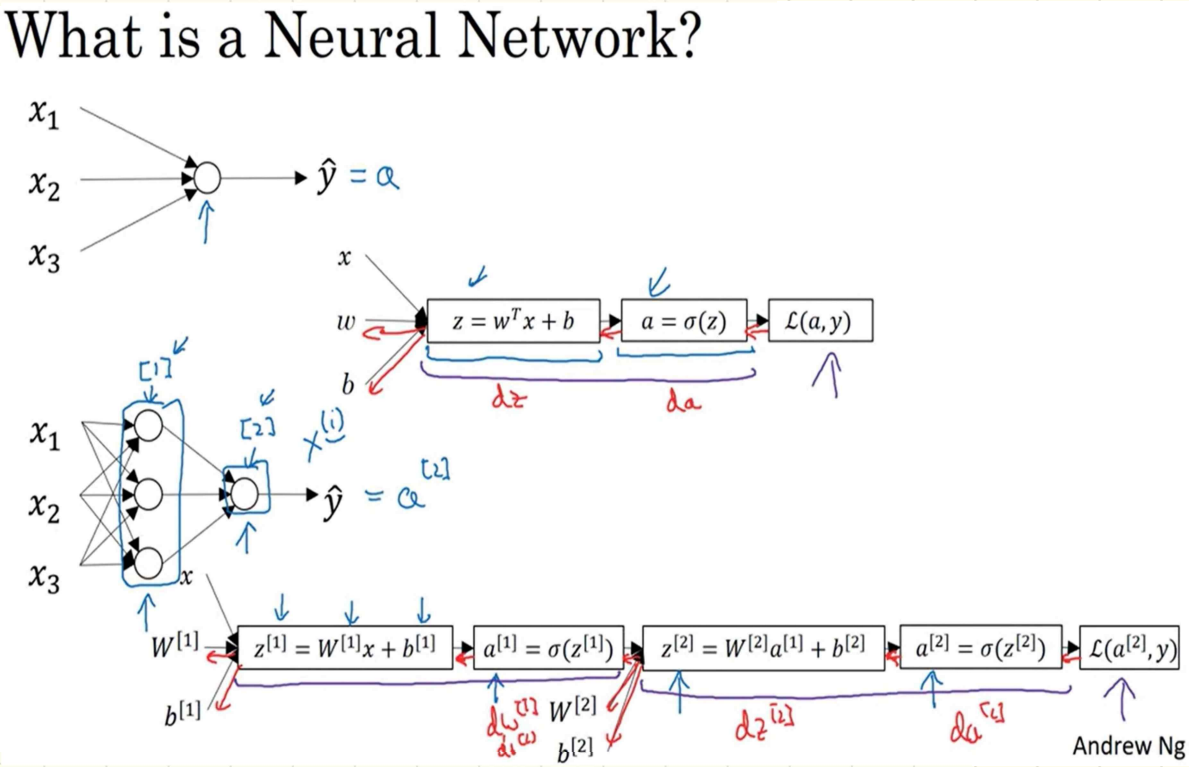
[2-1. Shallow Neural Network]

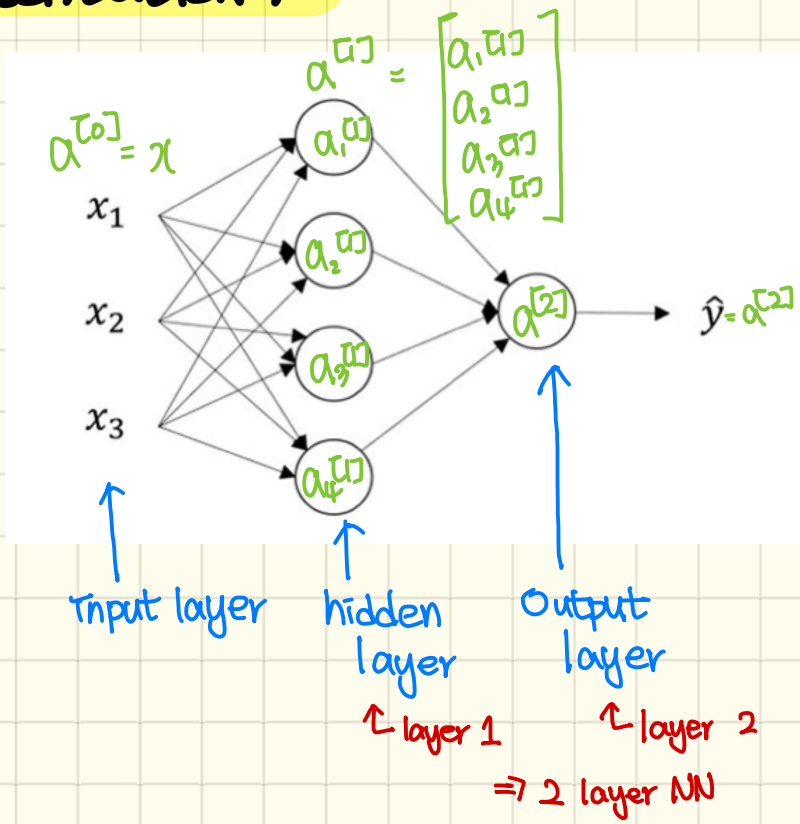
<Neural Networks Overview>



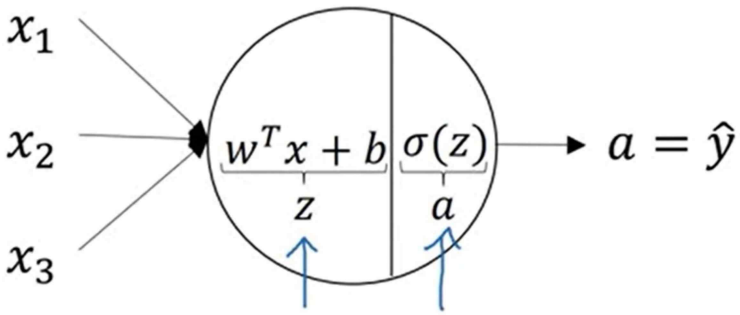
<Neural Network Representation>

* Input layer (입력층) : $a^{[0]}$
  Hidden layer (은닉층) : $a_n^{[1]}$
  Output layer (출력층)

* 신경망의 층의 개수를 셀 때는 입력층은 제외한다
  ∴ 오른쪽 사진은 2-layer Neural Network

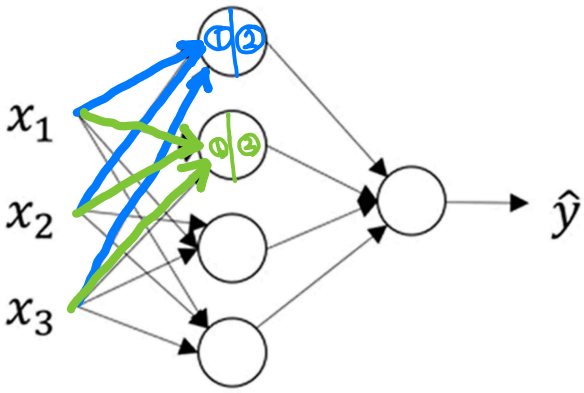


<Computing a Neural Network's Output>



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

$$a = \sigma(z)$$



* 파란색 표시

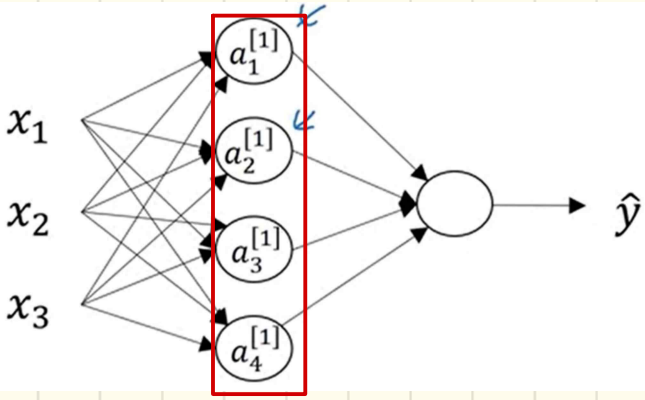
① $z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$

② $a_1^{[1]} = \sigma(z_1^{[1]})$

* 초록색 표시

① $z_2^{[1]} = w_2^{[1]T} + b_2^{[1]}$

② $a_2^{[1]} = \sigma(z_2^{[1]})$

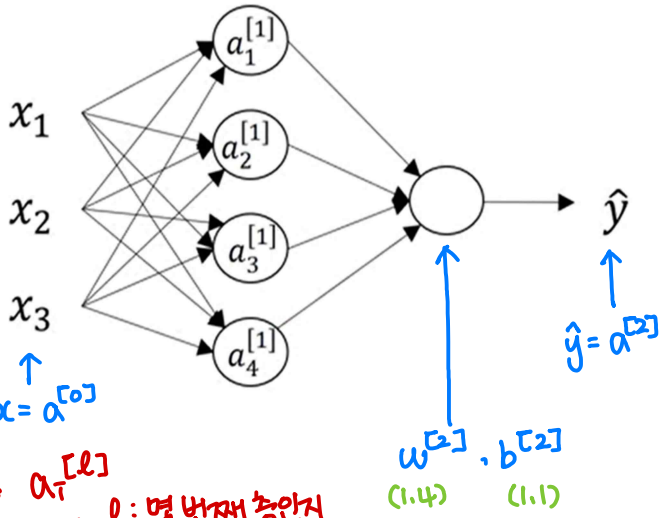


↑ $a^{[1]} = \sigma(z^{[1]})$

$$\begin{aligned}
z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]}, & a_1^{[1]} &= \sigma(z_1^{[1]}) \\
z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, & a_2^{[1]} &= \sigma(z_2^{[1]}) \\
z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, & a_3^{[1]} &= \sigma(z_3^{[1]}) \\
z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, & a_4^{[1]} &= \sigma(z_4^{[1]})
\end{aligned}$$

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{(4,3)} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{(3,1)} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{(4,1)} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ ↑ $b^{[1]}$



$a_i^{[l]}$
- l : 몇 번째 층인지
- i : 해당 층에서 몇 번째 노드인지

Given input x:

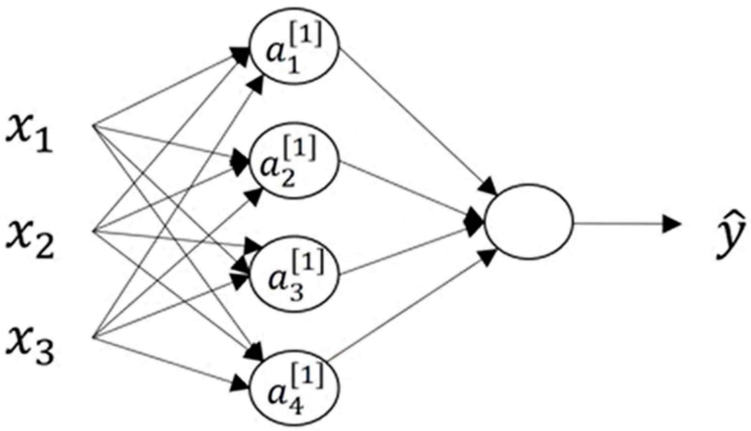
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1)

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

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

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

<Vectorizing Across Multiple Examples>



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

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

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

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

$x \longrightarrow a^{[2]} = \hat{y}$

m개의 training set 으로 확장

$\Rightarrow x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\cdots$

$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$

$a^{[i](j)}$
- i : 몇 번째 층인지
- j : 몇 번째 훈련샘플인지

$\Rightarrow$ for $i = 1$ to m,

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

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

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

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

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

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

Vectorizing →

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

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

(n_x, m)

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

hidden units

training unit

<Explanation for Vectorized Implementation>

$$z^{[1](1)} = w^{[1]}x^{(1)} + b^{[1]}, \quad z^{[1](2)} = w^{[1]}x^{(2)} + b^{[1]}, \quad z^{[1](3)} = w^{[1]}x^{(3)} + b^{[1]}$$

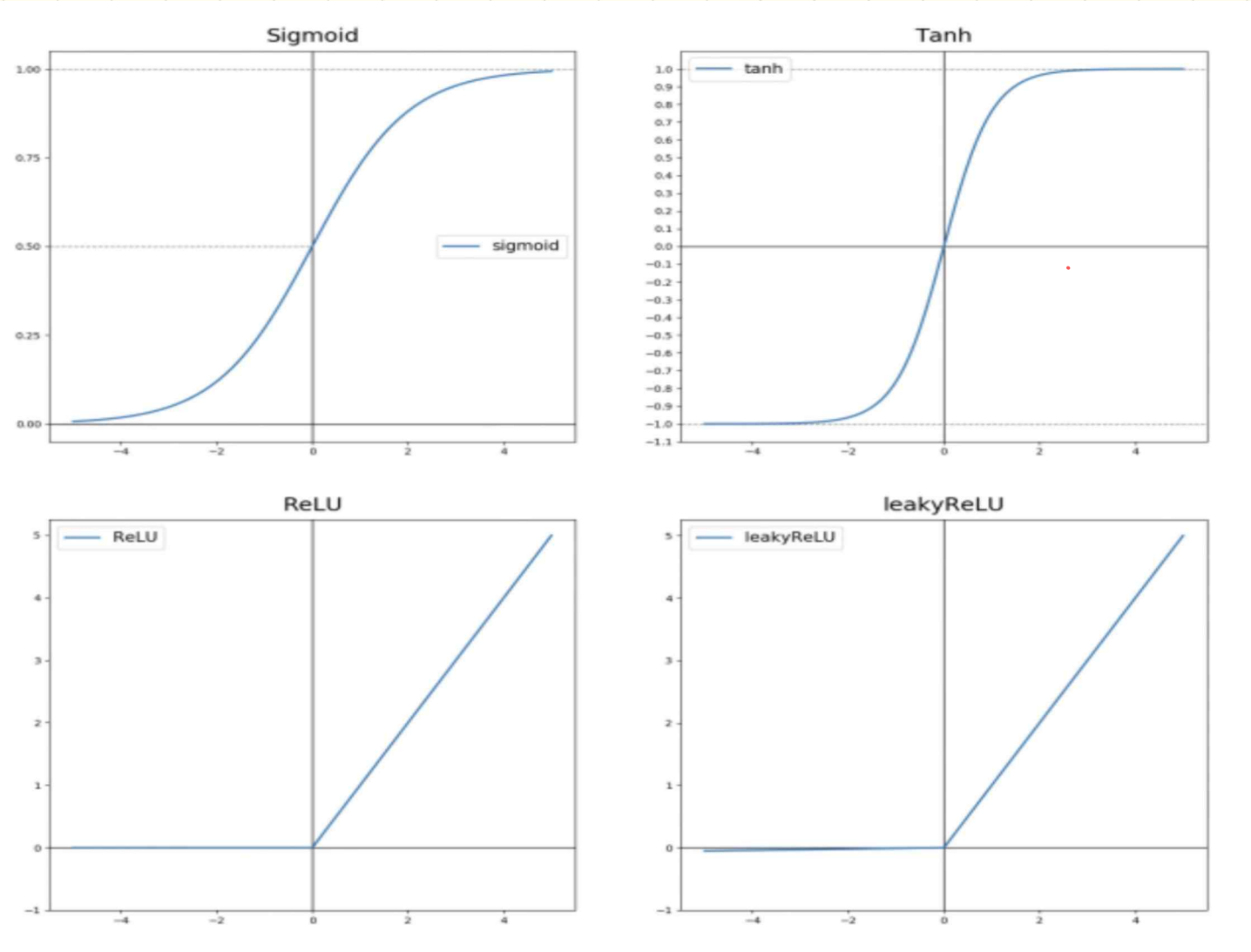
0이라고 가정

$$w^{[1]} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \quad w^{[1]}x^{(1)} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \quad w^{[1]}x^{(2)} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \quad w^{[1]}x^{3} = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$$w^{[1]}\begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)}\cdots \\ | & | & | \end{bmatrix} = \begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix} \cdots = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = Z^{[1]}$$

$$\therefore w^{[1]}x^{(i)} = z^{[1](i)}$$

<Activation Functions>



- sigmoid : $a = \frac{1}{1+e^{-z}}$
- ☆☆ tanh : $a = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$
- ☆☆☆ ReLU : $a = \max(0, x)$
- leaky ReLU : $a = \max(0.01x, x)$

tanh의 장점

· 값이 [-1, 1] 사이에 있고 평균이 0 이므로, 데이터를 이동하는 효과가 있으며, 평균이 0.5인 sigmoid 보다 더 효율적이다.

ReLU의 장점

: 0보다 큰 활성함수의 미분값이 다른 활성함수의 값보다 크기 때문에 더 효율적이다.

<Why do you need Non-Linear Activation Function?>

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

$a^{[1]} = g^{[1]}(z^{[1]})$ → 선형 활성화 함수 : $g(z) = z$를 사용한다면, $z^{[1]}$

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

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

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

$a^{[2]} = z^{[2]} = w^{[2]}a^{[0]} + b^{[1]}$

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

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

$= w'x + b'$

∴ 선형 활성화 함수를 사용한다면, $g(g(g(z))) = z$ 로 세개의 은닉층을 쌓아도 효과를 얻지 못한다.

<Derivatives of Activation Functions>

*시그모이드

- $g(z) = \frac{1}{1+e^{-z}}$ ← slope of $g(z)$ at z
- $g'(z) = \boxed{\frac{d}{dz}g(z)} = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1-g(z))$

* Tanh

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

→ $z = 10$, $\tanh(z) \approx 1$ → $g'(z) \approx 0$

$z = -10$, $\tanh(z) \approx -1$ → $g'(z) \approx 0$

$z = 0$, $\tanh(z) = 0$ → $g'(z) = 1$

* ReLU

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

← 실제로는 정의하기 어려움

*Leaky ReLU

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

<Gradient Descent for Neural Networks>

- Parameters : $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}$
  $(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$ $n_x = n^{[0]}$, $n^{[2]} = 1$
- Cost Function : $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}, y)$
  $\hat{y} = a^{[2]}$
- Gradient Descent :
  (반복 계산) $dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

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

<Forward propagation>

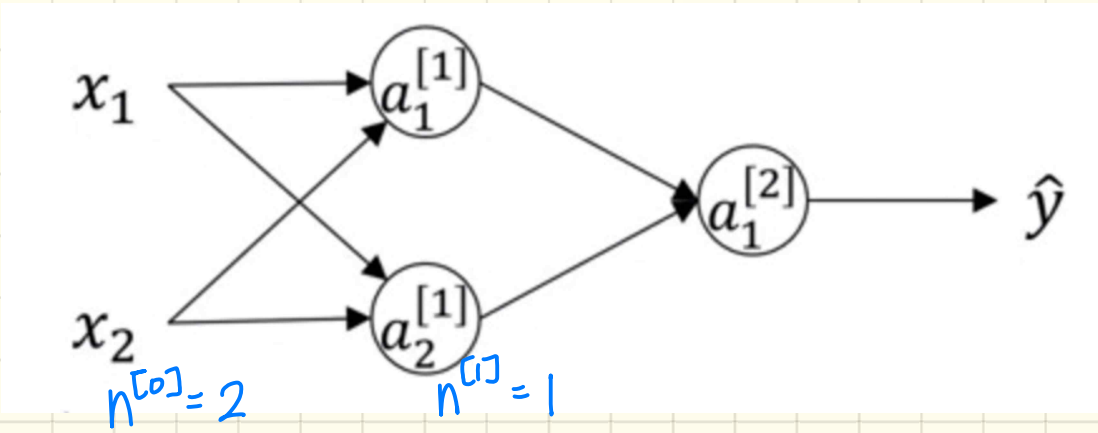
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$

<Back Propagation>

$$dZ^{[2]} = A^{[2]} - Y, \quad Y = [y^{(1)}\ y^{(2)}\ \cdots\ y^{(m)}]$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m}\, np.sum(dZ^{[2]}, axis=1, keepdims=True)$$
: $(n^{[1]},)$ 로 계산 X
$(n^{[1]}, 1)$ 로 계산 O

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^{T}$$
$$db^{[1]} = \frac{1}{m}\, np.sum(dZ^{[1]}, axis=1, keepdims=True)$$

<Random Initialization>



If) W의 초기값을 '0'으로 초기화 한다면?

$$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow a_1^{[1]} = a_2^{[1]}, \quad dz_1^{[1]} = dz_2^{[1]}$$

⇒ dw를 계산했을 때, 모든 행이 같은 값을 갖게된다.

∴0이 아니라 np.random.rand()를 이용하여 랜덤값을 부여해줘야 함.

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

매우 작은 값으로 만들어주기 위함.

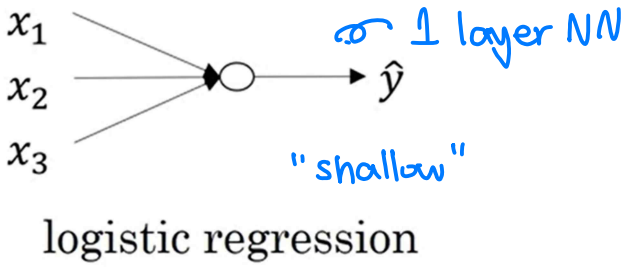
$b^{[1]}$ = np.zero((2,1))

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

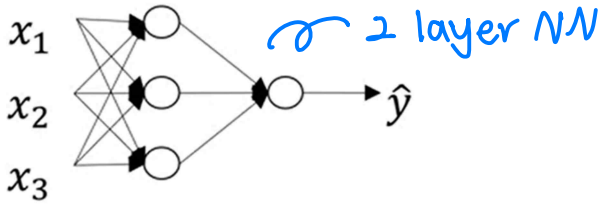
$b^{[2]}$ = 0

[4-1. Deep Neural Network]

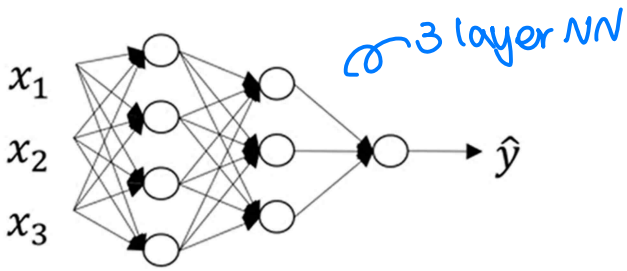
<Deep L-layer Neural Network>



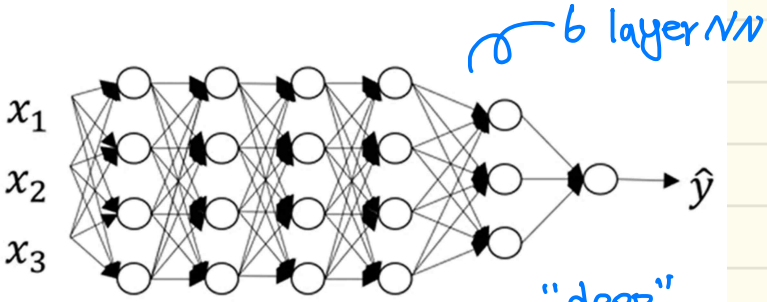
logistic regression



1 hidden layer



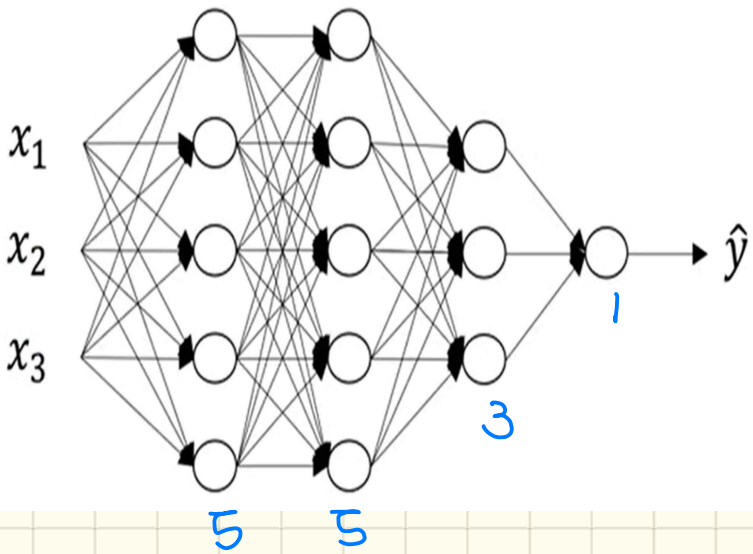
2 hidden layers



5 hidden layers

* 얼마나 깊은 신경망을 사용해야 하는지는 예측하기 어려우며, 하이퍼파라미터를 지정하면서 값이를 결정해야 한다.

ex) 4 layer NN



L = #layers = 4

$n^{[l]}$ = # units in layer l

$n^{[0]} = 3$, $n^{[1]} = 5$, $n^{[2]} = 5$

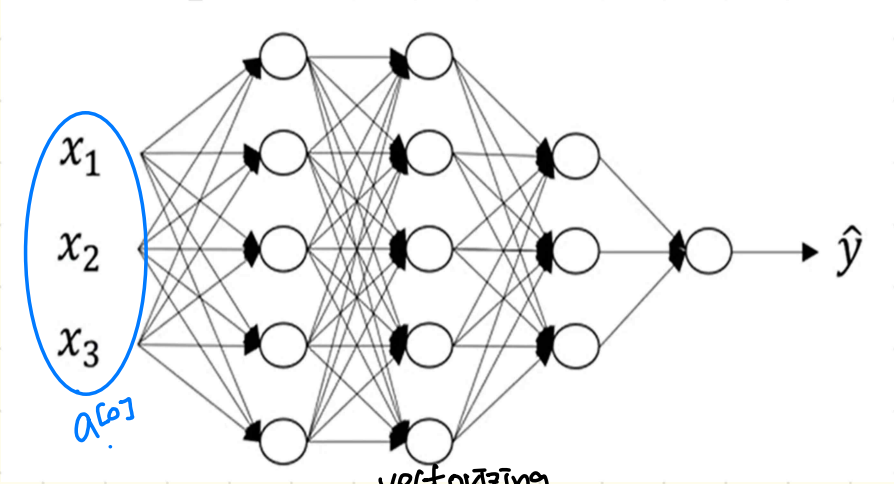
$n^{[3]} = 3$, $n^{[4]} = n^{[L]} = 1$

$a^{[l]} = g^{[l]}(z^{[l]})$ = activation in layer l

$a^{[0]} = x$, $a^{[L]} = \hat{y}$

$w^{[l]}, b^{[l]}$: weights for $z^{(l)}$

<Forward Propagation in a Deep Network>



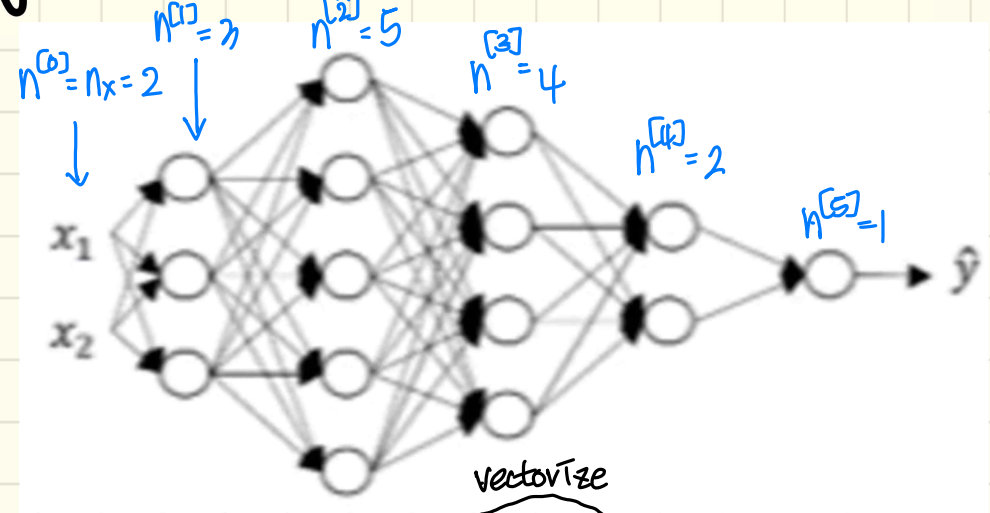
vectorizing

$$z^{[1]} = w^{[1]} a^{[0]} + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = w^{[2]} a^{[1]} + b^{[1]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$
$$\Rightarrow z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[1]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\hat{Y} = g(Z^{[4]}) = A^{[4]}$$

for $l = 1$ to 4

<Getting your Matrix Dimensions Right>

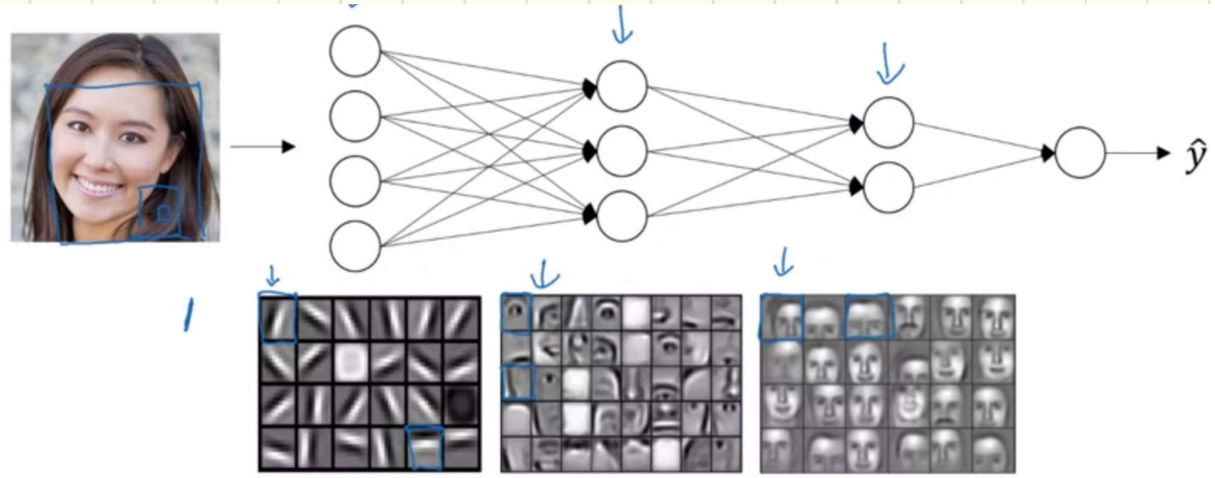


vectorize

$$z^{[1]} = w^{[1]} x + b^{[1]} \Rightarrow z^{[l]} = (n^{[l]}, 1)$$
$(3,1)\quad (3,2)\ (2,1)\quad (3,1)$
$$w^{[l]} = (n^{[l]}, n^{[l-1]})$$
$$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$
$(5,1)\quad (5,3)\ (3,1)\quad (3,1)$
$$b^{[l]} = (n^{[l]}, 1)$$
$$dw^{[l]} = (n^{[l]}, n^{[l-1]})$$
$$db^{[l]} = (n^{[l]}, 1)$$

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$(3, m)\quad (3,2)\ (2, m)\quad (3,1)$
$$\Rightarrow Z^{[l]}, A^{[l]} : (n^{[l]}, m)$$
$$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$$

<Why Deep Representations?>



직관 1) 네트워크가 깊어질수록 더 많은 특징을 잡아낼 수 있다. 낮은 층에서는 간단한 특징을 찾고 깊은 층에서는 낮은 층에서 탐지한 간단한 것들을 종합해서 더 복잡한 특징을 찾아낼 수 있다.

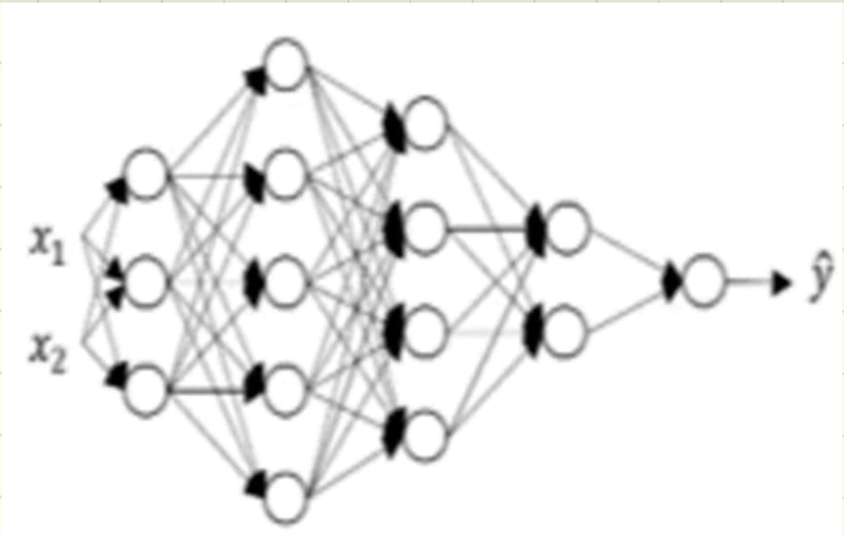
Circuit theory and deep learning

Informally: There are functions you can compute with a "small" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

직관 2) 순환이론에 따르면, 얕은 네트워크보다 깊은 네트워크에서 더 계산하기 쉬운 함수가 있다.

<Building Blocks of Deep Neural Networks>

* Forward and Backward functions



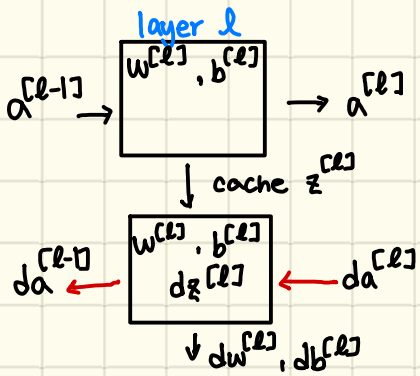
Layer l : $W^{[l]}$, $b^{[l]}$

Forward · Input $a^{[l-1]}$, Output $a^{[l]}$

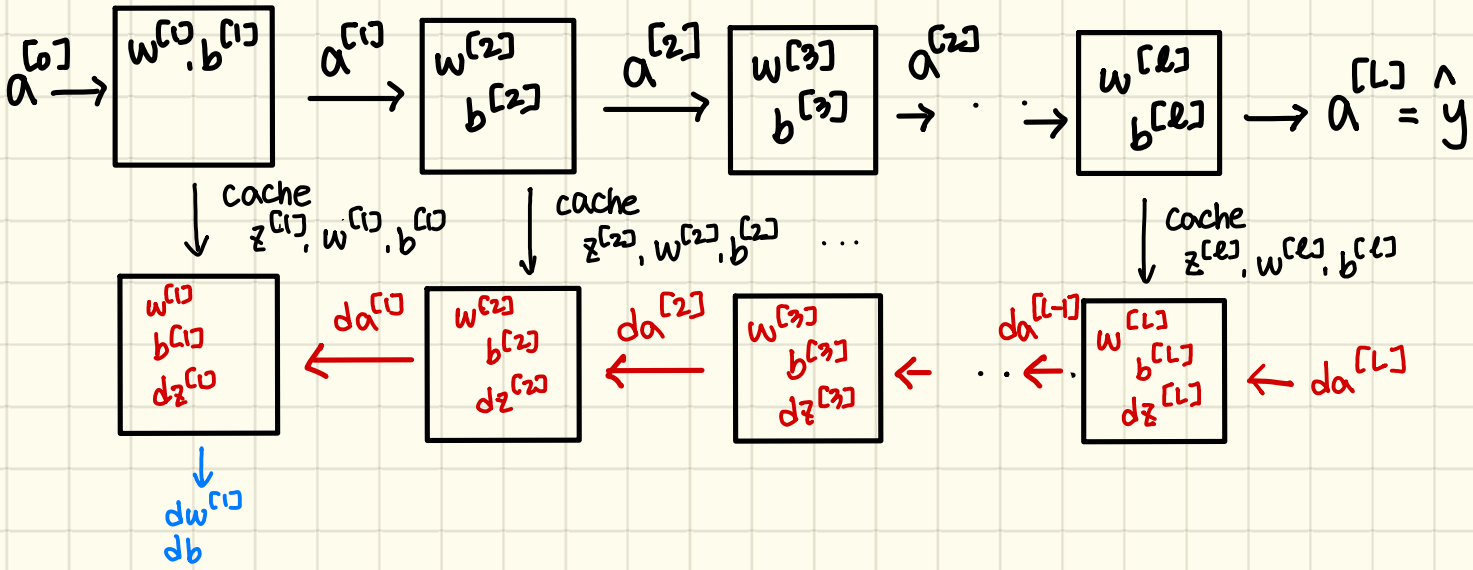
cache $z^{[l]}$

Backward · Input $da^{[l]}$

Output $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$



확장해보면,



<Foward and Backward Propagation>

* Forward propagation for layer l

- Input : $a^{[l-1]}$
- Output : $a^{[l]}$, cache ($z^{[l]}$) ⊕ coding 관점에서 cache $w^{[l]}, b^{[l]}$

$$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$

vectorize →

$$Z^{[l]} = W^{[l]} \cdot A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

* Backward propagation for layer l

- Input : $da^{[l]}$
- Output : $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l]} = W^{[l]T} \cdot dz^{[l]}$$
$$\rightarrow dz^{[l]} = W^{[l+1]T} \cdot dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$$

$$dz^{[l]} = dA^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dz^{[l]}, axis=1, keepdims=True)$$
$$dA^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

<Parameters VS Hyperparameters>

* What are hyperparameters?

- parameters : $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \cdots$
- Hyperparameters : learning rate α
  # iterations
  # hidden layer L
  # hidden units $n^{[1]}, n^{[2]}, \cdots$
  Choice of activation functions
  ...