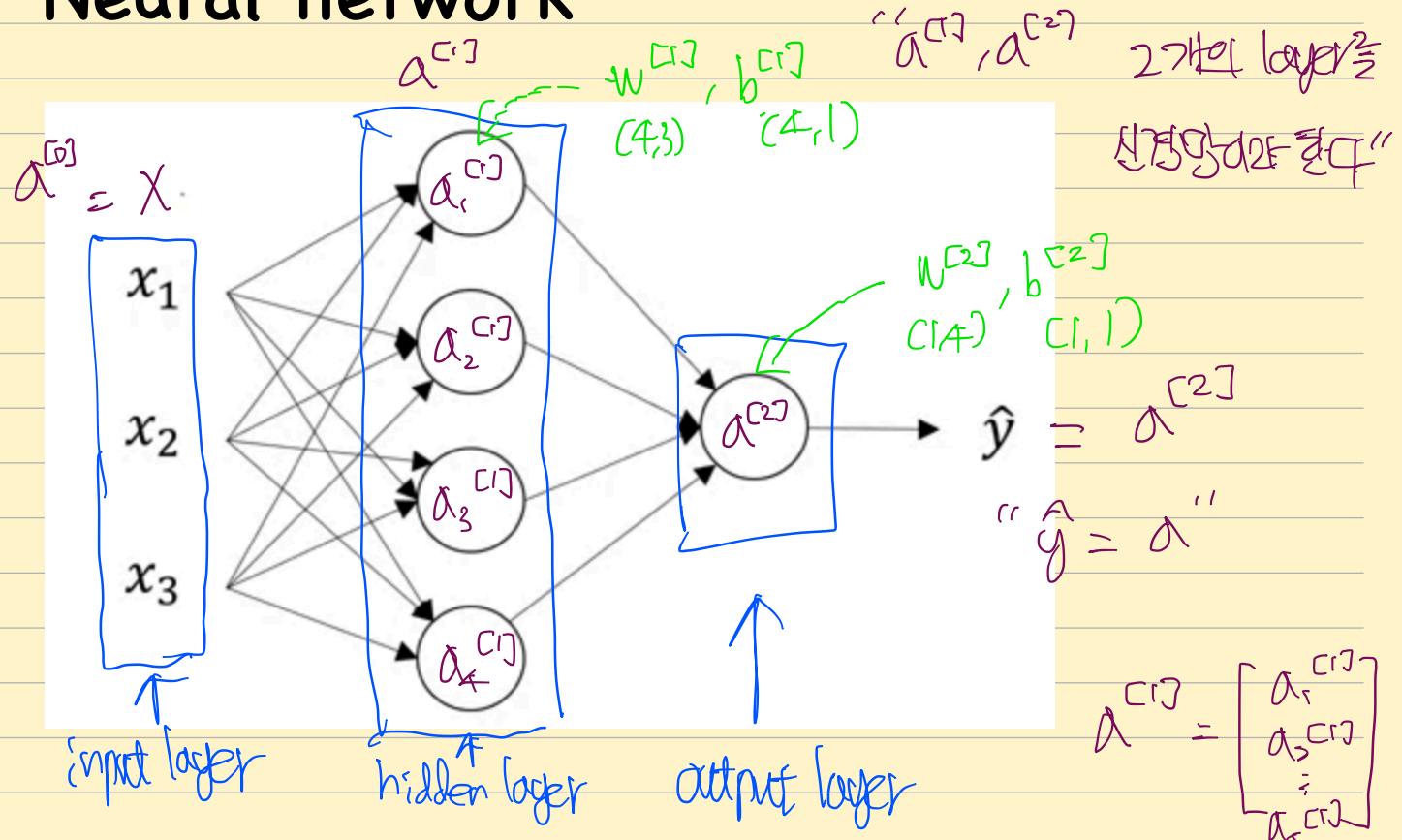


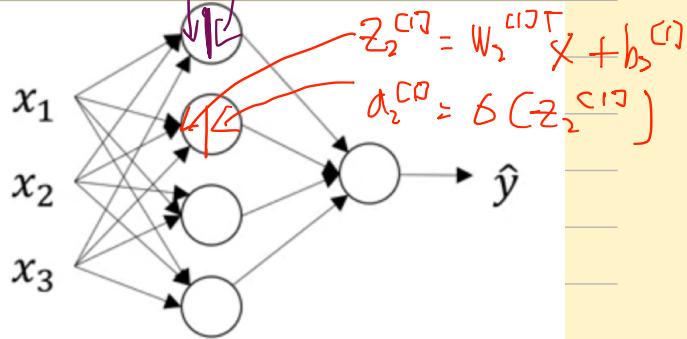
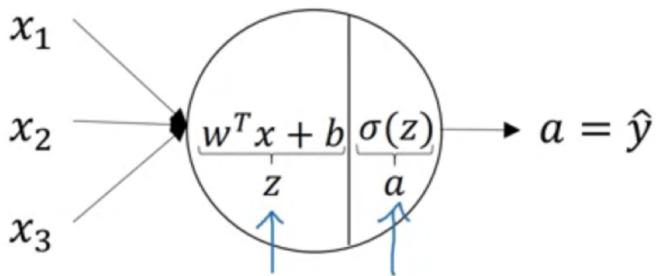
Neural network



- 위 그림의 neural network는 2 layer network이다.
input layer는 포함하지 않는다.
- hidden layer는 결과값을 위한 중간계산은 하지만 그 계산 값이 무엇인지 알기 때문에 hidden layer가 청한다.
- a^2 은 2 번째 layer의 모든 a 로 input의 4인 4차원 벡터이다.
- 2 번째 layer의 모든 w 를 dot transpose한 것을 w^T
2 번째 layer의 모든 b 를 b^T
- 선형변환을 위해 한 패션트로이 들어오는 input의 개수가 3개라면
 w 도 같은 수만큼의 element가 필요 \rightarrow 따라서 w^T 은 4×3 이다.
- b 는 scalar이다, 2 번째 layer의 패션트로이 개수이므로 4×1 벡터를 갖는다.

$$a_i^{[l]} \leftarrow \text{node in layer}$$

$$\begin{aligned} z_1^{[l]} &= w_1^{[l]T} x + b_1^{[l]} \\ a_i^{[l]} &= \sigma(z_1^{[l]}) \end{aligned}$$



$$z = w^T x + b$$

$$a = \sigma(z)$$

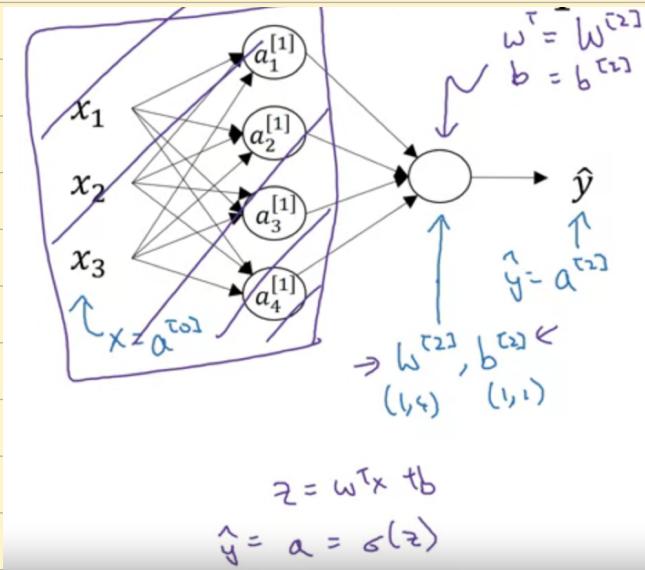
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$\underbrace{\begin{bmatrix} \dots & w_1^{[1]T} & \dots \\ \dots & w_2^{[1]T} & \dots \\ \dots & w_3^{[1]T} & \dots \\ \dots & w_4^{[1]T} & \dots \end{bmatrix}}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$



Given input x :

$$\begin{aligned} \rightarrow z^{[1]} &= W^{[1]} x + b^{[1]} \\ \rightarrow a^{[1]} &= \sigma(z^{[1]}) \\ \rightarrow z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\ \rightarrow a^{[2]} &= \sigma(z^{[2]}) \end{aligned}$$

- 2 번째 layer 를 보면, 표기법은 하단의 input이 4개이므로.

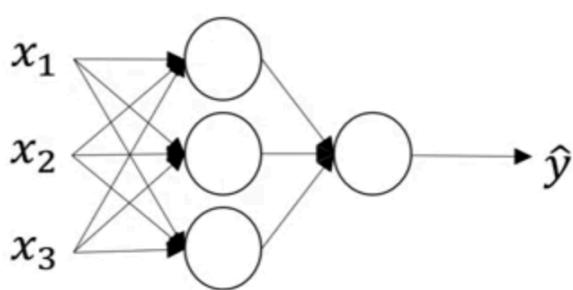
$(4, 1)$ 이고, 이를 transpose 하면 $W^{[2]} = (1, 4)$ 이다.

- 따라서 $a^{[1]}$ 이 $(4, 1)$ 로 쓰여야 되므로 $W^{[2]} a^{[1]}$ 은 $(1, 1)$ 임.

- $b^{[2]}$ 역시 $(1, 1)$ 이고 $z^{[2]}$ 는 $(1, 1)$.

- $z^{[2]}$ 는 non-linear 특성을 가진 $a^{[2]}$ 역시 $(1, 1)$

Activation function



Given x :

$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ \rightarrow a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ \rightarrow a^{[2]} &= \sigma(z^{[2]}) \end{aligned}$$

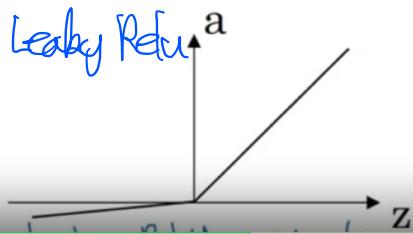
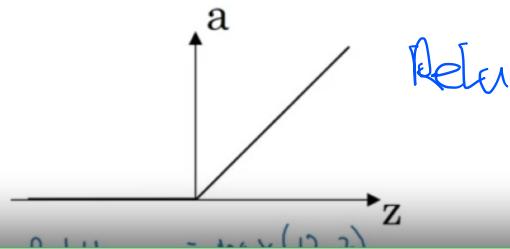
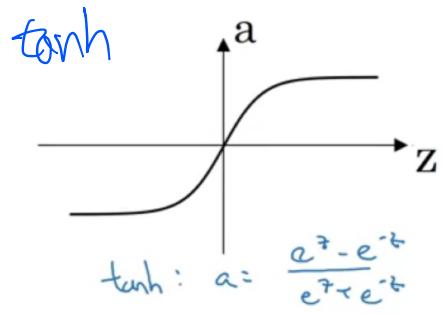
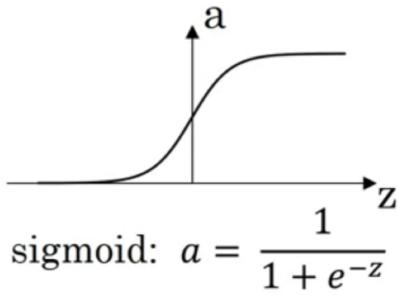
logistic regression에서
non-linear transformation인
sigmoid 향수

Activation function이

해당함.

- Activation function은 미분만 가능하다면, 즉 gradient 계산이 가능하다면 non-linear function을 사용하는 경우이다.
- Sigmoid 향수는 주로 output layer에만 사용되고, hidden layer에서는 잘 사용되지 않는다.

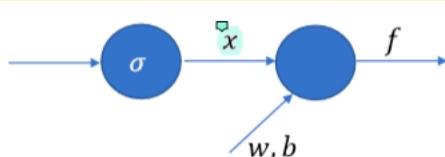
Pros and cons of activation functions



Sigmoid

- Sigmoid는 [0,1] 범위에 a를 분포시킨다.
- 하지만 exponent가 높기 때문에 계산시간이 오래걸리고,
그 뿐만 아니라 gradient가 매우 작아지는 vanishing gradient 문제가 있다.
따라서, Sigmoid의 결과는 zero-centered가 아닌 단점 존재.

- $f = \sum w_i x_i + b$
- $\frac{df}{dw_i} = x_i$
- $\frac{dLoss}{dw_i} = \frac{dLoss}{df} \frac{df}{dw_i} = \frac{dLoss}{df} x_i$



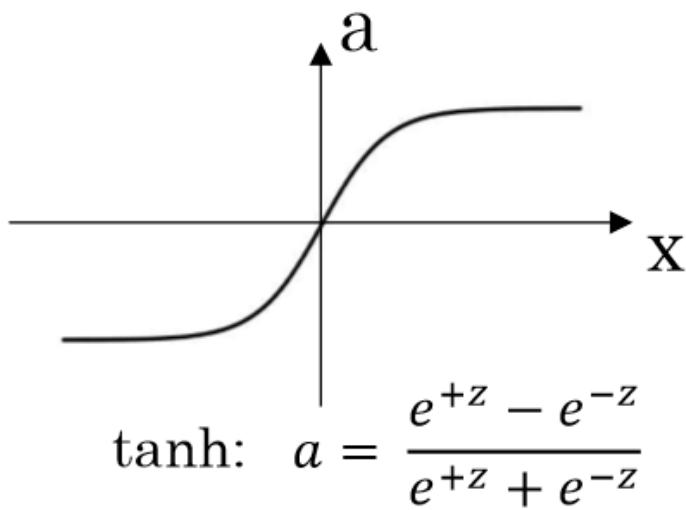
- Sigmoid를 통과한 output들은 zero-output이 아니라 전부 양수인가 때문에 다음과 같은 문제가 존재한다.
- Sigmoid를 통과하여 나온 x가 전부 양수가 되면서 모든 $\frac{dLoss}{dx}$ 의 부호가 $\frac{dLoss}{df}$ 의 부호에 따라 정해지게 된다.

- f 는 scalar 이기 때문에 $\frac{\partial \text{loss}}{\partial f}$ 도 항상 하나의 scalar 값으로 결정되면서 모든 $\frac{\partial \text{loss}}{\partial w}$ 가 양수 혹은 음수 하나의 부호를 가지게 된다.
- w 가 (m)개는 (m) 개는 전부 같은 부호를 가짐.
- 이를 w 값을 m 번적으로 나오면서 아래와 같이 지그재그 형식으로 learning 하게 된다.



이러한 경지 매우 비효율적이기 때문에
sigmoid는 output layer가 아닌
hidden layer에서 거의 쓰이지 않음.

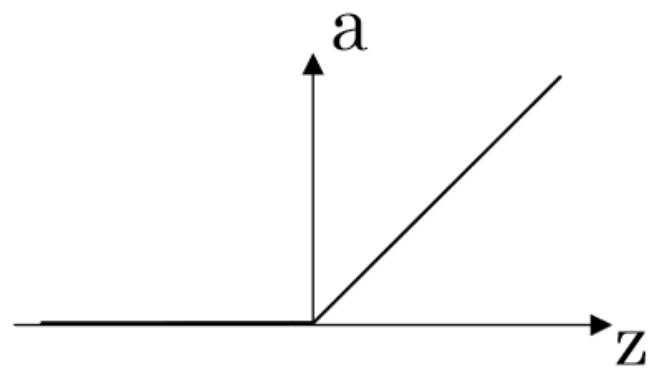
• tanh (Hyperbolic Tangent)



- tanh는 sigmoid가 shift된
함수이다.
- tanh는 [-1, 1] 사이의 값을
output으로 가지기 때문에,
zero-centered이다.

- sigmoid와 다른지 zig-zag dynamic이 없다.
- 따라서 tanh는 sigmoid보다 항상 더 좋은 결과를 낸다.
- 하지만 exponent 가 2개 있기 때문에, 시간이 오래걸리는 expense다.
방법,
- 또한 크가 너무크거나 작으면 기울기가 0에 수렴하기 때문에
vanishing gradient의 문제가 있다.

③ ReLU (Rectified Linear Unit)



$$\text{ReLU: } a = \max(0, z)$$

- ReLU는 most common activation function이다.
- 음수값은 전부 0으로 만들고 음수가 아니면 그대로 놔둔다.
- ReLU는 그가 음수일 때 vanishing gradient 문제가 있다.
- ReLU를 activation function으로 사용하는 이유는 쓸모있는 신호만 남겨두고, 나머지는 제거하기 때문이다.

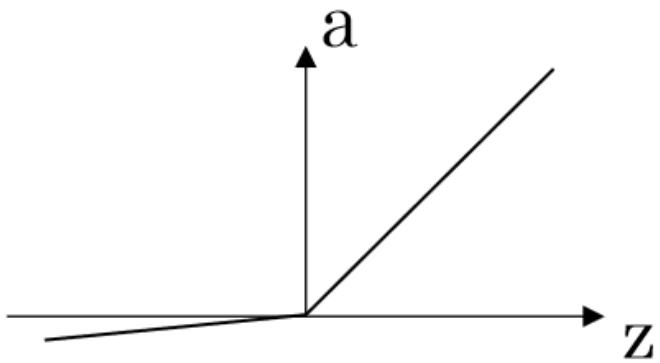
신호만 남겨두고, 나머지는 제거하기 때문이다.

⇒ ReLU를 통과하면 상대적으로 중요한 정보만 남는다.

⇒ 좋은 신호는 양수로 보내고 안 좋은 신호는 음수로 보내는 way to learning하기 쉽다.

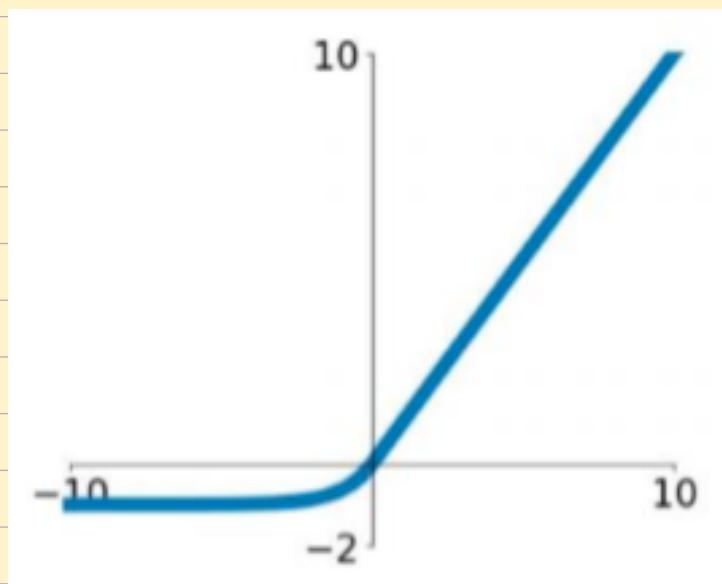
- ReLU는 zero-centered가 아님 때문에 zig-zag 문제도 어느정도 있지만, 계산이 험첨하게 바르고 꼴로 ~~있~~는 정보를 차단할 수 있기 때문에 자주 사용된다.
- 한수마다 경계점이 있고, 막상 사용하기 전까지 성능이 좋은지 결과를 알 수 없지만, ReLU는 많은 경우에 좋다고 알려져 있다.

• Leaky ReLU



$$\text{Leaky ReLU: } a = \max(0.01z, z)$$

- ReLU의 vanishing gradient를 해결하기 위해 고안된 activation function
- ELU (Exponential Linear Unit)



$$f(x) = \begin{cases} x & (\text{if } x > 0) \\ x(\exp(x) - 1) & (\text{else}) \end{cases}$$

- ReLU의 장점을 모두 가지고 있다.
- Leaky ReLU의 경우, 0은 값의 gradient를 가지기 때문에 이상한 값을 원하는가 하면 가지 못하는 단점 존재.
- 이를 해결하기 위해 고안된 function, 매우 expensive 한 편.

Why do we need Non-linear activation function?

activation function은 Non-linear인가 아닐 [linear을 썼다고 징]

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

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

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

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

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

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

$$= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]}$$

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

$$= W'x + b'$$

- linear function은 activation function을 사용하면 final output이 단순한 linear 형태를 가진다.

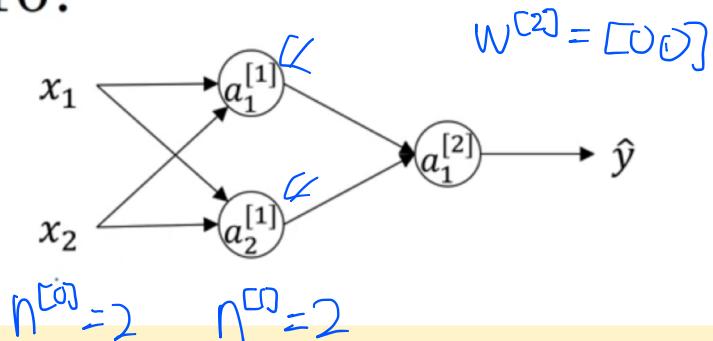
- 결론적으로, neural network가 N개의 node는 W개의

input에 대해 딱 한 번 linear transformation 한 것과 같기 때문에 Deep Neural Network을 사용하는のが 쉽다.

Random initialization

- 신경망을 빠르게 할 때, 가중치를 무작위로 초기화하는 것이 좋다.
- Logistic Regression의 경우, 가중치를 0으로 초기화해도 관계 없다.
- 하지만, 모든 parameters에 대한 가중치를 0으로 초기화한 다음 gradient descent를 적용하면 작동하지 않을 것.

What happens if you initialize weights to zero?



$W^{[1]} = [0 \ 0]$ 를 초기화 했더니 가중, $b^{[1]} = [0]$

$$\alpha_1^{[1]} = \alpha_2^{[1]}$$

$$d\alpha_1^{[1]} = d\alpha_2^{[1]}$$

$$W^{[0]} = W^{[1]} - \alpha dW$$

∴ 1. All hidden units will be completely identical (symmetric)
- compute exactly the same function.

2. On each gradient descent iteration all the hidden units will always update the same.

- 이러한 문제를 해결하려면, 무작위로 초기화 한다.

$$W^{[1]} = np.random.randn(2, 2) \times 0.01$$

$$b^{[1]} = np.zeros(2, 1)$$

왜 0이 아닌 0.01 같은 작은 값을 쓰는가?

⇒ sigmoid나 tanh에서 가중치가 너무 크면 $z^{[1]} = W^{[1]}X + b^{[1]}$

에 따라 너무 큰 z값을 가지게 되기 때문에 작은 값을
필요로 한다.

만약 큰 z값을 가지게 된다면 sigmoid나 tanh 같은 activation
function에서 saturation되어 학습 속도가 느려진다.

신경망 내에서 sigmoid나 tanh가 사용되지 않는다면 이 같은
문제가 가지 않는다.