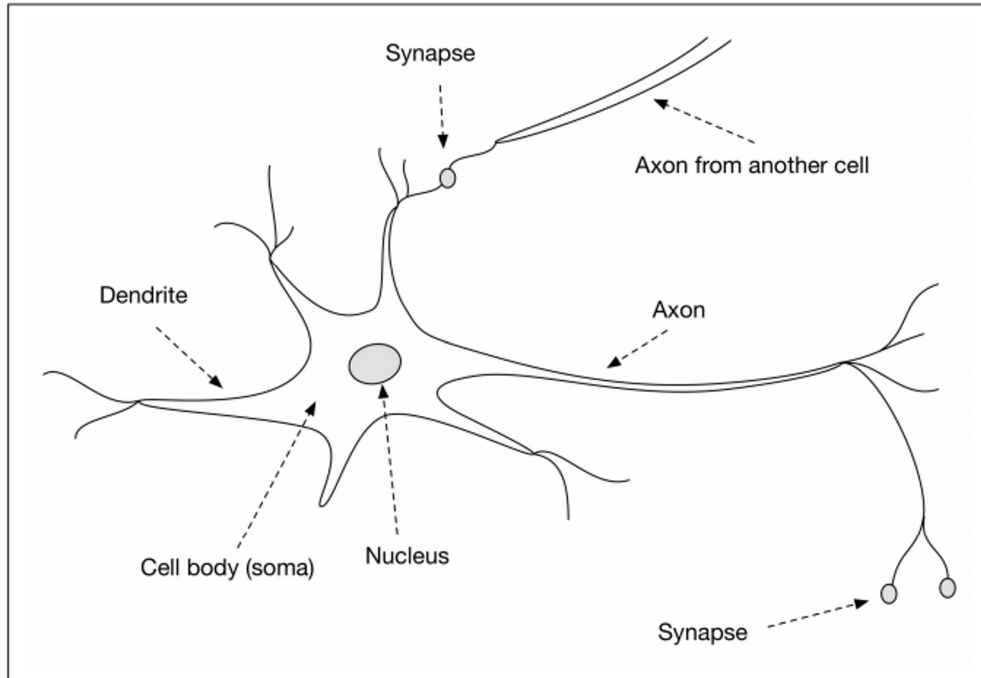


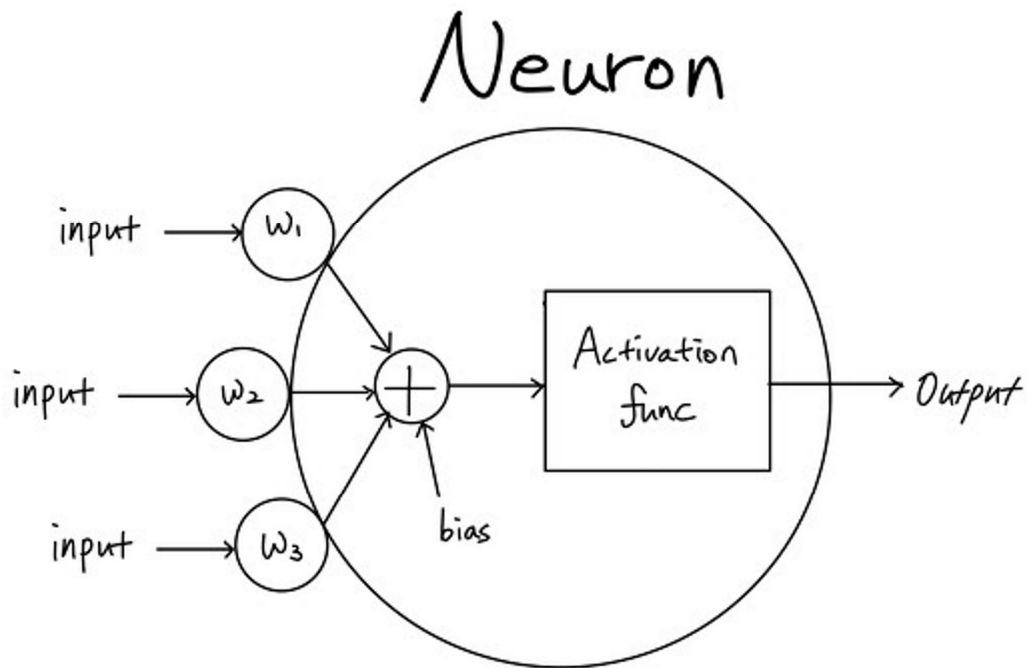
03. 신경망

2020년 12월 27일 일요일 오전 11:46

1. 신경망



2. 뉴런 모델



- 뉴런(node)으로의 입력을 $x_1, x_2, x_3 \dots, x_n$ (feature)
- 각 입력에 대응하는 가중치를 $w_1, w_2, w_3 \dots, w_n$ (weight)
- 각 입력에 대한 가중치 곱은 xw 로 표현
- 각 입력과 가중치의 갯수가 n 개 라고 하면 다음과 같이 일반화 가능

$$\sum_{k=1}^n x_k w_k$$

- 위 결과에 bias b 를 더한 결과는 다음과 같음

$$u = \sum_{k=1}^n (x_k w_k) + b$$

- 위 결과 u 를 활성화함수(Activation function) f 에 입력하여 출력된 결과를 y 라 하면

$$y = f(u) = f\left(\sum_{k=1}^n (x_k w_k) + b\right)$$

- 위의 입력과 가중치의 곱에 bias 를 더하는 부분을 행렬을 이용하면 간단하게 표현 가능

입력을 벡터 \vec{x} 로, 가중치를 벡터 \vec{w} 로 표현하면

$$\vec{x} = (x_1, x_2, x_3 \dots, x_n)$$

$$\vec{w} = (w_1, w_2, w_3 \dots, w_n)$$

이 둘을 행렬곱으로 표현

$$u = \vec{x}\vec{w} + b = (x_1, x_2, x_3 \dots, x_n) \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{pmatrix} + b$$

\vec{x} 와 \vec{w} 를 X, W로 사용하기도 함

$$u = XW + b$$

$$y = f(u) = f(XW + b)$$

3. 단일 신경망 구축

- 딥러닝에서 가중치와 bias는 임의의 수로 설정 후 학습하는 과정에서 갱신함
- 이번 예제는 가중치와 bias를 고정하고, 이 값들이 실제 신경망에서 어떤 의미인지를 확인
- 입력으로 단 두개의 값(feature) x_1, x_2
- 활성화함수로 시그모이드 함수 사용

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

# x1, x2 값
X1 = np.arange(-1.0, 1.0, 0.2) # 원소 수는 10개
X2 = np.arange(-1.0, 1.0, 0.2)

# 출력값을 저장할 10x10 그리드
Z = np.zeros((10,10))

# x1, x2 값의 입력 가중치
w_x1 = 2.5
w_x2 = 3.0

# 편향
bias = 0.1

# 그리드맵의 각 그리드별 뉴런의 연산
for i in range(10):
    for j in range(10):

        # 입력과 가중치 곱의 합 + 편향
        u = X1[i]*w_x1 + X2[j]*w_x2 + bias

        # 그리드맵에 출력 값 저장
        y = 1/(1+np.exp(-u)) # 시그모이드 함수
        Z[j][i] = y
```

```
# 그리드맵 표시
plt.imshow(Z, "gray", vmin = 0.0, vmax = 1.0)
plt.colorbar()
plt.show()
```

● numpy의 ndarray를 이용한 행렬연산

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

# x1, x2 값
X1 = np.arange(-1.0, 1.0, 0.2) # 원소 수는 10개
X2 = np.arange(-1.0, 1.0, 0.2)

# 출력값을 저장할 10x10 그리드
Z = np.zeros((10,10))

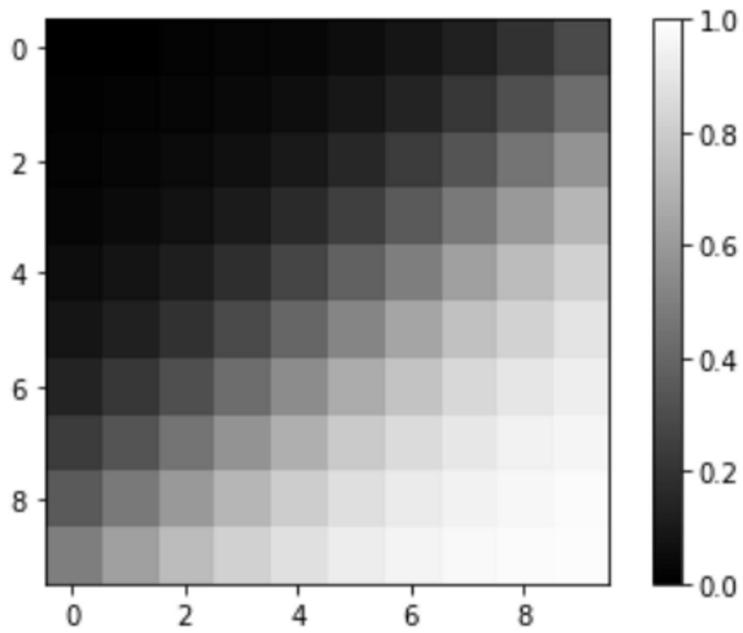
# x, y 값의 입력 가중치
W = np.array([2.5,3.0])

# 편향
bias = np.array([0.1])

# 그리드맵의 각 그리드별 뉴런의 연산
for i in range(10):
    for j in range(10):

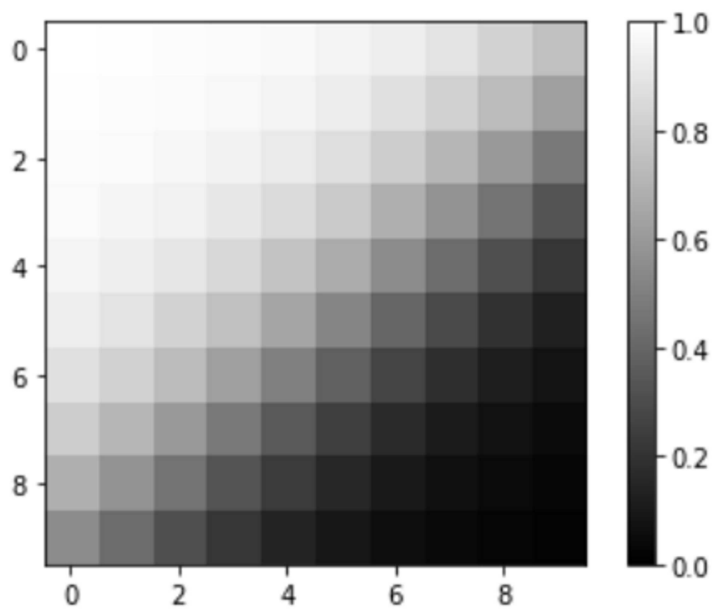
        # 입력과 가중치 곱의 합 + 편향
        list1 = [X1[i], X2[j]]
        X = np.array(list1)
        u = np.dot(X, W.T) + bias
        # 그리드맵에 출력 값 저장
        y = 1/(1+np.exp(-u)) # 시그모이드 함수
        Z[j][i] = y

# 그리드맵 표시
plt.imshow(Z, "gray", vmin = 0.0, vmax = 1.0)
plt.colorbar()
plt.show()
```

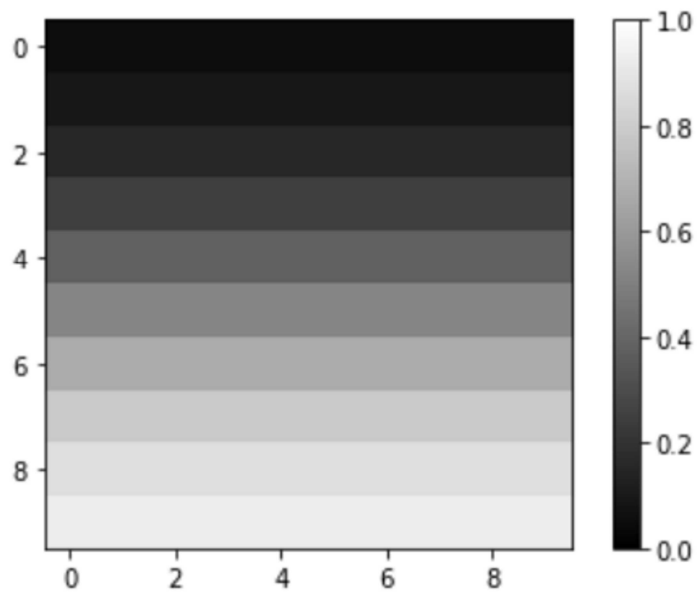


< 가중치와 bias의 영향 >

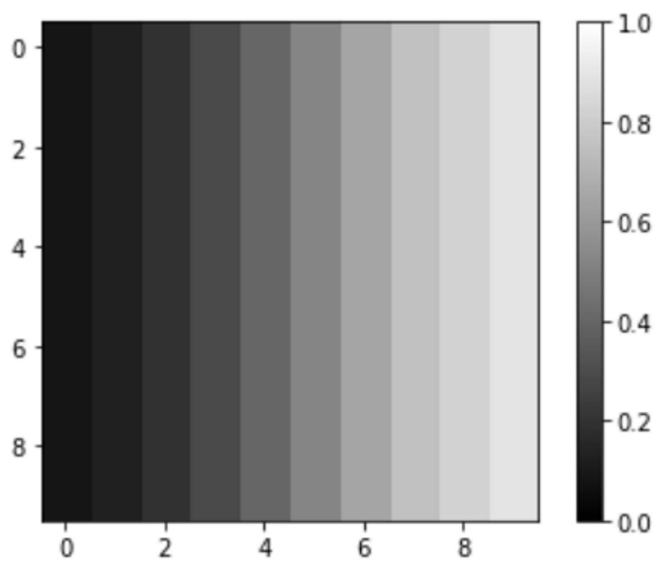
● $W = [-2.5, -3.0]$



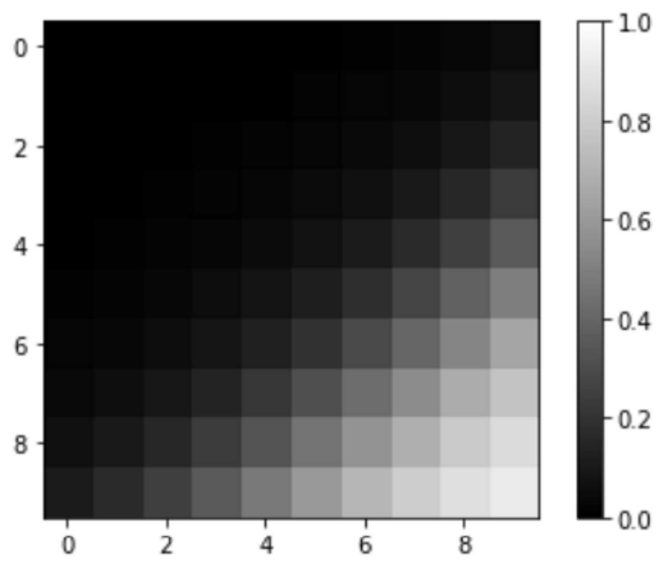
● $W = [0, 3.0]$



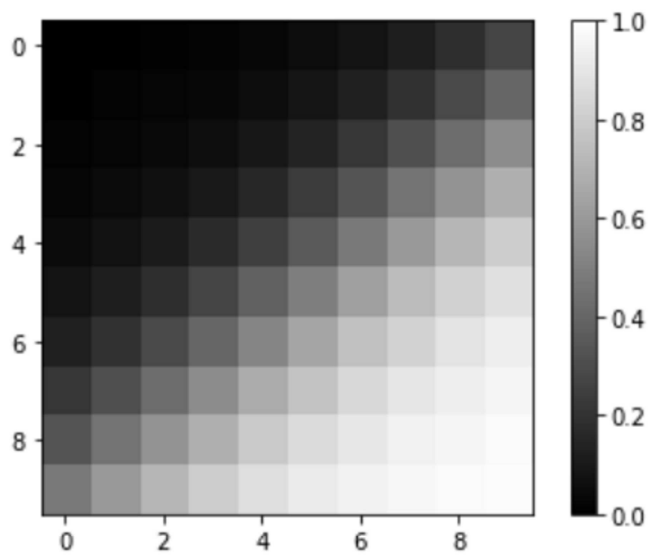
● $W = [2.5, 0]$



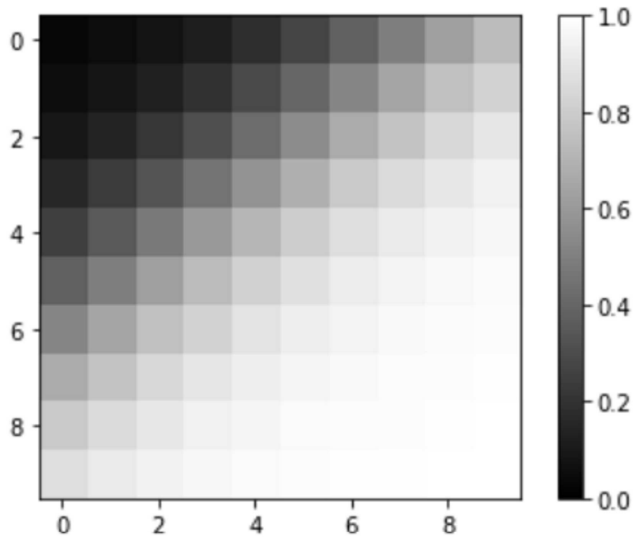
● $\text{bias} = [-2.0]$



● bias = [0.0]



● bias = [2.0]



4. 다중 신경망

- 뉴런(node)으로의 입력을 $X = [x_1, x_2, x_3 \dots, x_m]$ (feature)
- 출력 뉴런(노드)의 수를 n개라 가정
- 각 입력에 대응하는 가중치 W 행렬은 다음과 같음

$$\begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ w_{31} & w_{32} & \dots & w_{3n} \\ \vdots & \vdots & \dots & \vdots \\ w_{m1} & w_{m2} & \dots & w_{mn} \end{bmatrix}$$

- bias의 수도 뉴런의 수와 동일

$$B = [b, b_2, b_3 \dots, b_n]$$

- 각 입력과 가중치의 갯수가 n개 라고 하면 다음과 같이 일반화 가능

$$XW = \left[\sum_{k=1}^m x_k w_{k1}, \sum_{k=1}^m x_k w_{k2}, \sum_{k=1}^m x_k w_{k3}, \dots, \sum_{k=1}^m x_k w_{kn} \right]$$

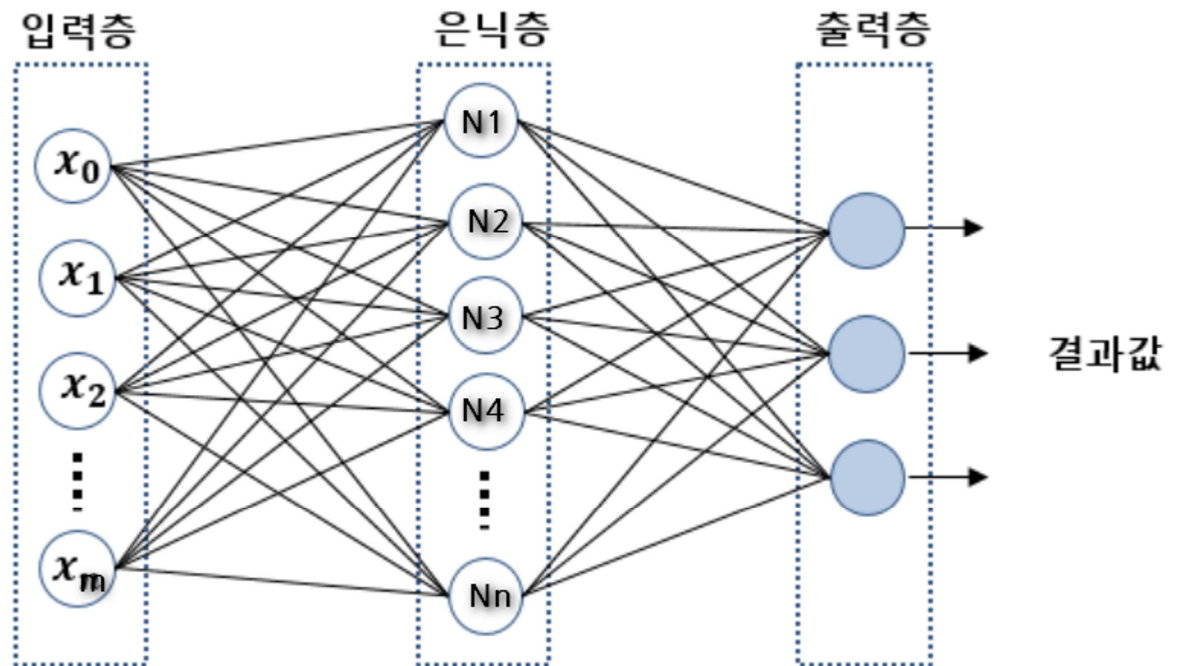
- 위 결과에 bias b를 더한 결과는 다음과 같음

$$U = \left[\sum_{k=1}^m (x_k w_{k1}) + b_1, \sum_{k=1}^m (x_k w_{k2}) + b_2, \sum_{k=1}^m (x_k w_{k3}) + b_3, \dots, \sum_{k=1}^m (x_k w_{kn}) + b_n \right]$$

- 위 결과 u를 활성화함수(Activation function) f에 입력하여 출력된 결과를 y라 하면

$$\begin{aligned} Y &= f(u) \\ &= \left[f\left(\sum_{k=1}^m (x_k w_{k1}) + b_1\right), f\left(\sum_{k=1}^m (x_k w_{k2}) + b_2\right), f\left(\sum_{k=1}^m (x_k w_{k3}) + b_3\right), \dots, f\left(\sum_{k=1}^m (x_k w_{kn}) + b_n\right) \right] \end{aligned}$$

- 위 결과를 그림으로 표현



- 지금 부터 구현하는 신경망을 Input Layer 입력 2개, Hidden Layer 노드 2개, Output Layer 노드 1개
- Hidden Layer 활성화 함수는 시그모이드, Output Layer 활성화 함수는 항등함수

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

# x, y 값
X = np.arange(-1.0, 1.0, 0.2) # 원소는 10개
Y = np.arange(-1.0, 1.0, 0.2)

# 출력을 저장하는 10x10 그리드
Z = np.zeros((10,10))

# 가중치
w_im = np.array([[4.0,4.0],
                  [4.0,4.0]]) # 은닉층 2x2 행렬
w_mo = np.array([[1.0],
                  [-1.0]]) # 출력층 2x1 행렬

# 편향
b_im = np.array([3.0,-3.0]) # 은닉층
b_mo = np.array([0.1]) # 출력층

# 은닉층
def middle_layer(x, w, b):
    u = np.dot(x, w) + b
    return 1/(1+np.exp(-u)) # 시그모이드 함수
```

```

# 출력층
def output_layer(x, w, b):
    u = np.dot(x, w) + b
    return u # 항등함수

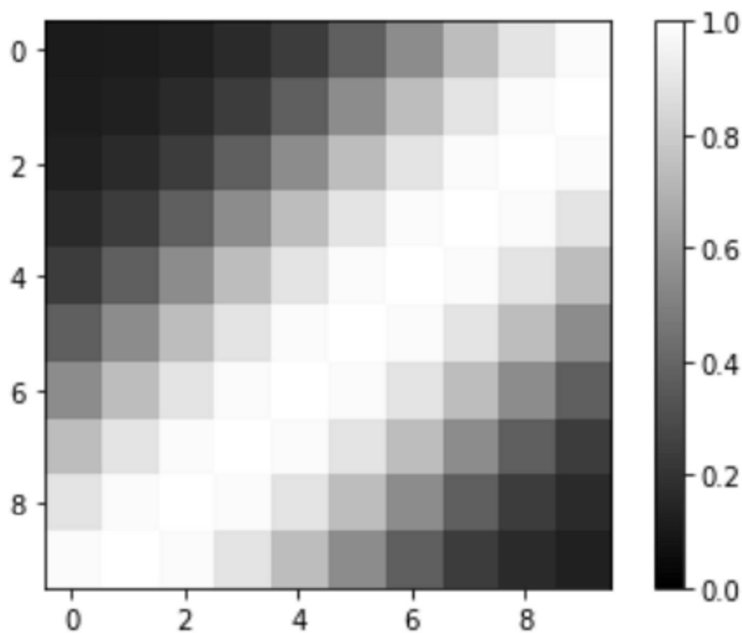
# 그리드맵의 각 그리드별 신경망 연산
for i in range(10):
    for j in range(10):

        # 순전파
        inp = np.array([X[i], Y[j]]) # 입력층
        mid = middle_layer(inp, w_im, b_im) # 은닉층
        out = output_layer(mid, w_mo, b_mo) # 출력층

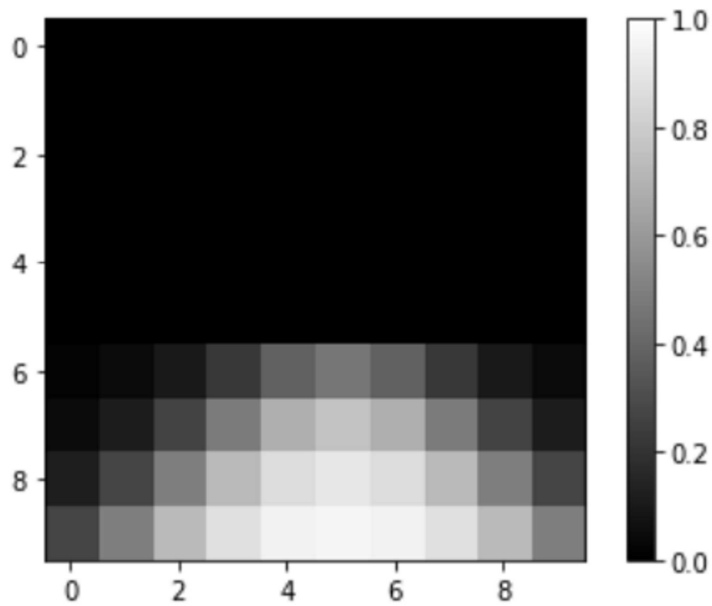
        # 그리드맵에 신경망 출력 값 저장
        Z[j][i] = out[0]

# 그리드맵으로 표시
plt.imshow(Z, "gray", vmin = 0.0, vmax = 1.0)
plt.colorbar()
plt.show()

```

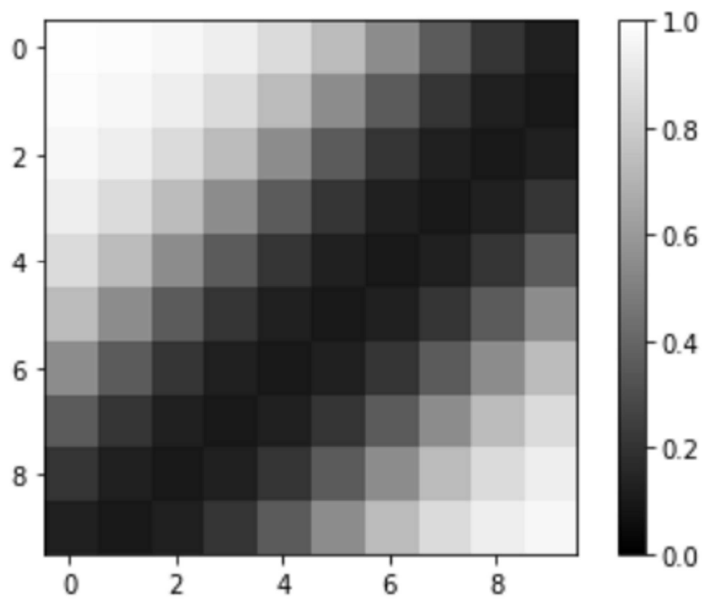


- `w_im = np.array([[-5.0,-5.0],
[5.0,-5.0]])`
`w_mo = np.array([[1.0],
[-1.0]])`
`b_im = np.array([0.0,0.0])`
`b_mo = np.array([0.0])`



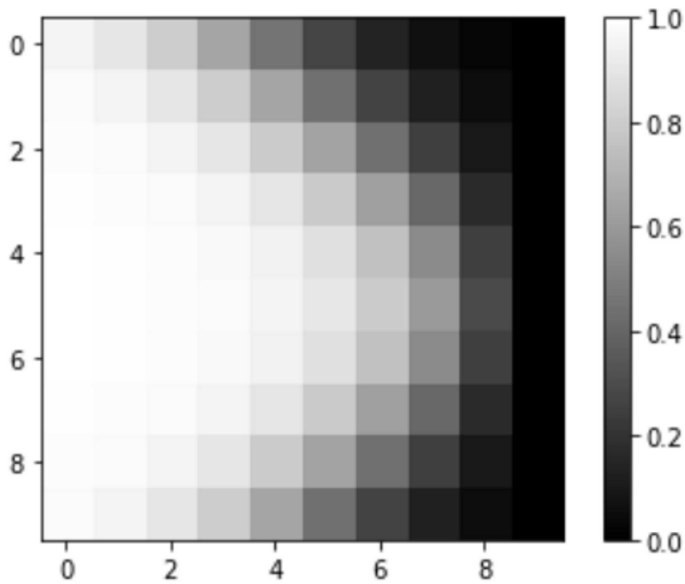
- `w_im = np.array([[4.0, 4.0],
[4.0,4.0]])`
`w_mo = np.array([[-1.0],
[1.0]])`

`b_im = np.array([3.0,-3.0])`
`b_mo = np.array([1.0])`



- `w_im = np.array([[-4.0, 4.0],
[-4.0,-4.0]])`
`w_mo = np.array([[1.0],
[-1.0]])`

`b_im = np.array([3.0,-3.0])`
`b_mo = np.array([0.0])`



5. 분류

- Input Layer 입력 2개, Hidden Layer 노드 2개, Output Layer 노드 2개
- Output Layer 활성화 함수는 소프트맥스 사용

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

# x, y 값
X = np.arange(-1.0, 1.0, 0.1) # 원소 수는 20개
Y = np.arange(-1.0, 1.0, 0.1)

# 가중치
w_im = np.array([[1.0, 2.0],
                  [2.0, 3.0]]) # 은닉층 2x2 행렬
w_mo = np.array([[-1.0, 1.0],
                  [1.0, -1.0]]) # 출력층 2x2 행렬

# 편향
b_im = np.array([0.3, -0.3]) # 은닉층
b_mo = np.array([0.4, 0.1]) # 출력층

# 은닉층
def middle_layer(x, w, b):
    u = np.dot(x, w) + b
    return 1/(1+np.exp(-u)) # 시그모이드 함수

# 출력층
def output_layer(x, w, b):
    u = np.dot(x, w) + b
    return np.exp(u)/np.sum(np.exp(u)) # 소프트맥스 함수
```

```

# 분류 결과를 저장하는 리스트
x_1 = []
y_1 = []
x_2 = []
y_2 = []

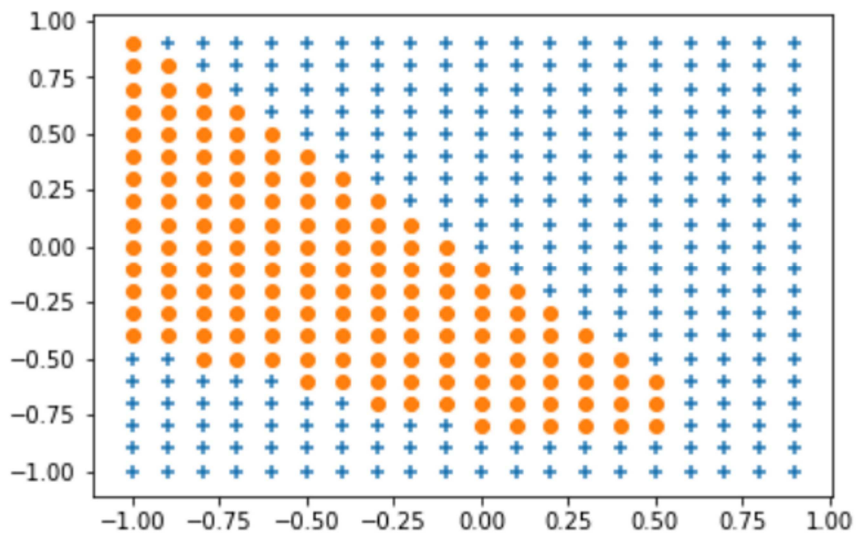
# 그리드맵의 각 그리드별 신경망 연산
for i in range(20):
    for j in range(20):

        # 순전파
        inp = np.array([X[i], Y[j]])
        mid = middle_layer(inp, w_im, b_im)
        out = output_layer(mid, w_mo, b_mo)

        # 확률의 크기를 비교해 분류함
        if out[0] > out[1]:
            x_1.append(X[i])
            y_1.append(Y[j])
        else:
            x_2.append(X[i])
            y_2.append(Y[j])

# 산포도 표시
plt.scatter(x_1, y_1, marker="+")
plt.scatter(x_2, y_2, marker="o")
plt.show()

```



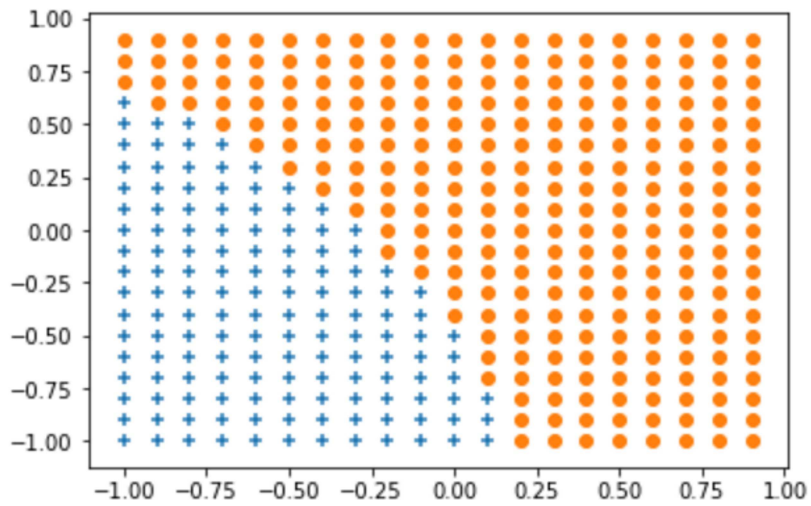
```

● w_im = np.array([[2.0,1.0],
                   [0.0,3.0]])
w_mo = np.array([[-2.0,1.0],
                 [-1.0,-1.0]])

# 편향

```

```
b_im = np.array([-0.3,-0.3])
b_mo = np.array([0.4,-1.2])
```



```
● w_im = np.array([[2.0,2.0],
                    [2.0,3.0]])
w_mo = np.array([[-1.0,1.0],
                  [1.0,-1.0]])
```

편향

```
b_im = np.array([0.3,-0.3])
b_mo = np.array([0.4,0.1])
```

