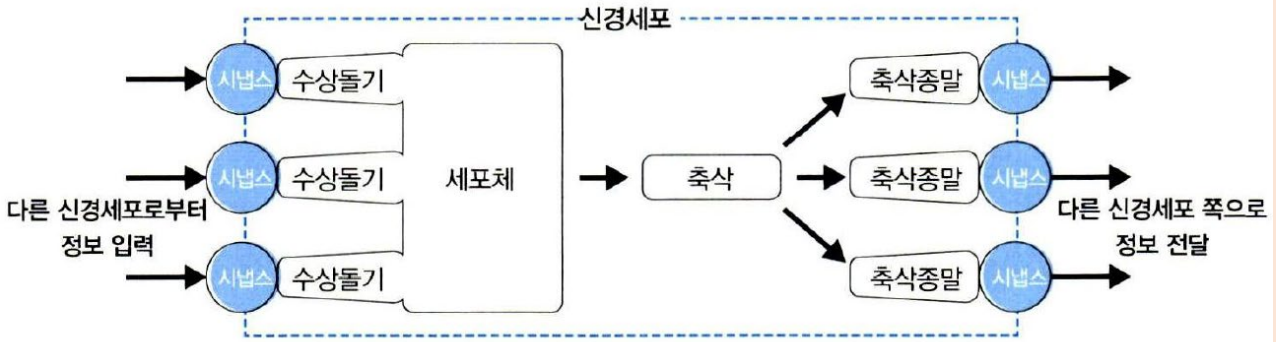
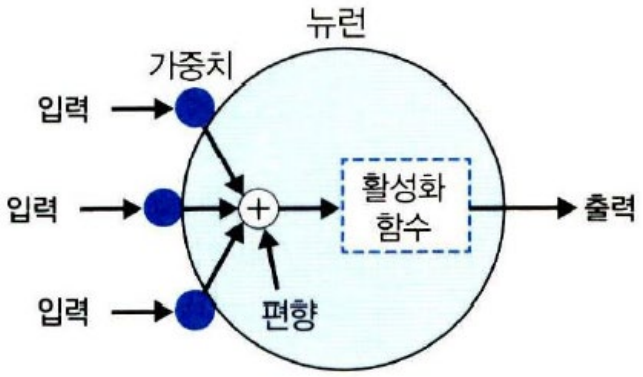
[DL] 신경망

『실체가 손에 잡히는 딥러닝』 Ch. 4

강재영, 구병모, 김영채



<신경세포 구조도>



<뉴런 모델>

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

- **신경세포(Neuron): 생물체의 신경계에서 정보를 전달하고 기억을 유지하는 역할 담당**

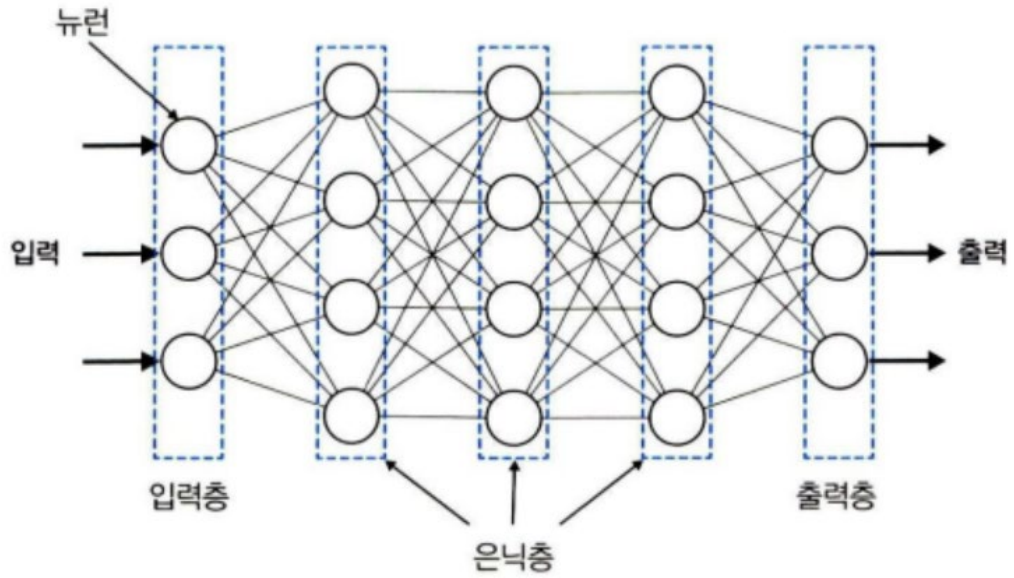
- **인공 신경망(ANN): 컴퓨터 상에서 모델링 된 신경세포 네트워크 (신경망)**

- **뉴런: 다수의 입력, 하나의 출력**
- 가중치: 각 입력에 곱해지는 상수

  -> 클수록 더 많은 정보 전달
- 편향: 뉴런의 감도를 나타냄

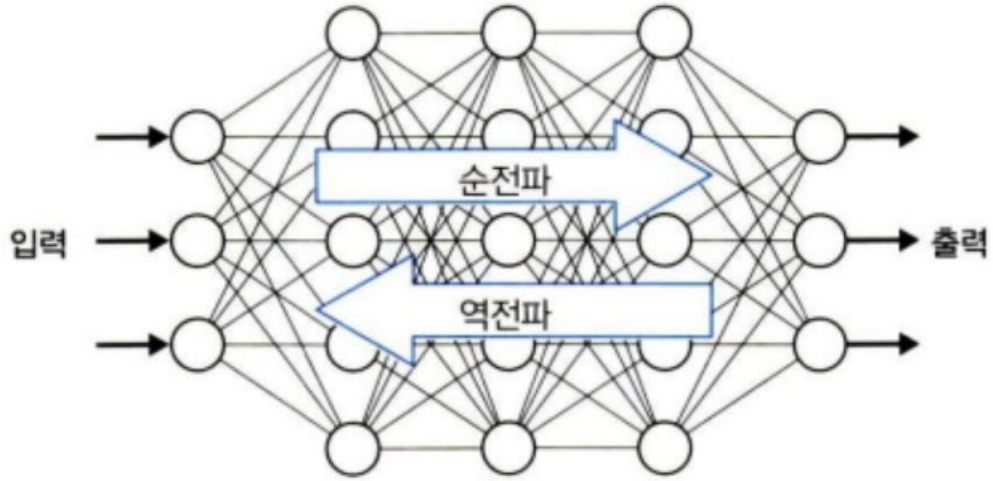
  -> 뉴런의 흥분 정도
- 활성화 함수: 뉴런의 흥분 상태를 표시하는 신호로 전환하는 함수

그림 4-5 신경망



- 입력층: 전체 신경망의 입력값을 받아들임
- 은닉층: 입력층과 출력층 사이에 위치한 층
- 출력층: 전체 신경망의 출력값을 내보냄

그림 4-6 순전파와 역전파

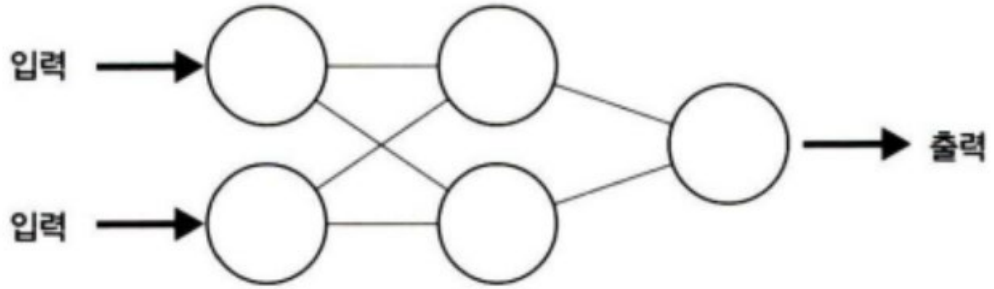


- 순전파: 입력 -> 출력 순서로 정보 전달

-> 예측값을 통해 실제값과 예측값의 차이 확인 가능

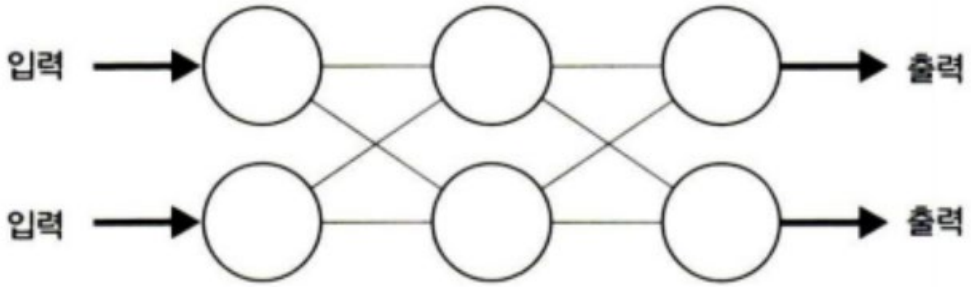
- 역전파: 출력 -> 입력 순서로 정보 전달

-> 출력층의 출력값에 대한 입력층의 입력값 기울기 확인 가능



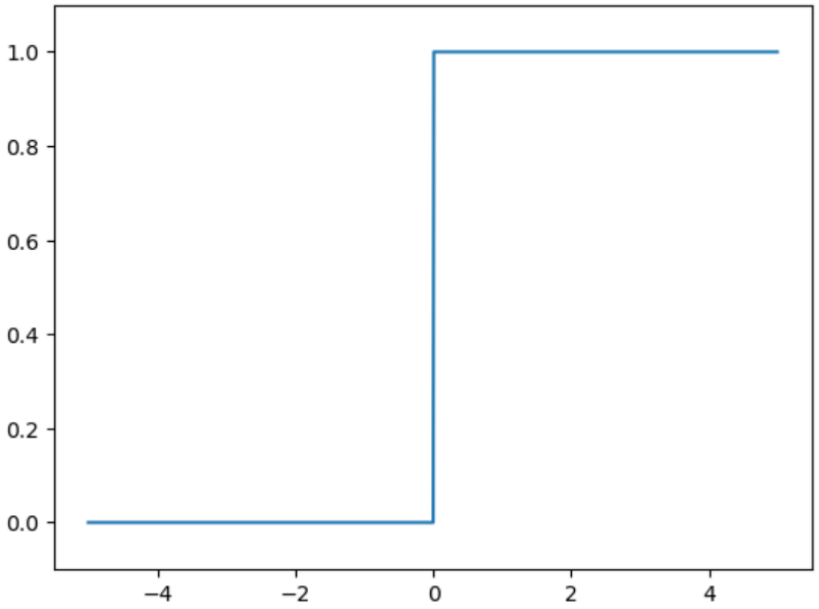
- **회귀(Regression): 데이터의 경향성(추세)으로 연속적인 수치를 예측하는 문제**

출력층의 출력이 그대로 예측값이 됨



- **분류(Classification): 데이터를 정해진 범주에 따라 분류하는 문제**

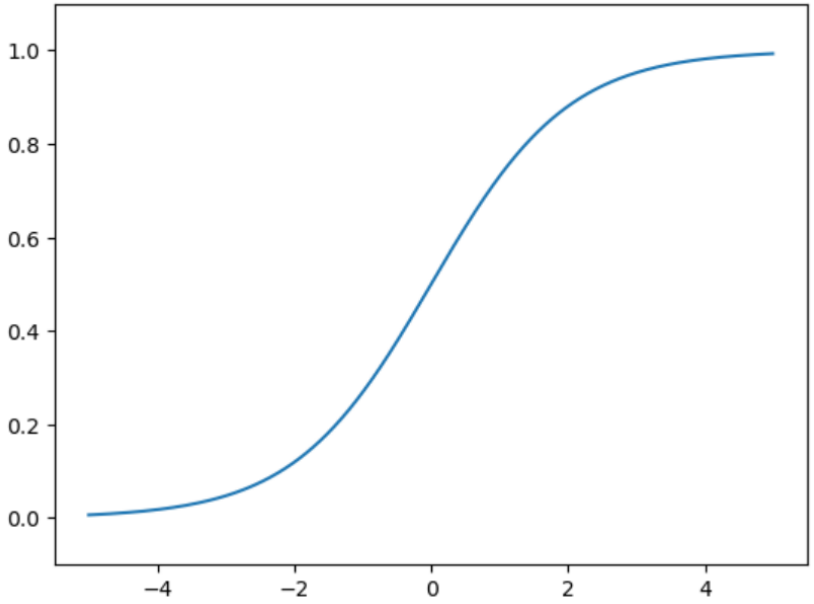
분류되는 범주의 개수만큼 출력층이 생성되고, 출력값은 각 범주로 분류될 확률



- **계단 함수 (Step function)**

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

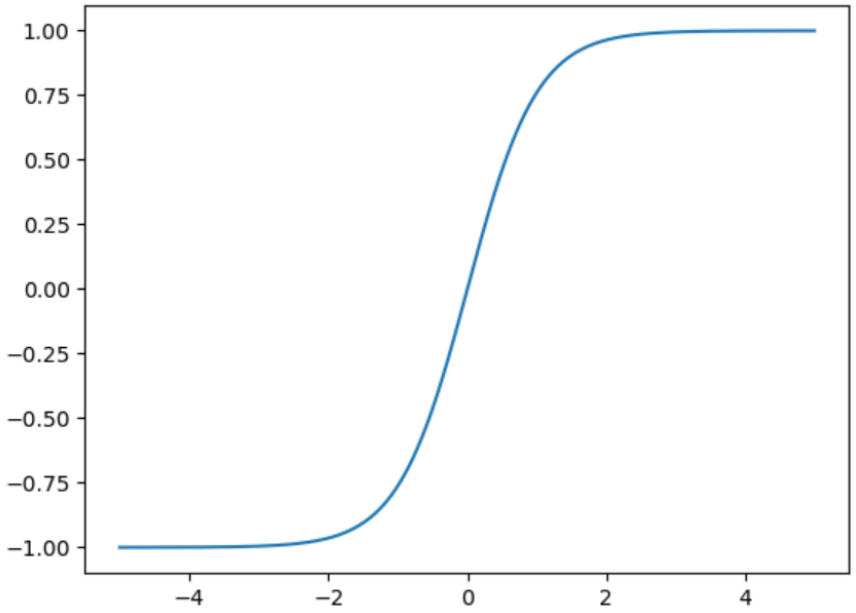
- 장점: 뉴런의 흥분 상태를 0과 1로 간단하게 표시 가능
- 단점: 0과 1의 중간상태를 나타낼 수 없음



- **시그모이드 함수 (Sigmoid function)**

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

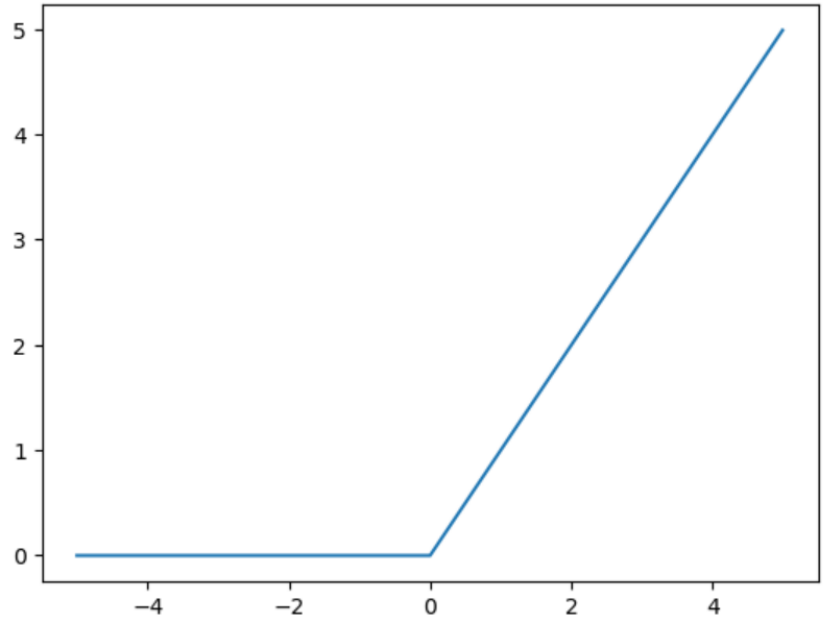
- 장점: 미분 가능
- 신경망에서는 활성화 함수로 시그모이드 함수를 이용하여 신호를 변환하고, 변환된 신호를 다음 뉴런에 전달



- **tanh 함수 (Hyperbolic tangent function)**

$$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

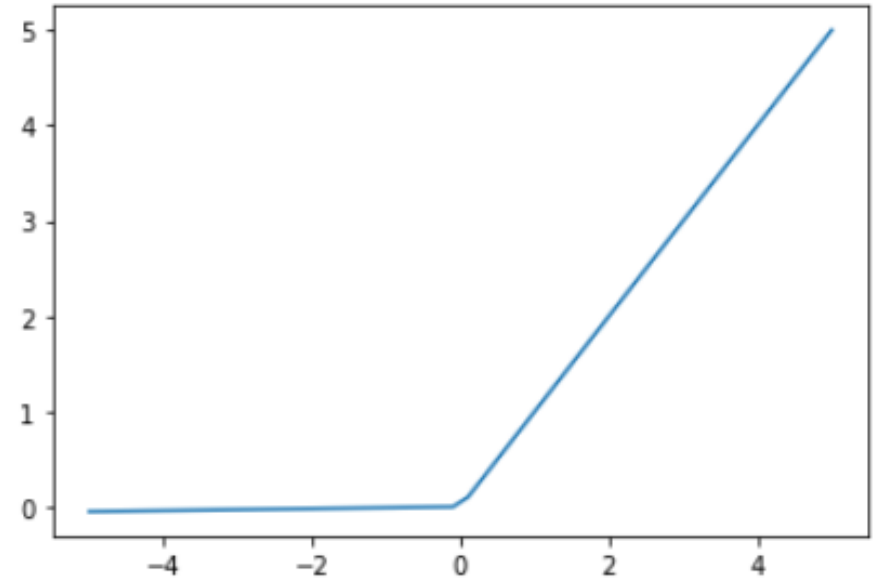
• 함수의 중심점을 0으로 옮겨 sigmoid function의 학습이 느려지는 단점을 해결
• 0을 중심으로 대칭 (기함수)



- **ReLU 함수 (Rectified Linear Unit, Ramp function)**

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

• 장점: 미분값이 x에 상관없이 안정적인 값을 얻어 연산이 빠름
• 단점: 무한히 커지기 때문에 회생하기 힘듦

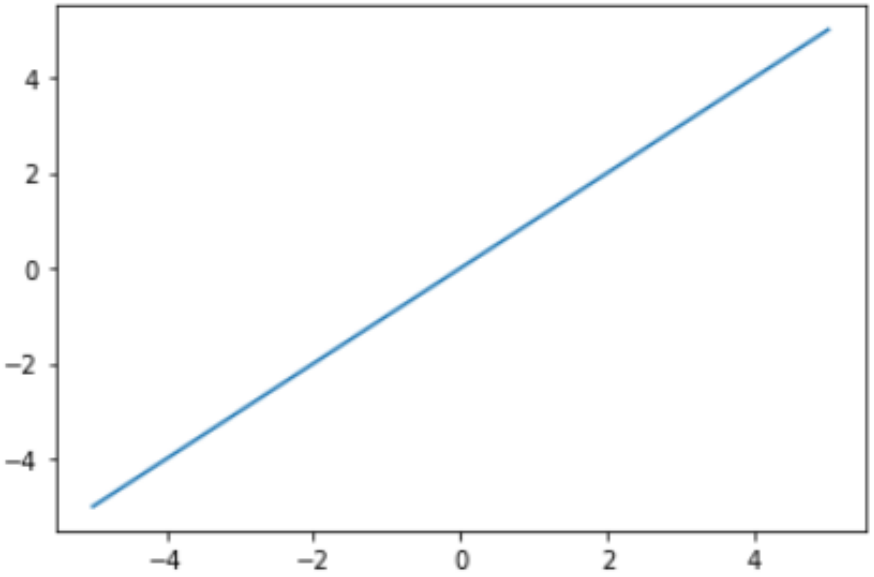


- **Leaky ReLU 함수**

$$y = \begin{cases} 0.01x & (x \leq 0) \\ x & (x > 0) \end{cases}$$

- ReLU와 비슷하지만 x가 음수인 경우 매우 작은 기울기를 만들어 dying ReLU 문제 해결

* Dying ReLU: 출력이 0이 되어 더 이상 학습이 진행되지 않는 뉴런이 다수 발생하는 문제



- **항등 함수 (Identify function)**

$$y = x$$

- 함수의 입력값이 그대로 출력값으로 반환
- 출력 범위에 제한이 없고 연속적이기 때문에

회귀 문제에서 많이 사용

```python
# 소프트맥스 함수 구현

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

    return y
```

- **소프트맥스 함수 (Softmax function)**

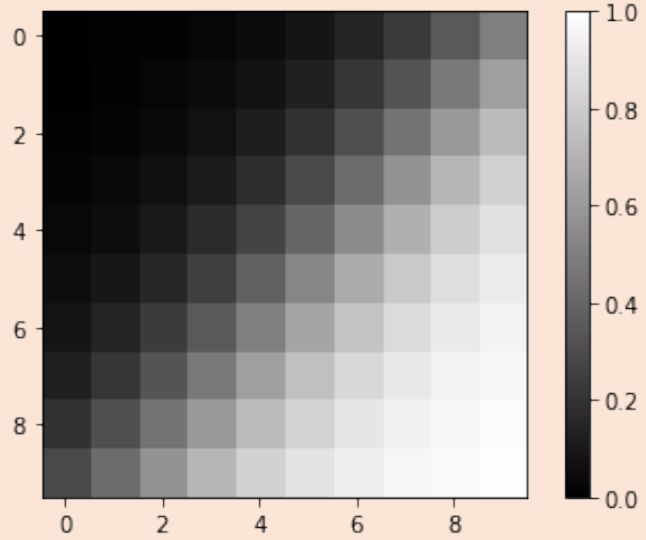
$$y = \frac{\exp(x)}{\sum_{i=1}^{n} \exp(x_k)}$$

- 같은 층의 모든 활성화 함수의 출력값을 모두 더하면 1
- 한 층에 여러 개의 뉴런이 존재, 각 뉴런에서 활성화 함수를 통과한 값은 0에서 1 사이 -> 분류 문제에 주로 사용
- 지수 함수를 사용하기 때문에 overflow 주의

- 개선식:

$$y = \frac{C\ exp(x)}{C\ \sum_{i=1}^{n} exp(x_k)} = \frac{exp(x+log\ C)}{\sum_{i=1}^{n} exp(x_k+log\ C\ )} = \frac{\exp(x+C')}{\sum_{i=1}^{n} \exp(x_k+C')}$$

```python
# -1 ~ 1까지 (간격: 0.2) 각 10개의 값을 넘파이 배열로 저장
X = np.arange(-1.0, 1.0, 0.2)
Y = np.arange(-1.0, 1.0, 0.2)

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

# x, y값의 입력 가중치
w_x = 2.5
w_y = 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 = X[i]*w_x + Y[j]*w_y + bias

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

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

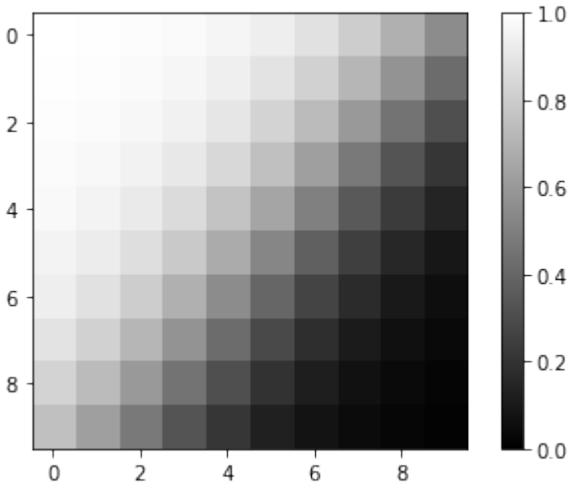


- **단일 뉴런 구현**

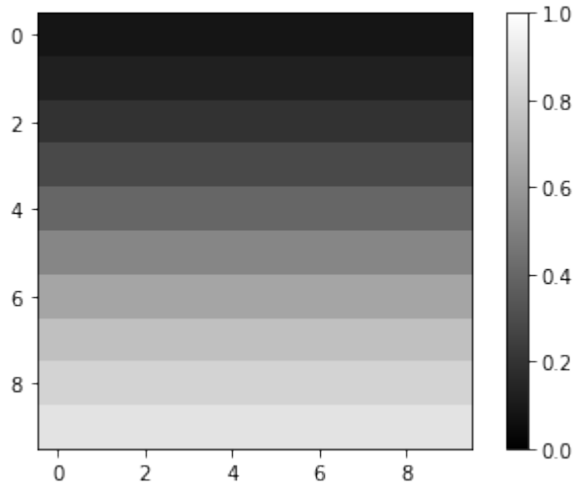
- 검정색 -> 출력값이 0 (뉴런이 흥분하지 않은 상태)
- 흰색 -> 출력값이 1 (뉴런이 흥분한 상태)
- 좌상단에서 우하단까지 연속적으로 변화하는 모습

-> 시그모이드 함수를 활성화 함수로 사용했기 때문

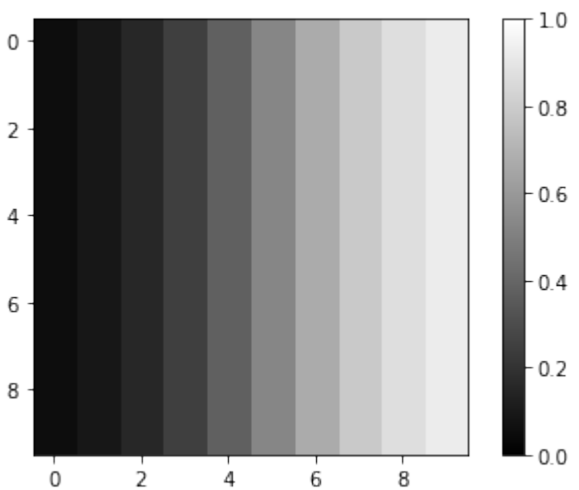
- 다양한 **가중치** 값에 따른 출력 그리드맵



w_x = −2.5, w_y = −3.0

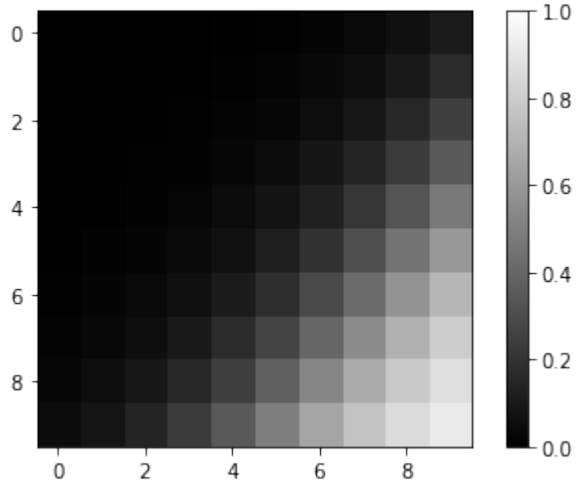


w_x = 0, w_y = 3.0

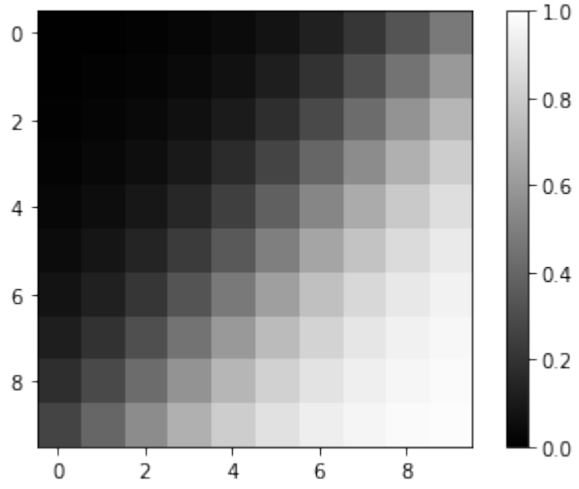


w_x = 2.5, w_y = 0

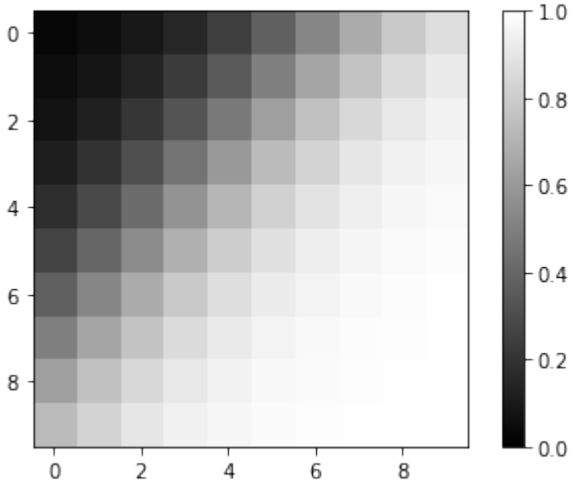
- 다양한 **편향** 값에 따른 출력 그리드맵



bias = −2.0



bias = 0.0

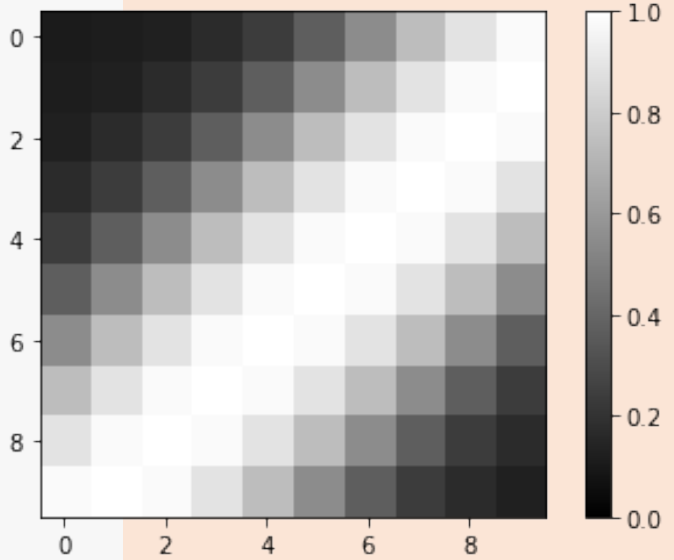


bias = 2.0

```python
# x, y값
X = np.arange(-1.0, 1.0, 0.2)
Y = np.arange(-1.0, 1.0, 0.2)

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

# 가중치
w_im = np.array([[4.0,4.0],
                 [4.0,4.0]])   # 은닉층: 2 * 2 행렬
w_mo = np.array([[1.0],
                 [-1.0]])      # 출력층: 2 * 1 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):     # 시그모이드 함수
    u = np.dot(x, w) + b
    return 1/(1+np.exp(-u))

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

```python
# 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 출력값 저장
        Z[j][i] = out[0]
```

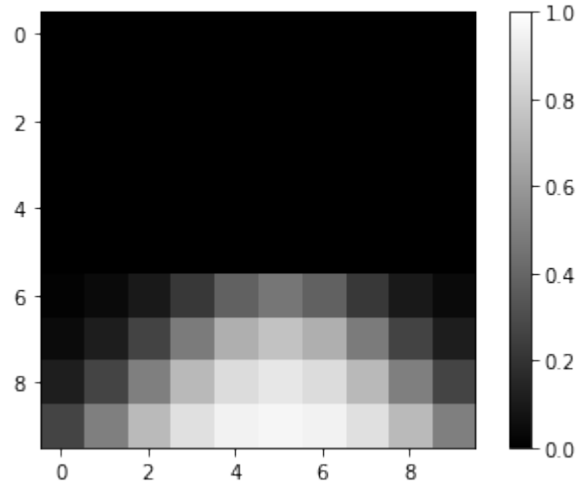
```python
# 그리드맵으로 표시
plt.imshow(Z, "gray", vmin = 0.0, vmax = 1.0)
plt.colorbar()
plt.show()
```



- **신경망 (회귀)**

- 입력층(뉴런 수 : n=2), 은닉층(n=2), 출력층(n=1)의 3층 구조
- 다수의 뉴런으로 형성되어 흰색 영역이 검정색 영역에 끼어있는 모습

- 다양한 **가중치** 값에 따른 출력 그리드맵

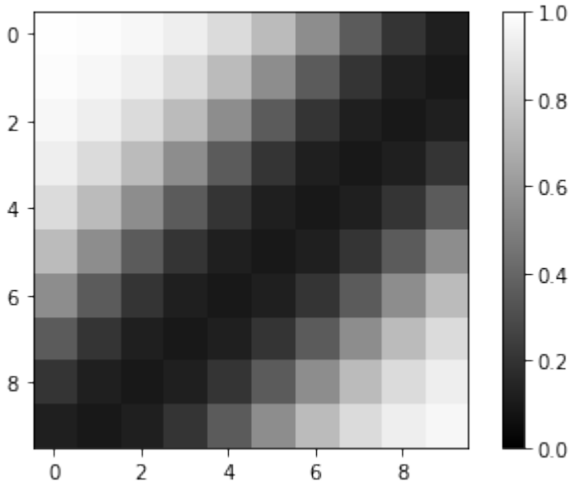


w_im = np.array([[−5.0, −5.0], [5.0, −5.0]])

w_mo = np.array([[1.0], [−1.0]])

b_im = np.array([0.0, 0.0])

b_mo = np.array([0.0])

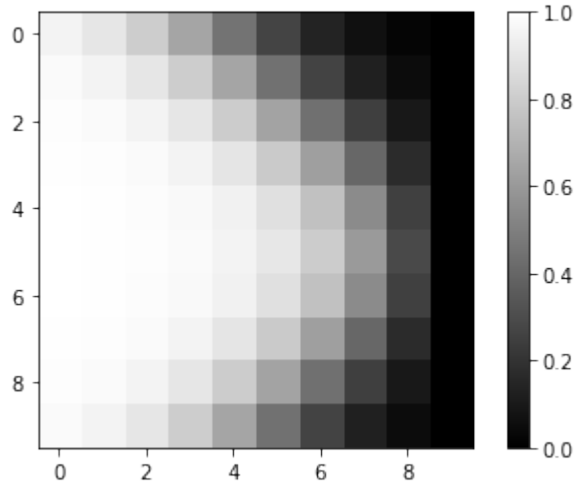


w_im = np.array([[4.0, 4.0], [4.0, 4.0]])

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]])

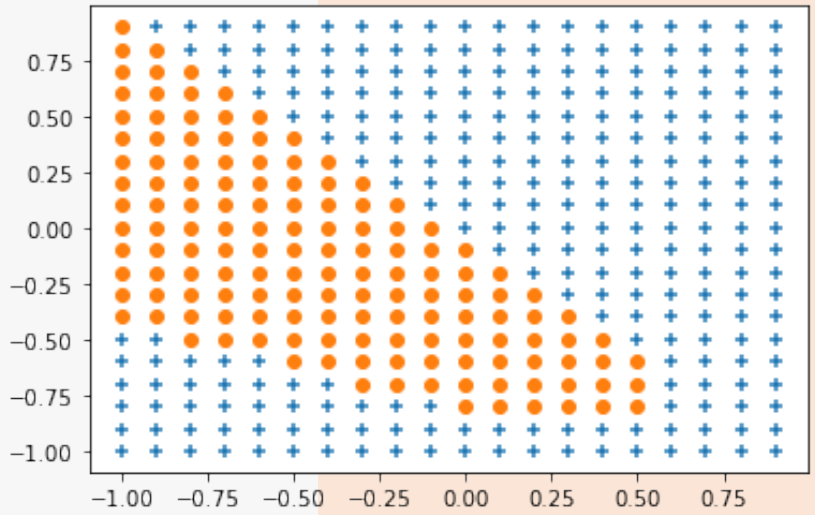
w_mo = np.array([[1.0], [−1.0]])

b_im = np.array([3.0, −3.0])

b_mo = np.array([0.0])

```python
# x, y값
X = np.arange(-1.0, 1.0, 0.1)
Y = np.arange(-1.0, 1.0, 0.1)

# 가중치
w_im = np.array([[1.0,2.0],
                 [2.0,3.0]])   # 은닉층: 2 * 2 행렬
w_mo = np.array([[-1.0,1.0],
                 [1.0,-1.0]])  # 출력층: 2 * 1 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       # 시그모이드 함수
    return 1/(1+np.exp(-u))

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

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

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

        # 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

        # 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()
```



- **신경망 (분류)**

- 회귀와 달리 결과값이 연속적이지 않고 경계가 명확히 구분됨
- 층을 더 쌓으면 표현력이 향상됨