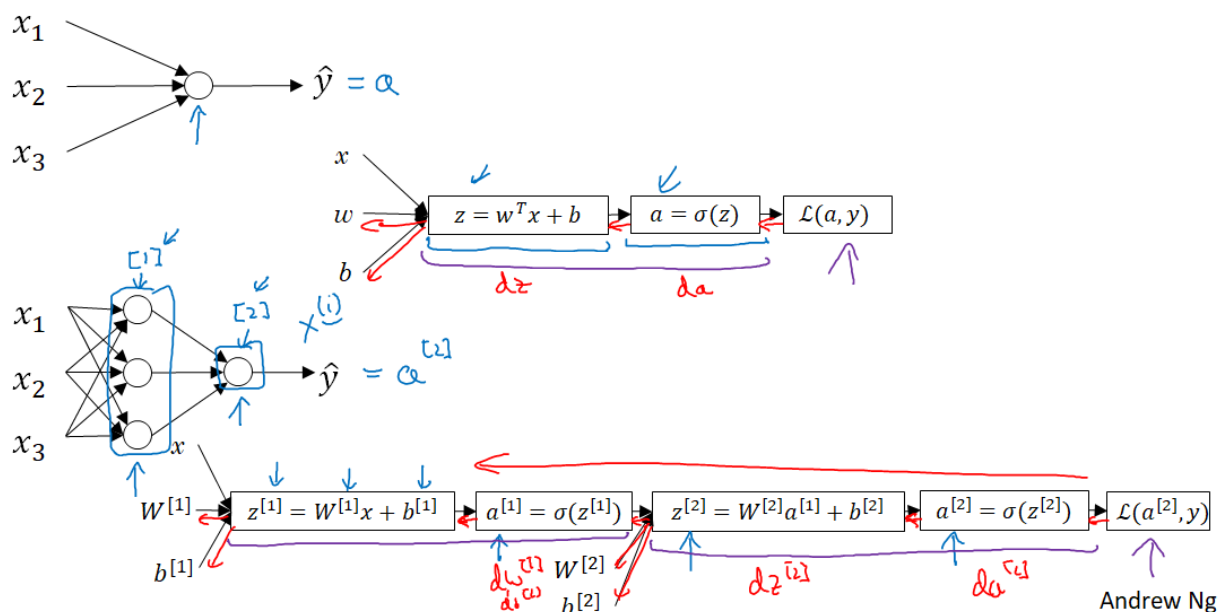




# [3주차] Shallow Neural Network

## What is a Neural Network?



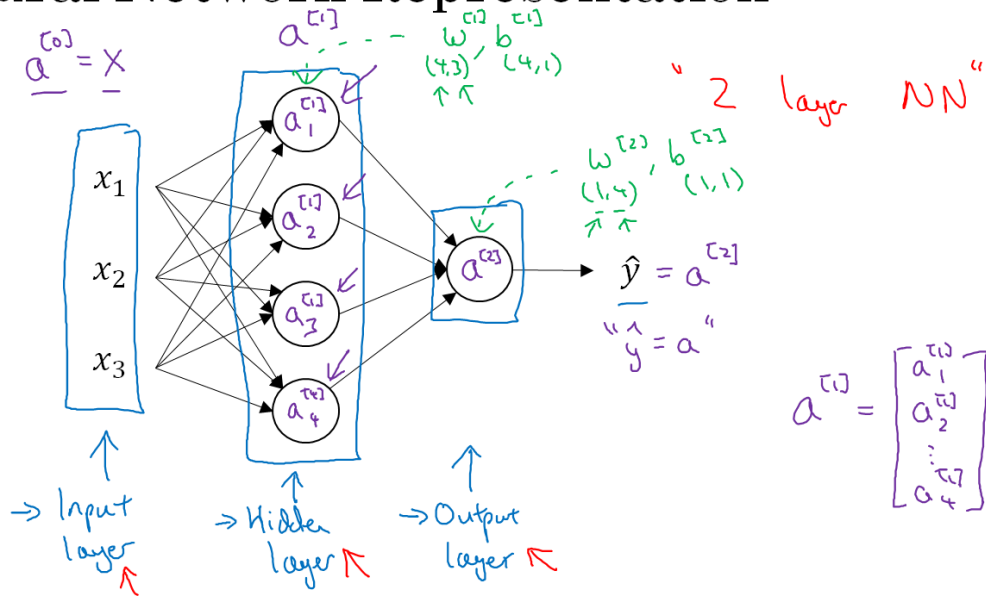
로지스틱 회귀분석법에 대해 이야기 했었는데, 이 모델이 어떻게 여기 보이는 산출 그래프로 이어지는지 알아보았습니다. 여기서는  $x$  특성을  $z$ 를 산출해주는  $w$ 와  $b$  파라미터에 넣지 않았는데요.

**신경망에서는** 처음에는 입력 특성  $x$ 을 넣을 것인데요. 그 다음에  $w$ 와  $b$ 의 파라미터와 함께 말이죠. 그러면 이것이  $z^1$ 을 계산하게 해줄 것인데요, 그러면 새로 소개할 표기는 위 첨자 괄호 1 여기 레어라고 부르는 노드 뭉치의 양을 나타낼 것인데요, 그리고 다음에는 위첨자 괄호 2로 Daniel이라고 불리는 것과 연관된 양입니다. 이것은 또 하나의 신경망 층입니다. 여기 있는 위첨자는 괄호는 개인 트레이닝 예시를 나타낼 때 쓰는 소괄호와 헷갈리시면 안됩니다. 반면에  $x$  위첨자 소괄호 1은 트레이닝 예시를 나타내고, 위첨자 대괄호 1과 2는 여기 이렇게 다른 층을 나타냅니다.

**여러분이 기억하실 핵심 부분**은 로지스틱 회귀분석에서는 여기 Z값과 그 다음에 a 계산이 따랐고, 그 다음에 신경망 네트워크는 여러번 진행합니다. z 다음에 a 계산 다음에 또, Z 계산 진행 후, 다음에 a 계산이 따릅니다. 그리고 최종적으로 loss의 값을 구합니다.

**로지스틱 회귀분석**에서는 여기 이런 backward calculation 이 있었습니다. derivative를 계산하기 위해서 말이죠. 여기서는 **da와 dz를 계산**하는 것이죠. 같은 방법으로, **신경망**에서는 backward calculation을 하텐데요, 이렇게 생겼습니다. 이 경우에는 **da2과 dz2**를 계산해서 **dw2와 db2의 값을 얻을 수 있게** 합니다.

## Neural Network Representation



Andrew Ng

이전에는 입력값을 x로 표현했는데요, 입력특성 값들에 대한 또 다른 표기법으로는, **a 위첨자 괄호 0**입니다. a는 여기서 activation을 상징합니다. 서로 다른 신경망들의 층이 다음 이어지는 신경망 층들로 전달하는 값을 뜻합니다.

여기 위 첨자 대괄호 1의 값은 4차원 벡터인데요, 파이썬에서 구현할 것입니다. 4 X 1 common 벡터 매트릭스 인데요. 이렇게 생겼습니다.

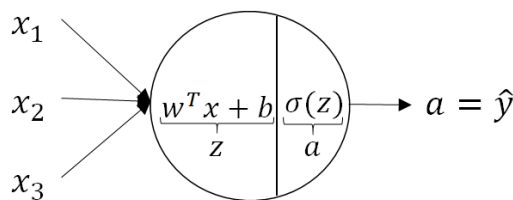
로지스틱 회귀분석때는 1개의 결과값 층밖에 없었습니다. 그렇기 때문에 위첨자 대괄호를 사용하지 않았습니다. 이제 새로운 신경망에서는 어떤 층에서 왔는지를 나타내기 위해서 위첨자를 사용할 것입니다.

여기에서 본 네트워크가 two layer 신경망인데, 신경망에서 그렇게 불리는 이유는 **입력 층을 세지 않기 때문**입니다. 그러므로 숨겨진 층 이 1번 층이고, 결과값 층이 2번 층입니다.

나중에 보겠지만  $w$ 는  $4 \times 3$  매트릭스 입니다.  $b$ 는  $4 \times 1$  벡터이구요. 4라는 첫번째 좌표는 4개의 숨겨진 층과 층이 있는 노드가 있다는 것을 나타내구요, 그리고 3이라는 좌표 숫자는 우리가 입력 특성이 3개를 가지고 있다는 것입니다.

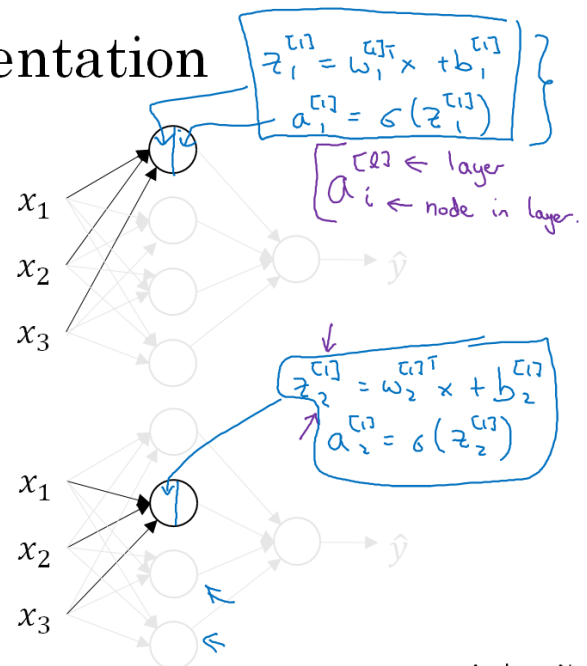
$w$  위첨자 대괄호 2 와  $b$  위첨자 대괄호 2와 연관되어 있을 수 있는데요, 각각의 dimension 은  $1 \times 4$ , 그리고  $1 \times 1$ 입니다. 그리고 여기서  $1 \times 4$ 인 이유는, 숨겨진 층이 4개 있기 때문입니다. 그리고 결과값은 1개의 유닛이 있습니다.

## Neural Network Representation



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

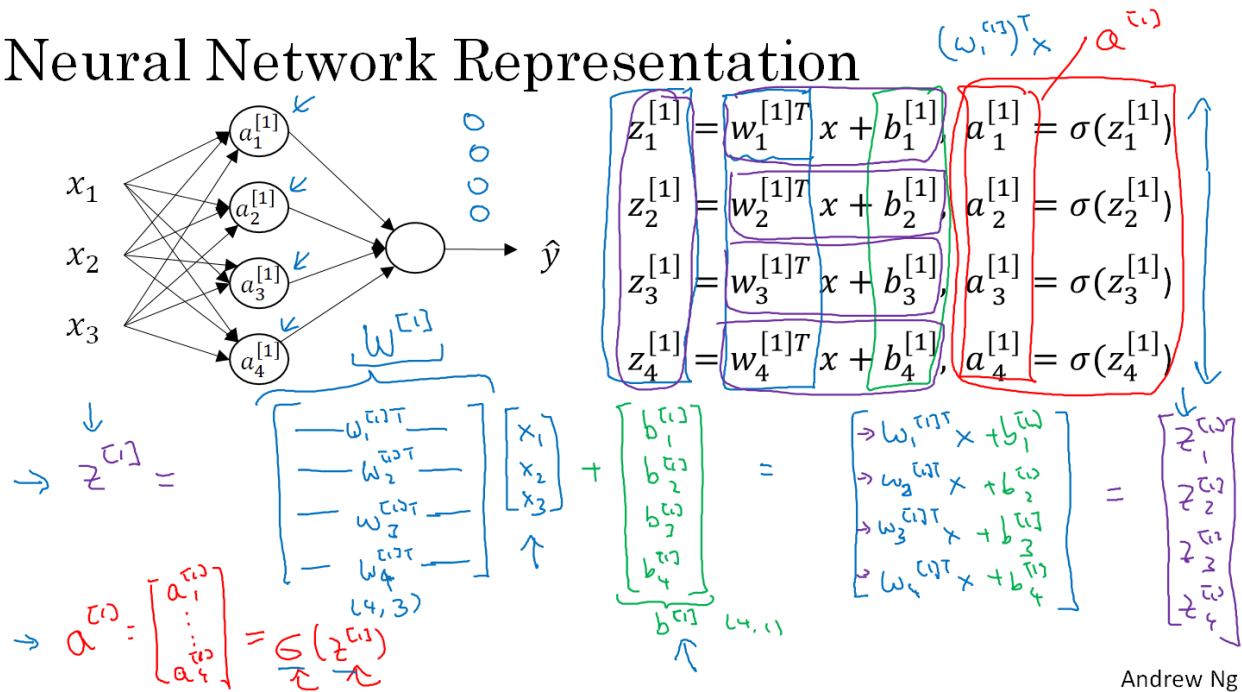
$$a = \sigma(z)$$



Andrew Ng

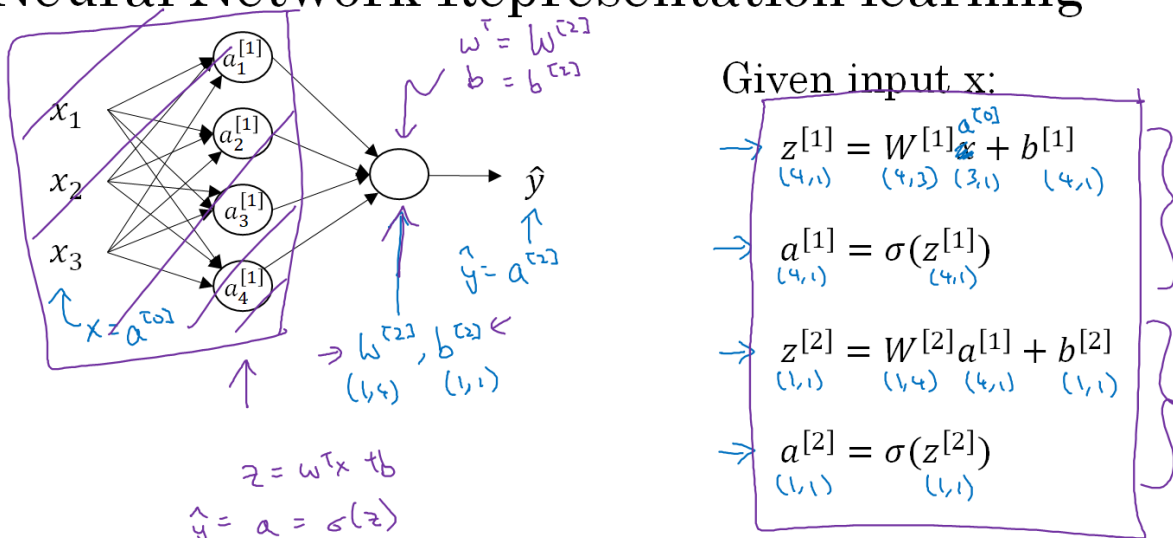
$a, l, i$  위첨자 대괄호  $l$ 은 층의 번호를 나타내고 여기  $i$  첨자는 층에 있는노드를 표현합니다.

# Neural Network Representation



여러분도 짐작하셨겠지만, 이것을 신경망 네트워크에 도입하는 경우에, 이것을 4개의 루프로 하는 것은 비효율적이라 생각하실 것입니다. 그렇기 때문에 이 4개의 공식을 갖고 벡터화시킬 것입니다. 그러면 저는 먼저  $z$ 를 벡터로 산출하는 방법을 보여드릴 것입니다.

## Neural Network Representation learning



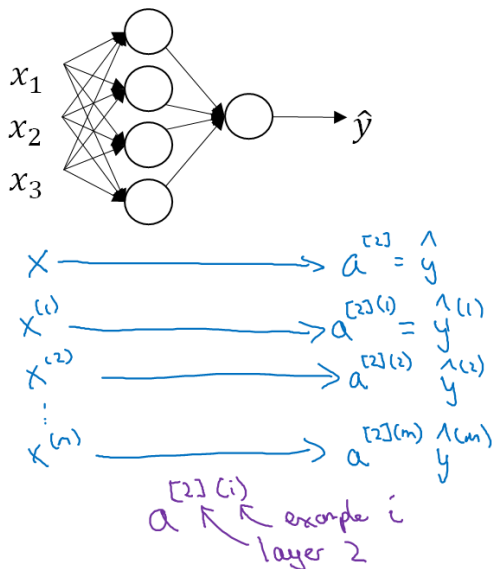
Andrew Ng

기억하시겠지만,  $x$ 는 0이라고 했죠.  $\hat{y}$ 이  $a_2$ 인 것과 같이 말이죠. 그렇기 때문에 원하면 이  $x$  값을  $a_0$ 로 대입해도 됩니다.

결과값이 하는 것은,  $w_2$ 와  $b_2$ 와 연계가 될텐데요, 이런 매개 변수의 경우, 이것은  $1 \times 4$  매트릭스가 될 것입니다. 그리고  $b_2$ 는  $(1, 1)$  실수가 됩니다 그렇기 때문에  $z_2$  가  $(1, 1)$  실수인 매트릭스가 될 것입니다. 이것은 1, 4가 될 것이고, 이것은 4, 1 이었죠. 더하기  $b_2$ 는 1, 1입니다. 그렇기 때문에 이것은 또한 실수가 됩니다.

네트워크의 이 왼쪽에 있는 부분을 일단 무시하겠습니다. 그러면 여기 이 마지막 결과값은 로지스틱 회귀분석방식과 매우 유사한데요, 단지  $w$ 와  $b$ 로 매개 변수를 갖는 대신에, 변수를  $w_2$ 와  $b_2$ 로 쓰는 것입니다.  $1 \times 4$  다이멘션과  $1 \times 1$  다이멘션으로 말이죠.

## Vectorizing across multiple examples



$$\left\{ \begin{array}{l} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{array} \right\} \leftarrow$$

→ for  $i = 1$  to  $n$ ,

$$\begin{array}{l} z^{[1]^(i)} = W^{[1]}x^{(i)} + b^{[1]} \\ a^{[1]^(i)} = \sigma(z^{[1]^(i)}) \\ z^{[2]^(i)} = W^{[2]}a^{[1]^(i)} + b^{[2]} \\ a^{[2]^(i)} = \sigma(z^{[2]^(i)}) \end{array}$$

Andrew Ng

# Vectorizing across multiple examples

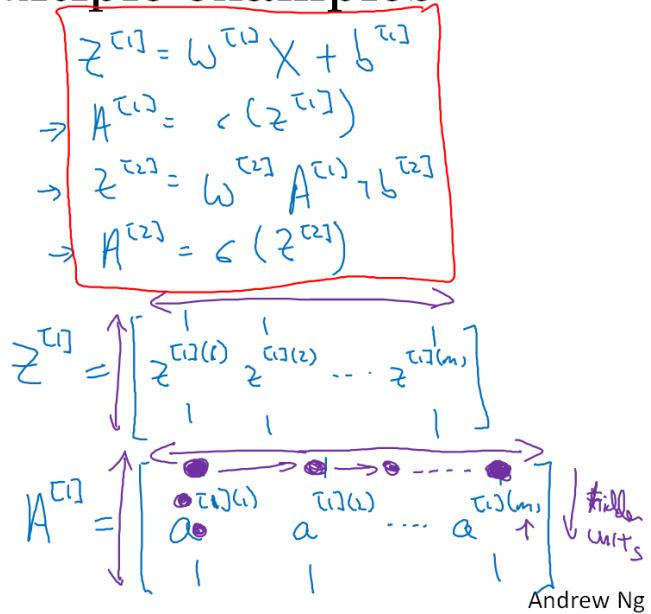
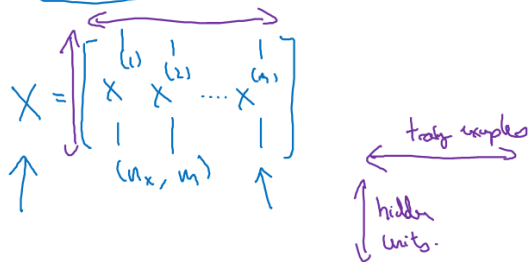
for  $i = 1$  to  $m$ :

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

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

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

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



여러분이 모든 결과값을  $m$ 트레이닝 샘플에 대해 계산하고 싶으면 여기 전체 값을 벡터화시킵니다.

그러면 저희가 소문자  $x$ 에서 대문자  $X$ 로 간 것과 같이, 소문자  $z$ 에서 대문자  $Z$ 가 된 것과 같이 말이죠, 이것은 소문자  $a$ 벡터에서, 대문자  $A1$ 이 됩니다. 여기  $A1$ 과 같이 말이죠. 또 비슷하게,  $z[2]$  와  $a[2]$ 는 여기 이 벡터들을 쌓아서 얻어지는데요, **가로로 쌓아서** 말이죠.

여기  $Z$ 와  $A$ 와 같은 매트릭스는 트레이닝 샘플들을 **가로로 인덱싱**할 것인데요, 그렇기 때문에 **가로 인덱스가 다른 트레이닝 샘플을 나타내는 것**입니다.

만약 왼쪽에서 오른쪽을 가는 경우, 트레이닝 세트를 스캐닝하는 것과 마찬가지입니다. **세로**로는 여기 **세로 인덱스가 여러분의 신경망에 있는 여러개의 노드**를 나타냅니다. 예를 들어, 여기 이 노드는, 가장 위에 있는데요, 가장 위 코너에 있는 매트릭스 값은 첫번째 샘플에서 숨겨진 유닛의 activation을 나타냅니다. 그 다음 값은 두번째 첫번째 샘플에서 두번째 숨겨진 유닛의 activation을 나타냅니다. 그리고 첫번째 샘플의 세번째 숨겨진 유닛 도 이렇게 나타나겠죠. 이렇게 스캔을 해보면, 이것이 숨겨진 유닛에 해당하는 인덱스 숫자이구요.

반대로 가로로 움직이는 경우, 첫번째 트레이닝 샘플의 첫번째 숨겨진 유닛에서, 두번째 트레이닝 샘플의 첫번째 숨겨진 유닛으로 이렇게 세번째 트레이닝 샘플 등등 말이죠. 여기 이 노드가 마지막 트레이닝 샘플의 첫번째 숨겨진 유닛의 activation까지 나타날때까지  $n$ 번째 트레이닝 샘플로 이어집니다.

Handwritten diagram illustrating the vectorization of a linear transformation. The diagram shows the transformation of a vector  $x^{(1)}$  into  $z^{(1)}$  using a weight matrix  $W^{(1)}$  and a bias  $b^{(1)}$ .

The transformation is shown as:

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

The diagram uses color-coding to show the row-by-row calculation of  $z^{(1)}$ :

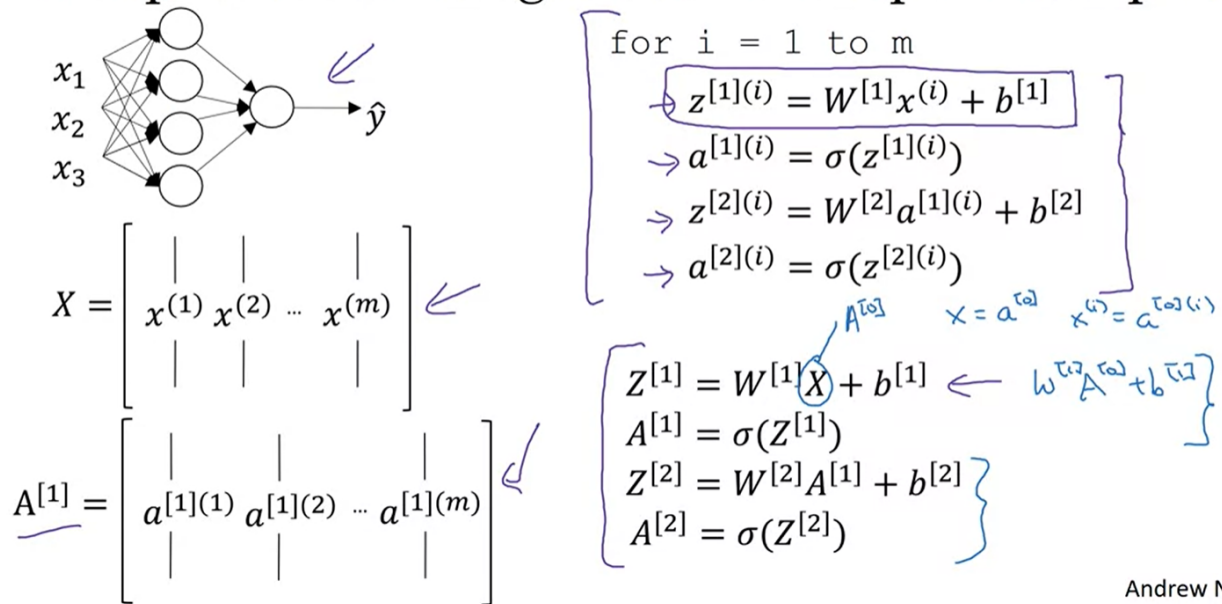
- Purple:** The first row of  $W^{(1)}$  is multiplied by  $x^{(1)}$  to produce the first element of  $z^{(1)}$ .
- Green:** The second row of  $W^{(1)}$  is multiplied by  $x^{(1)}$  to produce the second element of  $z^{(1)}$ .
- Yellow:** The third row of  $W^{(1)}$  is multiplied by  $x^{(1)}$  to produce the third element of  $z^{(1)}$ .

The final result is the vector  $z^{(1)}$ , which is the output of the transformation.

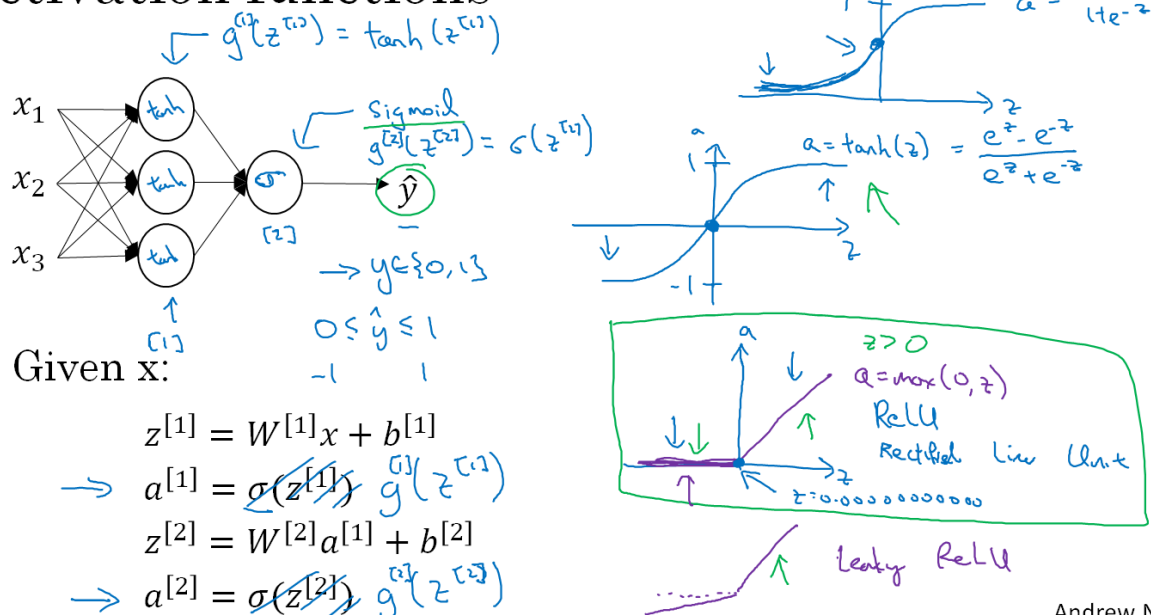
이 슬라이드 내용을 간략하기 위해 일단  $b$ 는 무시할 것입니다. 공식을 조금 더 심플하게 하기 위해서,  $b$ 를 0으로 만들 것입니다.

### [3주차] Shallow Neural Network

## Recap of vectorizing across multiple examples



## Activation functions



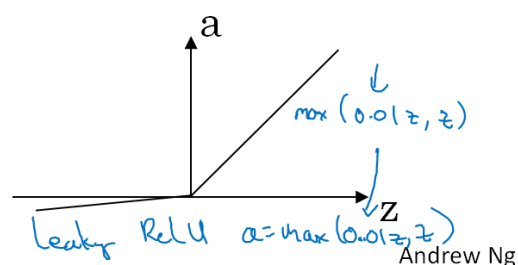
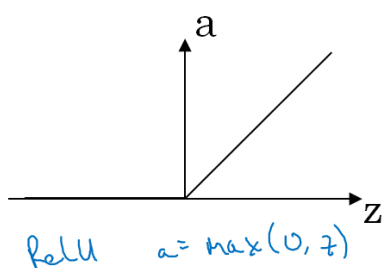
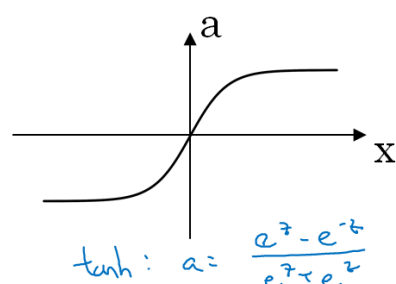
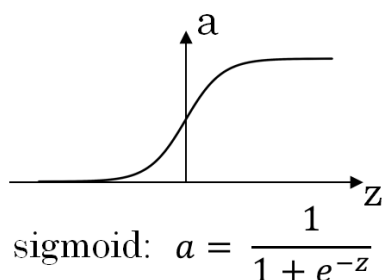
한 가지 상기시킬 부분은 저는 시그모이드 함수는 더 이상 거의 사용하지 않는다는 것입니다.  $\tanh$  함수는 거의 항상 우월합니다. 한 가지 예외는 출력층일 때입니다. 왜냐하면  $y$ 는 0 이



거나 1 인데, 그럴 경우  $\hat{y}$  ( $y$ 의 예측 값) 은 0과 1 사이의 출력 값이며 -1 과 +1 사이는 아니기 때문입니다. 제가 시그모이드 활성화함수를 사용하는 한 가지 예외는 이진 분류에 사용하는 경우입니다. 그런 경우는 출력층에 시그모이드 함수를 사용하게 될 것입니다. 따라서  $g(z[2])$  는  $\text{sigma}(z[2])$  가 됩니다. 이 예제에서 본 것 처럼  $\tanh$  를 은닉층의 활성화함수로 갖고 있으며, 출력층은 시그모이드입니다. 그래서 층이 다를 경우 다른 활성화함수를 가질 수 있습니다. 때로 활성화함수가 층이 다를 경우 같지 않기 때문에 우리는 대괄호(square bracket) 를 위첨자로 써서  $g^{[1]}$  과  $g^{[2]}$  이 다르다는 것을 표시합니다.  $[1]$  이 이 층을 나타내고, 그리고  $[2]$  이 출력층을 나타냅니다.

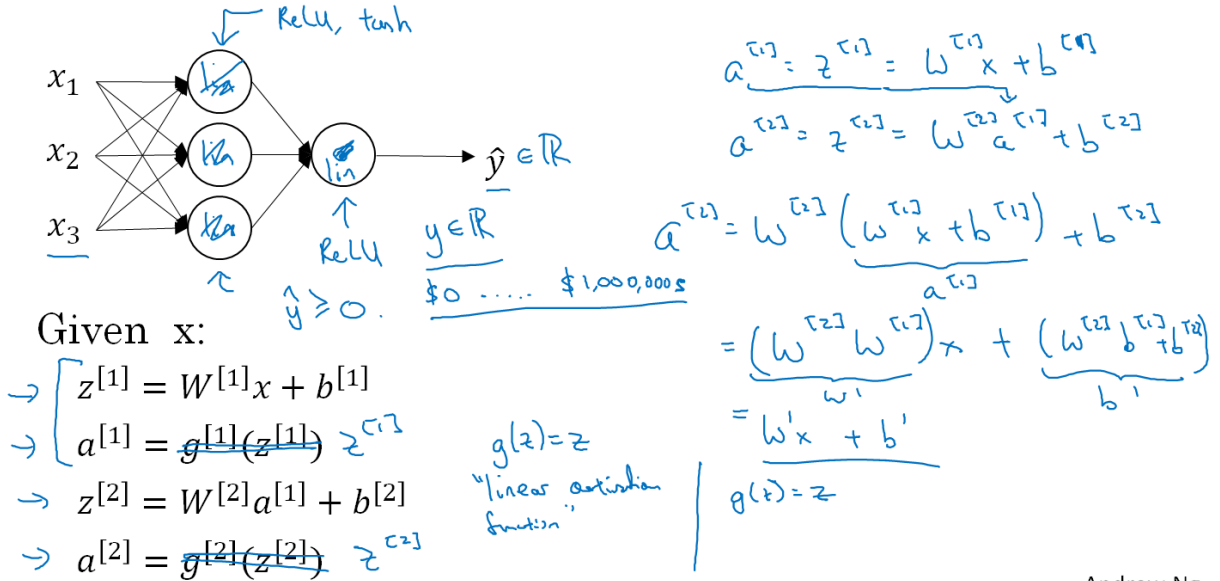
여기 **시그모이드 활성화함수**가 있습니다. 저는 출력층이 아니라면 쓰지 말라고 얘기하고 싶습니다. 여러분이 **이진 분류**를 하는 경우라면 **가능하겠지만, 그렇지 않다면 사용하지 마시기** 바랍니다.

## Pros and cons of activation functions



여러분이 어떤 걸 써야 할지 잘 모르겠다면, 이것을 사용하면 됩니다. 그리고 leaky ReLU 를 사용하는 것도 좋습니다. 아마도  $\max(0.01z, z)$  로 표현할 수 있겠네요  $a$  는  $\max(0.01z, z)$  인데, 여기 조금 굽혀진 함수를 제공합니다. 여러분은 아마도 왜 0.01 을 사용하는 거죠? 라고 물어볼 수 있습니다. 음, 어떤 사람들은 다른 파라미터 값을 사용할 수도 있을 겁니다. 더 잘 작동하는 경우도 있을 수 있습니다. 하지만 저는 그렇게 하는 사람들은 거의 보지 못했습니다.

# Activation function

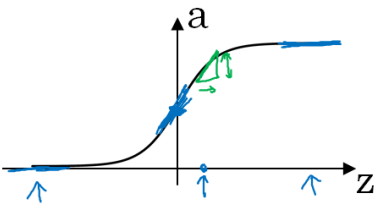


Andrew Ng

$g$ 의 함수를 빼겠습니다. 그리고  $a_1$ 을  $z_1$ 으로 지정하겠습니다. 또는  $g(z)$ 가  $z$ 와 동일하다고 표현해도 됩니다. 가끔씩은 이것이 **선형 activation 함수**라고 표현하는데요, 더 나은 이름은 **identity activation 함수**일 것입니다. 이유는 이것들은 단지 입력값이 들어간 그대로 결과값이 나오기 때문입니다.

중요한 부분은 선형에서의 숨겨진 레이어는 거의 쓸모가 없다는 것입니다. 왜냐면 2개의 선형 함수의 구성요소는 그 자체가 선형함수이기 때문입니다. 그러므로 비선형 특성을 기입하지 않는 이상은 더 흥미로운 함수를 산출하는 것이 아닙니다. 신경망이 더 깊어질 때 말이죠.

# Sigmoid activation function



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

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

$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

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

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

$$= \boxed{a(1-a)} \quad \left| \begin{array}{l} g'(z) = a(1-a) \\ \uparrow \end{array} \right.$$

$$z = 10, \quad g(z) \approx 1$$

$$\frac{d}{dz} g(z) \approx 1(1-1) \approx 0$$

$$z = -10, \quad g(z) \approx 0$$

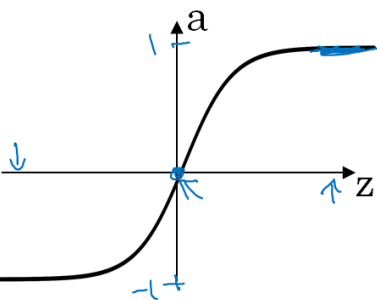
$$\frac{d}{dz} g(z) \approx 0(1-0) \approx 0$$

$$z = 0, \quad g(z) = \frac{1}{2}$$

$$\frac{d}{dz} g(z) = \frac{1}{2} \left( 1 - \frac{1}{2} \right) = \frac{1}{4}$$

Andrew Ng

# Tanh activation function



$$g(z) = \tanh(z)$$

$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

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

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

$$z = 10, \quad \tanh(z) \approx 1$$

$$g'(z) \approx 0$$

$$z = -10, \quad \tanh(z) \approx -1$$

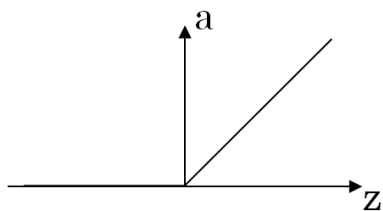
$$g'(z) \approx 0$$

$$z = 0, \quad \tanh(z) = 0$$

$$g'(z) = 1$$

Andrew Ng

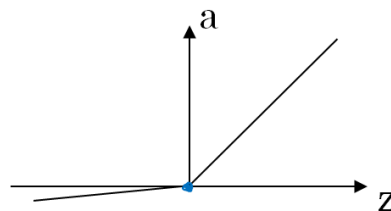
## ReLU and Leaky ReLU



ReLU

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

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



## Leaky ReLU

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

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

Andrew Ng

## Formulas for computing derivatives

Forward propagation:

$$Z^{in} = W^{in} X + b^{in}$$

$$A^{\tau_{12}} = g^{\tau_{12}}(z^{\tau_{12}}) \leftarrow$$

$$z^{[2]} = w^{[2]} A^{[1]} + b^{[2]}$$

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

Back propagation:

Back propagation:

$$\delta z^{[2]} = A^{[2]} - Y \quad \swarrow$$

$$d\omega^{[2]} = \frac{1}{m} dz^{[2]} A^{[n]T}$$

$$b^{[2]} = \frac{1}{n} \text{np.sum}(Z^{[2]}, \text{axis}=1, \text{keepdims}=\text{True})$$

$$dz^{[1]} = \underbrace{W^{[2]T}}_{(n^{[2]}, m)} dz^{[2]} \otimes \underbrace{g^{[2]'}(z^{[2]})}_{\text{element-wise product}} (n^{[2]}, m)$$

$$dw^{(2)} = \frac{1}{n} dz^{(2)} X^T$$

$$\underline{d\mathbf{b}^{(i)}} = \frac{1}{n} \text{np.sum}(d\mathbf{z}^{(i)}, \text{axis}=1, \text{keepdims}=\text{True})$$

Andrew Ng

오른쪽은 **후 방향전파** 인데요,  $dz2$ 를 계산할 것인데요,  $a2$  - ground truth  $Y$ 인데요, 다시 말씀드리자면 이것은 예시에 대해 **vectorize**된 것입니다. **매트릭스  $Y$ 는  $1 \times m$  매트릭스** 인데

요 모든 M 예시에 대해 가로로 정리합니다. dw2는 이거와 같은데요, 여기 첫번째 3개의 공식은 로지스틱 회귀의 기울기 강하와 매우 유사합니다. axis= 1, keepdims= true인데요, 여기 상세 내용인데요, NP.sum은 이번 예제에서의 경우, **가로로 더하는 것**을 뜻합니다.

여기 이 keepdimms=true를 진행함으로써 파이썬이 db2의 결과값으로 벡터 (n, 1)의 값을 출력하게 해줍니다. 엄밀히 이야기하면 이것은 (n2, 1)이 될것입니다. 이 경우에는 어차피 (1, 1) 이여서 상관은 없으나 나중에 이것이 중요한 때 다시 이야기하겠습니다.

keepdimms parameter 대신에 사용할 수 있는 방법은 **reshape**를 불러오는 것입니다. NP.sum 결과값을 다시 reshape 하는 것입니다 db가 원하는 디아멘션으로 말이죠.