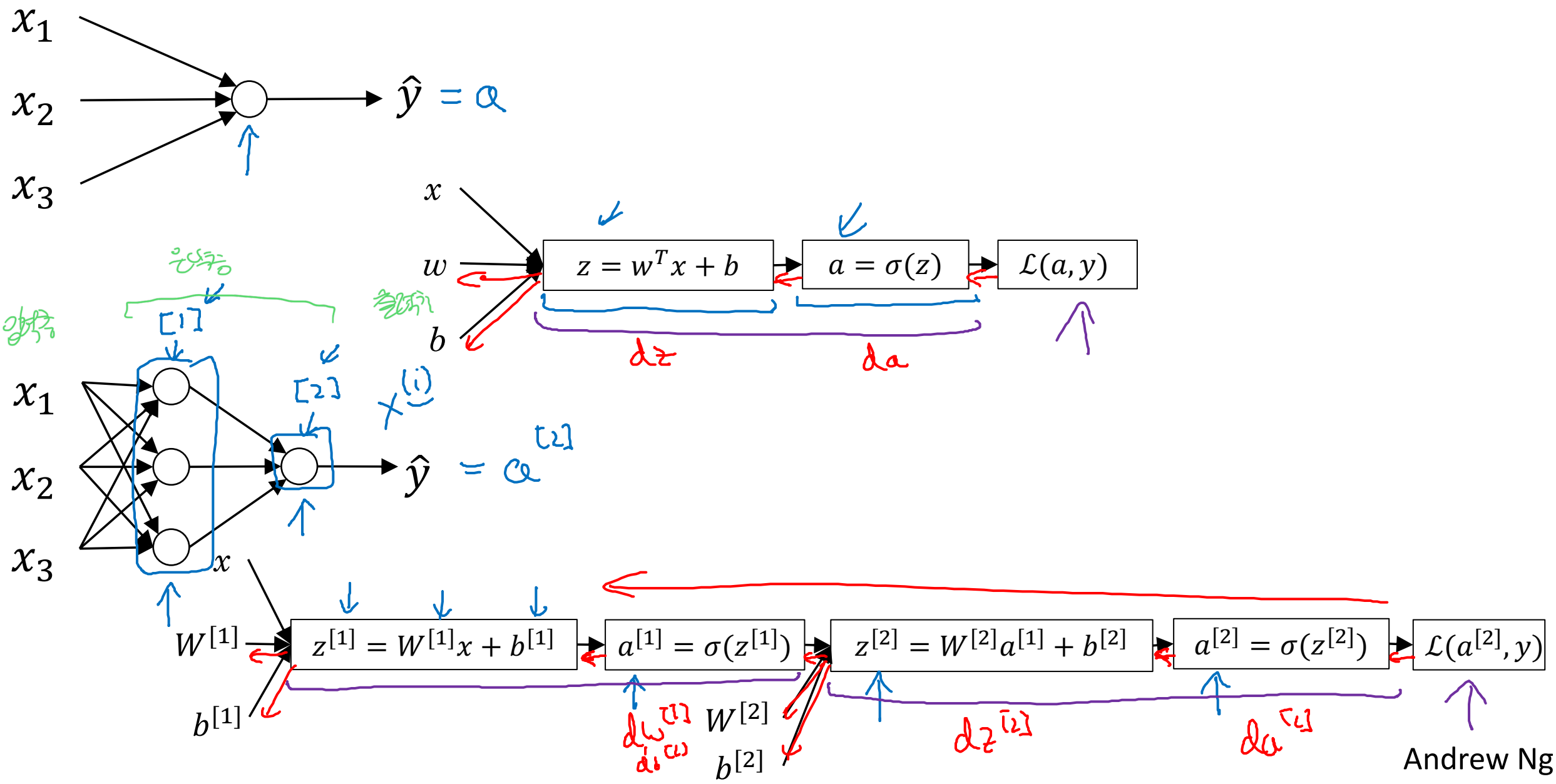
Copyright Notice

deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?



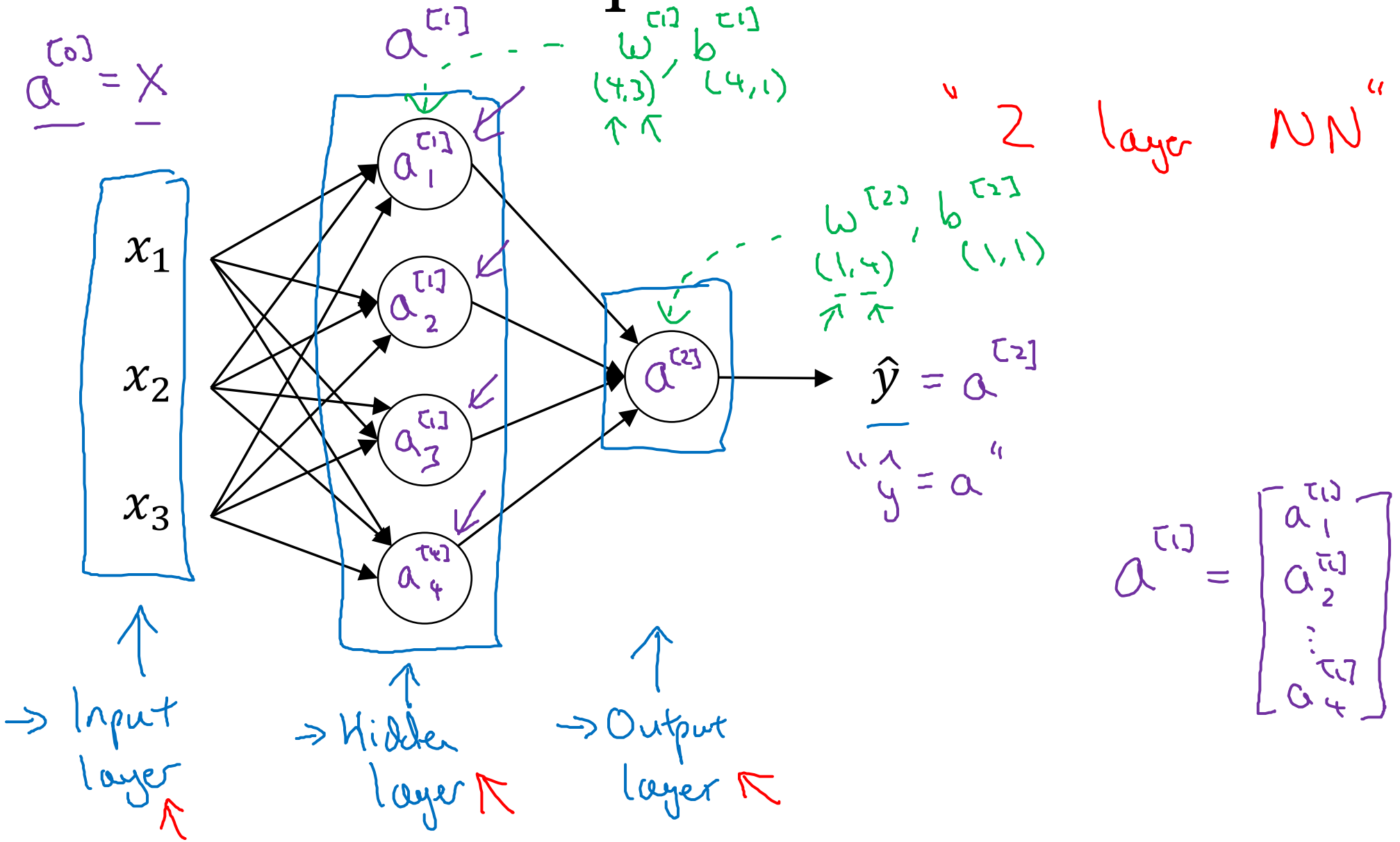
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation



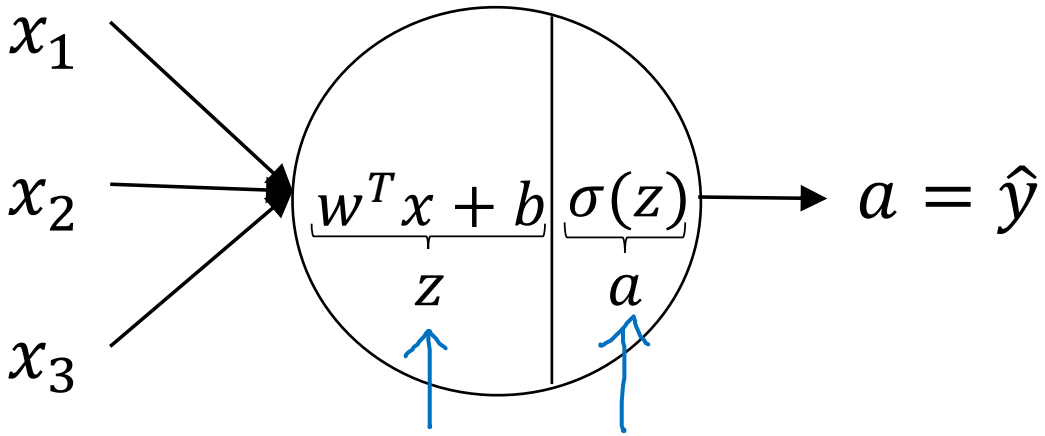
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

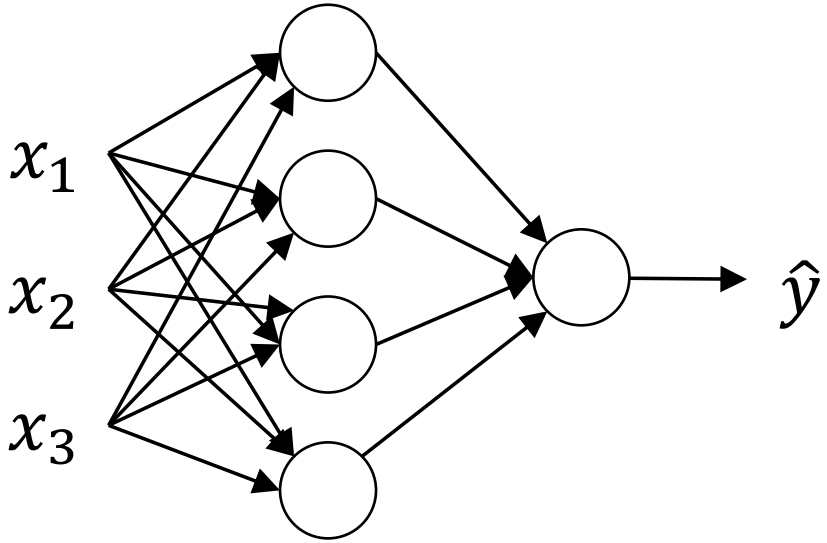
Computing a Neural Network's Output

Neural Network Representation

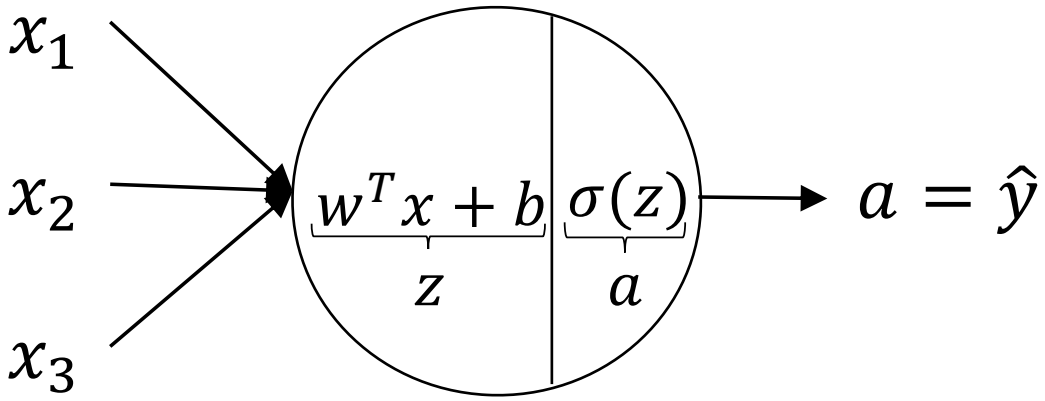


$$z = w^T x + b$$

$$a = \sigma(z)$$

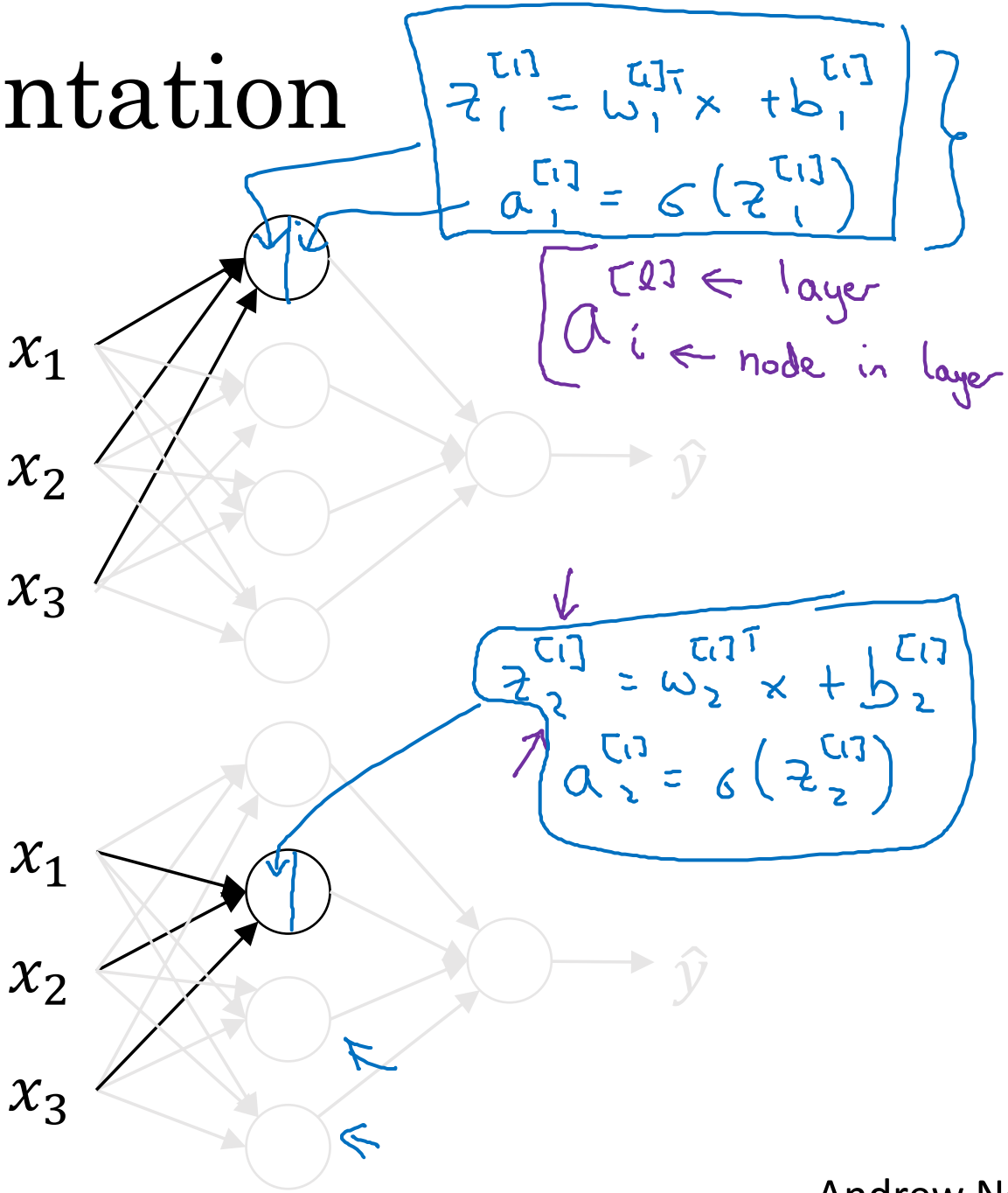


Neural Network Representation



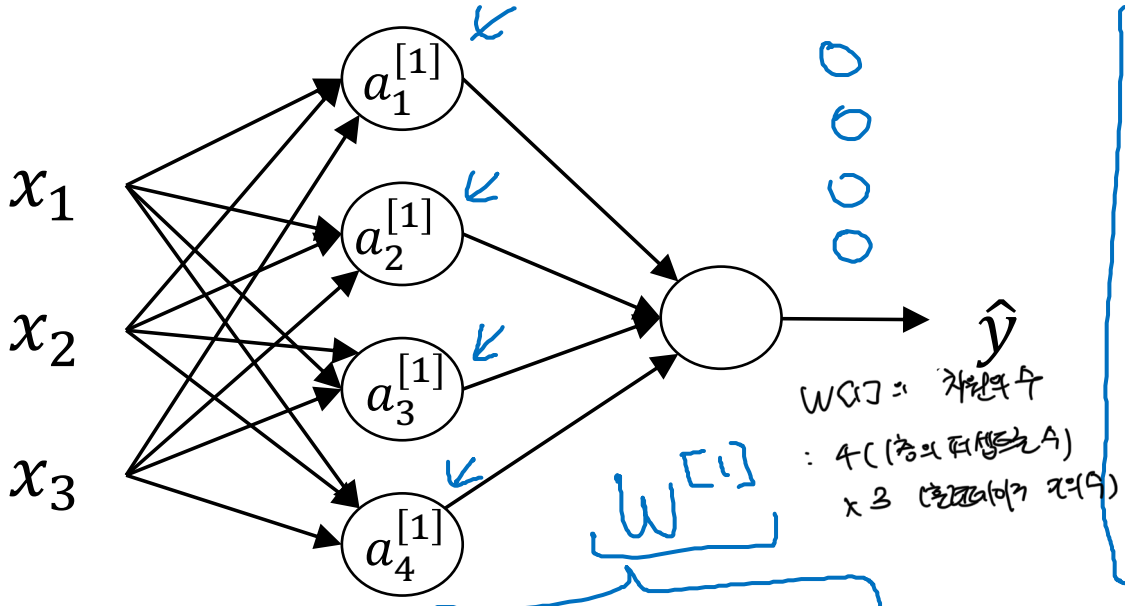
$$z = w^T x + b$$

$$a = \sigma(z)$$



Andrew Ng

Neural Network Representation



$W^{[1]}$: 차원의 수 : 4(층의 퍼셉트론 수) x 3 (입력데이터 개수)

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

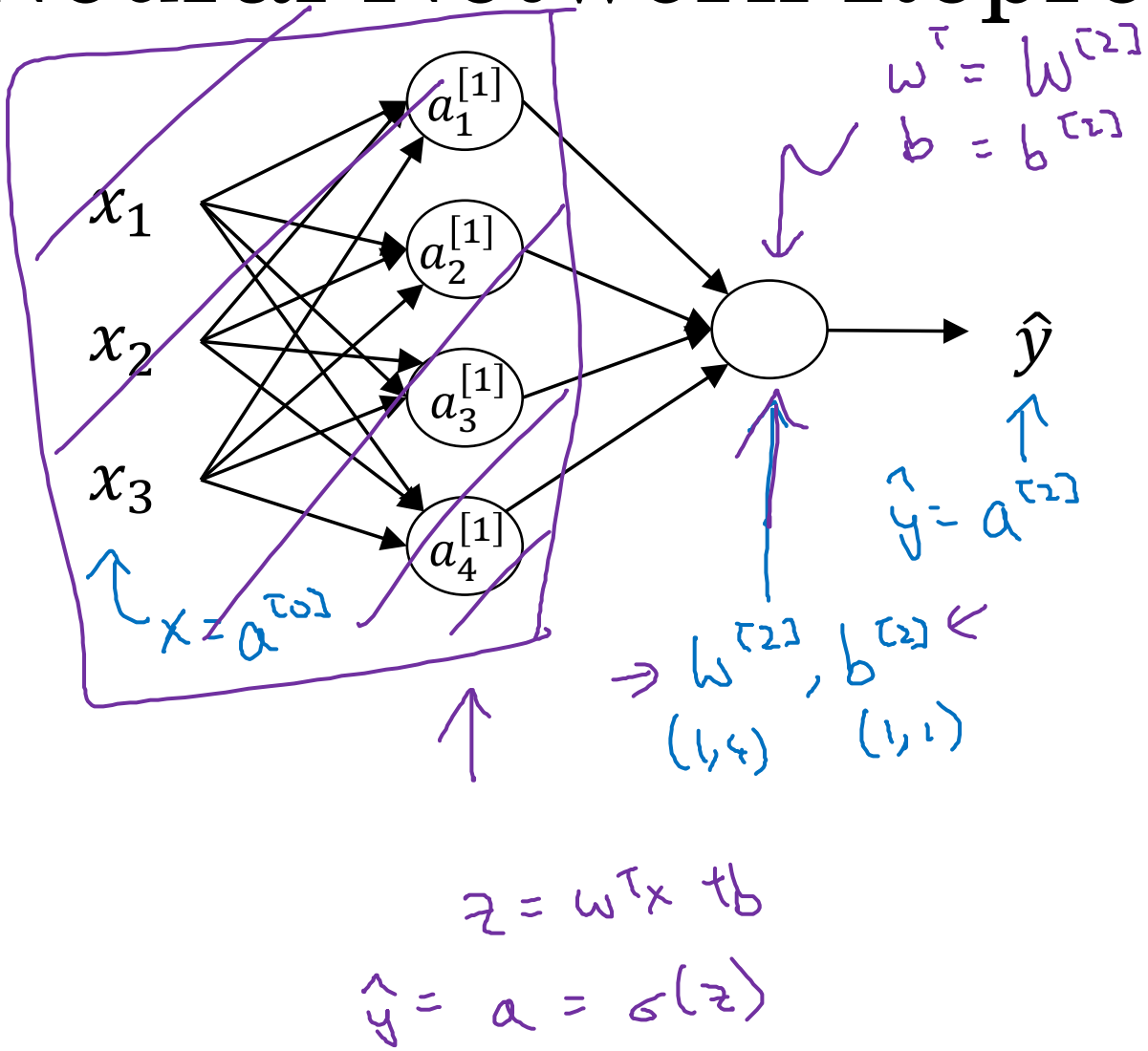
$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x +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

(4, 3) · x → (3, 1) + $b^{[1]}$ (4, 1) → (4, 1)

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

Andrew Ng

Neural Network Representation learning



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)

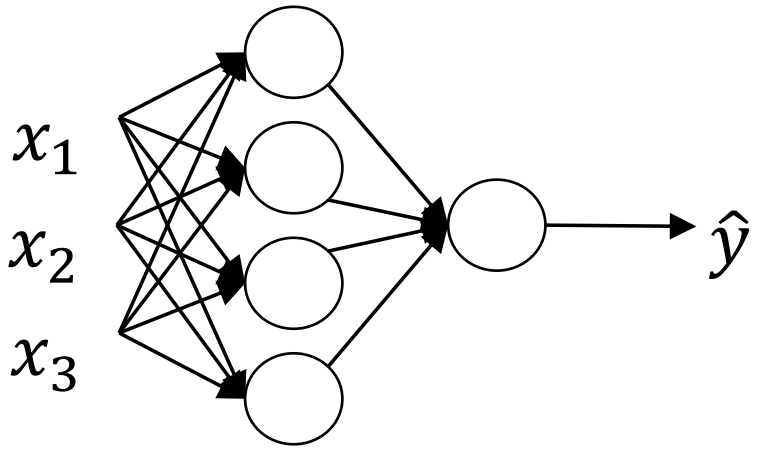
$$a^{[2]} = \sigma(z^{[2]})$$
(1,1) (1,1)

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



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

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

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

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

$\vdots$

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

$a^{[2](i)}$ ← example i, layer 2

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 across multiple examples

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

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

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

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

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

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

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

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

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

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

(n_x, m)

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

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

training examples

hidden units.

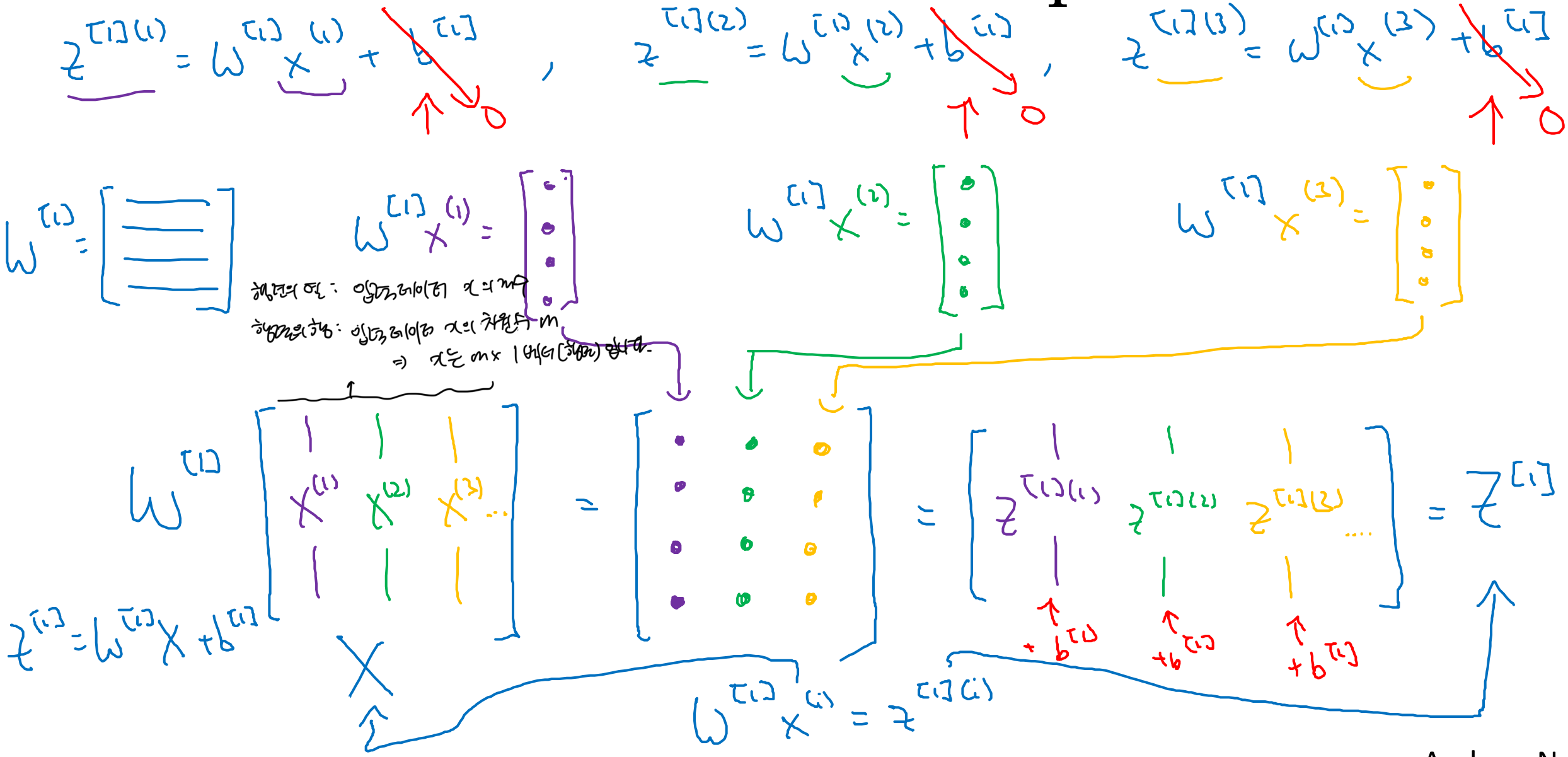
#hidden units

deeplearning.ai

One hidden layer Neural Network

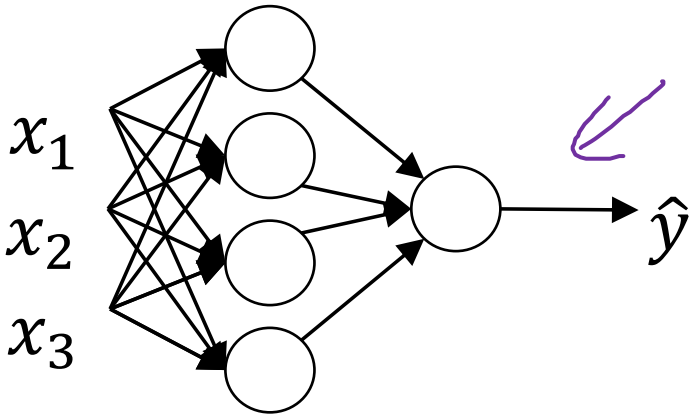
Explanation for vectorized implementation

Justification for vectorized implementation



Andrew Ng

Recap of vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

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

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

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

$A^{[0]} = X = a^{[0]}$ $\quad x^{(i)} = a^{[0](i)}$

$$Z^{[1]} = W^{[1]}X + b^{[1]} \quad \leftarrow \quad W^{[1]}A^{[0]} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

Andrew Ng

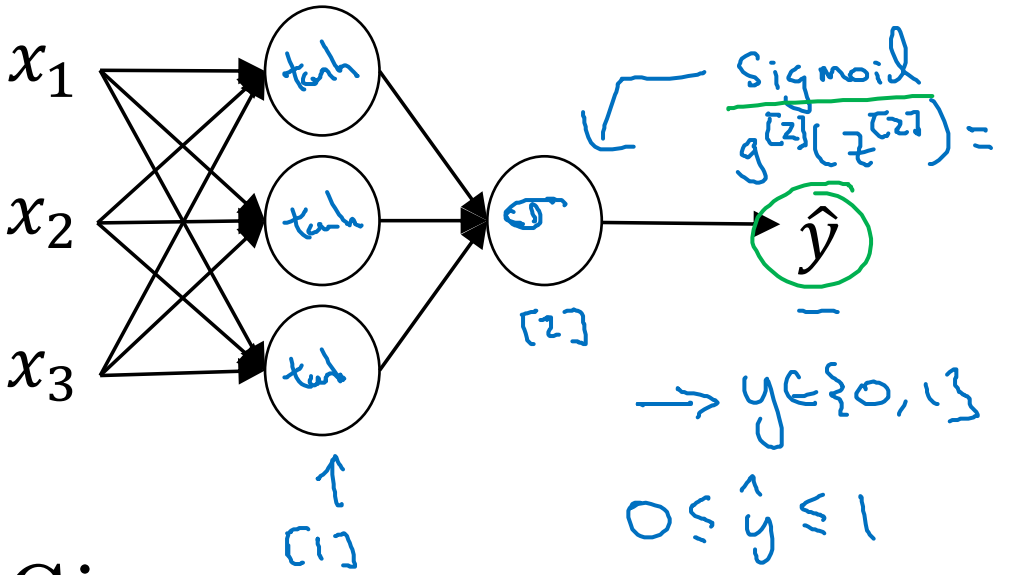
deeplearning.ai

One hidden layer Neural Network

Activation functions

Activation functions

: 활성화 함수 (Activation function)을 통과
이때 변수 w, b를 학습하고 함수를 결정할 수 있다.



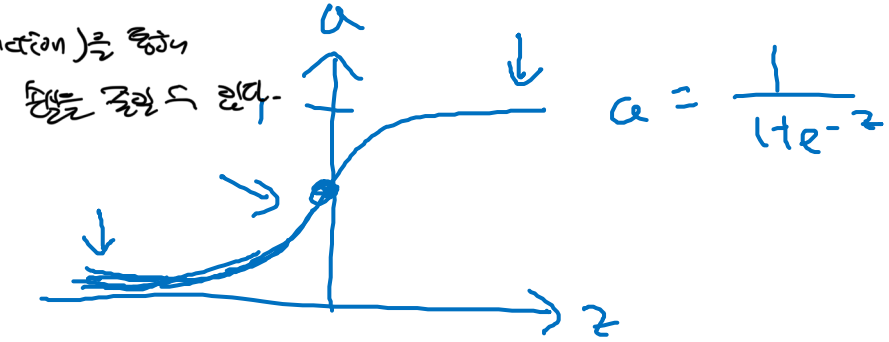
$g^{[1]}(z^{[1]}) = \tanh(z^{[1]})$

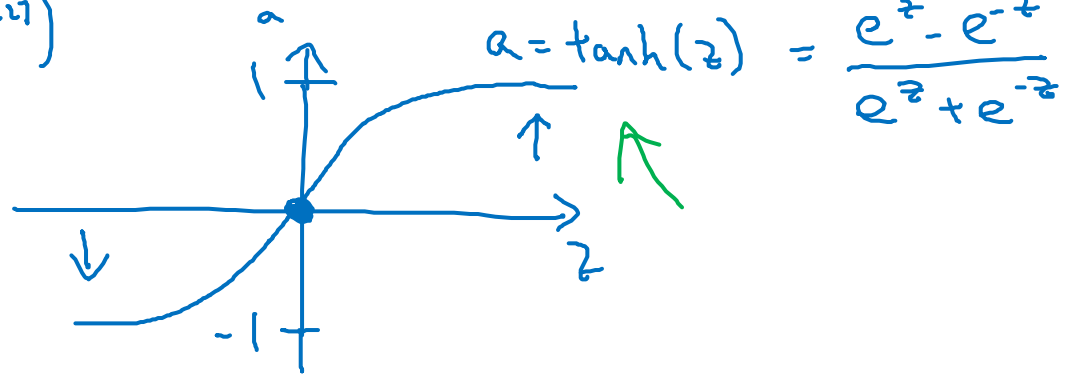
Sigmoid
$g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

$\rightarrow y \in \{0, 1\}$

$0 \le \hat{y} \le 1$

$-1 \quad 1$





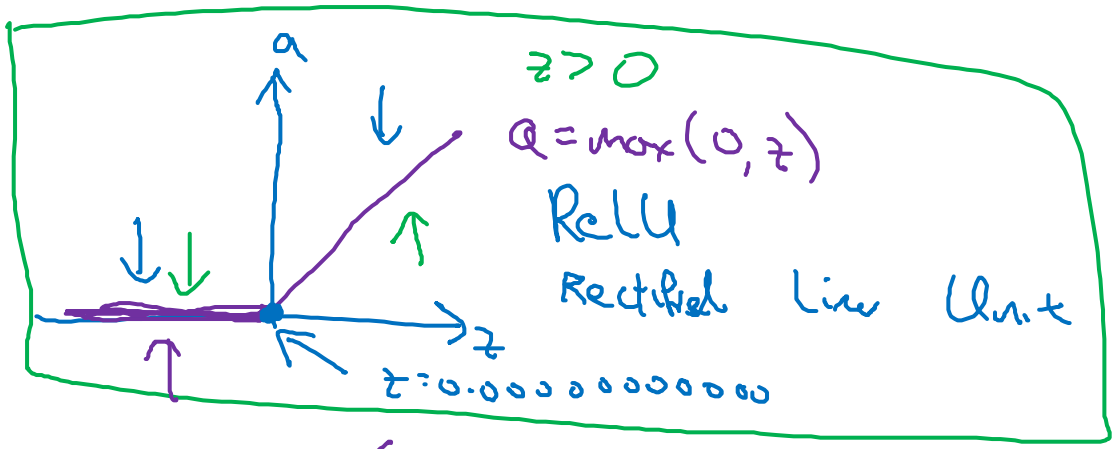
Given x:

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

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

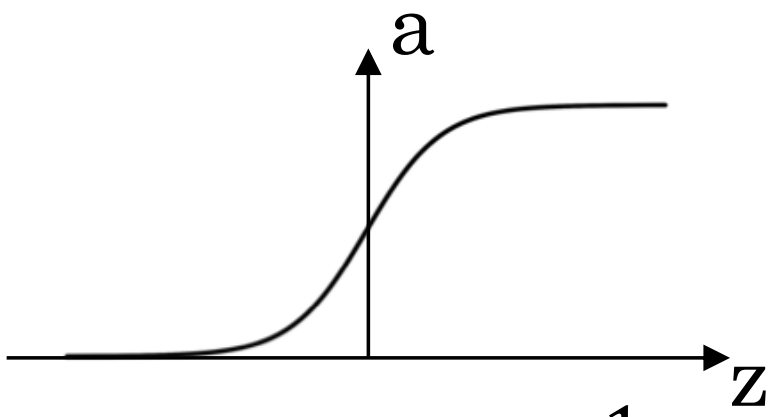
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

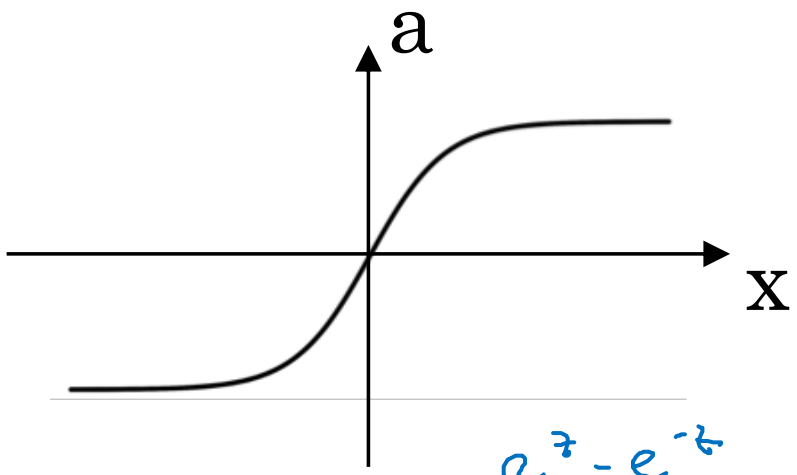


Andrew Ng

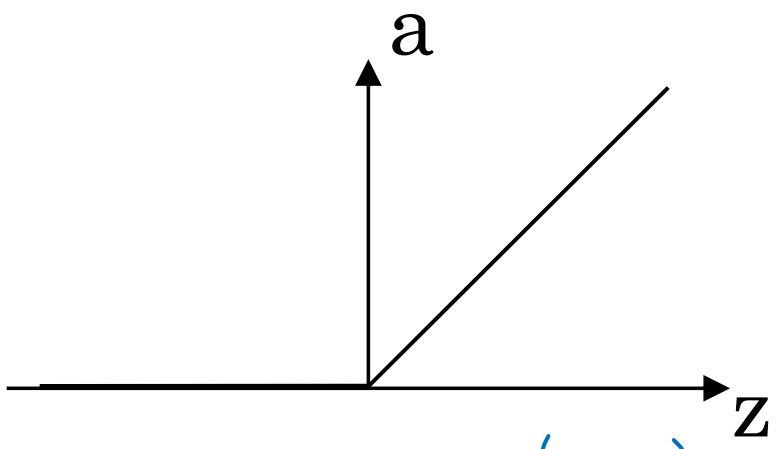
Pros and cons of activation functions



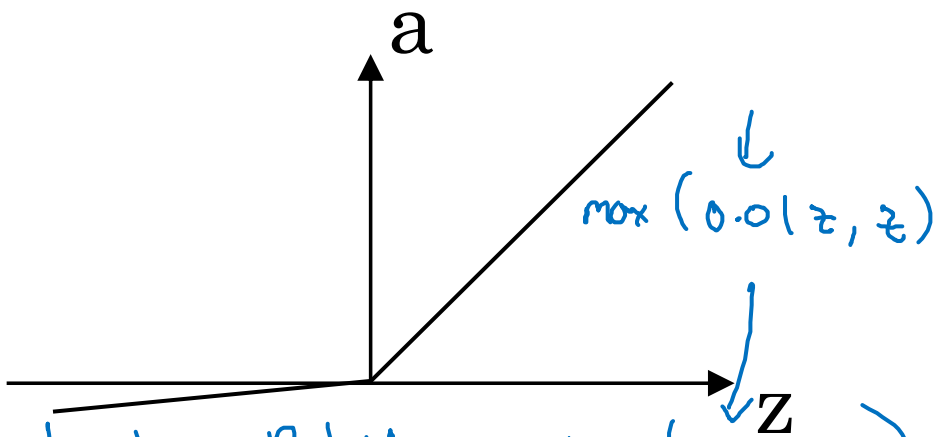
sigmoid: $a = \frac{1}{1+e^{-z}}$



tanh: $a = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



ReLU $a = \max(0, z)$



Leaky ReLU $a = \max(0.01z, z)$

deeplearning.ai

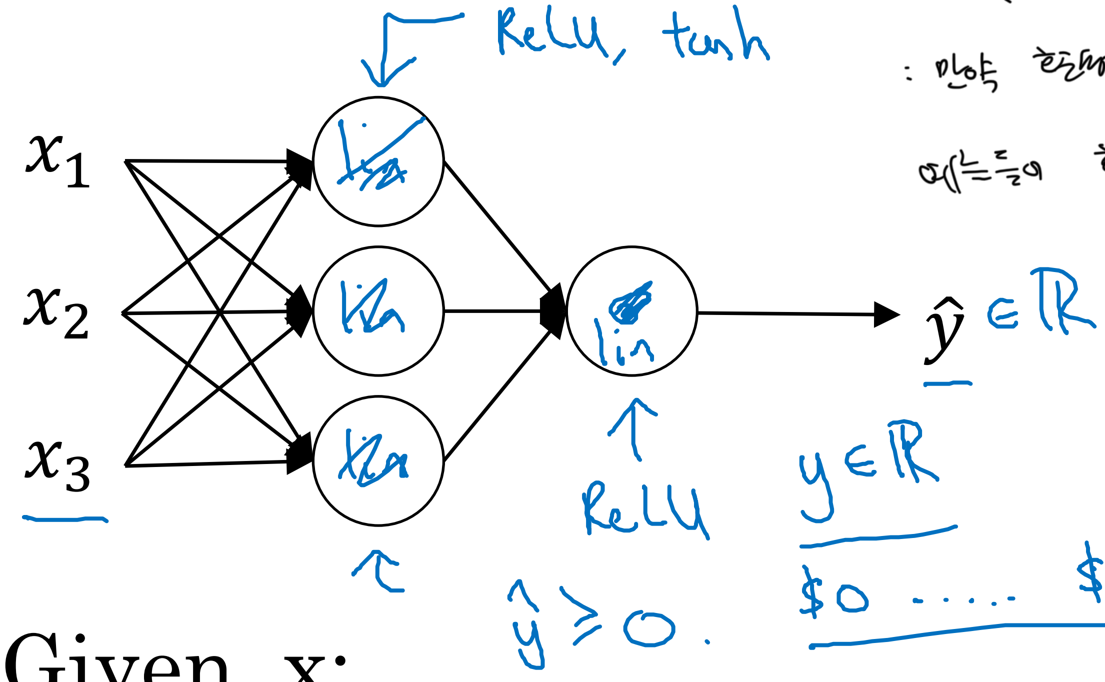
One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function

< 활성화 함수가 모두 비선형 함수인 이유 >

: 만약 활성화 함수가 선형함수라면, 신경망을 거쳐 학습하는 의미가 사라지기 때문



예를 들어 활성화 함수가 y=x라면 ⇒

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

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

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

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

$$= W'x + b'$$

↳ 신경망을 거쳐도 학습을 하지 않는다

$W^{[1]}x + b^{[1]}$로 계산이 된다.

$g(z) = z$

신경망을 거치는 학습에 의미가 사라지고

기계학습이 보다 간단 로지스틱 회귀와 동일한 의미가 되게 된다.

Given x:

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

non linear activation function 이다

g( ) 나 σ( ) 사용함,

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

$g(z) = z$

"linear activation function"

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

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

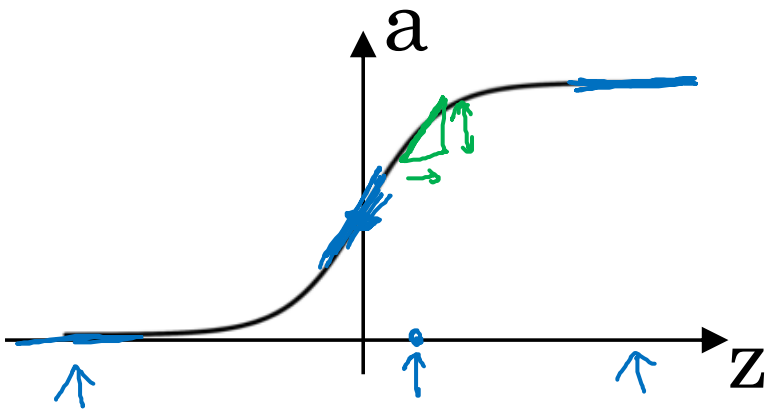
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Derivatives of activation functions

Sigmoid activation function



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

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

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

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

$= g(z)\left(1 - g(z)\right)$ ←

$= a(1-a)$

$g'(z) = a(1-a)$ ↑

$z = 10$. $g(z) \approx 1$

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

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

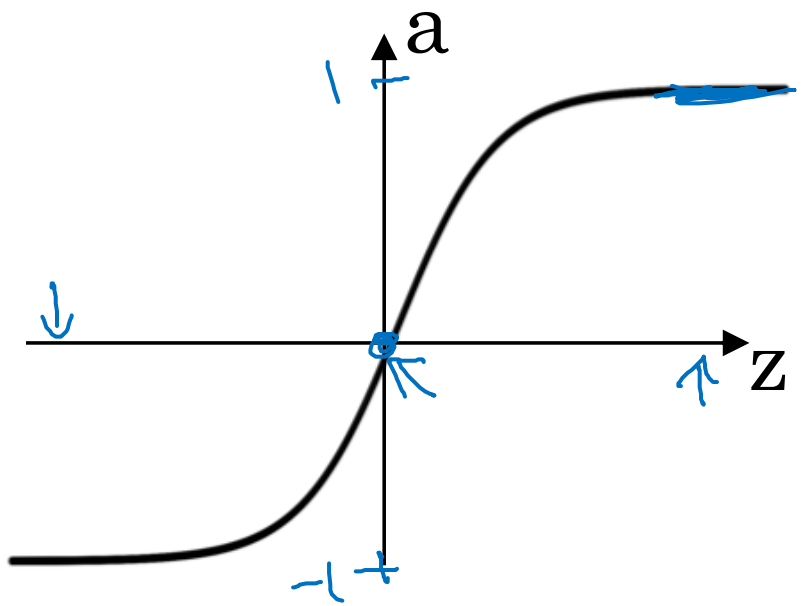
$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

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

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

Andrew Ng

Tanh activation function



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

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

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

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

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

$g'(z) \approx 0$

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

$g'(z) \approx 0$

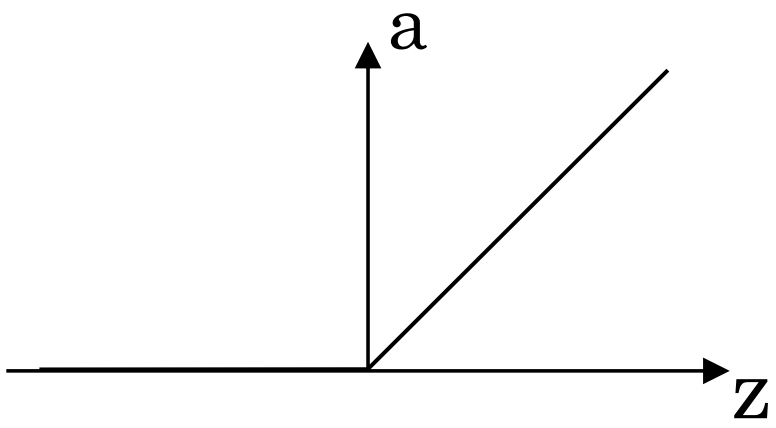
$z = 0 \quad \tanh(z) = 0$

$g'(z) = 1$

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

Andrew Ng

ReLU and Leaky ReLU



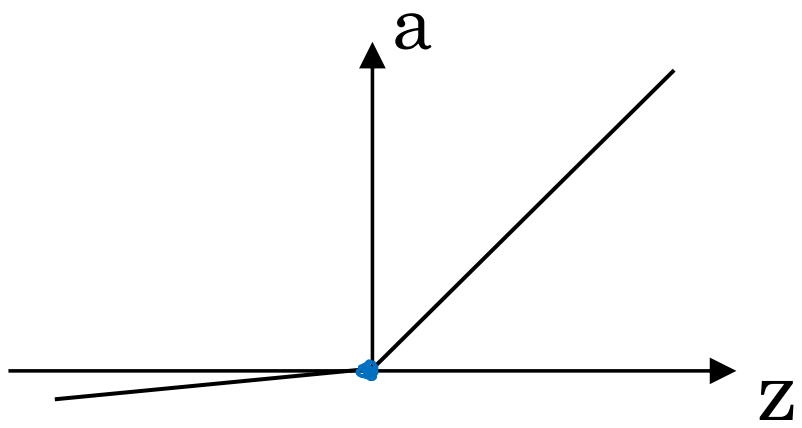
ReLU

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

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

~~undef if $z = 0$~~

$z = 0.000\ldots0$



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

deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks

< 은닉계층이 1개인경우 >

Parameters: $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

$n_x = n^{[0]}$ (input feature), $n^{[1]}$ (hidden feature), $n^{[2]} = 1$ (output feature)

이진분류시

Cost function: $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{n} \mathcal{L}(\hat{y}, y)$

$\hat{y} \leftarrow a^{[2]}$

Gradient descent:

→ Repeat {

수렴할때까지 아래반복 →

Compute predicts $(\hat{y}^{(i)}, i = 1 \ldots, m)$

$dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

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

$b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$

$W^{[2]} := \ldots$ $b^{[2]} := \ldots$

}

Andrew Ng

Formulas for computing derivatives

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:

$dz^{[2]} = A^{[2]} - Y$

$Y = [y^{(1)}\ y^{(2)} \cdots y^{(m)}]$

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

$(n^{[2]},)$

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

$(n^{[2]}, 1)$

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

$(n^{[1]}, m)$ $(n^{[1]}, m)$

element-wise product

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

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

$(n^{[1]}, 1)$ $(n^{[1]},)$

reshape

Andrew Ng

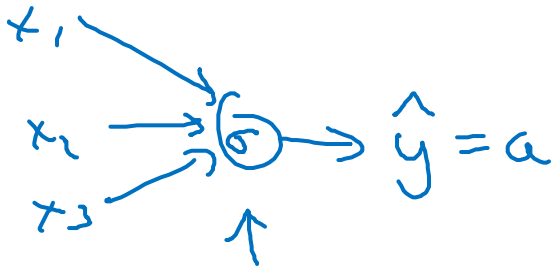
deeplearning.ai

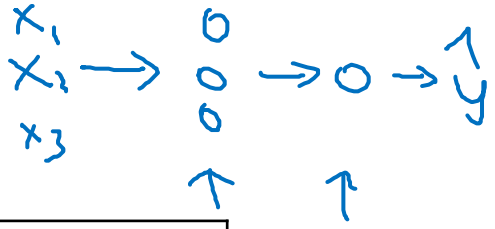
One hidden layer Neural Network

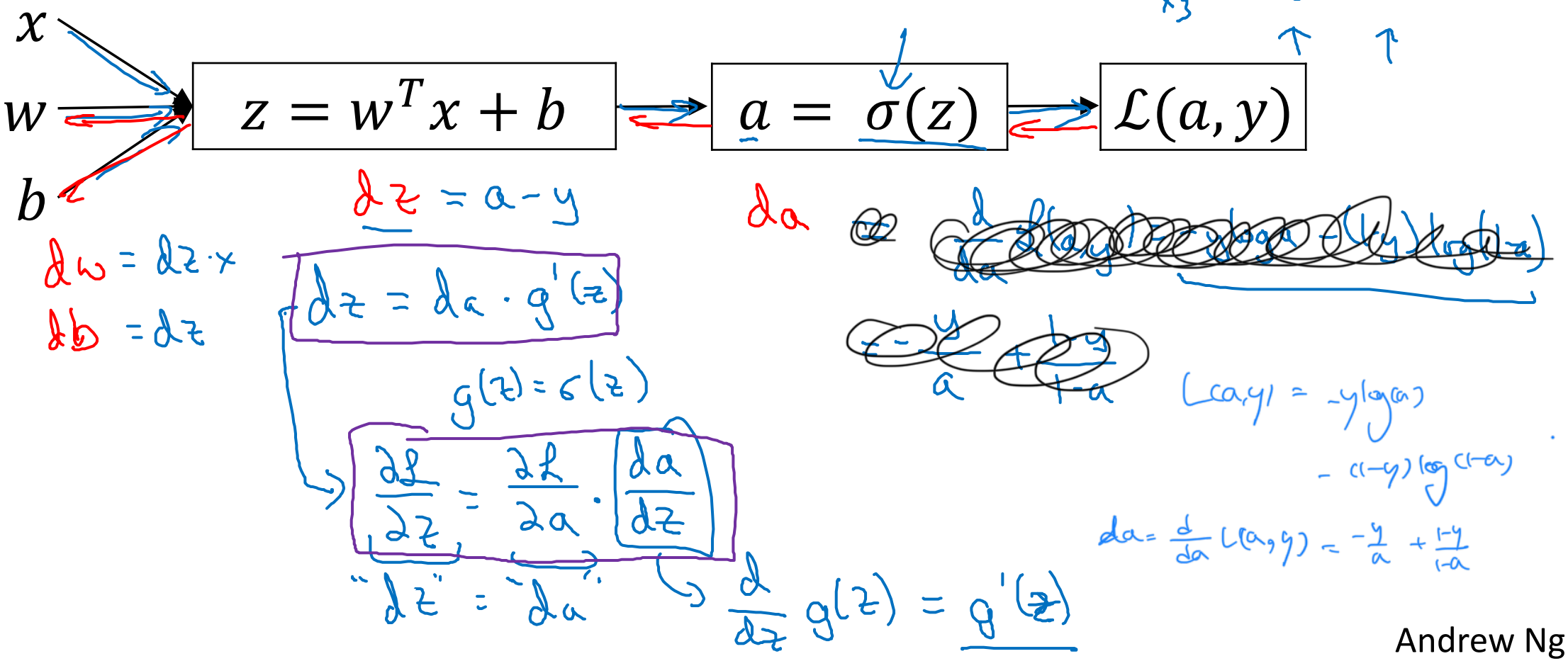
Backpropagation intuition (Optional)

Computing gradients

Logistic regression

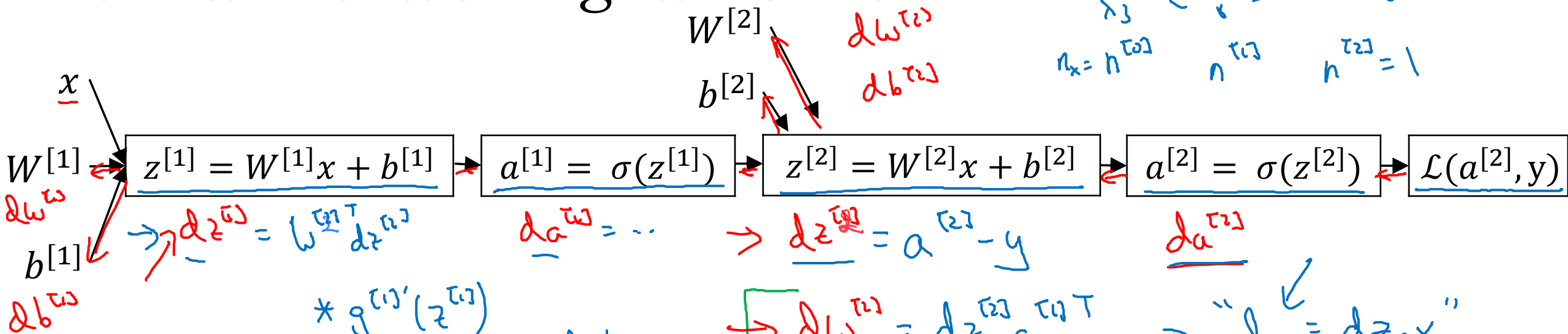


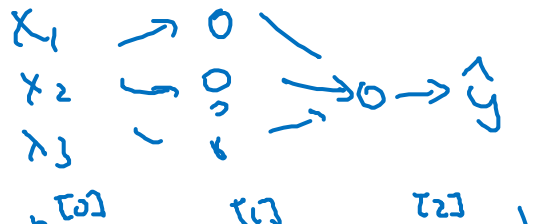




Andrew Ng

Neural network gradients





$dW^{[2]}$ $db^{[2]}$ $dW^{[1]}$ $db^{[1]}$

$dz^{[1]} = W^{[2]T} dz^{[2]}$ $da^{[1]} = \cdots$ $dz^{[2]} = a^{[2]} - y$ $da^{[2]}$

$* g^{[1]\prime}(z^{[1]})$ ← element-wise product

$dW^{[2]} = dz^{[2]} a^{[1]T}$ → "$dw = dz \cdot x$"

$db^{[2]} = dz^{[2]}$

$W^{[2]}$의 차원: $(n^{[2]}, n^{[1]})$

$z^{[2]}, dz^{[2]}$의 차원: $(n^{[2]}, 1)$ $-(1,1)$

$z^{[1]}, dz^{[1]}$의 차원: $(n^{[1]}, 1)$

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$

$(n^{[1]}, 1) = (n^{[1]}, n^{[2]})\ (n^{[2]}, 1) * (n^{[1]}, 1)$

foo
dfoo

$W^{[1]}$
$dW^{[2]}$

$dW^{[1]} = dz^{[1]} \cdot x^T$

$db^{[1]} = dz^{[1]}$ $a^{[0]T}$

* back propagation 시 차원 맞추는 연습하기 필요

Andrew Ng

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^{T}$$

$$db^{[1]} = dz^{[1]}$$

Vectorized Implementation:

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

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

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

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

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

Andrew Ng

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$(n^{[1]}, 1)$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

<vectorized implementation>

$$dZ^{[2]} = A^{[2]} - Y$$

$$J(\cdot) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$$

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

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

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

elementwise product

$(n^{[1]}, m)$ $(n^{[1]}, m)$ $(n^{[1]}, m)$

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

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

Andrew Ng

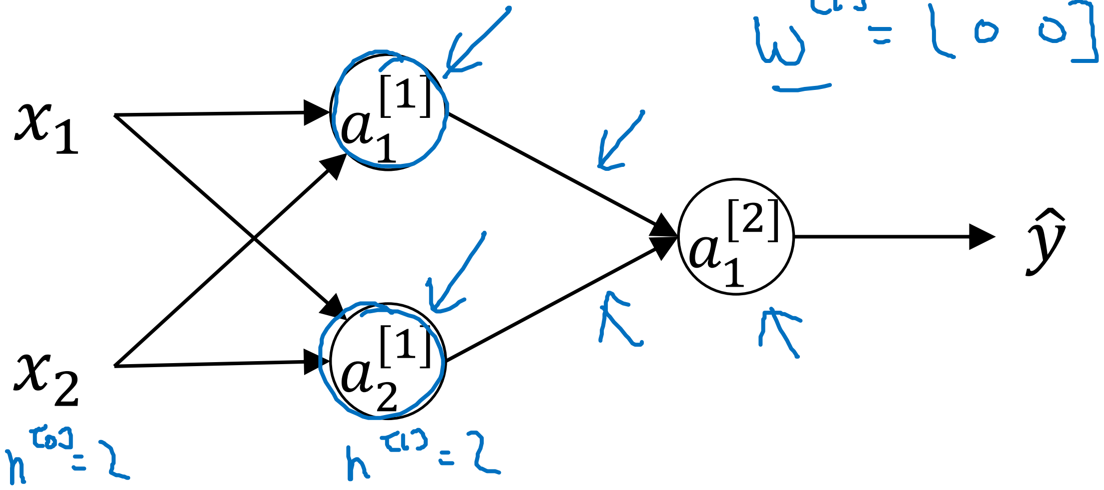
deeplearning.ai

One hidden layer Neural Network

Random Initialization

What happens if you initialize weights to zero?

< 매개변수 W를 0으로 초기화하면 안 되는 이유 >



$W^{[2]} = [0 \; 0]$

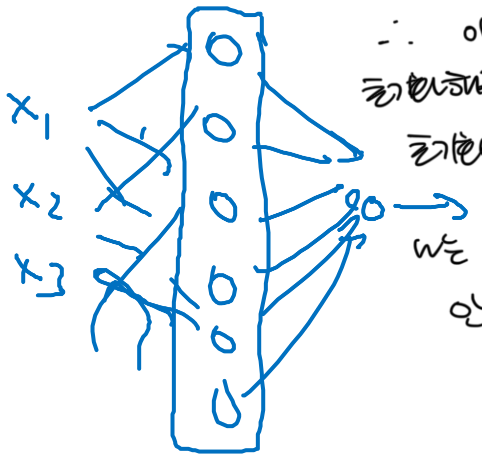
$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$a_1^{[1]} = a_2^{[1]}$ $dz_1^{[1]} = dz_2^{[1]}$

$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $W^{[1]} = W^{[1]} - \alpha dW$

매개변수를 초기화 할때, w, b의 값을 0으로 초기화하면, (활성화함수로 선형함수를 사용할때처럼) 신경망은 계속 학습을 하나 안하나 결과가 같게 된다.

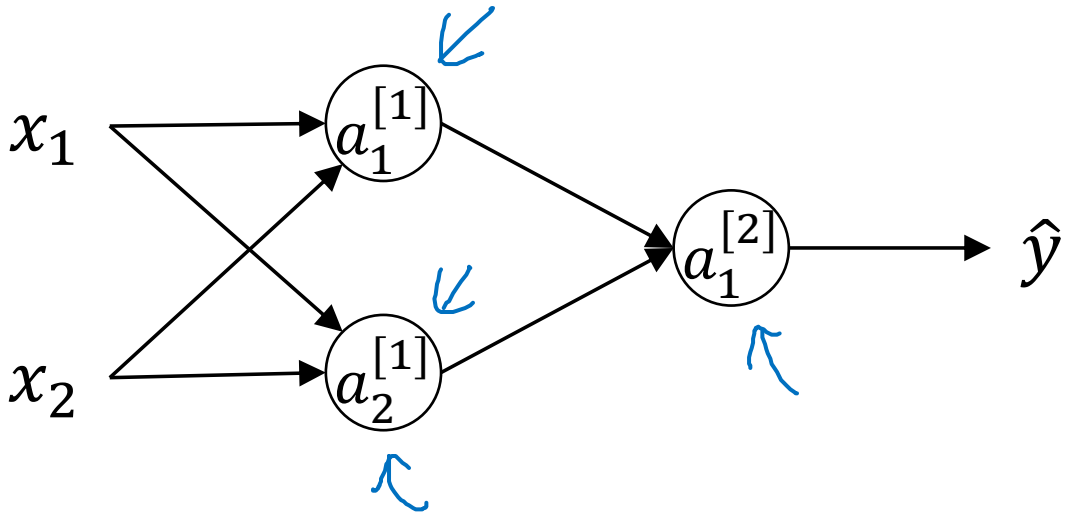
Symmetric



∴ 이러한 이유로 매개변수를 초기화할때, b는 0으로 초기화 해도 되지만 w는 0으로 초기화하면 안된다.

$W^{[1]} = \begin{bmatrix} \cdots \\ \cdots \end{bmatrix}$

Random initialization



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

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

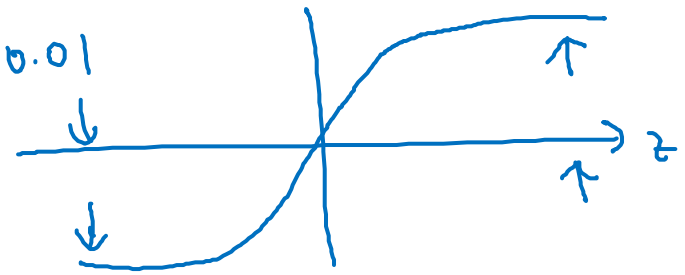
$W^{[2]} = np.random.randn((1,2)) * 0.01$

$b^{[2]} = 0$

0.01 / 100 ?

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

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



Andrew Ng