("B.shape =", B.shape)
```

```
B = [[1 2]
 [3 4]
 [5 6]]
B.ndim =  2
B.shape = (3, 2)
```

- 3 X 2 배열 ?
  - 처음 차원에는 원소가 3개, 다음 차원에는 원소가 2개 있다는 의미
  - 2차원 배열은 행렬(matrix)라고 부르며 배열의 가로 방향은 행(row), 세로 방향은(column)



Array

❖ 행렬의 내적(행렬 곱)

- 2 x 2 행렬의 내적 구하기

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

- 파이썬으로 구현해보기
  - 넘파이 내적 계산: np.dot()
  - np.dot(A, B), np.dot(B, A) 값은 어떤가?

```
In [19]: A = np.array([[1, 2], [3, 4]])
         B = np.array([[5, 6], [7, 8]])
         print("A.shape =", A.shape)
         print("B.shape =", B.shape)

         A.shape = (2, 2)
         B.shape = (2, 2)

In [20]: result = np.dot(A, B)
         print("A·B =",result)
         print("result.shape =", result.shape)

         A·B = [[19 22]
          [43 50]]
         result.shape = (2, 2)
```

Array

❖ 행렬의 내적(행렬 곱)

- 2 x 3 행렬 A와 3 x 2 행렬B 의 내적을 구해보세요

```
In [21]: A = np.array([[1, 2, 3], [4, 5, 6]])
         B = np.array([[1, 2], [3, 4], [5, 6]])
         print("A.shape =", A.shape)
         print("B.shape =", B.shape)

         A.shape = (2, 3)
         B.shape = (3, 2)
```

```
In [22]: result = np.dot(A, B)
         print("A·B =",result)
         print("result.shape =", result.shape)

         A·B = [[22 28]
          [49 64]]
         result.shape = (2, 2)
```

- 2 x 3 행렬 A와 2 x 2 행렬B 의 내적을 구해보세요
  - 값이 어떻게 나오나요?

Array

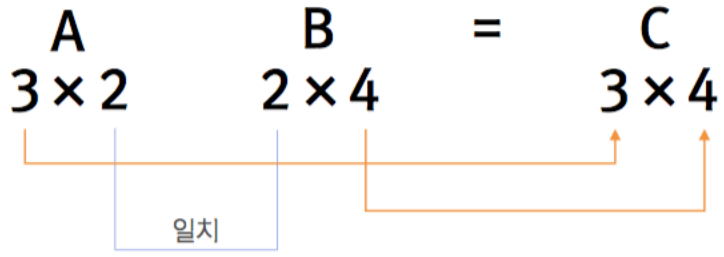
❖ 행렬의 내적(행렬 곱)

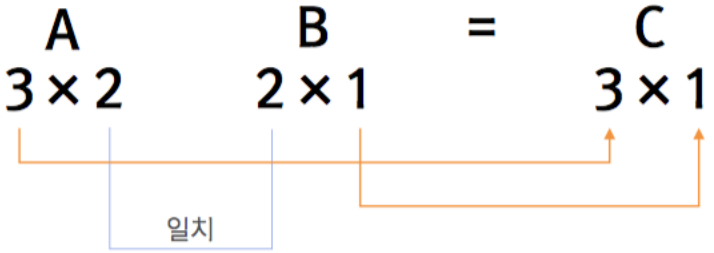
- 2 x 3 행렬 A와 2 x 2 행렬B 의 내적을 구해보세요

```
In [23]: C = np.array([[1, 2], [3, 4]])
         print("C.shape =", C.shape)
         print("A·C =", np.dot(A, C))

         C.shape = (2, 2)

         ---------------------------------------------------------------------------
         ---
         ValueError                                Traceback (most recent call la
         st)
         <ipython-input-23-d0e86aa025e2> in <module>()
               1 C = np.array([[1, 2], [3, 4]])
               2 print("C.shape =", C.shape)
         ----> 3 print("A·C =", np.dot(A, C))

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

????

- 다차원 배열의 내적(곱셈)에는 두 행렬의 대응하는 차원의 원소 수를 일치하여야 한다!
- 행렬의 형상(Shape)에 주의*

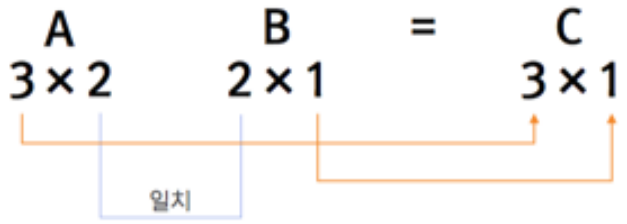




Array

❖ 행렬의 내적(행렬 곱)

- 3 x 2 행렬 A와 2 x 1 행렬B 의 내적을 구해보세요



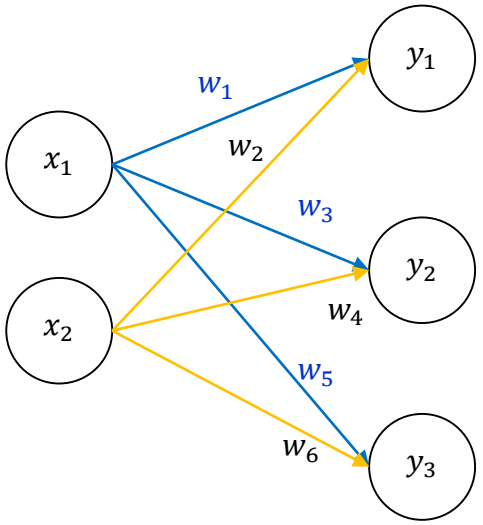

```
In [24]: A = np.array([[1, 2], [3, 4], [5, 6]])
         print("A.shape =", A.shape)
         B = np.array([7, 8])
         print("B.shape =", B.shape)
         print("A·B = ", np.dot(A, B))
```

```
A.shape = (3, 2)
B.shape = (2,)
A·B =  [23 53 83]
```

Array

❖ 신경망의 내적

- 넘파이를 가지고 어떻게 신경망을 구현할 수 있을까



$$\begin{pmatrix} w_1 & w_3 & w_5 \\ w_2 & w_4 & w_6 \end{pmatrix}$$

X W = Y

1 × 2 2 × 3 1 × 3

일치

Array

❖ 신경망의 내적 Code

- 넘파이를 가지고 어떻게 신경망을 구현할 수 있을까

```
In [25]: x = np.array([1, 2])
         print("shape of input =", x.shape)
         weight = np.array([[1, 3, 5], [2, 4, 6]])
         print("weight =", weight)
         print("shape of weight =", weight.shape)
         y = np.dot(x, weight)
         print("y =", y)
```

```
shape of input = (2,)
weight = [[1 3 5]
 [2 4 6]]
shape of weight = (2, 3)
y = [ 5 11 17]
```

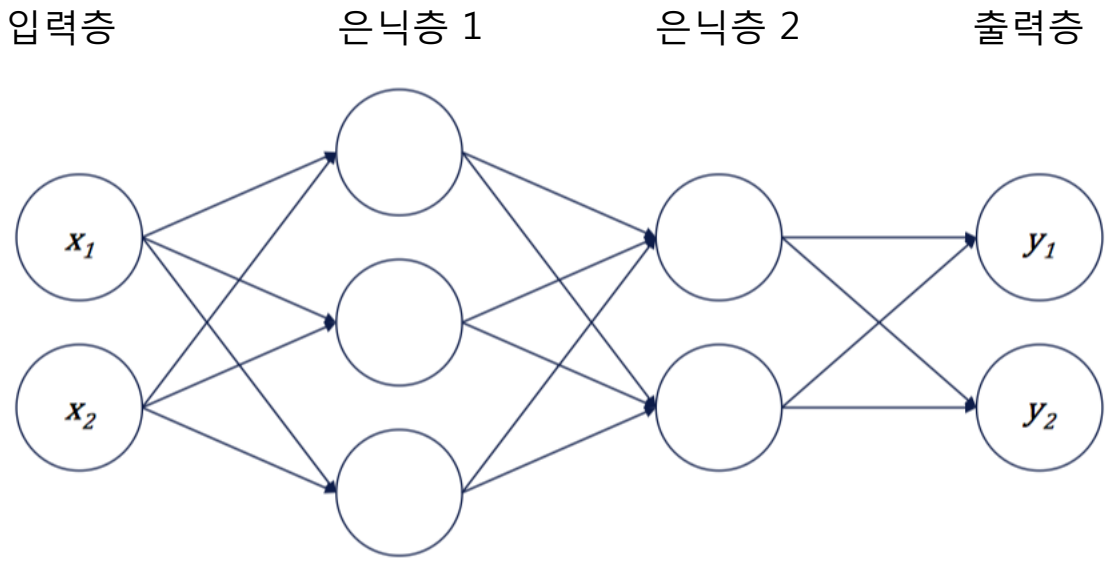
2

4. 신경망 구현하기

신경망 구현하기

❖ 은닉층이 2개인 신경망 구현하기

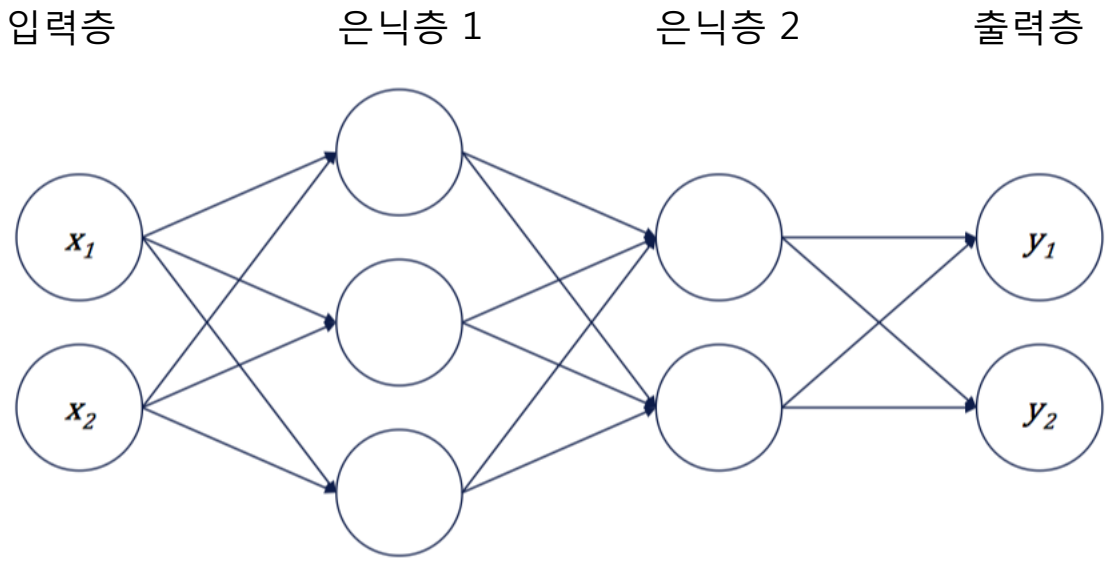
- 넘파이를 이용하여 입력부터 출력까지 순방향 신경망 구현
- 구현 예정 신경망



신경망 구현하기

❖ 은닉층이 2개인 신경망 구현하기

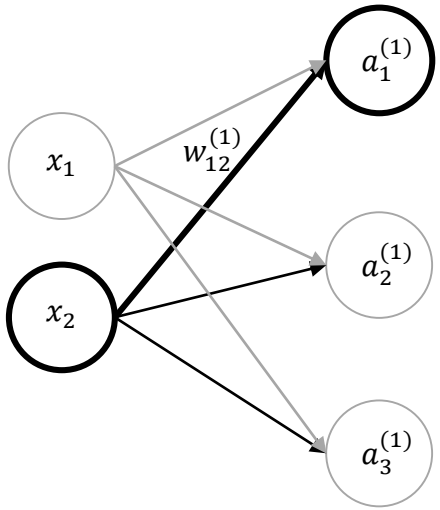
- 넘파이를 이용하여 입력부터 출력까지 순방향 신경망 구현
- 구현 예정 신경망



신경망 구현하기

❖ 은닉층이 2개인 신경망 구현하기

- 표기법



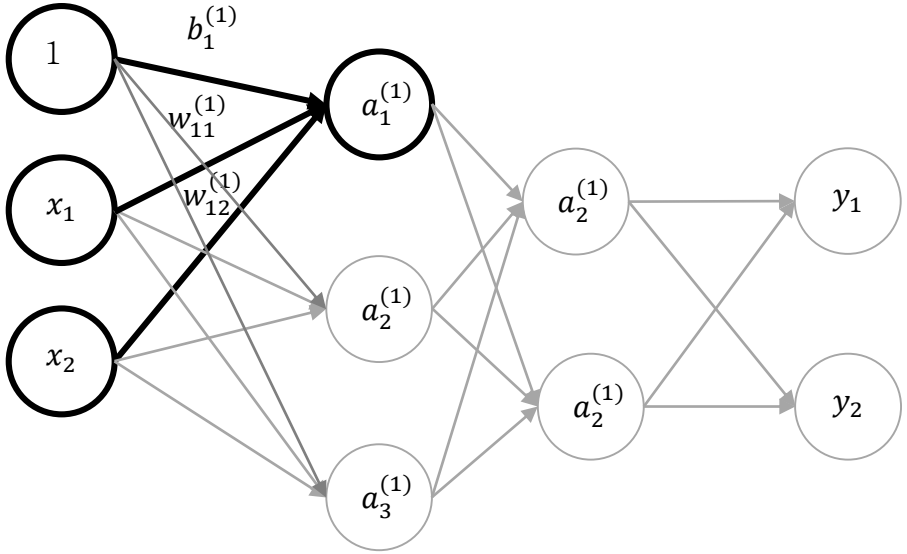
$$w_{12}^{(1)}$$

- (1) → 은닉층 1의 가중치
- 12 → 이전 층의 2번째 뉴런, 다음 층의 1번째 뉴런

신경망 구현하기

❖ 각 층별 신호 전달 구현

- 입력층에서 은닉층 1의 첫번째 뉴런으로 가는 신호
- Bias(편향) 추가



- $a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$
- 행렬의 내적을 이용하여 식을 간소화: $\mathbf{A}^{(1)} = \mathbf{W}\mathbf{X}^{(1)} + \mathbf{B}^{(1)}$
- $\mathbf{A}^{(1)} = \left(a_1^{(1)}, a_2^{(1)}, a_3^{(1)}\right)$, $\mathbf{W} = (\boldsymbol{x_1}, \boldsymbol{x_2})$, $\mathbf{B}^{(1)} = \left(b_1^{(1)}, b_2^{(1)}, b_3^{(1)}\right)$
- $\mathbf{W}^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$

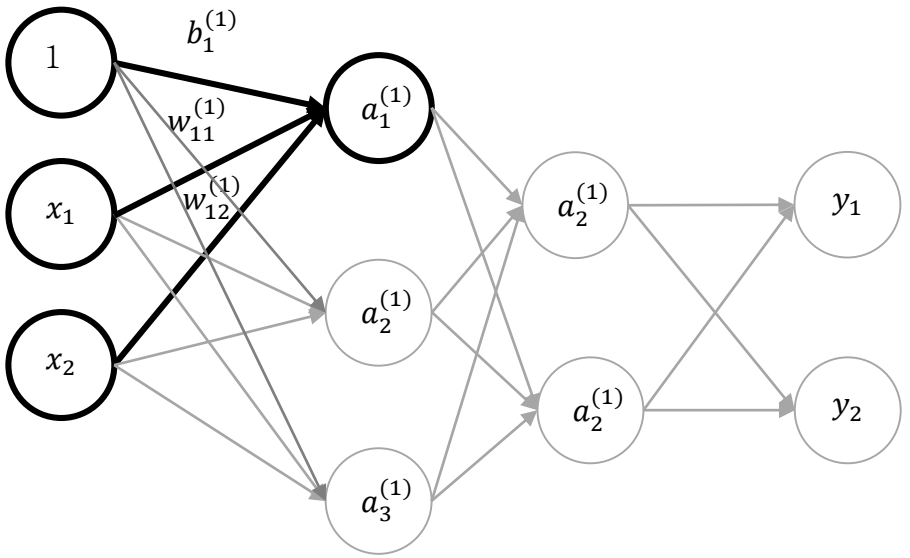
신경망 구현하기

❖ 각 층별 신호 전달 구현 Code

- 입력층에서 은닉층 1층의 첫번째 뉴런으로 가는 신호
  - W1은 2 x 3 행렬
  - X는 원소가 2개인 1차원 배열
  - W1과 X의 차원의 원소 수 일치

```
In [12]: 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('W1.shape :', W1.shape)
         print('X.shape :', X.shape)
         print('B1.shape :', B1.shape)

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

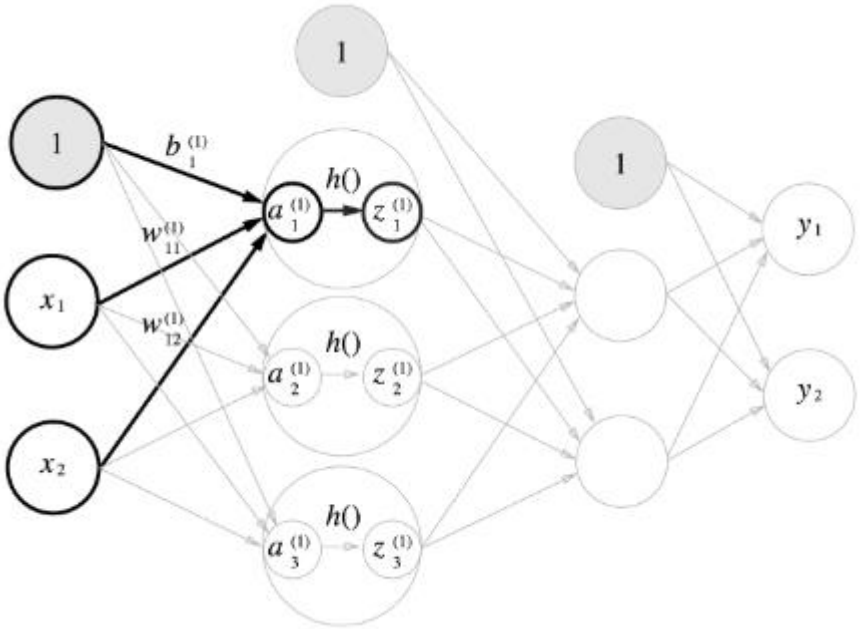
```
W1.shape : (2, 3)
X.shape : (2,)
B1.shape : (3,)
A1.shape : (3,)
A1 : [0.3 0.7 1.1]
```



신경망 구현하기

❖ 각 층별 신호 전달 구현: 활성화 함수 처리

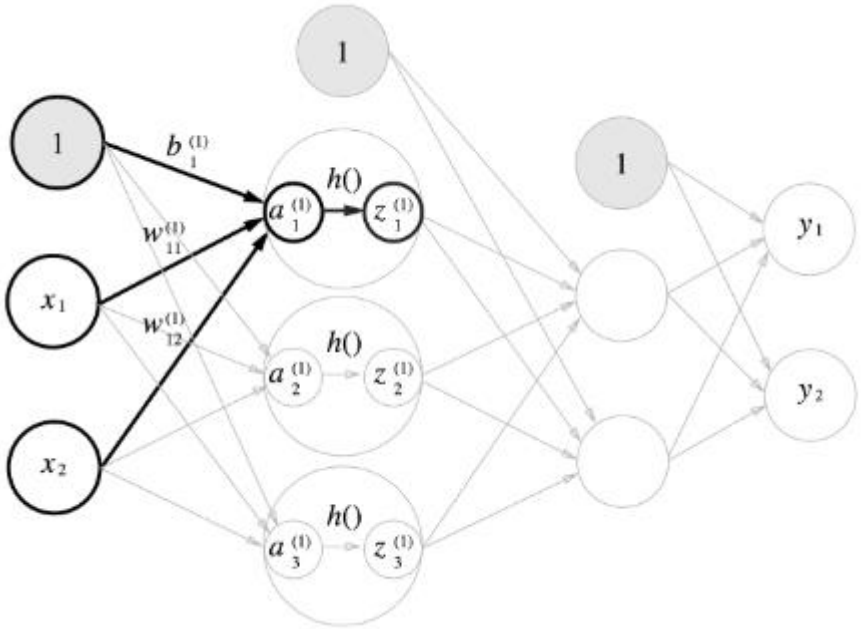
- a 는 은닉층에서의 가중치 합(가중신호와 편향의 총합)으로 표기
- z 는 활성화 함수 $h()$로 변환된 신호
- 활성화 함수로 시그모이드를 사용



신경망 구현하기

❖ 각 층별 신호 전달 구현: 활성화 함수 처리 Code

- a 는 은닉층에서의 가중치 합(가중신호와 편향의 총합)으로 표기
- z 는 활성화 함수 $h()$로 변환된 신호
- 활성화 함수로 시그모이드를 사용



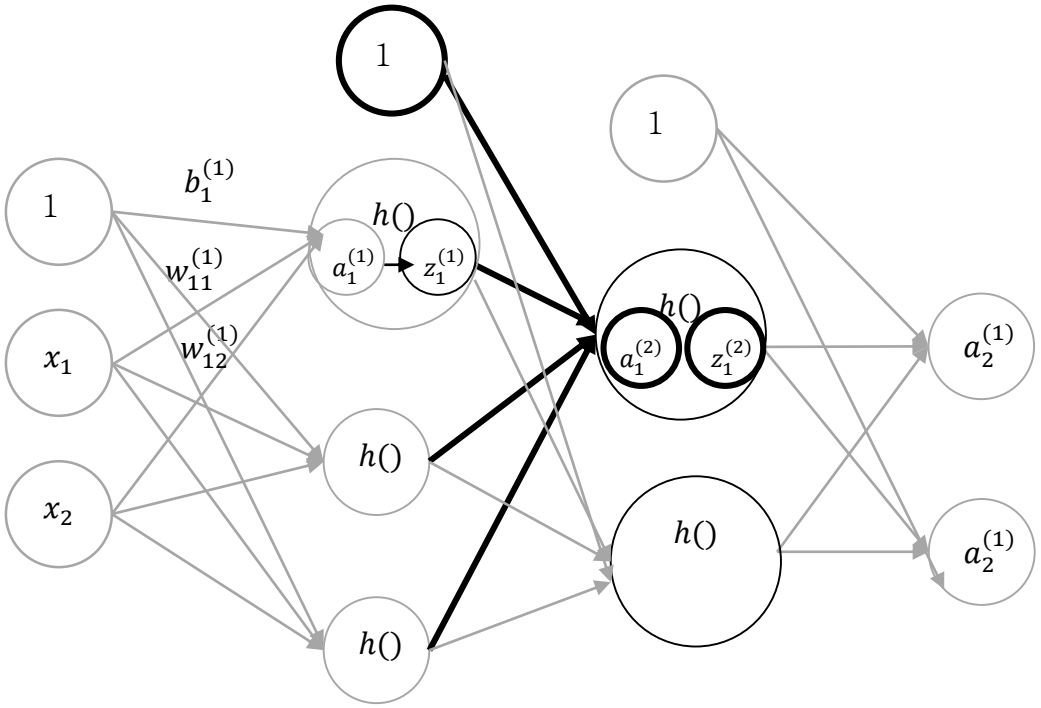

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

Z1 = sigmoid(A1)

print('A1 :', A1)
print('Z1 :', Z1)
```

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

신경망 구현하기

❖ 각 층별 신호 전달 구현: : 은닉층 1에서 은닉층 2로...

- a 는 은닉층에서의 가중치 합(가중신호와 편향의 총합)으로 표기
- z 는 활성화 함수 $h()$로 변환된 신호
- 활성화 함수로 시그모이드를 사용

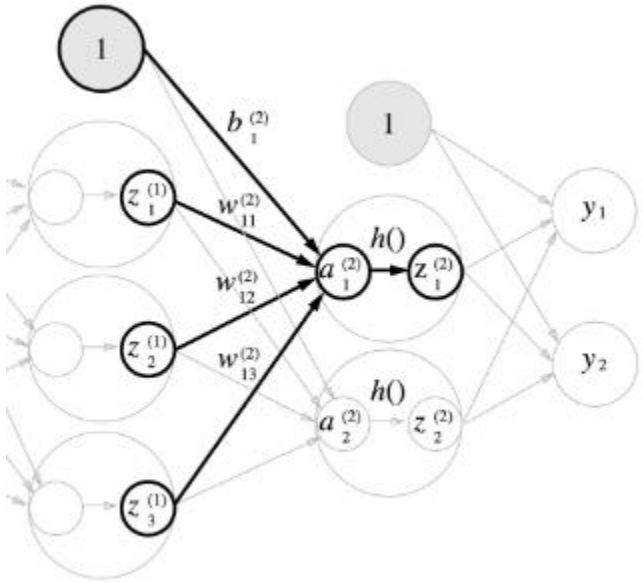


신경망 구현하기

❖ 각 층별 신호 전달 구현: 은닉층 1에서 은닉층 2로... Code

```
In [16]: W2 = np.array([[0.1, 0.4],
                        [0.2, 0.5],
                        [0.3, 0.6]])
         B2 = np.array([0.1, 0.2])

         print('Z1.shape :', Z1.shape)
         print('W2.shape :', W2.shape)
         print('B2.shape :', B2.shape)

         A2 = np.dot(Z1, W2) + B2  # 신호 합
         Z2 = sigmoid(A2)  # 활성화함수 값

         print('A2 :', A2)
         print('Z2 :', Z2)
```

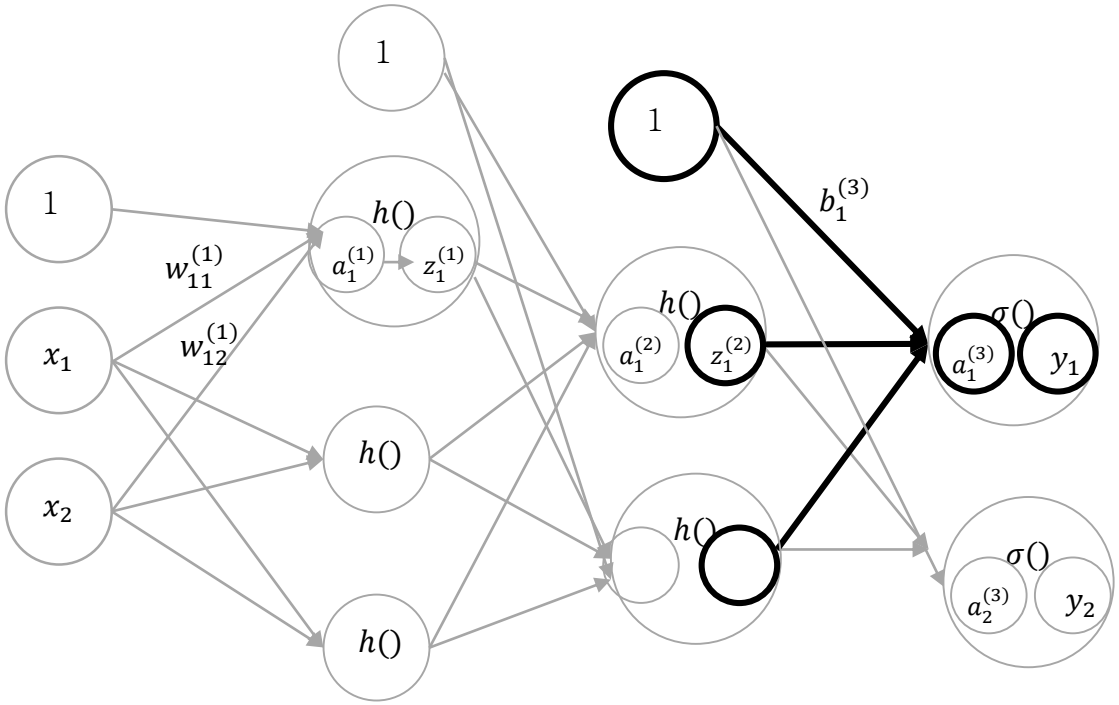
```
Z1.shape : (3,)
W2.shape : (3, 2)
B2.shape : (2,)
A2 : [0.51615984 1.21402696]
Z2 : [0.62624937 0.7710107 ]
```



신경망 구현하기

❖ 각 층별 신호 전달 구현: 출력층 구현

- 출력층



신경망 구현하기

❖ 각 층별 신호 전달 구현: 출력층 구현 Code

- 출력층

```
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)  # Y = A3

print('W3.shape :', W3.shape)
print('B3.shape :', B3.shape)
print('A3.shape :', A3.shape)
print('Y.shape :', Y.shape)

print('A3 :', A3)
print('Y :', Y)
```

```
W3.shape : (2, 2)
B3.shape : (2,)
A3.shape : (2,)
Y.shape : (2,)
A3 : [0.31682708 0.69627909]
Y : [0.31682708 0.69627909]
```

신경망 구현하기

❖ 구현 정리

- 은닉층이 2개인 신경망
- Init_network(): 가중치와 편향을 초기화 후 network에 저장
- forward(): 입력을 출력으로 변환하는 처리 과정

```
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  # dictionary return


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]
```

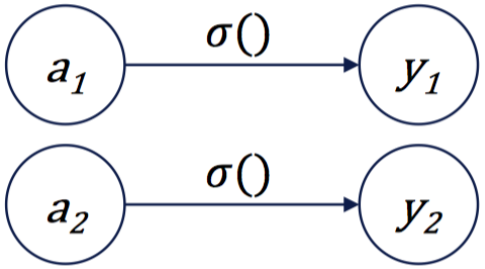
2

5. 출력층 구현하기

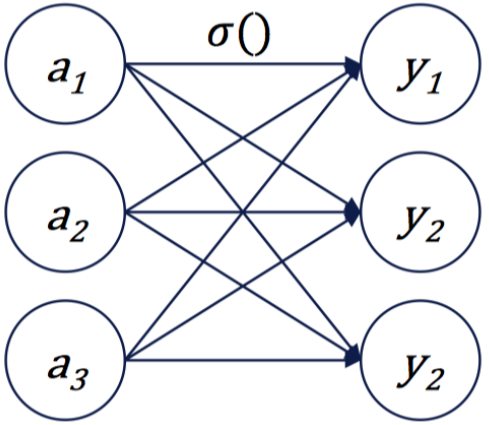
출력층 구현하기

❖ 항등 함수(Identity Function) 와 소프트맥스 함수(Softmax Function)

- 항등 함수?
  - 항등 함수는 입력을 그대로 출력하는 함수
  - 즉 입력과 출력이 항상 같은 함수



- 소프트맥스 함수
  - 분류에서 사용되는 함수
  - $\exp(x)$ 는 e^x를 뜻하는 지수 함수
  - n은 출력층의 뉴런 수이며, y_k는 그중 k번째 출력을 뜻함
  - $y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} exp(a_i)}$



출력층 구현하기

❖ 소프트맥스 함수(Softmax Function) Code

- n은 출력층의 뉴런 수이며, y_k는 그중 k번째 출력을 뜻함
- $y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} exp(a_i)}$

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


a = np.array([0.3, 2.9, 4.0])
print('a :', a)
print('softmax(a) :', softmax(a))
```

```
a : [0.3 2.9 4. ]
softmax(a) : [0.01821127 0.24519181 0.73659691]
```

출력층 구현하기

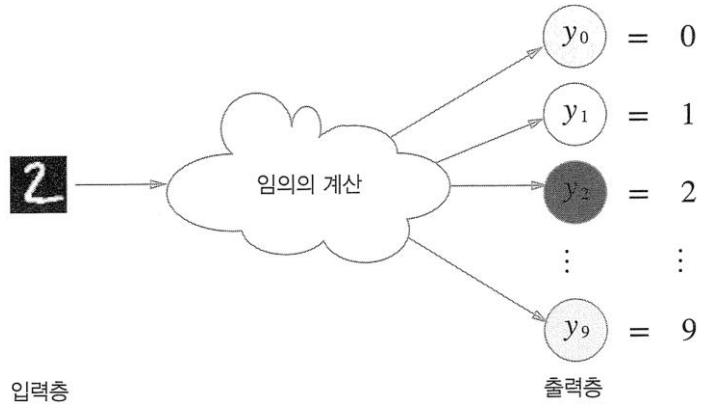
❖ 소프트맥스 함수(Softmax Function) 특징

- 소프트맥스 함수의 출력: 0에서 1.0 사이의 실수
- 소프트맥스 함수 출력의 총합은 1
  - 출력 총합이 된다는 점은 중요한 성질 중 하나
  - 확률과 유사하게 보임

```
In [7]: matrix = np.array([0.3, 2.9, 4.0])
y = softmax(matrix)
print("y =", y)
print("sum of y =", np.sum(y))
```

```
y = [0.01821127 0.24519181 0.73659691]
sum of y = 1.0
```

- 그렇다면 출력층에서 어떻게 소프트맥스 함수를 사용할까?
  - 분류의 경우 분류하고 싶은 클래스 수로 설정
  - Q) 입력 이미지가 숫자 0 ~ 9 중 하나로 분류하는 문제라면?
    - 출력층의 뉴런을 10개로 설정



2

6. MNIST 맛보기

MNIST 맛보기

❖ 손글씨 숫자 인식

- 손글씨 데이터를 분류하는 신경망 구조를 구현해 봅시다.
  - 추론 과정만 구현(학습은 다음 챕터에서 자세하게!)
  - 이러한 추론 과정을 신경망의 forward propagation 이라고 함

- MNIST 데이터 셋이란?
  - 미국 국립표준기술연구소(NIST)의 손글씨 데이터베이스
  - MNIST 손글씨 숫자 이미지 집합
  - 0 부터 9까지 숫자 이미지로 구성
  - Train data: 60,000장, Test data: 10,000장
  - 28*28 회색조 이미지
  - 각 이미지에는 이미지의 정답 레이블이 함께 있음

MNIST 맛보기

❖ 손글씨 숫자 인식

- 데이터 불러오기

```
[12]  #구글 드라이브 연결
      from google.colab import drive

      drive.mount('/gdrive')
```

```
Mounted at /gdrive
```

```
[6]   import sys, os
      import pickle
      import numpy as np
      import matplotlib.pyplot as plt
      sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
      import keras
      from keras.datasets import mnist #keras.datasets의 mnist 이용
```

```
[56]  (x_train, y_train), (x_test, y_test) = mnist.load_data()  #mnist 로드

      print('x_train.shape :', x_train.shape)
      print('t_train.shape :', y_train.shape)
      print('x_test.shape :', x_test.shape)
      print('t_test.shape :', y_test.shape)
```

MNIST 맛보기

❖ 손글씨 숫자 인식

```
#이미지 확인
plt.figure(figsize=(4, 2))
plt.subplot(121)
img = x_train[0].reshape(28, 28)
plt.imshow(img, 'gray')
plt.xticks([]), plt.yticks([]);
plt.subplot(122)
img = x_train[1].reshape(28, 28)
plt.imshow(img, 'gray')
plt.xticks([]), plt.yticks([]);
print("y_train[0] (x_train[0] label) =", y_train[0])
print("y_train[1] (x_train[1] label) =", y_train[1])
```

```
y_train[0] (x_train[0] label) = 5
y_train[1] (x_train[1] label) = 0
```

MNIST 맛보기

❖ 손글씨 숫자 인식

- 함수 정의

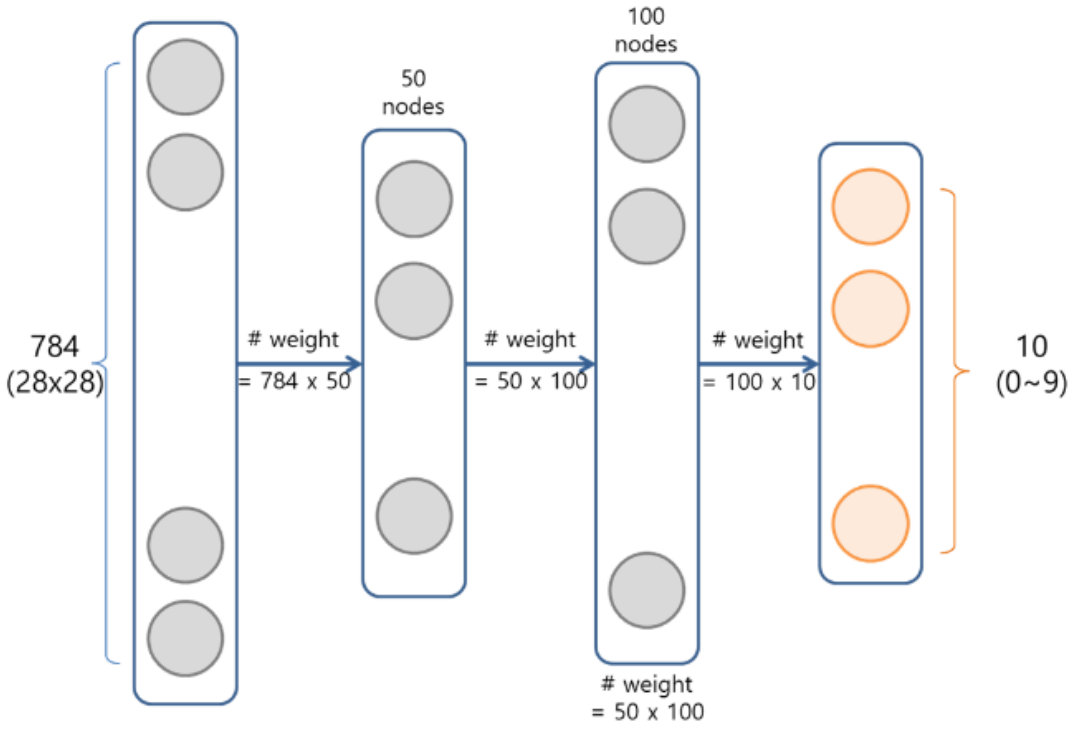
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

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

```
# mnist에서 테스트 데이터만 반환
def get_data():
    (x_train, y_train), (x_test, y_test) = mnist.load_data()
    return x_test, y_test
#sample_weight.pkl 에 저장된 학습된 가중치 가져오기
def init_network():
    with open('/gdrive/My Drive/DeepLearning/sample_weight.pkl', 'rb') as f:
        network = pickle.load(f)
    return network
#학습된 network에 x 값을 입력하여 결과 y 반환
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
```

MNIST 맛보기

❖ 손글씨 숫자 인식

- 신경망 구조



sample_weight.pkl

MNIST 맛보기

❖ 손글씨 숫자 인식

- 성능 확인

```
#(1)테스트 데이터만 가져오기
x, t = get_data()
print("x.shape=", x.shape)  #28*28 이미지 10000개
print("t.shape=", t.shape)

#(2) 28*28 2차원을 784 1차원으로 변경(flatten)
x = np.array(x).reshape(x.shape[0], x.shape[1]*x.shape[2])
print("x.shape=", x.shape)
print("t.shape=", t.shape)
#print(x[:28*28])   #첫번째 이미지 값 확인

#(3) 0~255  값을 0~1 실수값으로 변경 (normalization)
x = x / 255.0
#print(x[:28*28])   #첫번째 이미지 값 확인

#학습된 네트워크 얻기
network = init_network()
```

```
x.shape= (10000, 28, 28)
t.shape= (10000,)
x.shape= (10000, 784)
t.shape= (10000,)
```

```
accuracy_cnt = 0
for i in range(len(t)):
    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)))
```

```
Accuracy:0.9352
```