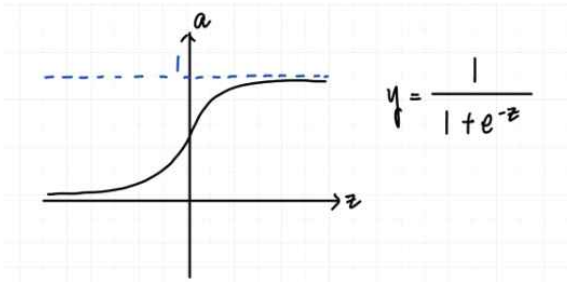


[Week 4] 1. Activation Function

우리는 신경망을 만들 때 은닉층과 출력층에서 어떤 활성화 함수를 사용할지를 결정해야 한다. 앞에서는 시그모이드 함수를 사용하였지만, 상황에 따라 적절한 활성화 함수가 존재한다. 이번에는 활성화 함수의 종류와 그 특징들에 대해 자세히 알아보자.

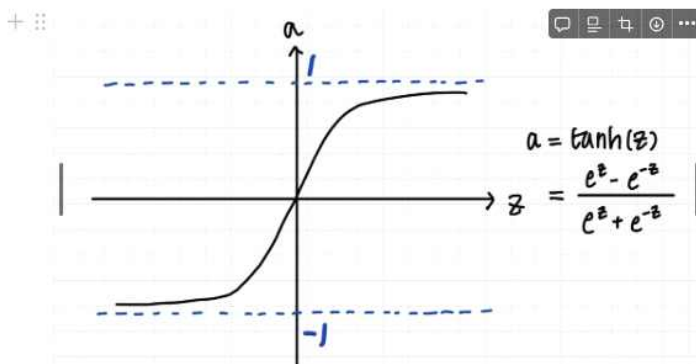
Sigmoid



이진분류의 output layer에는 쓰지 않는 것을 추천한다. Tanh가 거의 항상 더 좋지 때문이다.

*이진분류의 output layer에는 sigmoid 함수를 사용한다. 다른 경우에는 ReLU가 활성화함수로 많이 사용된다.

Tanh



tanh 함수는 수학적으로 시그모이드 함수를 조금 옮긴 형태인데, sigmoid 함수와는 다르게 원점을 지나고, 범위가 (-1,1)이며, 비율이 달라졌다. 그리고 hidden unit에 대해 활성화 함수는 sigmoid 보다 tanh가 거의 항상 성능이 좋다. 왜냐하면 값이 (-1,1) 사이이므로 평균값이 0에 더 가깝기 때문이다. 학습 알고리즘을 훈련 할 때 평균값의 중심이 시그모이드는 0.5에 가깝지만, tanh는 0에 가깝다.

((더 정확히 이해할 필요가 있다))

** 그러나 출력층의 활성화 함수라면 이야기가 다르다. 출력층의 값은 -1과 1 사이의 값보다 0과 1 사이의 값으로 출력하는 것이 좋지 때문이다.

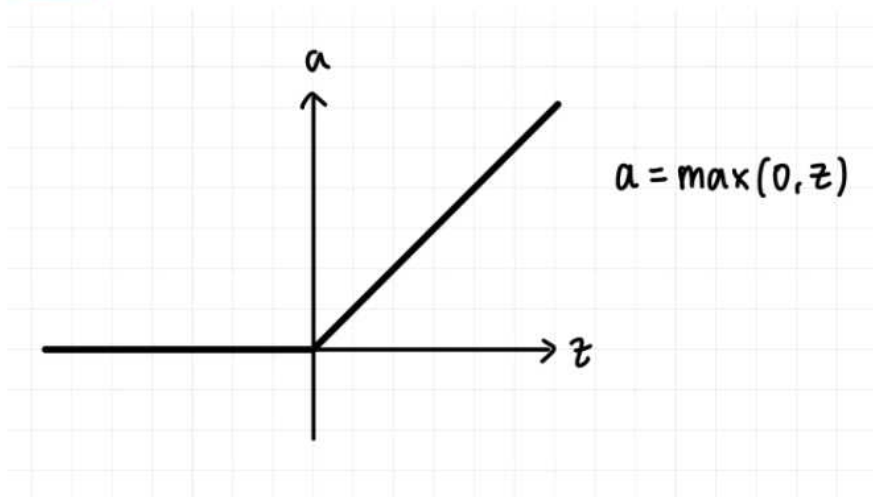
** 서로 다른 층에는 서로 다른 활성화 함수를 적용할 수 있다.

[sigmoid 와 tanh 함수의 단점]

z값($W^T X + b$)이 굉장히 크거나 작으면 함수의 미분계수(기울기)가 굉장히 작아진다는 것이다. 기울기가 0에 가까워지기 때문에, 경사하강법을 적용했을 때, 파라미터 값의 변화가 작게 됨. 따라서 학습속도가 느릴 수 있다.

$$w : w - \alpha \frac{dJ(w, b)}{dw}$$

ReLU

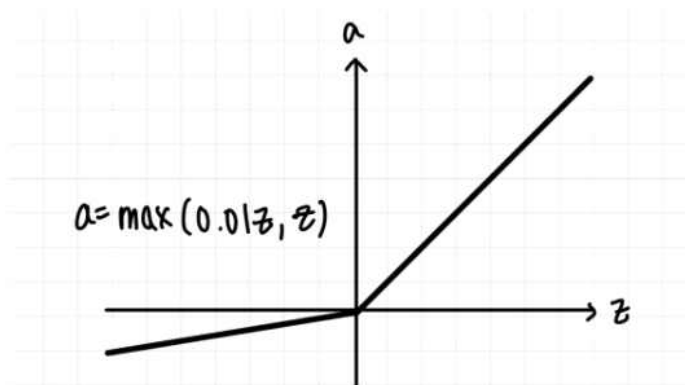


0과 z 값 중 큰 값을 취하는 함수이다. z 값이 양수이면 미분계수가 1이고, z 값이 음수이면 미분계수가 0이다. 엄밀하게는 z 값이 0일때의 미분계수가 지정되지는 않았으나 컴퓨터에서 구현하면 z 가 정확히 0이 될 확률은 매우 낮으므로 걱정하지 않아도 된다. 또한 실제로 대부분의 hidden unit의 z 값은 0보다 크기 때문에 실제로 ReLU는 잘 동작한다.

왜? z 값이 대부분 0보다 크지?

이는 대부분의 상황에서는 사실일 수 있지만, ReLU의 한계로 인해 발생할 수 있는 'dying ReLU' 문제를 고려하지 않은 것이다. 이 문제는 네트워크의 일부 뉴런이 0 이하의 값만을 나타내어 기울기가 계속 0이게 되면, 해당 뉴런의 가중치는 더이상 업데이트가 이루어지지 않아 학습과정에 기여할 수 없게 된다. 이는 특히 네트워크의 초기 가중치가 잘못 설정되었거나, 매우 큰 학습률을 사용할 때 발생할 수 있습니다.

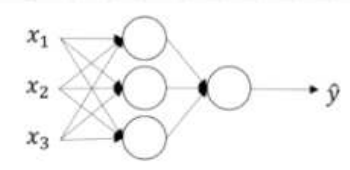
+ leaky ReLU



z 가 음수일 때 도함수가 0이 아닌 약간의 기울기를 준 함수이다. 대부분의 z 에 대해 기울기 값이 0이 아니게 되므로, 변화율이 커져 학습속도가 훨씬 빠르게 된다. ReLU와 leaky ReLU의 장점은 대부분의 z 값에 대해 activation function의 기울기가 0과 매우 다르다는 것이다. 이로 인해 신경망을 더 빠르게 학습할 수 있다.

Why NON-LINEAR activation function?

활성화함수 g 를 $g(z)=z$ 라고 해보자. 당연히 이는 선형(여기서는 단순히 항등함수를 예시로 설명하겠다)함수이다. 이러한 함수 g 를 linear activation function이라고 부른다. 선형 활성화 함수를 사용하면 출력값을 선형으로 계산하게 된다. 이를 아래의 수식으로 설명해본다면,



Given x :

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

→

$$\begin{aligned}
 a^{[2]} &= z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\
 &= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} \\
 &= \underbrace{W^{[2]}W^{[1]}}_{W'}x + \underbrace{W^{[2]}b^{[1]} + b^{[2]}}_{b'} \\
 &= W'x + b' \quad (\text{linear})
 \end{aligned}$$

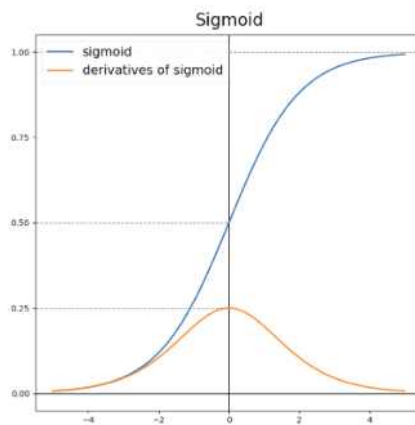
따라서 신경망은 입력의 선형식만을 출력하게 된다. 우리는 앞으로 더 깊은(=레이어가 많은) 신경망을 배우게 될 것이다. 이러한 신경망에서 선형으로만 계산하게 된다면, 층이 얼마나 많은 간에 신경망은 선형으로만 계산하기 때문에 은닉층이 없는 것과 다름이 없게 된다. 우리가 위의 선형 활성화 함수만 존재하는 신경망의 출력층에 sigmoid를 준다면 결국 위의 모델은 로지스틱 회귀모델보다 나아진게 없는 것이다.

두 선형 함수의 조합은 하나의 선형 함수가 되므로, 선형의 은닉층은 쓸모가 없다. 단, 선형 함수를 사용할 때가 있다. 회귀 문제에 대한 머신 러닝을 할 때 사용한다. y 가 실수값이라면(ex. 집값 예측) 선형 활성화 함수를 써도 괜찮을 수 있다. 하지만 hidden unit은 비선형 함수를 대부분 사용해야 한다.

[Week 4] 2. Derivatives of Activation Functions

Back Propagation을 구현하려면 activation function의 도함수를 우리가 앞에서 배운 activation function들과 그 함수들의 기울기를 어떻게 구하는지 알아보자.

Sigmoid



< Sigmoid 미분 >

* 몫의 미분 $\frac{d}{dx}\left(\frac{f}{g}\right) = \frac{f'g - fg'}{g^2}$

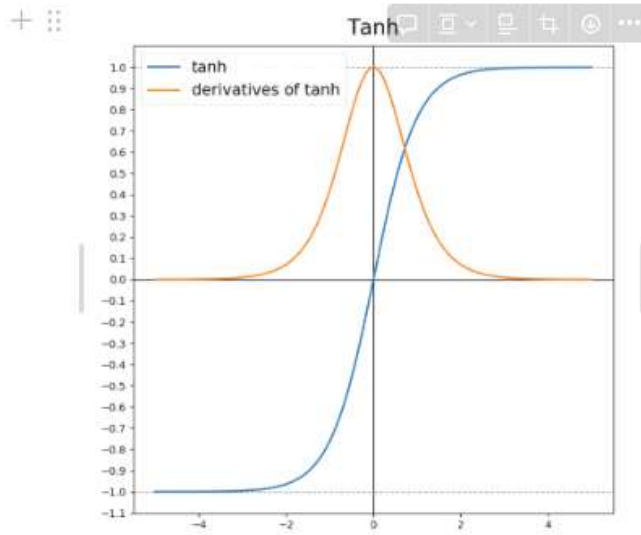
$$g(z) = \frac{1}{1+e^{-z}} \quad \Rightarrow \quad g'(z) = -\frac{-e^{-z}}{(1+e^{-z})^2} = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} \cdot \frac{e^{-z}}{(1+e^{-z})}$$
$$= \underline{g(z)} \cdot \underline{(1-g(z))}$$

$$g'(z) = g(z) \cdot (1-g(z))$$

$$= a \cdot (1-a) \quad (+ a = g(z) = \frac{1}{1+e^{-z}}) \quad \text{바로 표현}$$

Tanh

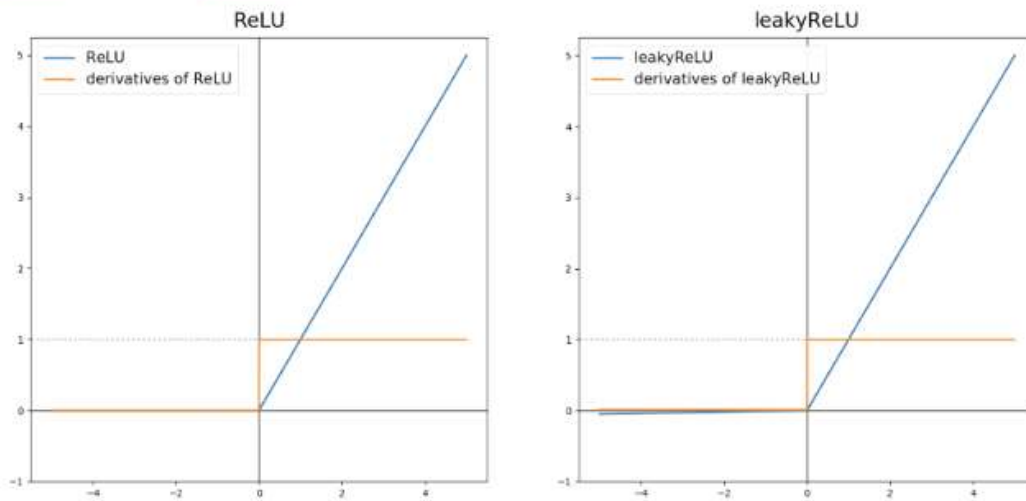


< Tanh 미분 >

$$\begin{aligned} g'(z) = \left[\tanh(z) \right]' &= \left[\frac{e^z - e^{-z}}{e^z + e^{-z}} \right]' \\ &= 1 - \tanh(z)^2 \\ &= 1 - g(z)^2 \end{aligned}$$

$$g'(z) = 1 - a^2$$

ReLU , Leaky ReLU



< ReLU 와 Leaky ReLU 미분 >

$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & (\text{if } z < 0) \\ 1 & (\text{if } z \geq 0) \end{cases}$$

~~undef (if z=0)~~

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & (\text{if } z < 0) \\ 1 & (\text{if } z \geq 0) \end{cases}$$

** z가 정확히 0일 때의 도함수가 정의되지 않았지만, 수학적으로는 정확하지는 않으나 z가 0 일때 기울기를 0 또는 1로 두어도 문제가 되지 않는다. z가 정확히 0일 확률(0.00000...)은 매우 작기 때문.

[Week 4] 3. Gradient Descent for Neural Networks

우리는 파라미터 값을 훈련하기 위해 gradient descent 를 사용한다.

**주의 : 변수를 0이 아닌 값으로 초기화 하는 것이 중요하다.

* 행렬의 크기: $(n^{[1]}, n^{[2]})$ $(n^{[2]}, 0)$ $(n^{[2]}, n^{[3]})$ $(n^{[3]}, 0)$

- Parameters : $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$
 $n_x : n^{[0]}, n^{[1]}, n^{[2]}$
 (우측 4)
- Cost Function : $J(w, b) = \frac{1}{m} \sum_{i=1}^n L(\hat{y}, y)$
- Gradient Descent : $dW^{[1]} = \frac{dJ}{dW^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$, ... 각 변수에 대한
(derivative (편미분) 구하기)

$$\left. \begin{aligned} W^{[1]} &= W^{[1]} - \alpha dW^{[1]} \\ b^{[1]} &= b^{[1]} - \alpha db^{[1]} \\ &\vdots \end{aligned} \right\}$$
파라미터 update.
각 파라미터 값이 수렴할 때까지 반복

중요한 것은 우리가 편미분을 어떻게 계산하느냐이다. ($dW^{[1]}$, $db^{[1]}$, .. 을 어떻게 계산할 것인지)

먼저 Forward Propagation에 대한 식을 정리한 후, Back Propagation에서 사용하는 수식을 알아보자.

Forward Propagation

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

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

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

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

Back Propagation

$$dZ^{[2]} = A^{[2]} - Y \quad Y = [y^{[0]}, y^{[1]}, \dots, y^{[m]}]$$

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

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

가도도 필요한 것 rank 1 행렬 ($n^{[2]} \times 1$) 가 아닌 ($n^{[2]}, 1$) 행렬로 출력될 수 있게 함

elementwise product

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

($n^{[1]}, m$) ($n^{[2]}, m$) hidden layer에서 사용했던 activation function의 derivative.

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

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

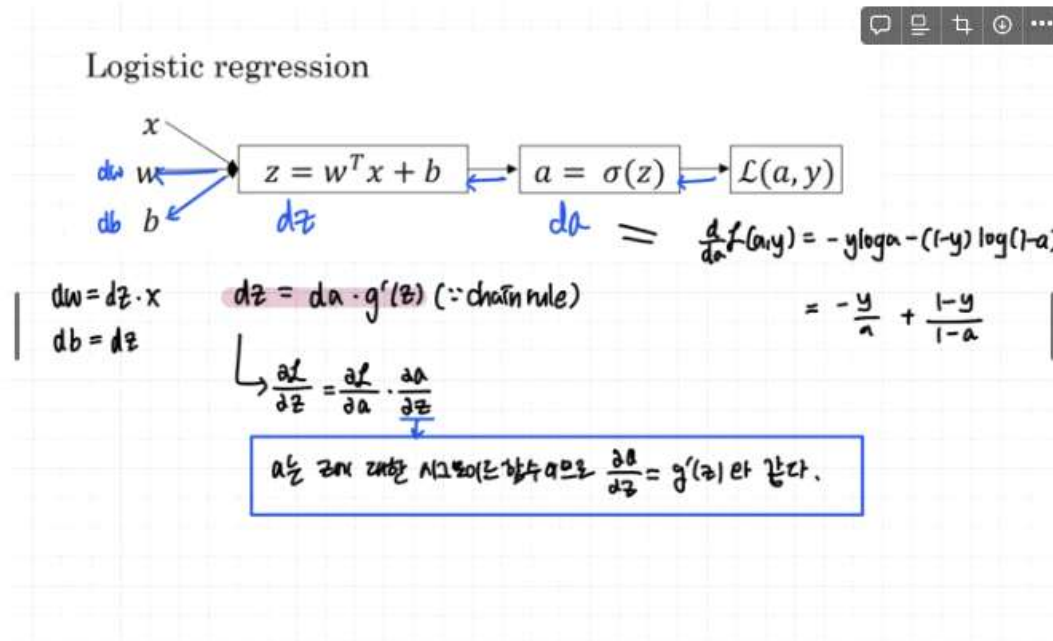
derivative = 계산해낸다

n개의 레이어로 구성된 신경망이라면 1,2,3,...n까지의 연산을 반복하면 된다.

- Elementwise product : 두 개의 동일한 차원을 갖는 행렬 또는 벡터 간에 원소별로 곱셈을 수행하는 것을 의미

그러면 이러한 식들은 어떻게 유도하는 것일까?

Computation graph를 이용하여 앞서 배웠던 로지스틱 회귀에서 이러한 식들이 어떻게 유도되는지 알아보자.



우리는 이러한 계산을 두번만에 수행할 수 있다. 왜냐하면 이 로지스틱 회귀 신경망은 2개의 layer로 이뤄져 있기 때문이다.

- 입력값 x 는 지도학습에서 고정된 값이기 때문에 x 에 대한 도함수는 구할 필요가 없다 (당연)

그동안 하나의 샘플에 관해 계산해보았다. 우리는 m 개의 샘플을 한번에 훈련하도록 식을 벡터화 하여 나타낼 수 있다. 이는 아래와 같다.

Summary of gradient descent

$$\begin{aligned}
 dz^{[2]} &= a^{[2]} - y \\
 dW^{[2]} &= dz^{[2]} a^{[1]T} \\
 db^{[2]} &= dz^{[2]} \\
 dz^{[1]} &= W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]}) \\
 dW^{[1]} &= dz^{[1]} x^T \\
 db^{[1]} &= dz^{[1]}
 \end{aligned}$$

$J(\cdot) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\hat{y}_i, y_i)$
 $dZ^{[2]} = A^{[2]} - Y$
 $dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$
 $db^{[2]} = \frac{1}{m} \text{np.sum}(dZ^{[2]}, \text{axis} = 1, \text{keepdims} = \text{True})$
 $dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$
 $dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
 $db^{[1]} = \frac{1}{m} \text{np.sum}(dZ^{[1]}, \text{axis} = 1, \text{keepdims} = \text{True})$

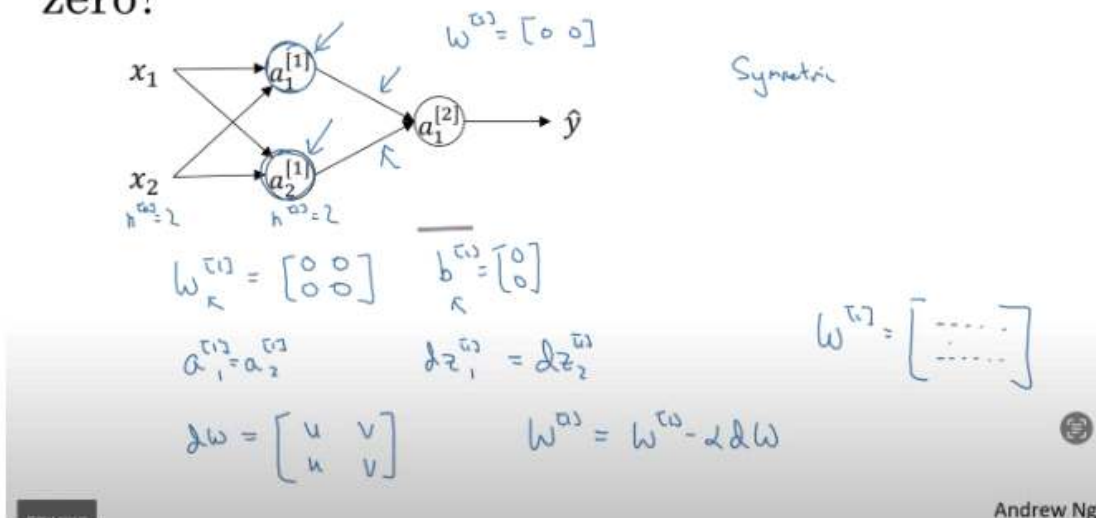
Andrew Ng

[Week 4] 4. Random Initialization

신경망에서 훈련할 때 모든 변수를 랜덤으로 초기화하는 것이 중요하다.

만약 신경망에서 모든 파라미터를 0으로 초기화하고 경사하강법을 적용할 경우 어떤 문제가 발생하는지 알아보자.

What happens if you initialize weights to zero?



여기서의 문제는 $a_1^{[1]}$ 과 $a_2^{[1]}$ 의 값이 같다는 것이다. 왜냐하면 두 유닛은 같은 레이어에 있으므로 정확히 같은 함수를 계산하게 되기 때문이다. 따라서 두 유닛은 **symmetric**이 된다. 따라서 가중치를 업데이트해도 둘은 항상 같은 함수를 계산하기 때문에 출력유닛에 같은 영향을 주게 되고, 첫번째 iteration 이후에 같은 상태가 계속해서 반복된다. 따라서 얼마나 연산을 반복하든지 두 유닛은 항상 같은 함수를 연산하게 된다. 따라서 유닛이 한개인것과 다름이 없어진다. $dz_1^{[1]}, dz_2^{[1]}$ 도 마찬가지로 같아진다.

이를 방지하기 위해 우리는 임의로 변수를 초기화 하는 것이다. 그렇다면 random initialization은 어떻게 하는걸까?

```
W[1]=np.random.randn((2,2)) * 0.01
```

```
b[1]=np.zeros((2,1)) #bias의 경우 이처럼 대칭의 문제를 가지지 않기 때문에 0으로 초기화 해도 된다.
```

왜 0.01을 곱할까?

가중치 값이 너무 큰 경우에 활성화 함수를 계산하게 되면 w 값이 크므로 z 값이 매우 크거나 매우 작게 되고, 활성화 함수에서 변화율(기울기)가 낮은 부분에 속하게 되기 때문에 훈련속도가 매우 느려지게 된다.

