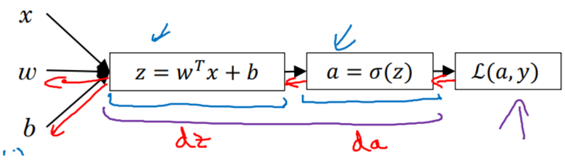
얕은 신경망 네트워크

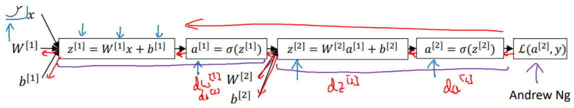
개요

로지스틱 회귀에서는 z와 a를 한 번씩 계산했지만, 신경망에서는 여러 번 계산함.

- 로지스틱 회귀

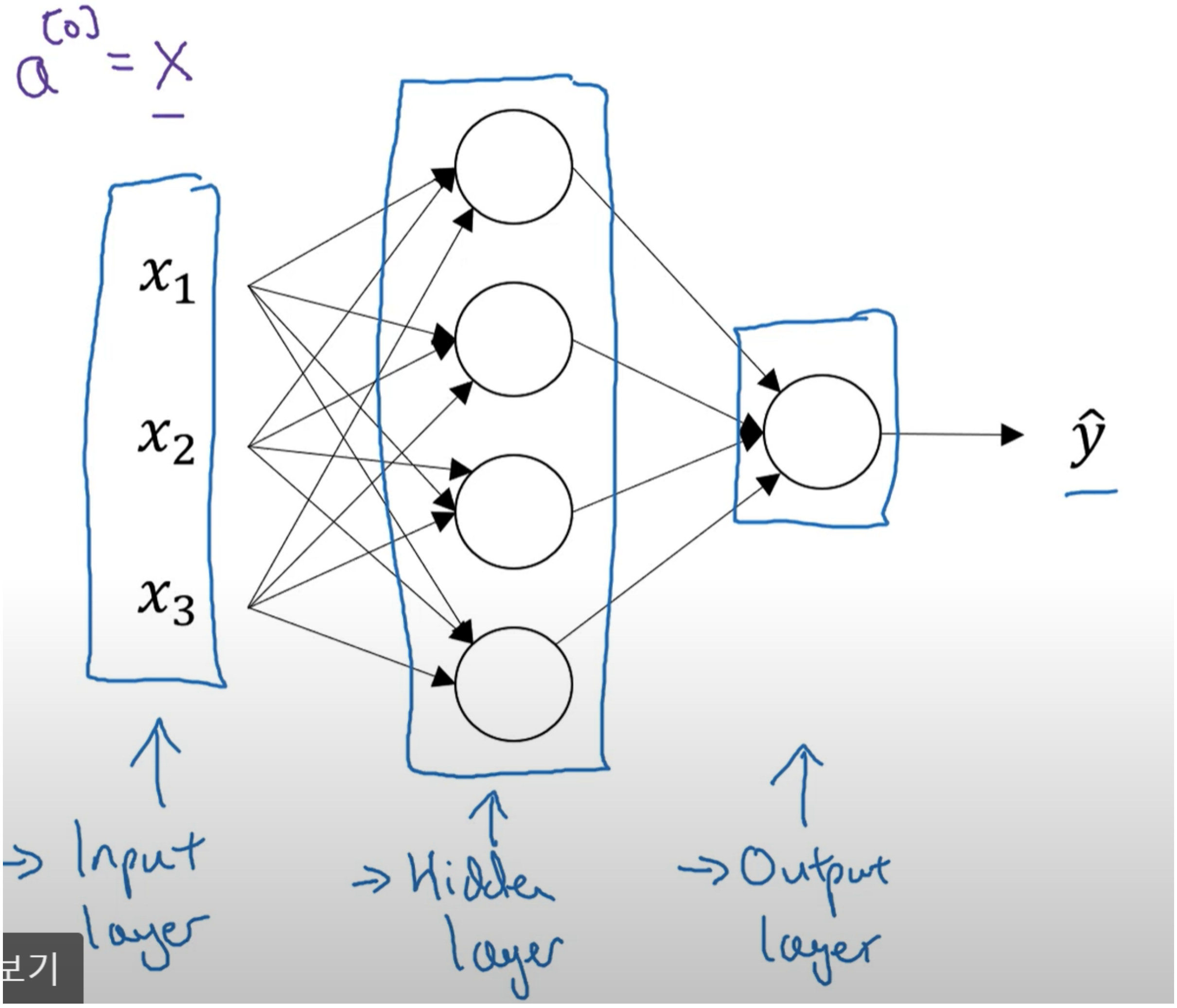


- 신경망

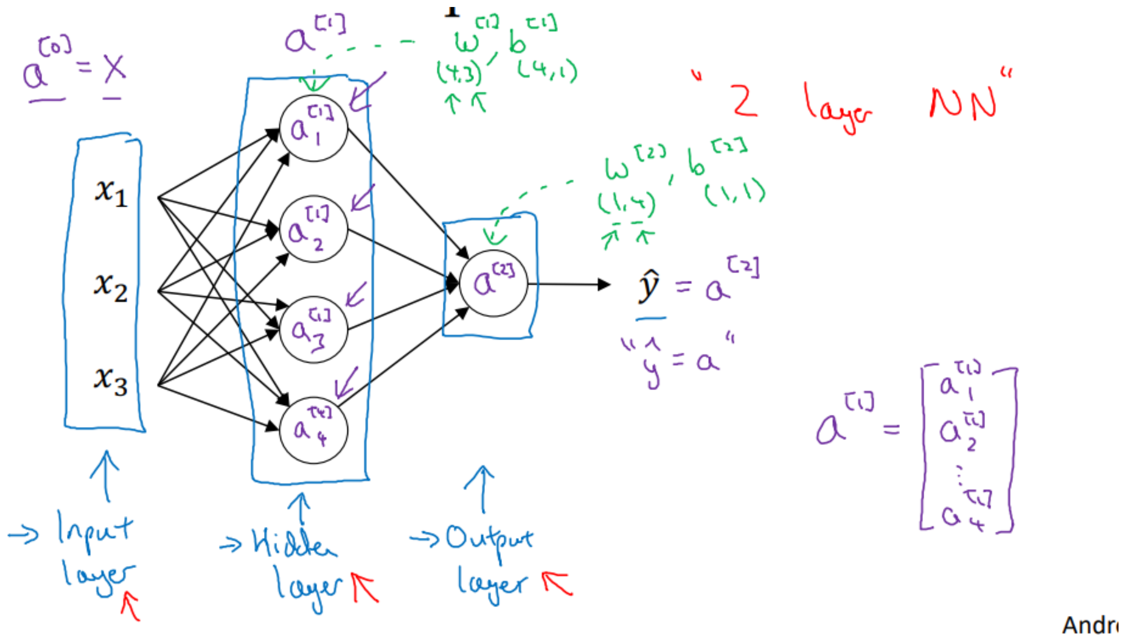


신경망 네트워크의 구성

예) 은닉층이 하나인 신경망



- 입력층
- 은닉층
    - 훈련 세트는 (입력, 출력)으로 이루어져 있기 때문에, 훈련 세트에서 볼 수 없다는 뜻
- 출력층
    - 노드 하나로 이루어짐



값 표기법

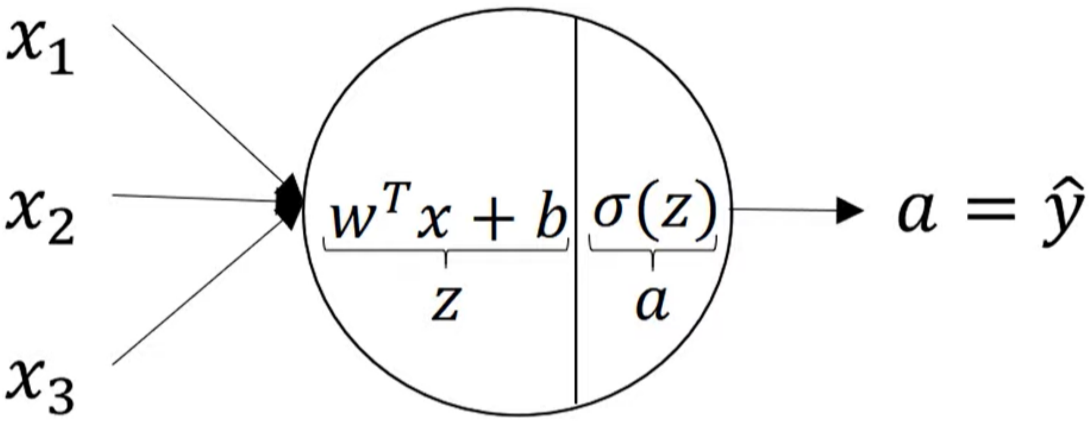
- **a**: 활성값. 신경망의 다음 층으로 전달해주는 값
- 위첨자: 생성된 층의 위치
- 아래첨자: 해당 층에서의 노드 번호
- 층을 셀 때는 입력층은 세지 않는다. 예) 이 예시는 2 layer NN

각 층에는 연관된 매개변수가 있다.

- 예) 첫 번째 은닉층은 매개변수 w^[1]과 b^[1]에 관련됨
- w: (4, 3) 행렬 - 은닉 노드 4개, 입력 특성(n) 3개
- b: (4, 1) 벡터 - 은닉 노드 4개, 출력 노드 1개

신경망 네트워크 출력의 계산

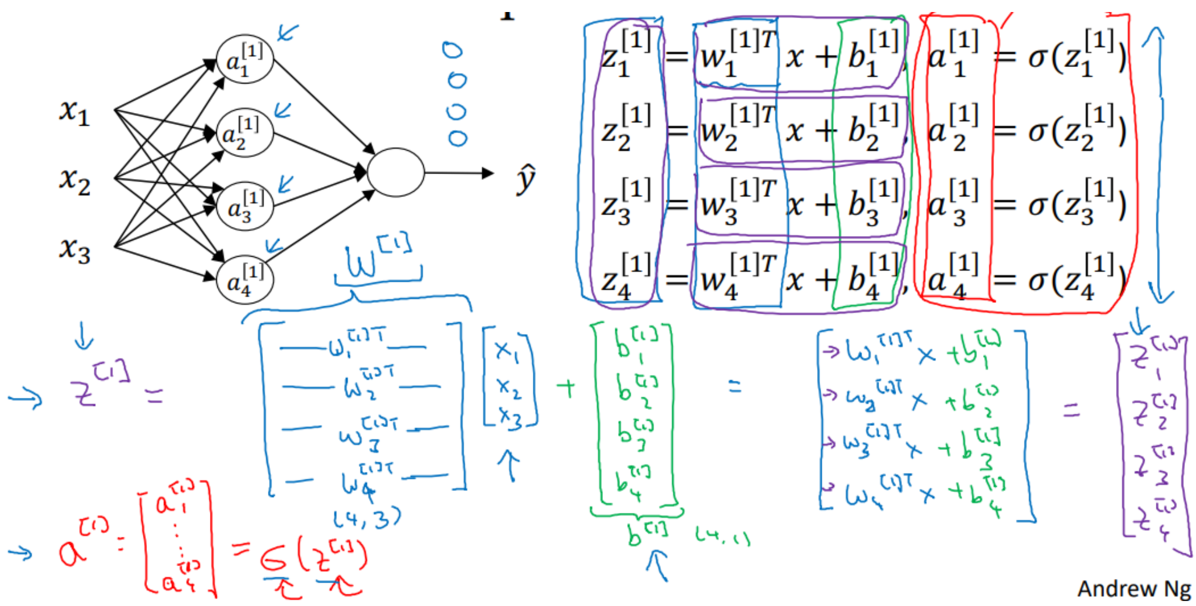
신경망의 노드는 아래 계산을 수행한다.



- z를 계산한 후, z를 활성화함수에 통과해 a를 계산한다.

매 층마다, 매 노드마다 z와 a를 계산해야 하므로 반복적인 구현이 필요하다. 지난 주에 배운 벡터화를 이용하면, 식이 아래 손글씨와 같이 바뀐다.

- Tip: 한 층에 노드가 여러 개이면 세로로 쌓는다.



Given input x:

$$\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}}\underset{(3,1)}{\overset{a^{[0]}}{\cancel{x}}} + \underset{(4,1)}{b^{[1]}}$$

$$\underset{(4,1)}{a^{[1]}} = \sigma(\underset{(4,1)}{z^{[1]}})$$

$$\underset{(1,1)}{z^{[2]}} = \underset{(1,4)}{W^{[2]}}\underset{(4,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}}$$

$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

→ 벡터화하면 이렇게 한꺼번에 계산 가능!

많은 샘플에 대한 벡터화

$a^{[2](i)}$

example i

layer 2

[j]: 층 번호

(i): i번째 훈련 샘플

for문을 이용한 계산

```
for i = 1 to m:
```

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

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

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

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

훈련 샘플 반복 삭제

- 단일 훈련 샘플 x가 아닌 전체 훈련 샘플 X로 벡터화

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

z와 a도 벡터화

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

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

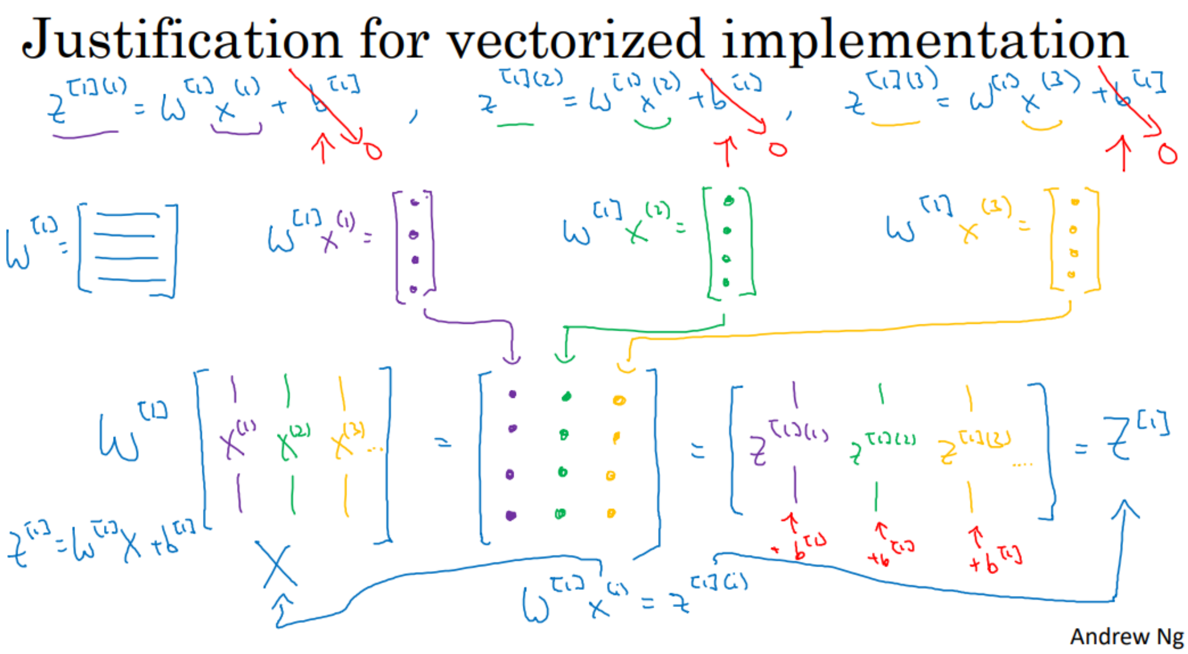
And

- 가로: 훈련 샘플 번호

- 세로: 신경망의 노드들 (A)
  입력층의 노드들 (Z)

예) 맨 위에서 가로로 움직이면 은닉 유닛은 첫 번째로 고정, 훈련 샘플이 바뀜

벡터화 구현에 대한 설명



첫 훈련 샘플에 대해 Z^1 = W^[1] x^(1) + b^[1]을 계산하고,

두 번째 훈련 샘플에 대해선 Z^1 = W1 x^(2) + b^[1]을 계산하고,
세 번째 훈련 샘플에선 Z^1 = W^1 x^(3) + b^[1]을 계산해야 한다.

(b는 0이라고 생각)

W^[1]: 행이 여러 개인 행렬, x^(1)은 상수 → W^[1]와 x^(1)의 곱은 열 벡터

훈련 샘플을 모두 쌓아 만든 훈련 세트 X는 x^(1), x^(2), x^(3)를 모두 가로로 쌓아 만든 것

따라서 X와 W를 곱한다면 나오는 결과인 Z도 개별 원소로 계산했을 때의 z가 그대로 열로 쌓인다.

심층 신경망에서 볼 것

$$
\begin{cases}
Z^{[1]} = W^{[1]}X + b^{[1]} \\
A^{[1]} = \sigma(Z^{[1]}) \\
Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} = \sigma(Z^{[2]})
\end{cases}
$$

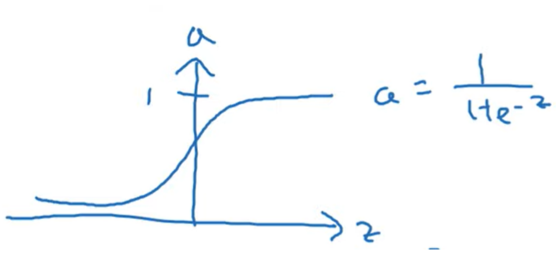
$\leftarrow W^{[1]}A^{[0]} + b^{[1]}$

2-layer NN에서 각 층의 계산은 차수만 다르고 형태가 완전히 같음. 심층 신경망에서도 이 과정을 반복하는 것뿐임.

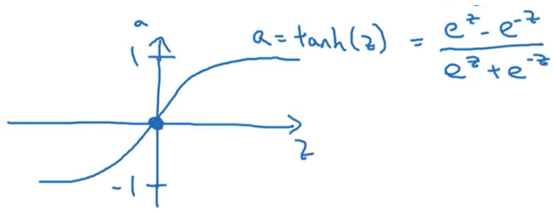
활성화 함수

- 은닉층과 출력층에서 어떤 활성화 함수를 쓸 것인가?

Sigmoid 함수와 tanh 함수



- 범위: 0~1
- 평균값의 중심: 0.5
- 이진 분류의 출력층 외에는 사용 X



- 범위: -1~1
- 평균값의 중심: 0
- 원점 대칭
- 은닉층에 두면 대부분 시그모이드보다 좋음

→ 다음 층의 학습을 더 쉽게 함

교수님은 은닉층에서 시그모이드 함수를 거의 쓰지 않고, tanh 함수를 사용하심.

출력층은 예외! 이진 분류에서 y는 0이나 1이기 때문에 0~1 사이로 출력하는 게 좋다.

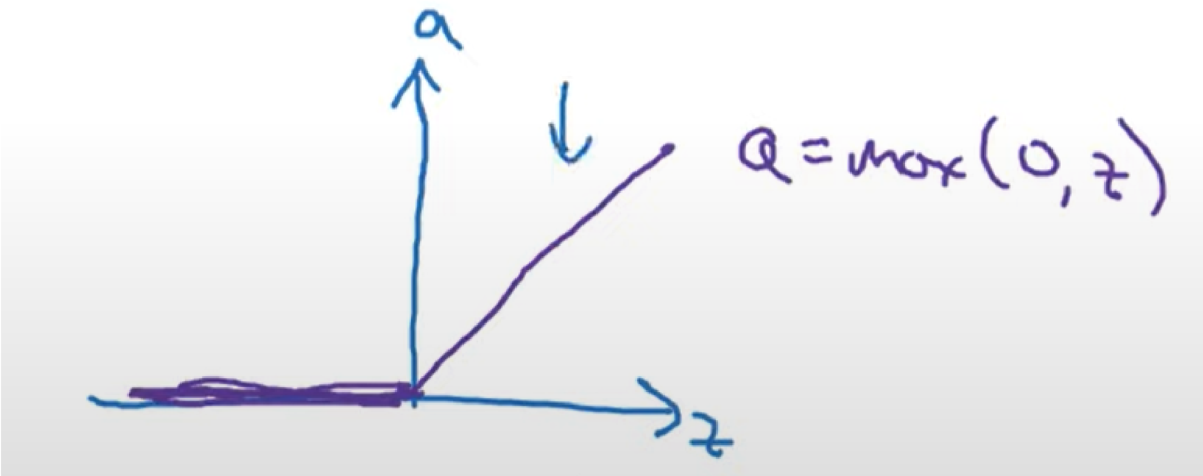
→ **은닉층에선 tanh 활성화 함수를 쓰고 출력층에서는 시그모이드 함수를 쓴다.**

(활성화 함수 g의 위첨자: 다른 층에 다른 활성화 함수가 쓰였다는 걸 보여주기 위함)

시그모이드와 tanh의 단점

z가 굉장히 크거나 작으면, 도함수가 0에 가까워져 경사 하강법 속도가 매우느려짐.

ReLU

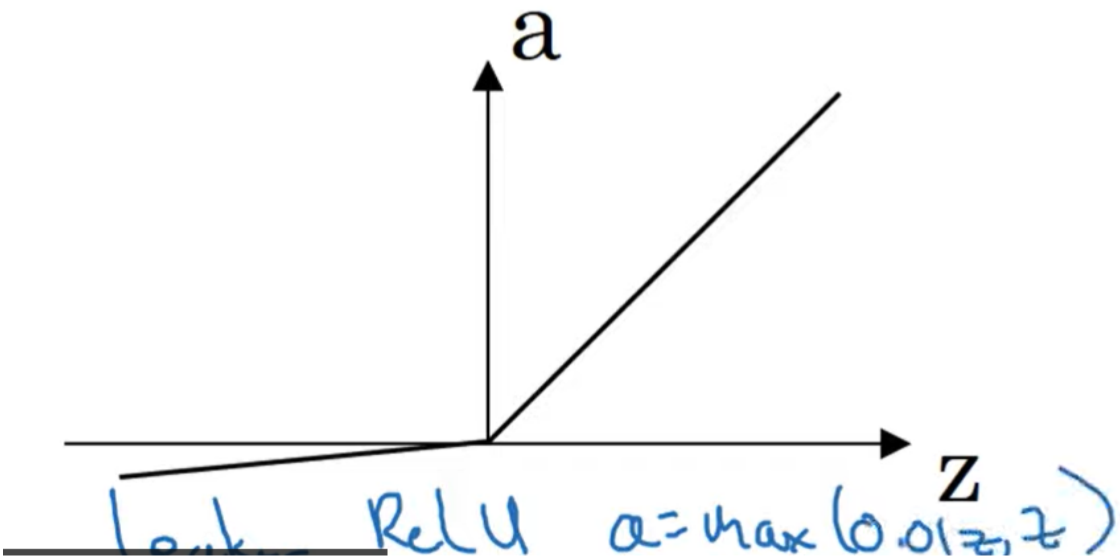


- z > 0: 도함수 = 1
- z < 0: 도함수 = 0
- z가 0일 때는 도함수를 1/0 중 어떤 것으로도 가정해도 됨

→ 이진 분류의 출력층에는 시그모이드 함수 사용

→ 은닉층의 기본값 함수는 ReLU. 대부분의 z에 대해 기울기가 0이 아님. 신경망을 훨씬 빠르게 학습시킬 수 있음.

Leaky ReLU



z가 음수일 때 도함수에 약간의 기울기를 줌

계수는 학습 알고리즘의 변수로 넣을 수는 있지만, 보통 0.01로 쓰는듯

신경망을 구현할 때는 은닉층의 수, 활성화 함수의 종류, 가중치 초기화값 등 여러 가지 선택사항이 있다.

특정 신경망에 어떤 활성화 함수가 좋을지는 가이드라인이나 정답이 없어서, 모두 시도해보고 ㅎ결과가 가장 좋은 걸 선택하면 된다.

왜 비선형 활성화 함수를 써야할까요?

은닉층에 선형 활성화 함수나 항등 활성화 함수를 사용한다면 (예) g(z) = z),

은닉층을 많이 쌓아도 하나의 선형 함수로 표현할 수 있기 때문에 (예) $g(g(g(z)))=z)$

아무런 효과가 없다.

y가 실수 전체이면 **출력층에서** 선형 활성화 함수를 써도 됨

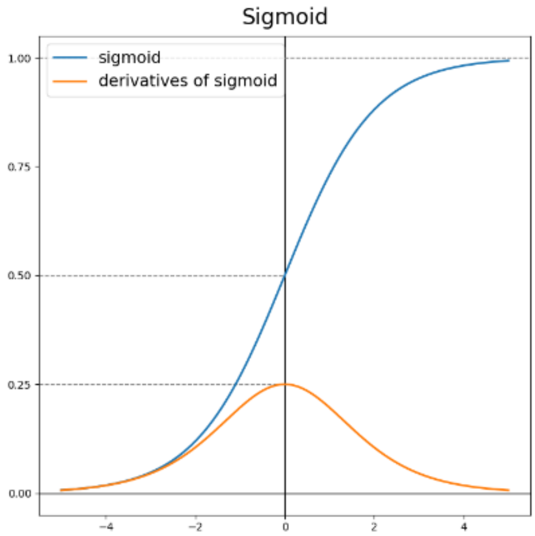
- **선형 함수**란?
  - 출력이 입력의 상수배만큼 변하는 함수
- **항등 함수**란?

- 자기 자신과 같은 값을 대응시키는 함수

활성화 함수의 미분

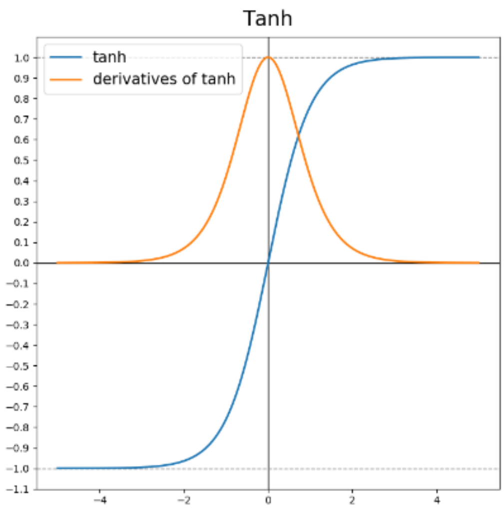
1. 시그모이드

$$g(z) = \frac{1}{1+e^{-z}}$$
$$g'(z) = \frac{d}{dz}g(z) = g(z)(1-g(z))$$



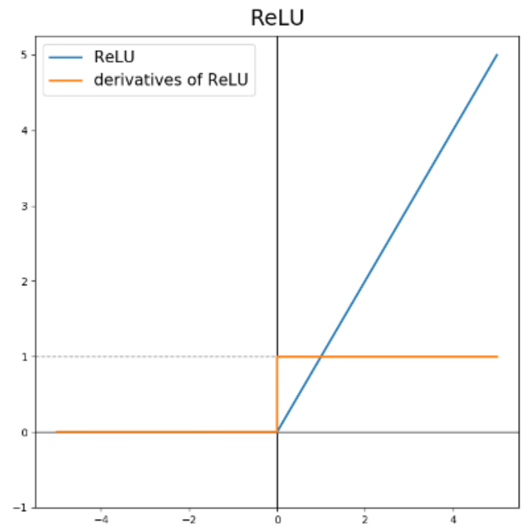
2. tanh

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$
$$g'(z) = 1 - (g(z))^2$$



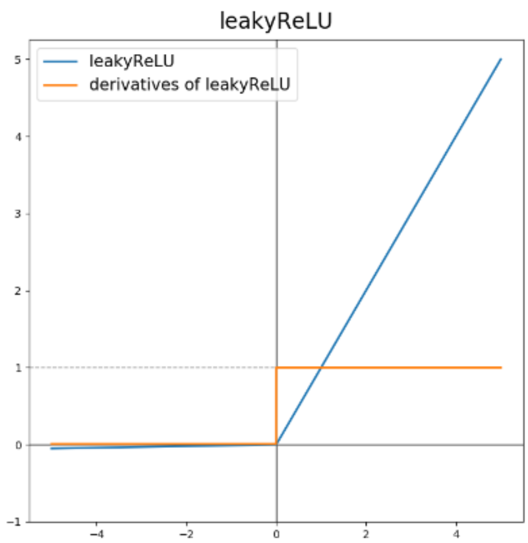
3. ReLU

$$g(z) = max(0, z)$$
$$g'(z) = 0 \quad \text{(z < 0인 경우)}$$
$$g'(z) = 1 \quad \text{(z >= 0인 경우)}$$



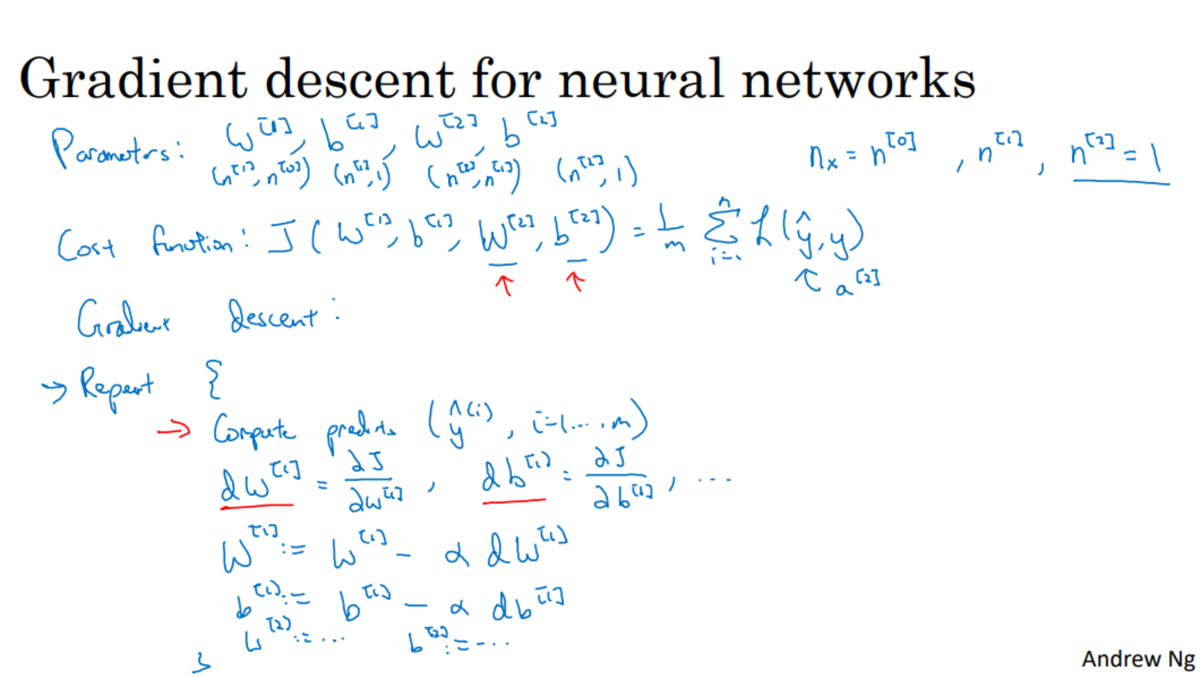
4. Leaky ReLU

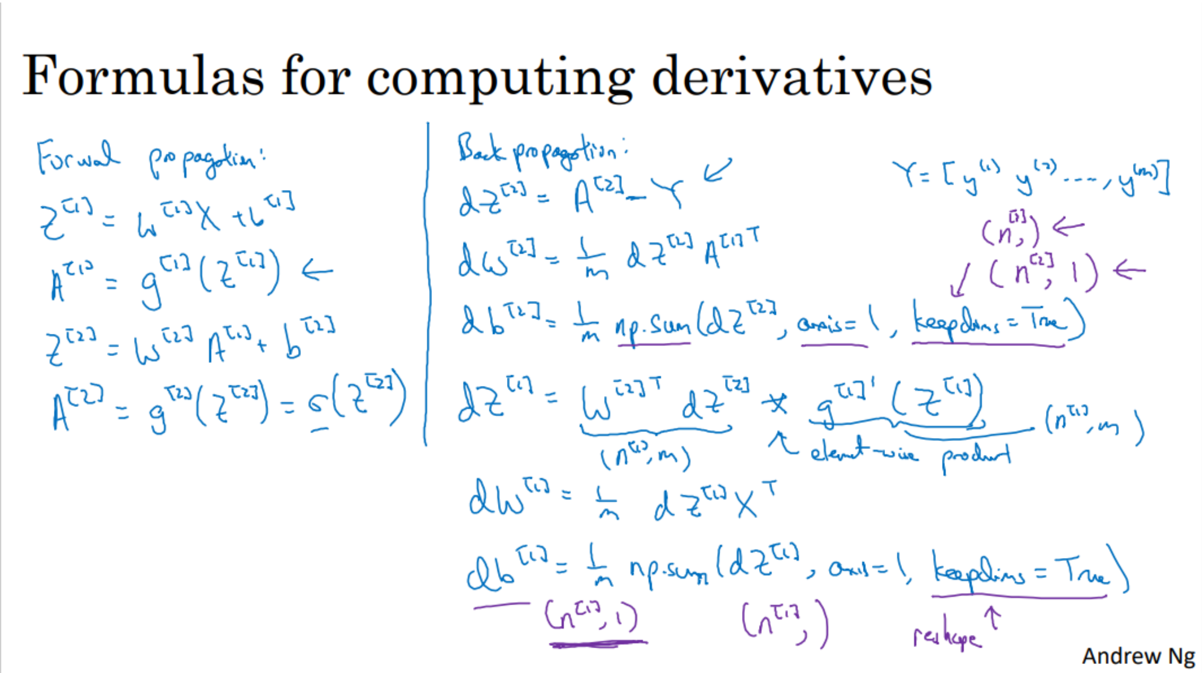
$$g(z) = max(0.01z, z)$$
$$g'(z) = 0.01 \quad \text{(z < 0인 경우)}$$
$$g'(z) = 1 \quad \text{(z >= 0인 경우)}$$



신경망 네트워크와 경사 하강법

- 변수들이 수렴할 때까지 반복
- 변수를 0이 아닌 값으로 초기화하는 게 중요 (나중에 설명)



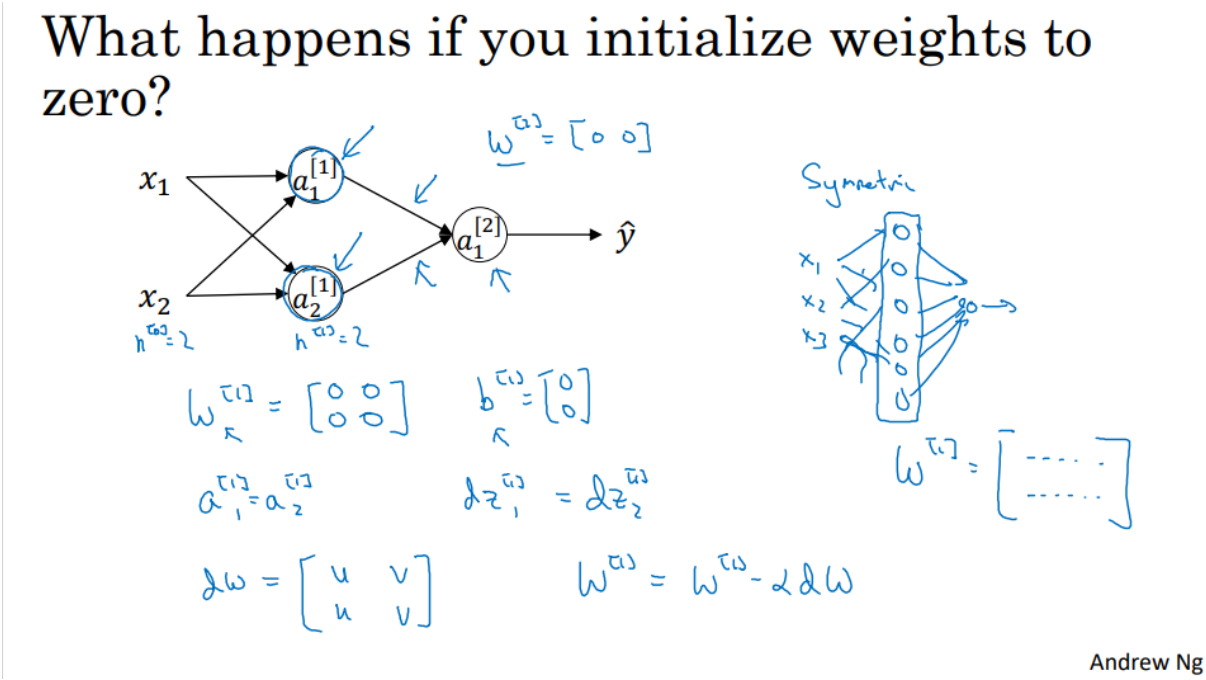


- np.sum 행렬의 어떤 축 방향으로 계산
  - keepdms = True: rank 1 배열을 출력하지 않게

랜덤 초기화

신경망에서 모두 0으로 초기화하고 경사 하강법을 적용할 경우 올바르게 동작하지 않음

b를 0으로 초기화하는 건 괜찮지만, w를 0으로 초기화하면 문제가 됨.



→ 어떤 훈련 샘플에도 a^[1]_1과 a^[1]_2가 같은 값을 가짐. 역전파를 계산할 때 dz의 값 또한 같음.

→ 모든 값을 0으로 초기화한다면 모든 은닉 유닛은 대칭이 되고 경사 하강법을 얼마나 적용시키는지에 상관 없이 모든 유닛은 항상 같은 함수를 계산하게 될 것

→ 변수를 임의로 초기화하는 것이 중요!

- w^[1]을 np.random.randn으로 설정 → 가우시안 랜덤 변수 생성
- 이 값에 0.01과 같이 작은 수를 곱해줘서, 굉장히 작은 임의의 수를 초기값으로 만들어낼 수 있음.
- w가 큰 값을 가지는 경우에 z도 굉장히 큰 값을 가질 수 있는데, tanh와 시그모이드에서 양끝값의 기울기는 0에 수렴하기 때문에 학습 속도가 느려질 수 있음.