신경망의 이해

로지스틱 회귀

회귀 문제는 보통 연속형 분류문제를 다루지만 로지스틱 회귀는 참 거짓과 같은 범주형 분류 문제를 해결하기 위해 사용 된다. 선형 회귀문제와 마찬가지로 독립 변수와 종속 변수 간의 관계를 모델링하는데 독립 변수의 결합을 로지스틱 함수(시그모이드 함수)를 통해 변환하여 종속 변수의 확률을 예측한다.

시그모이드 함수

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

이때 z는 $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$ 독립 변수들의 선형 결합이다.

출력 값은 시그모이드 함수의 특성으로 인해 0과 1사이의 값으로 변환된 확률 값이며 특정 임계값을 넘으면 클래스 1로 분류하고 아니면 클래스 0으로 분류한다.

손실함수

log-loss와 cross entropy를 사용한다

log-loss

$$\text{logloss}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0 \end{cases}$$

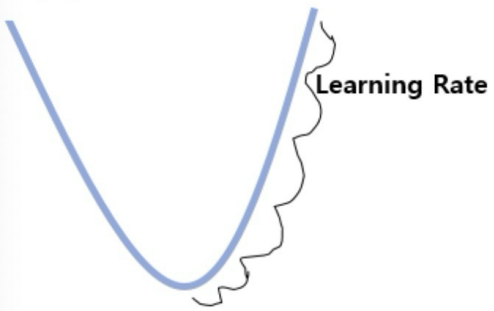
근데 if문을 고려해서 코딩하면 복잡해지니 한줄로 표현해서

$$\text{logloss}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y) \log(1 - h_\theta(x))$$

왜 최소 제곱법 안쓰는가?

$$Cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^i) - (y^i))^2$$

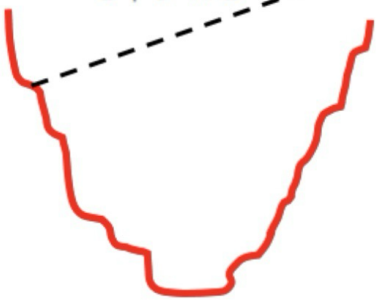
$H(X) = Wx + b$



Linear Regression

$H(X) = \frac{1}{1 + e^{-Wx}}$

평평한 부분을 가장 오차가 적은 부분으로 생각할 수 있음

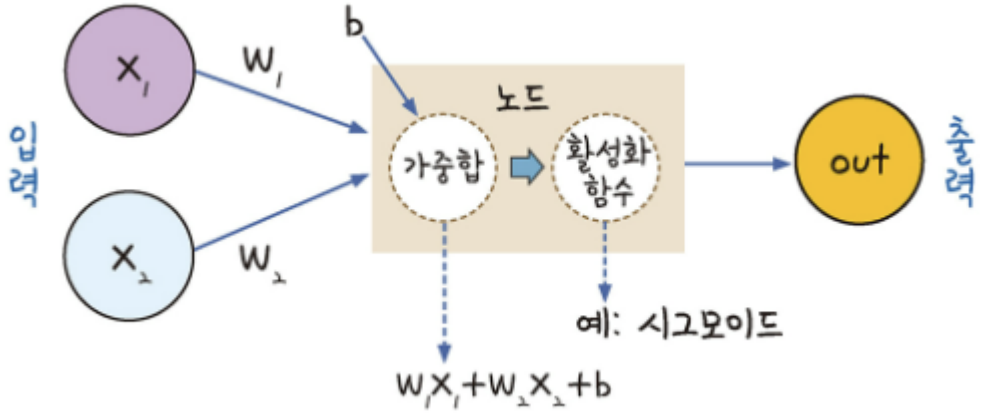


Logistic Regression

단일 퍼셉트론

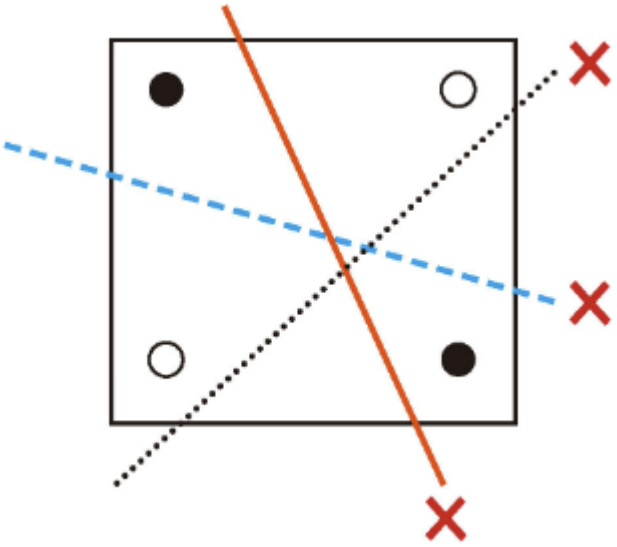
신경망을 이루는 가장 기본 단위

입력 값과 활성화 함수를 사용하여 출력 값을 다음으로 넘기는 가장 작은 신경망 단위



단일 퍼셉트론의 문제

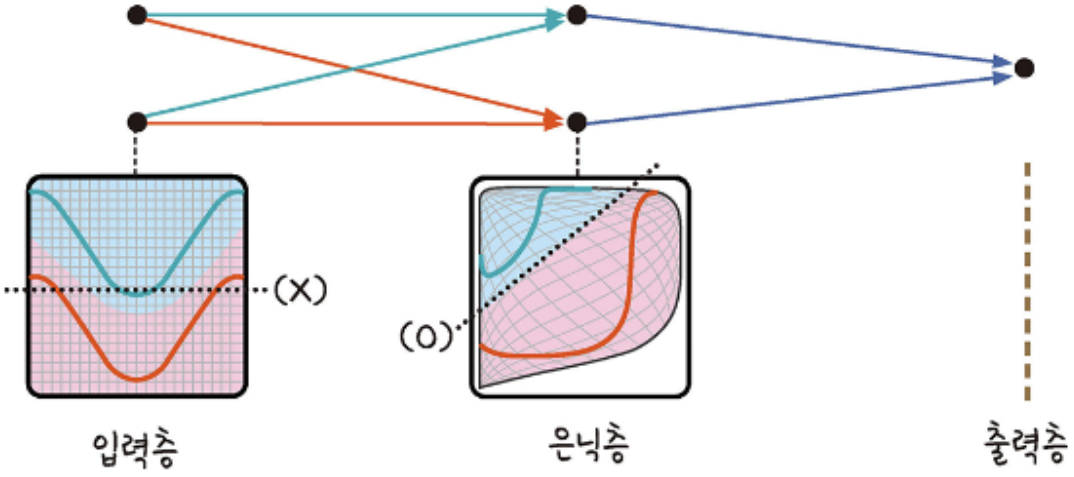
XOR 문제를 풀지 못한다.



2차원에서 줄 하나 그어서는 검은 점과 흰점을 구분하지 못한다.

이 문제를 해결하기 위해 **다층 퍼셉트론**을 사용한다

은닉층(hidden layer)을 만들어서 차원을 늘림



드디어 XOR 문제를 해결 할 수 있게 됐다. 그런데 과연 해결됐을까?

은닉층이 많이지면 생기는 문제점

1. 모델의 복잡성이 올라간다
    a. 오버피팅될 확률이 올라간다
    b. 드롭아웃 사용
2. 기울기 소멸 문제 발생

a. 출력층에서 은닉층으로 전달되는 오차가 크게 줄어들어 학습이 원활X
b. 활성화 함수를 시그모이드나 하이퍼볼릭 탄젠트 대신 렐루를 사용

3. 성능이 나빠진다
    a. 극복
        i. 확률적 경사하강법
        ii. 미니 배치 경사하강법

번외 차원 정리

간단 정리

1. size는 총 element의 개수
2. dim은 텐서가 존재하는 축의 개수
3. shape는 각 축에 존재하는 원소의 개수
4. rank는 텐서가 몇 차원인지 나타낸다

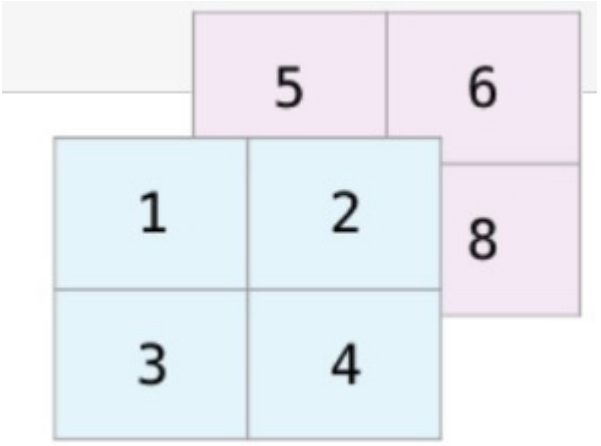
```
b = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print(b)

print("size :", b.size)
print("dim :", b.ndim)
print("shape :", b.shape)

#실행 결과
[[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
```

```
size : 8
dim : 3
shape : (2, 2, 2)
```



dim

1차원 [1,2] [3,4] [5,6] [7,8] **>>> dim1**

2차원 [[1,2], [3,4]] [[5,6],[7,8]] **>>> dim2**

3차원 **[** [[1,2] , [3,4]] **,** [[5,6] , [7,8]] **]** **>>> dim3**

(dim3,dim2,dim1)

dim은 축이 아니라 차원이라는 단어로도 쓰여서 헷갈릴 수 있는데 벡터나 행렬의 차원과 다르다

벡터, 행렬의 차원은 선형 독립을 이루는 열벡터의 개수를 의미한다

shape

1차원 벡터 안에는 원소가 2개씩 있다 **>>> 2**

2차원 행렬 안에는 1차원 벡터들이 2개씩 있다 **>>> 2**

3차원에는 2차원 행렬이 2개 있다 **>>> 2**

rank

scalar는 rank가 0인 텐서 ⇒ shape []

[1,2,3] 벡터는 rank가 1인 텐서 ⇒ shape [3]

[[1,2,3] , [4,5,6]] 행렬은 rank가 2인 텐서 ⇒ shape [2,3]

[[[1,2,3]],[[7,8,9]]] 3차원 배열은 rank가 3인 텐서 ⇒ shape [2,1,3]

행렬의 rank와 개념이 다르다

https://www.youtube.com/watch?v=GKkKkXtb_18&t=200s