3주차

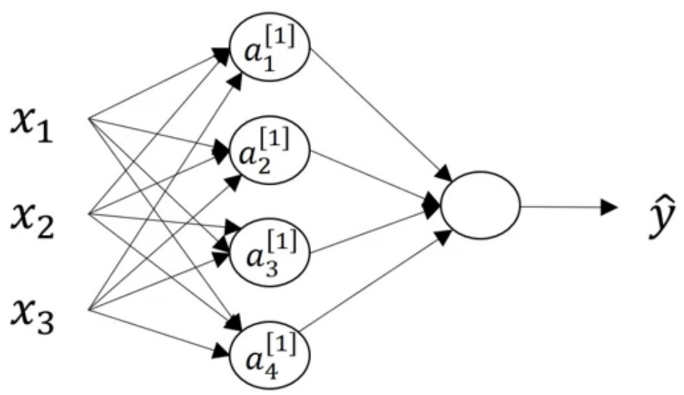
4. 얕은 신경망 네트워크

Neural Networks Overview

- 대괄호 위첨자 : 신경망의 i번째 레이어
- 소괄호 위첨자 : i번째 훈련 샘플
- 신경망에서는 z, a를 여러번 계산

Neural Network Representation

은닉층이 하나인 신경망



- 입력층 (Input layer)
    - 신경망의 입력 특성들의 층
    - 표기법 $a^{[0]} = X$
        - a는 활성값을 의미하고 신경망의 층들이 다음 층으로 전달해주는 값
        - $a^{[0]}$은 입력층의 활성값
- 은닉층 (Hidden layer)
    - 훈련 세트에서 볼 수 없다는 것을 의미
        - 입력값, 출력값은 알 수 있지만 은닉층의 값들은 알 수 x

- $a^{[1]}$: (1,4) 행렬
  - 노드 1은 $a_1^{[1]}$ 노드 2는 $a_2^{[1]}$ 노드 3은 $a_3^{[1]}$ 노드 4는 $a_4^{[1]}$
- 4차원인 이유는 은닉층에 은닉 노드가 4개 있기 때문
- $w^{[1]}$는 (4,3) $b^{[1]}$는 (4,1)벡터
  - 4는 은닉노드 4개, 3은 입력특성 3개

- 출력층 (Output layer)
  - 노드 1개, 예측값 $\hat{y}$ 계산
  - $\hat{y} = a^{[2]}$
  - w는 (1,4), b는 (1,1)
    - (1,4)는 은닉 노드 4개, 출력층 노드 1개,
- 로지스틱 회귀에서는 출력층 1개만 있어서 대괄호 위첨자를 사용하지 않았지만 신경망에서는 위첨자를 사용해 어떤 층에서 만들어진 건지 표기
- 이 신경망은 2층 신경망
  - 신경망의 층을 셀 때 입력층은 세지 않기 때문
  - 입력층 : 0번째 층, 은닉층 : 1번째 층, 출력층 : 2번째 층

Computing a Neural Network's Output

- 신경망의 출력값 계산 - 로지스틱 회귀와 비슷하지만 여러번 반복
- 로지스틱 회귀 : (1) $x = W^T x + b$ (2) $a = \sigma(z)$
- 신경망의 첫 번째 은닉층 노드 1개 계산
  - (1) $z_1^{[1]} = w_1^{[1]T} + b_1^{[1]}$
  - (2) $a_1^{[1]} = \sigma(z_1^{[1]})$

> 💡 $a_i^{[l]}$
>
> - i : 층 안의 노드 번호
> - l : 층 번호

은닉층이 하나인 신경망 출력값 계산

Given input x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

- 벡터화된 구현
- 마지막 출력 유닛은 로지스틱 회귀와 굉장히 흡사
    - $w^T = w^{[2]}$
    - $b = b^{[2]}$

Vectorizing across multiple examples

> 💡 $a^{[k][i]}$
>
> - k : layer 번호
> - i : 훈련 샘플 번호

for문으로 구현

for $i = 1$ to m,

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

벡터화한 구현

$$X = \begin{bmatrix} | & | & & | \\ X^{(1)} & X^{(2)} & \dots & X^{(m)} \\ | & | & & | \end{bmatrix}$$

- (n_x, m) 벡터

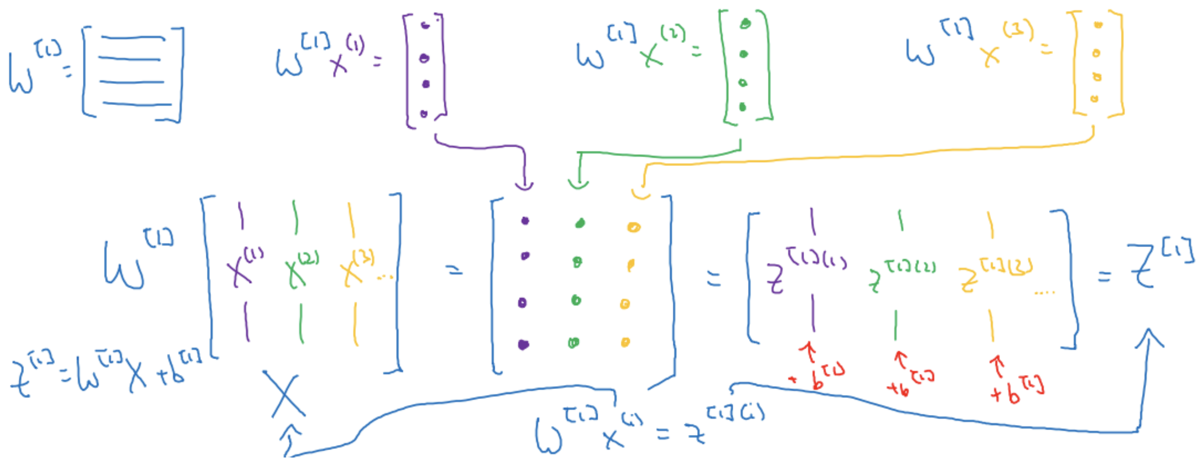
$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ Z^{[1](1)} & Z^{[1](2)} & \dots & Z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

- 가로는 훈련 샘플의 번호
- 세로는 은닉유닛의 번호

Explanation for vectorized implementation

왜 우리가 썼던 등식이 여러 훈련 샘플에 대한 정확한 벡터화인가?



- $Z^{[1]} = W^{[1]}X + b^{[1]}$
- 입력값을 열로 쌓는다면 결과도 열로 쌓인 값이 나옴

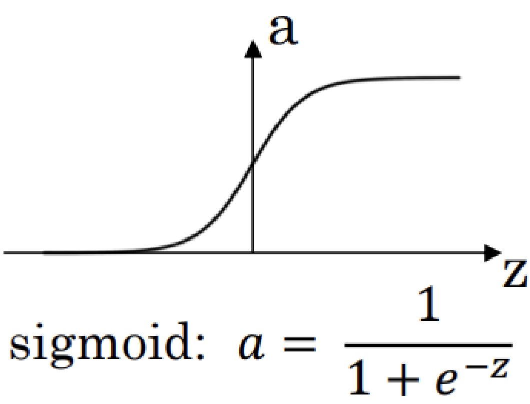
복습

- 한번에 하나의 훈련 샘플에 대해 정방향 전파를 한다면 i가 1부터 m까지 코드 실행
- 비슷한 방법으로 나머지 줄도 위의 코드에 맞는 구현임을 보일 수 있음
- $X = A[0]$
  - 입력특성벡터 $x = a^{[0]}$
  - $x^{(i)} = a^{[0](i)}$

Activation functions

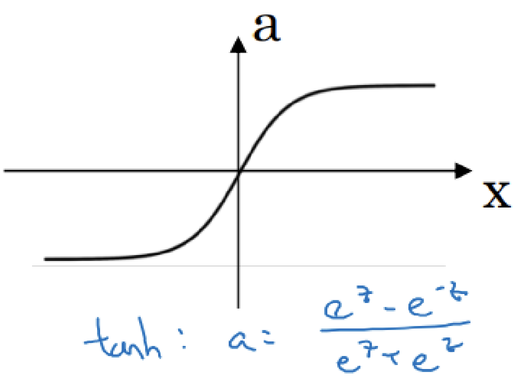
활성화 함수

- sigmoid 함수 대신 다른 함수 g 사용 가능
- sigmoid 함수



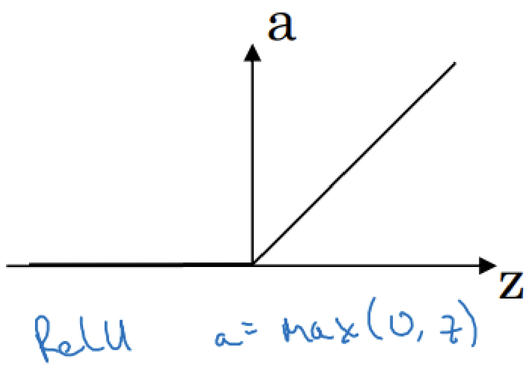
- ◦ 시그모이드 함수 사용 예외
    - ▪ 이진분류 출력층으로 사용
    - ▪ 나머지는 거의 쓰지 않는게 좋음

- tanh 함수

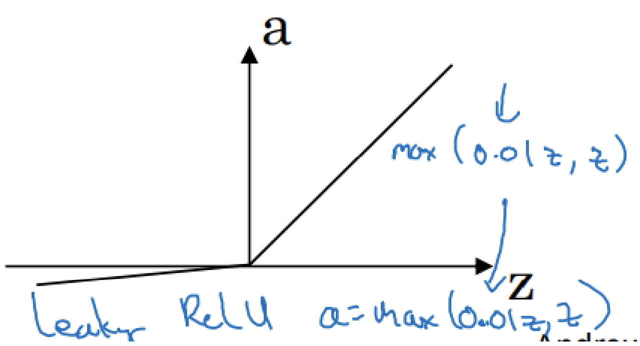


- ◦ sigmoid 함수와 비슷하지만 원점을 지나고 비율이 달라짐
- ◦ 은닉 유닛에 대해 $g(z^{[1]}) = tanh(z)$로 놓는다면 거의 시그모이드보다 좋음
    - ▪ 값이 +1과 -1 사이이기 때문에 평균값이 0에 더 가깝기 때문
        - 데이터의 중심을 0.5대신 0으로 만드는 효과 있음
- ◦ 출력층은 예외
    - ▪ y가 0이나 1이라면 $\hat{y}$도 0과 1사이로 출력하는게 더 좋기 때문
    - ▪ 단점
        - z가 굉장히 크거나 작으면 함수의 도함수가 굉장히 작아짐
        - z가 크거나 작으면 함수 기울기 0에 가까워지고 경사하강법 느려짐

- ReLU 함수



    - 머신러닝 인기 함수: ReLu a=max(0,z)
        - z가 양수일 때 도함수 1
        - z가 음수일때 도함수가 0
        - z가 정확히 0이 된 확률이 굉장히 낮아서 걱정 x
    - Relu 활성화 함수 기본값
    - Relu 단점
        - z가 음수일때 도함수가 0
- Leaky ReLU



    - leaky Relu z가 음수일때 도함수 0 대신 약간의 기울기 존재
    - ReLU, Leaky ReLu의 장점
        - 대부분의 z에 대해 기울기가 0과 매우 다름
        - 더 빠르게 학습 가능

Why do you need non-linear activation functions?

- $\hat{y}$을 입력 특성인 x에 대한 선형 함수로 계산
    - g(z)=z 선형 활성화 함수/항등 함수

$$a^{[1]} = \underbrace{z^{[1]} = W^{[1]}x + b^{[1]}}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = W^{[2]}(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}) + b^{[2]}$$

$$= \underbrace{(W^{[2]}W^{[1]})}_{W'}x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$$

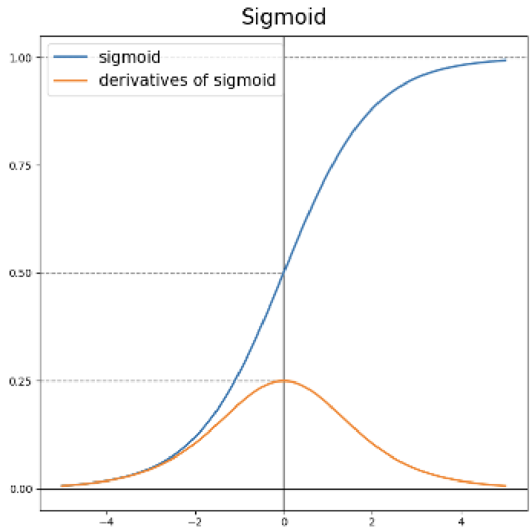
$$= \underline{W'x + b'}$$

- 선형 활성화 함수 사용/ 활성화 함수 없으면 은닉층이 없는것과 동일
    - 두 선형 함수의 조합은 하나의 선형 함수가 되기 때문
    - 은닉 유닛은 비선형 함수 사용해야함
- y가 실수값이라면 선형 활성화 함수 써도 괜찮음
    - $\hat{y}$ -∞ ~∞
    - 출력층일 때

Derivatives of activation functions

- 신경망의 역방향 전파를 구현하려면 활성화 함수 도함수 구해야함

Sigmoid

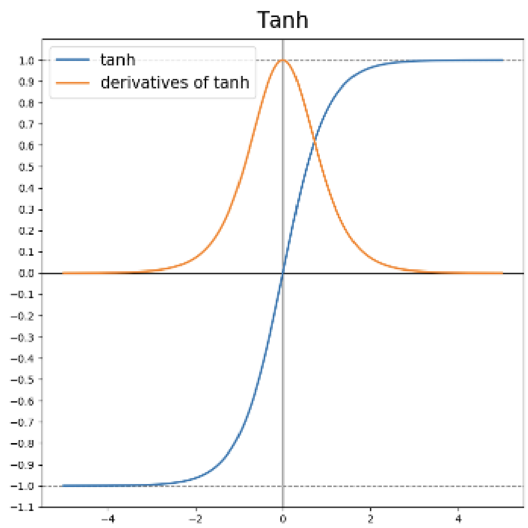


💡 $g\prime(z) = \frac{d}{dz}g(z)$ =slope of g(z) of z

$= g(z)(1 - g(z))$

$= a(1 - a)$

Tanh



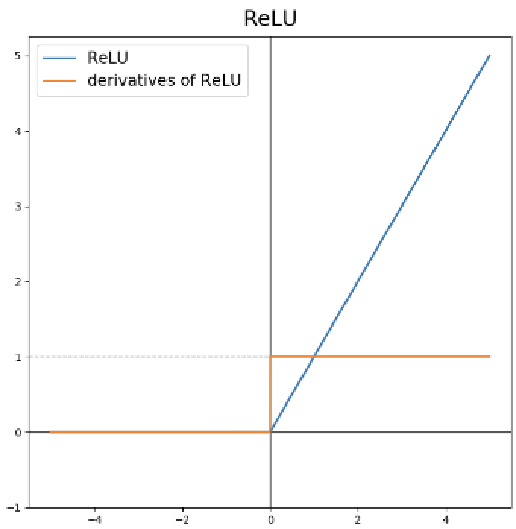
💡 $g\prime(z) = \frac{d}{dz}g(z)$ =slope of g(z) of z

$= 1 - (tanh(z))^2$

$= 1 - a^2$ #a=g(z)이므로

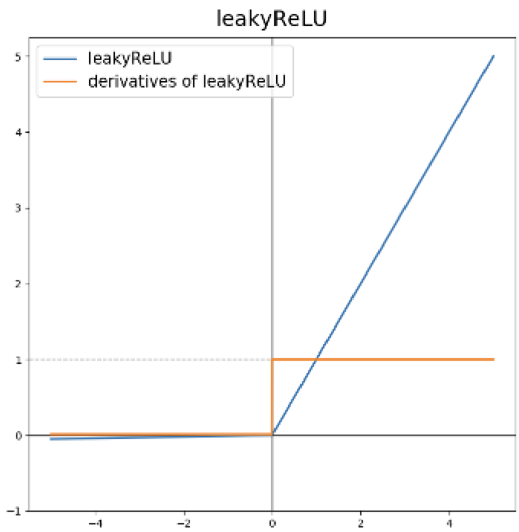
ReLU and Leaky ReLU

- ReLU
    - $g(z) = max(0, z)$



> 💡 $g\prime(z)$
>
> = 0 if z<0
>
> 1 if z≥0

- Leaky ReLU
    - $g(z) = max(0.01z, z)$



> 💡 $g\prime(z)$
>
> = 0.01 if z<0
>
> 1 if z≥0

- z가 정확히 0이라면 도함수가 정의되지 않았지만 둘 중 아무렇게나 설정해도 괜찮음

Gradient descent for neural networks

Gradient descent

Repeat {

Compute predict($\hat{y}$, i=1…m)

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$

$db^{[1]} = \frac{dJ}{db^{[1]}}$

$w^{[1]} = w^{[1]} - \alpha dw^{[1]}$

$b^{[1]} = b^{[1]} - \alpha db^{[1]}$

}

Back propagation:

$dZ^{[2]} = A^{[2]} - Y$

$Y = [y^{(1)}\ y^{(2)} \ldots, y^{(m)}]$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$

$(n^{[1]},)$

$(n^{[2]}, 1)$

$db^{[2]} = \frac{1}{m}$ np.sum$(dZ^{[2]}$, axis=1, keepdims=True)

$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$

element-wise product

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

$db^{[1]} = \frac{1}{m}$ np.sum$(dZ^{[1]}$, axis=1, keepdims=True)

$(n^{[1]}, 1)$ $(n^{[1]},)$ reshape

- keepdims=True
    - 파이썬이 잘못된 2차원 배열 출력 못하게 함
    - reshape를 대신 사용 가능
- $g^{[1]}{}'$: 은닉층에서 사용한 활성함수 도함수

Backpropagation intuition

- 입력값 X에 대한 도함수는 계산할 필요 x
    - 지도학습에서 고정된 값이기 때문
- 역전파를 구현할 때는 차원이 정확하게 일치하는지 확인해야함
- 벡터화

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$\underbrace{dz^{[1]}}_{(n^{[1]},1)} = W^{[2]T} dz^{[2]} * g^{[1]}{}'(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$J(\,..\,) = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

elementwise product

$$\underbrace{dZ^{[1]}}_{(n^{[1]},m)} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]}{}'(Z^{[1]})}_{(n^{[1]},m)}$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

    - $dW^{[2]}$ 1/m 추가 : 비용 함수 J가 1부터 m까지의 손실 함수의 합을 m으로 나눈것이기 때문

Random Initialization

모든 가중치를 0으로 초기화한다면?

- b를 0으로 초기화하는건 괜찮음
- w를 0으로 초기화하면 문제 발생
    - $a_1^{[1]} = a_2^{[2]}$
    - $dz_1^{[1]} = dz_2^{[2]}$

- 두 은닉 유닛이 같은 값으로 초기화 되어 가중치 결과값 항상 같음
- 완전 대칭 : 두 은닉 유닛이 항상 같은 함수 계산
    - 출력 유닛에 항상 같은 영향을 미침
    - 신경망이 얼마나 학습을 하는지와 상관없이 같은 상태가 반복
    - 은닉유닛이 실제로는 1개인 것과 같은 상태

가중치 임의로 초기화

> 💡 $w^{[1]}$=np.random.randn((2,2))*0.01
> $b^{[1]}$=np.zeros((2,1))
> $w^{[2]}$=np.random.randn((1,2))*0.01
> $b^{[2]}$=0

- w 초기화
    - 가우시안 랜덤 함수 사용
    - 작은 값(0.01) 곱해줌
        - w가 너무 크면 z 값이 매우 크거나 작은 상태가 될 수 있음
        - tanh, sigmoid 활성 함수 경사 기울기 낮아서 학습 속도 느려짐
        - 이진 분류 출력 유닛도 시그모이드 함수를 사용하므로 z값이 너무 크거나 작은 상태가 되면 x
        - 얕은 은닉층이면 0.01 괜찮지만 매우 깊으면 다른 수 선택해야 할수도 있음
- b 초기화
    - b는 0으로 초기화해도 괜찮음
    - w를 이미 다른 값으로 초기화해서 대칭 회피 문제가 해결