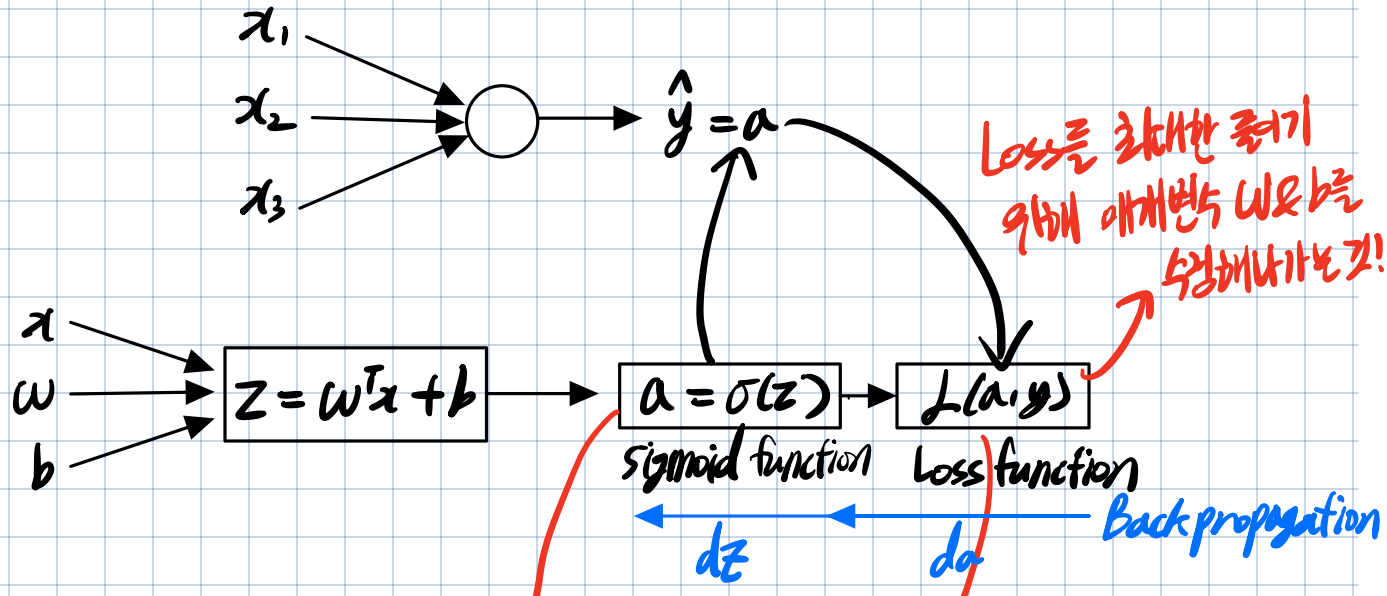
Week 3. Shallow Neural Networks

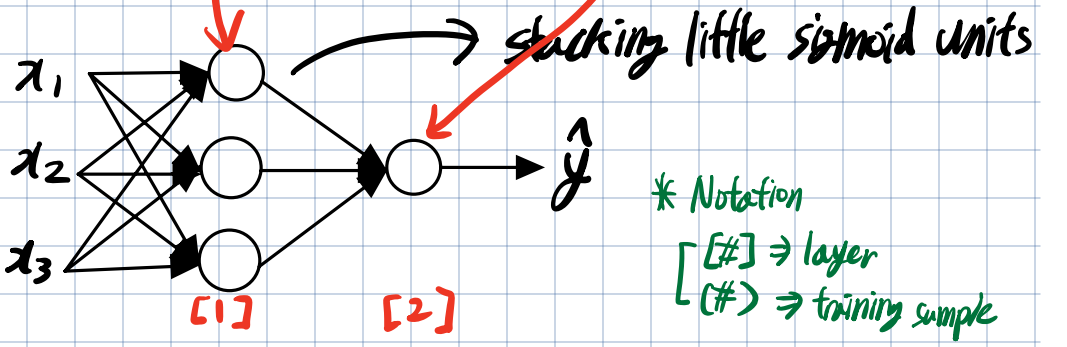
1. One hidden layer Neural Network

• Neural Networks Overview

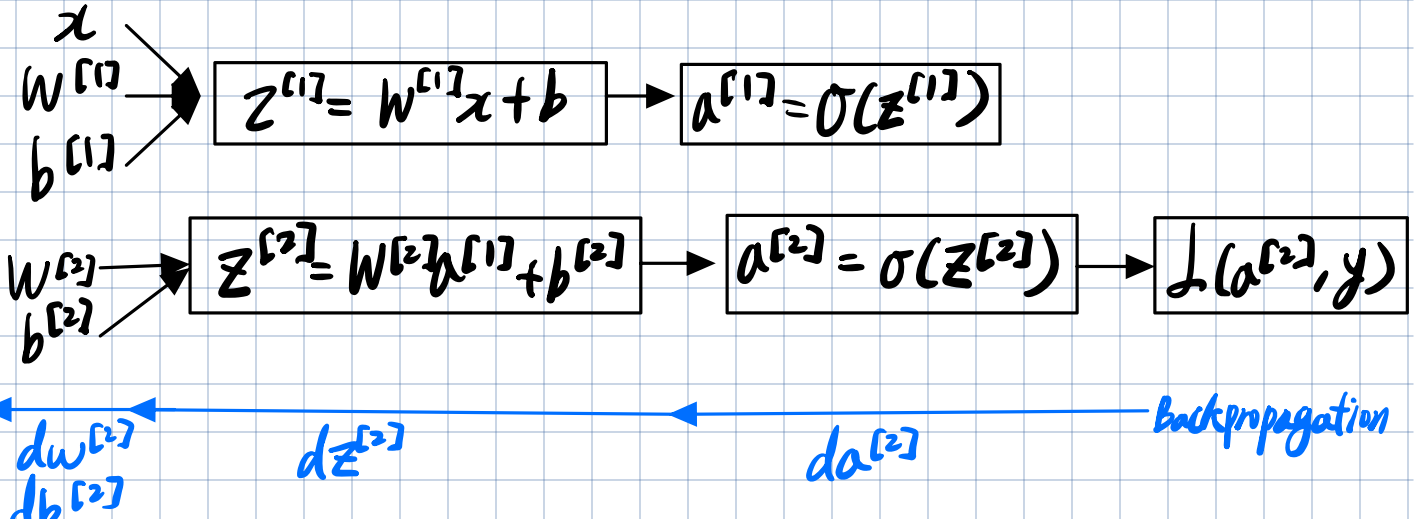
– Logistic Regression Model



– Neural Network



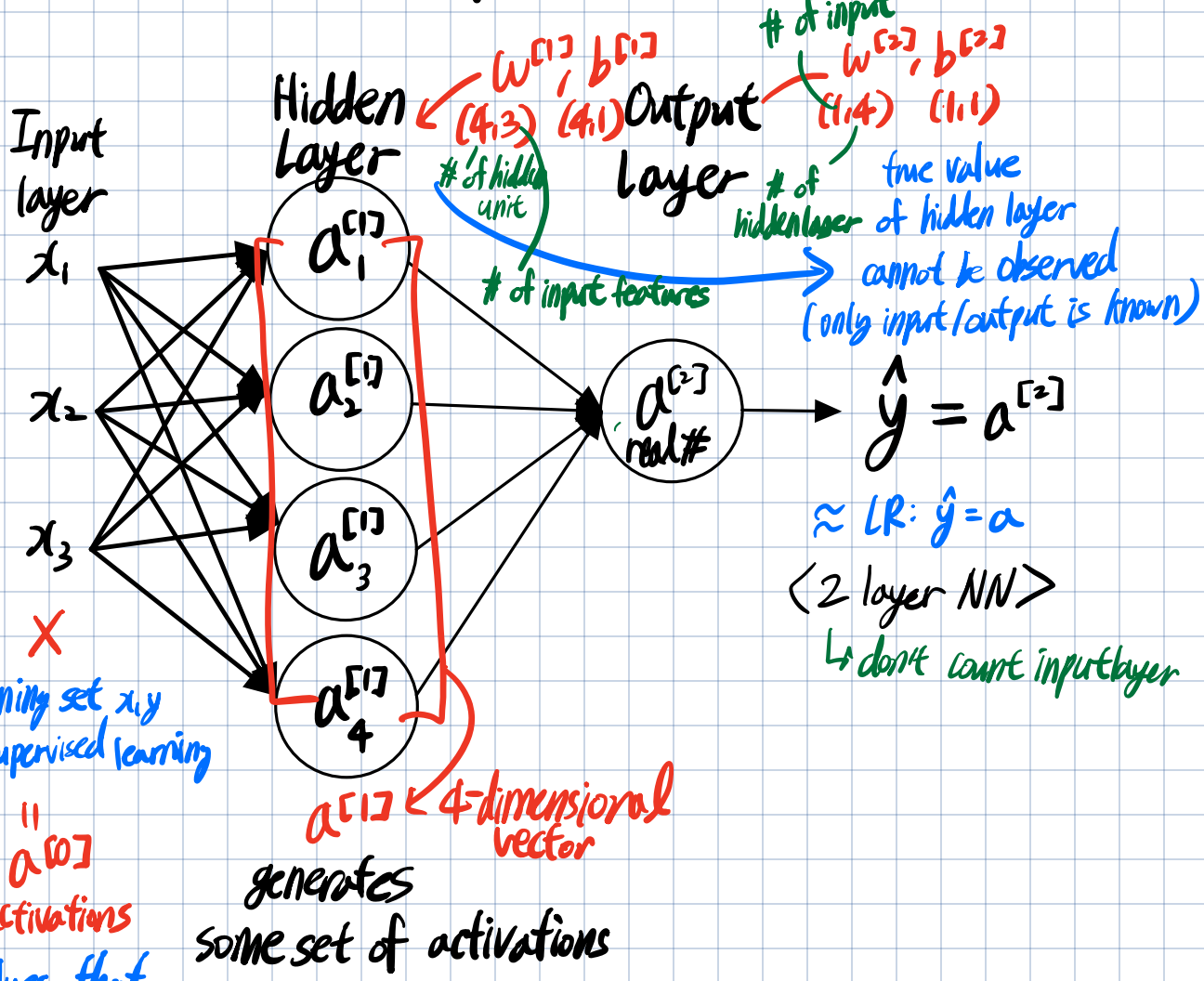
* Notation
  - [#] ⇒ layer
  - (#) ⇒ training sample



Logistic Regression: calculate z & a

Neural Network: calculate z & a multiple times
& finally compute loss function

• Neural Network presentation



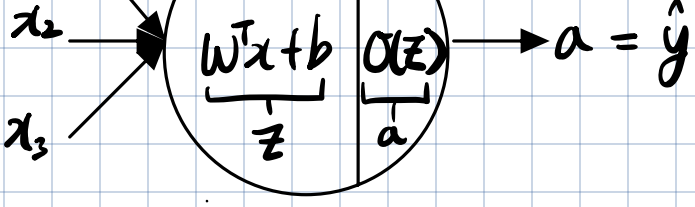
X
training set x, y
in supervised learning

||

$a^{[0]}$
activations
values that
different layers of NN
passing on to the
subsequent layers

generates
some set of activations

• Computing a Neural Network's output.



$z = w^T x + b$

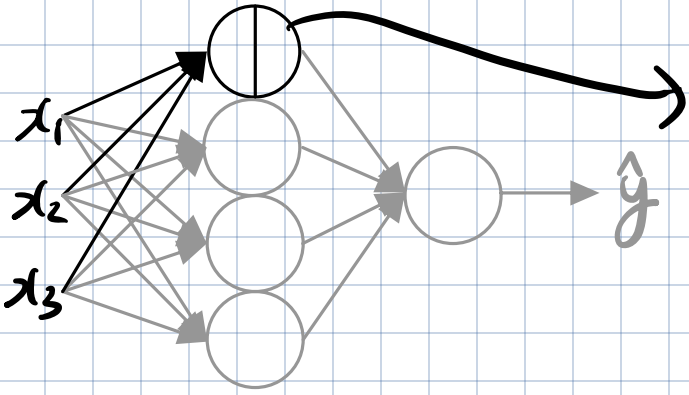
$a = \sigma(z)$

→ In NN, this computation repeats

— 1st node in hidden layer

* Notation

a^{m}_{n} ⇒ mth layer, nth node

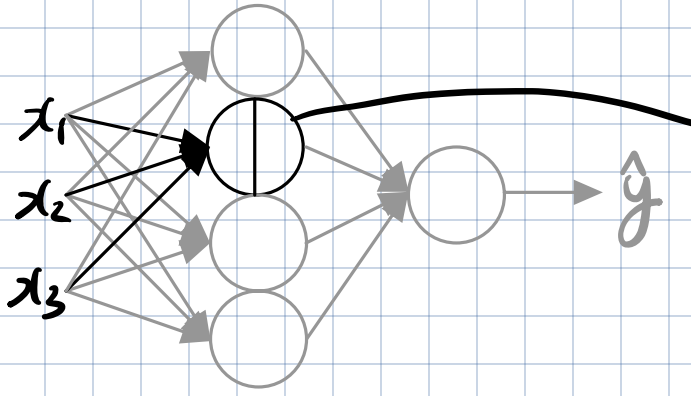


→ 2 steps of computation

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

R: $a_1^{[1]} = \sigma(z_1^{[1]})$

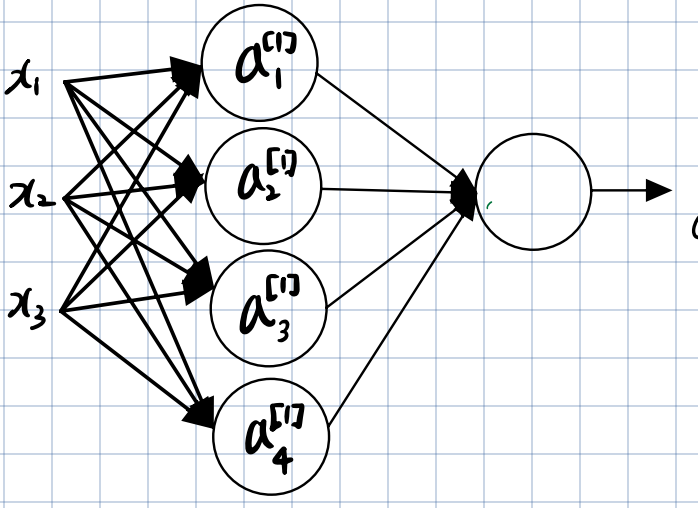
— 2nd node in hidden layer



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

R: $a_2^{[1]} = \sigma(z_2^{[1]})$

— Vectorization of z, w



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

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

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

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

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

⇒ stack row vectors as a matrix

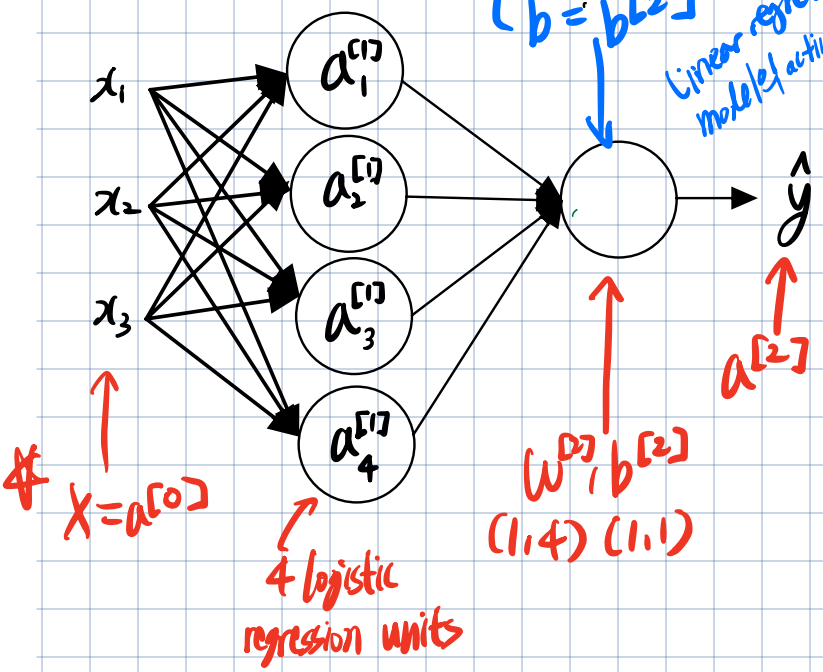
$$\begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ — & w_3^{[1]T} & — \\ — & w_4^{[1]T} & — \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T}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

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

4x3 matrix

$W^{[1]}$

$b^{[1]}$



LR: $w = w^{[2]}$, $b = b^{[2]}$

Linear regression model of activation 값

In 1 hidden layer NN...

Given input x:

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

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

hidden layer computation

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

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

output layer computation

In LR...

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$

• Vectorizing across multiple examples

<One training sample>

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

<m training samples>

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

[Unvectorized]

~~for~~ $i = 1$ to m,

$z^{[1](i)} = w^{[1]} x^{(i)} + b^{[1]}$ ($x^{(i)} = a^{[0](i)}$)

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

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

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

[Vectorized]

stack vector x horizontally to form matrix

$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$, $X \in \mathbb{R}^{n_x \times m}$

different input features

$Z^{[1]} = w^{[1]} X + b^{[1]}$ ($X = A^{[0]}$)

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

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

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

각 training sample에 대한 activation

$A^{[2]}$, $Z^{[2]}$도 마찬가지로 stacking을 통해 matrix로 나타내는 것임

각 training sample에 대한 feature로 계산한 것임!

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

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

activation of 1st training sample of first hidden unit (node)

hidden units

node

training samples

- Explanation for vectorized implementation
  - Justification for vectorized implementation

$z^{[1](1)} = w^{[1]} x^{(1)} + b^{[1]}$ (for simplification $b^{[1]} = 0$)

$z^{[1](2)} = w^{[1]} x^{(2)} + b^{[1]}$ ($b^{[1]} = 0$)

$z^{[1](3)} = w^{[1]} x^{(3)} + b^{[1]}$ ($b^{[1]} = 0$)

$w^{[1]}$: matrix

$w^{[1]} x^{(1)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $w^{[1]} x^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $w^{[1]} x^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$$W^{[1]}\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ b & b & b \end{bmatrix} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix} = Z^{[1]}$$

$$+b^{(1)} \quad +b^{(2)} \quad +b^{(3)}$$

X: training set

broadcasting

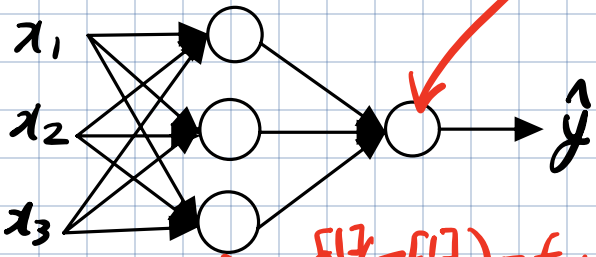
single train sample : $z^{[1](i)} = W^{[1]}x^{(i)}$ 일때, training sample인 x를 몇개 쌓든지 간에, $Z^{[1]}$도 z vector가 옆로 쌓여 matrix를 만들어짐 $+b^{(i)}$도 broadcasting을 통해 대해서 같은 결과

∴ forward propagation 시 training sample을 모두 옆로 쌓아 Z, a 값들로 옆로 쌓는다.

• Activation function

- Sigmoid function

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



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

Given x:

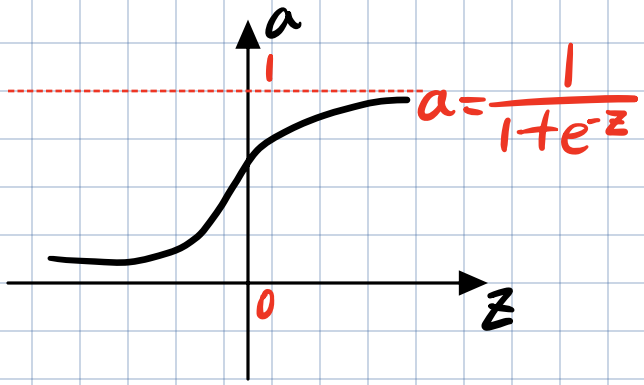
$z^{[1]} = W^{[1]}x + b^{[1]}$

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

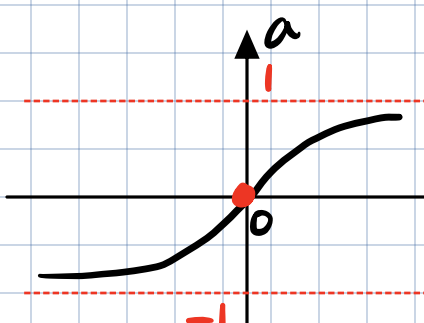
$z^{[2]} = W^{[2]}x + b^{[2]}$

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

↑ Sigmoid가 아닌 다른 비선형 함수가 올수도 있다.



- tanh function



almost always work better than sigmoid in hidden layer

the mean of activation ≈ 0

⇒ 데이터의 중심 : 0.5 → 0

다음층의 학습이 쉬워짐

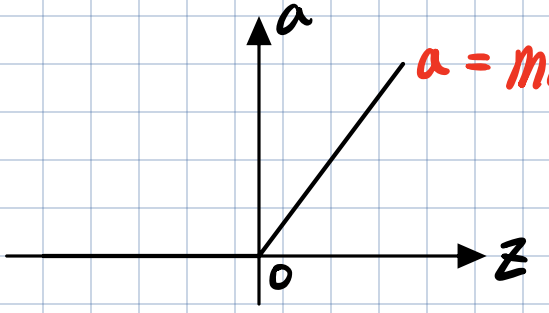
⇒ but output layer에서는 사용x

∵ y∈{0,1}이면 $-1 \le \hat{y} \le 1$보다 $0 \le \hat{y} \le 1$로 출력하는 것이 더 좋기 때문

∴ Sigmoid activation을 쓰는 한 가지 예외 : binary classification

- Sigmoid & tanh 모두 z가 굉장히 크거나 작으면 derivative(slope)는 0에 가까워진다 ⇒ slow down gradient descent

- ReLu(Rectified Linear unit) function



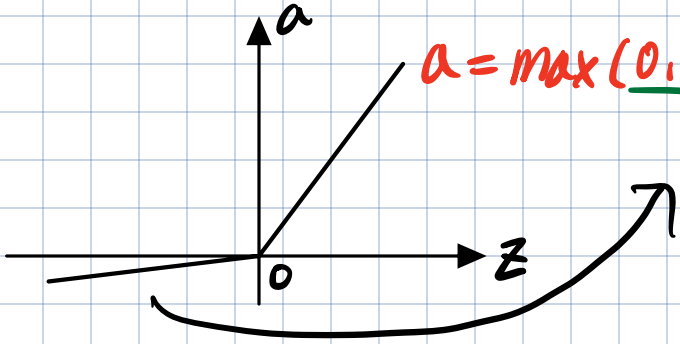
z=0 인경우의 dz는 컴퓨팅되지 않음

$dz \begin{cases} z>0 : 1 \\ z<0 : 0 \end{cases}$ ⇒ 이진 분류의 출력층이 아닌 경우, Relu function이 은닉층의 기본값으로 잘 쓰인다.

⇒ z가 음일때 dz가 0이 된다는 단점이 있다.

대안

- Leaky ReLu (ReLU보다 성능이 좋지만 잘 안쓰임)



parameter로 구현가능

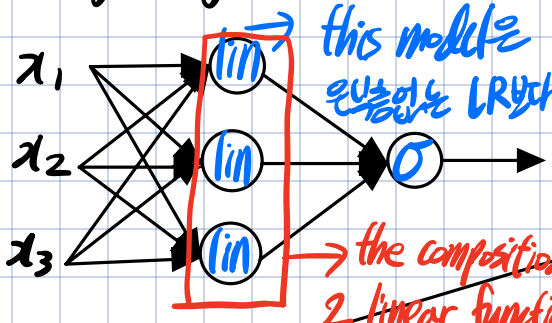
z가 음일때 약간 기울기를 준다.

- ReLu와 Leaky ReLu 둘다 대부분의 z에 대해 기울기(dz)가 0과 매우다름 ↗ 0이되면? gradient descent 속도↓

⇒ NN의학습속도↑

- 신경망을 고를 때에는 여러 option을 시도해보고 선택!

• Why do you need non-linear activation functions?



this model은 은닉층없는 LR보다 나아지지 않는다.

the composition of 2 linear functions is one linear function.

∴ NN이 필요없는 interesting 계산X

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

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

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

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

Let's say W', b'

$= W'x + b'$

⇒ $\hat{y}$를 x에 대한 선형 함수로 계산한다면?

<Forward Propagation>

Given x:

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

$a^{[1]} = g^{[1]}(z^{[1]})$ → Let's say $z^{[1]}$ (linear activation function / Identity activation function) $(g(z) = z^{[1]})$

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

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

If you use linear/identity activation functions,

⇒ the NN will output linear function of the input

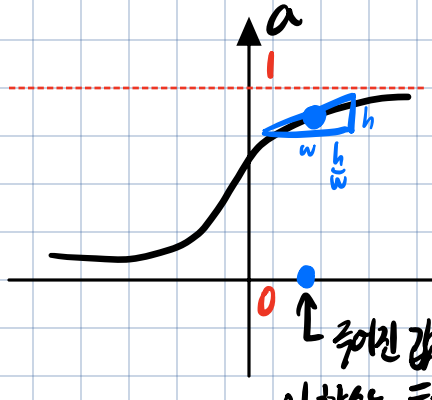
↳ 나중에 hidden layer가 많은 deep NN을 다룰 때 linear function을 쓰거나 activation function을 쓰지 않는다면 층이 없나 있는 것에 NN은 항상 항등 함수만 계산하기 때문에 은닉층이 없는 거나 다름없다.

- $g(z) = z$는 regression problem에 대한 machine learning을 (즉, y=실수 일 때) 할 때 or 대부분 output layer에 사용

e.g. house pricing: 은닉층은 linear ok, and 끝에 양수 내도록 출력층은 Relu 이용

• Derivatives of activation functions

- NN의 Back propagation을 계산하기 위해서는 활성화 함수의 도함수를 구해야 함
- Sigmoid activation function



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

주어진 값 z에 대해 이 함수는 특정한 기울기를 가짐

$\frac{d}{dz}g(z)$

= slope of $g(x)$ of x

$= g'(z)$

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

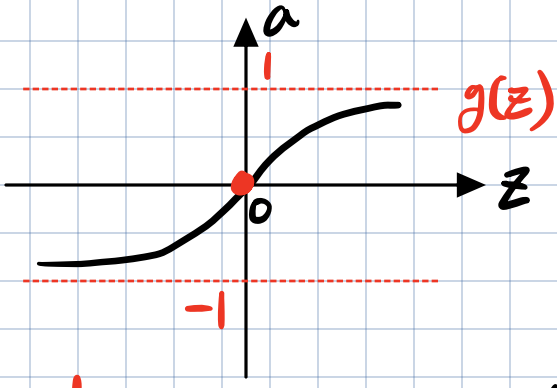
$= g(z)(1-g(z))$

$= a(1-a)$

e.g. when z is large $z=10$, $g(z)\approx 1$ — when z is small $z=-10$ $g(z)\approx 0$ — when z is zero $z=0$ $g(z)=\frac{1}{2}$

$\frac{d}{dz}g(z)\approx 1(1-1)\approx 0$ — $\frac{d}{dz}g(z)\approx 0\cdot(1-0)\approx 0$ — $\frac{d}{dz}g(z)=\frac{1}{2}(1-\frac{1}{2})=\frac{1}{4}$

- Tanh activation function

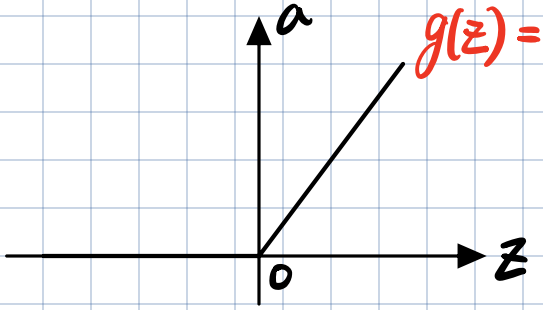


$$g(z)=\tanh(z)=\frac{e^z-e^{-z}}{e^z+e^{-z}}$$

$\frac{d}{dz}g(z)$ = slope of $g(x)$ of x = $g'(z)=1-(\tanh(z))^2=1-a^2$

e.g. when z is large $z=10$, $g(z)\approx 1$, $g'(z)=1-1^2\approx 0$ — when z is small $z=-10$, $g(z)\approx -1$, $g'(z)\approx 0$ — when z is zero $z=0$ $g(z)=0$, $g'(z)=1$

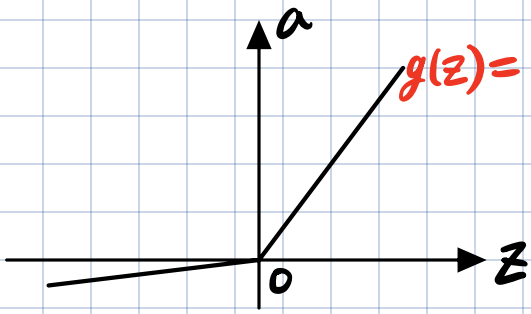
- ReLU and Leaky ReLU



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

(undefined if $z=0$) 미분 안 해도 됨

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



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

(undefined if $z=0$)

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

• Gradient descent for Neural Networks

- Equations for implementing Gradient Descent ⇒ back propagation
- Parameters of single layer Neural Network :

$$w^{[1]} \quad b^{[1]} \quad w^{[2]} \quad b^{[2]}$$

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

$n_x = n^{[0]}$ (input), $n^{[1]}$ (hidden), $n^{[2]} = 1$ (output) → : single hidden unit

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

  Let's say you're doing binary classification

  ($\hat{y}$ ↑ $a^{[2]}$, y ↑ true)

- Use Gradient descent to train algorithm

① Initialize parameters randomly other than 0

② Repeat Gradient descent to compute predicts ($\hat{y}^{[i]}$, i-1 ~ m)

③ Compute derivatives ($w^{[2]}$, $b^{[2]}$에 대한 cost function의 도함수)

$$dw^{[1]} = \frac{dJ}{dw}, \quad db^{[1]} = \frac{dJ}{db^{[1]}}, \ \dots$$

$$w^{[1]} := w^{[1]} - \alpha dw^{[1]}$$ ... 변수들이 수렴할때까지 반복하여 계산

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

how to compute partial derivative term?

• Formulas for computing derivatives

- Forward propagation:

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

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

(activation function)

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

- Back propogation

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

(Y: truth) $Y = [y^{(1)}, y^{(2)}, \dots y^{(m)}]$ (1, m)

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

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

(잘못된 1차원 배열 출력X, $(n^{[2]}, 1)$의 벡터출력)

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

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

- $dZ^{[1]}$: $(n^{[1]}, m)$ matrix
- $*$: element-wise product
- $g^{[1]\prime}$: 활성화함수의 도함수
- $g^{[1]\prime}(Z^{[1]})$: $(n^{[1]}, m)$ matrix

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

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

- $db^{[1]}$: $(n^{[1]}, 1)$
- keepdims: np.sum의 출력값은 reshape로 재배열하면 안 쓸 수 있다.

• Back propogation intuition

$x, w, b \rightarrow$ $z = w^T x + b$ $\rightarrow$ $a = \sigma(z)$ $\rightarrow$ $\mathcal{L}(a, y)$

$$\mathcal{L}(a,y) = -y\log a - (1-y)\log(1-a)$$

$$da = \frac{d}{da}\mathcal{L}(a,y) = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$dz = a - y = da \cdot g'(z) \quad (g = \sigma(z))$$

$$dw = dz \cdot x$$

$$db = dz$$

Chain Rule

$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}, \quad \frac{da}{dz} = \frac{d}{dz} g(z) = g'(z)$$

— 은닉층의 개수만큼의 단계를 신경망에서 계산하게 됨.

x ← fixed in supervised learning

$x, W^{[1]}, b^{[1]} \rightarrow$ $Z^{[1]} = W^{[1]}x + b^{[1]}$ $\rightarrow$ $a^{[1]} = \sigma(z^{[1]})$ $\rightarrow$ $Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$ (with $W^{[2]}, b^{[2]}$) $\rightarrow$ $a^{[2]} = \sigma(Z^{[2]})$ $\rightarrow$ $\mathcal{L}(a, y)$

$$da^{[2]}$$

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

$$dw^{[2]} = dz^{[2]} a^{[1]T} \rightarrow \text{“}dw = dz \cdot x\text{”}$$

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

- a는 column vector이므로 Transpose 해준다.
- LR의 경우 하나의 출력일 때 w는 row vector

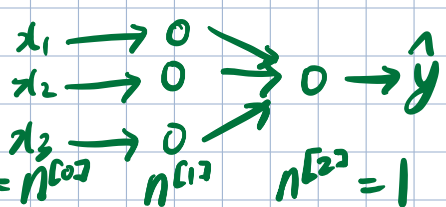
$dz^{[1]}$ (먼저 계산: $da^{[1]} = W^{[2]T} dz^{[2]}$)

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

$$dw^{[1]} = dz^{[1]} \cdot x^T \quad (x \rightarrow a^{[0]})$$

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

\# dimension check ← 여러 matrix의 dimension이 일치하는지 확인



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

$$Z^{[2]}, dZ^{[2]} \Rightarrow (n^{[2]}, 1) - (1, 1) \quad \text{binary classification}$$

$$Z^{[1]}, dZ^{[1]} \Rightarrow (n^{[1]}, 1)$$

⇒ foo, dfoo는 항상 차원이 같다.

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

- Summary of gradient descent

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

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

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

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

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

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

Vectorized implementation

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

$Y = [y^{(1)}, y^{(2)}, \dots y^{(m)}]$ $(1,m)$ truth

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

잘못된 1차원 배열 출력X
$(n^{[2]},1)$ 의 벡터출력

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

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

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

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

Vectorized implementation (multiple training samples)

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

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

stack z's columns

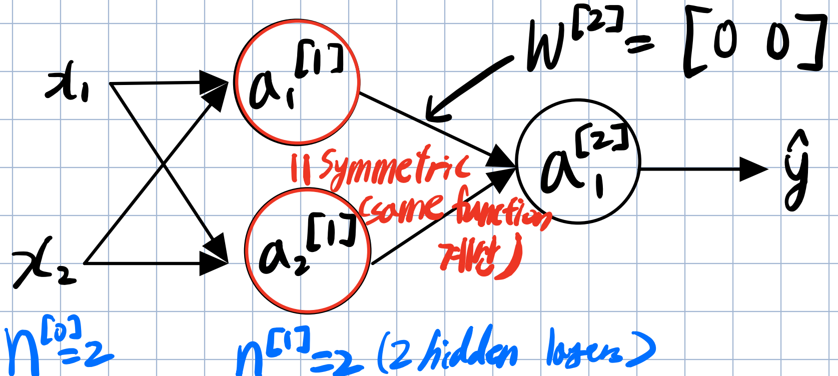
$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

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

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

- **Random Initialization**

- What happens if you initialize weights to zero?



$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ → 사실 b는 0으로 해도 ok

⇒ 어떤 샘플의 경우에도 $a_1^{[1]} = a_2^{[1]}$, 두 개의 활성이 같은 것으로 계산된다.

backpropagation 계산시 $dz_1^{[1]} = dz_2^{[1]}$ 두 유닛이 같은 값으로 초기화

→ 가중치의 결과값이 항상 같음.

let's say every row same value

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ → weight update → $W^{[1]} = W^{[1]} - \alpha dW$ → $W^{[1]} = \begin{bmatrix} \cdots \\ \cdots \end{bmatrix}$

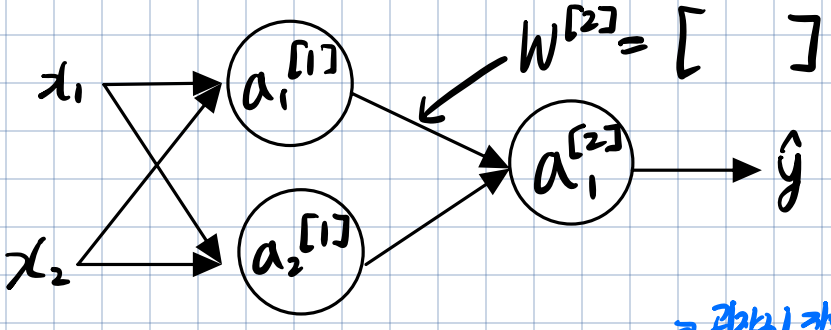
첫번째 열의 두번째 열과 같아진다.

∴ W값을 모두 0으로 초기화 → 두 유닛이 같은 함수를 계산하는 것으로 시작

→ 두 은닉유닛이 출력 유닛에 항상 같은 영향을 주는 것이 반복됨

∴ 두 유닛을 항상 같은 함수를 가지게 하므로 은닉유닛이 하나인 것, 다른 함수는 계산하기 위한 각각 다른 유닛이 필요하기 때문에 반드시 임의로 초기화한다.

— Random initialization



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

↳ 랜덤이 작은 임의의 수로 초기값을 만들기 위해 매우 깊은 깊이의 NN에서는 0.01이 아닌 다른 수를 선택할 수 있음.

↳ 가중치의 초기값은 매우 작게 설정해야함

tanh / sigmoid activation함수를 사용시 가중치가 너무 크면 곧 큰 상태로 훈련을 시작한다.

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

↳ b does not have symmetry breaking problem

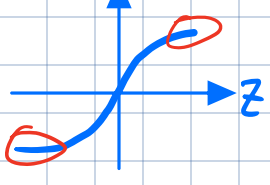
∴ W가 randomly set되었기 때문

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

$b^{[2]} = 0$

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

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



small gradient
⇒ slow learning