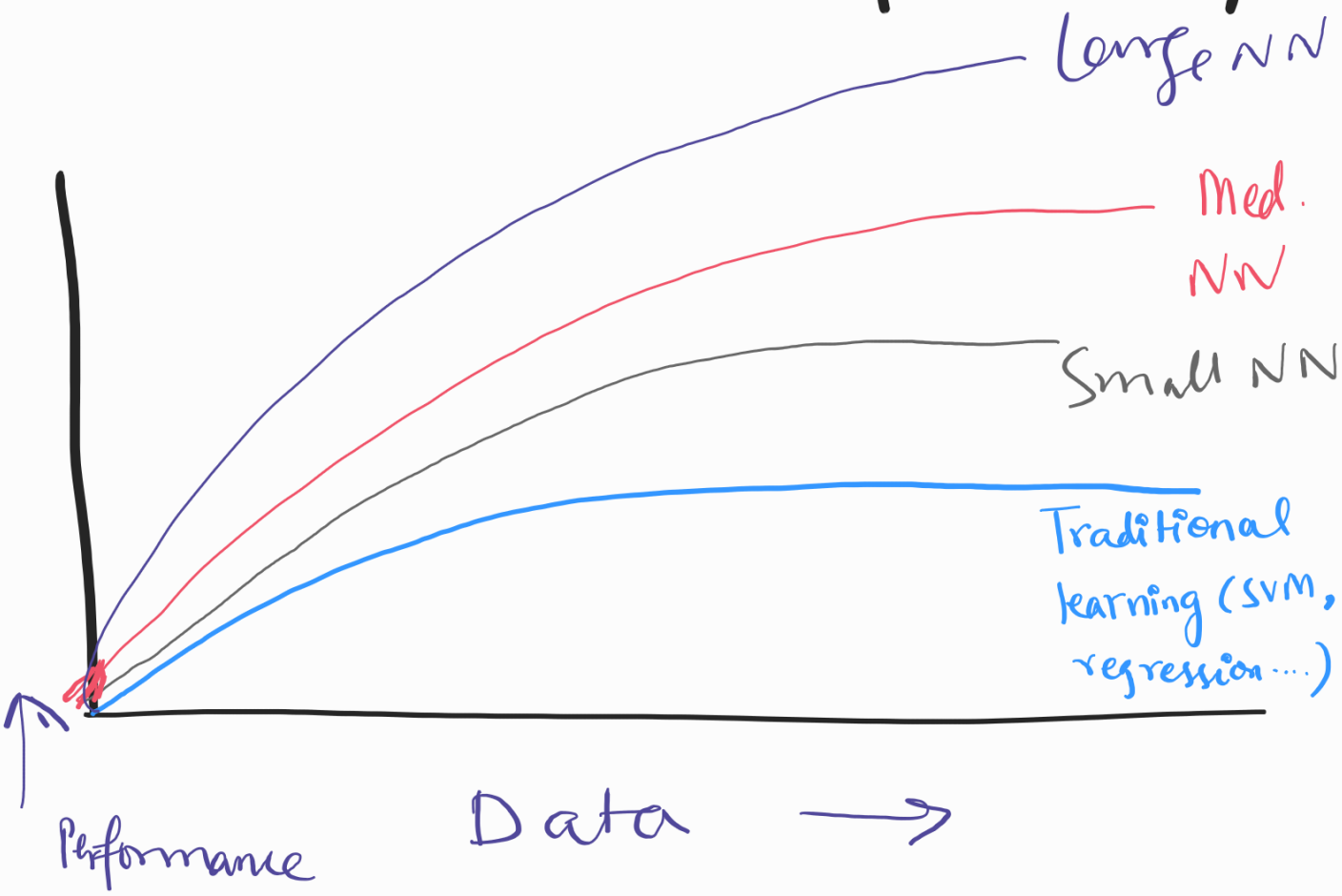
AI Theory

Follow up: Neural Nets

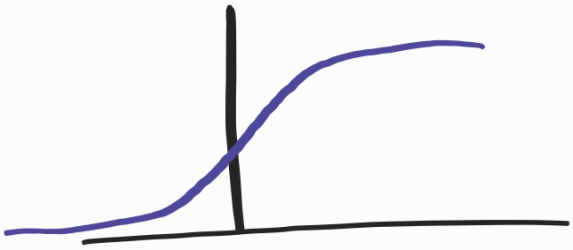
Ref: Intro to deep learning by Andrew Ng, Coursera

Introduction:

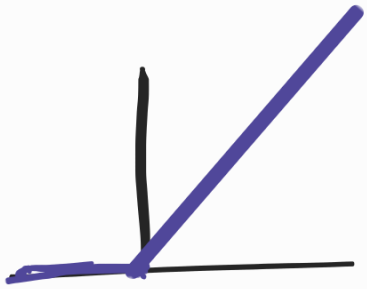
→ Scale drives deep learning



→ Activation functions:

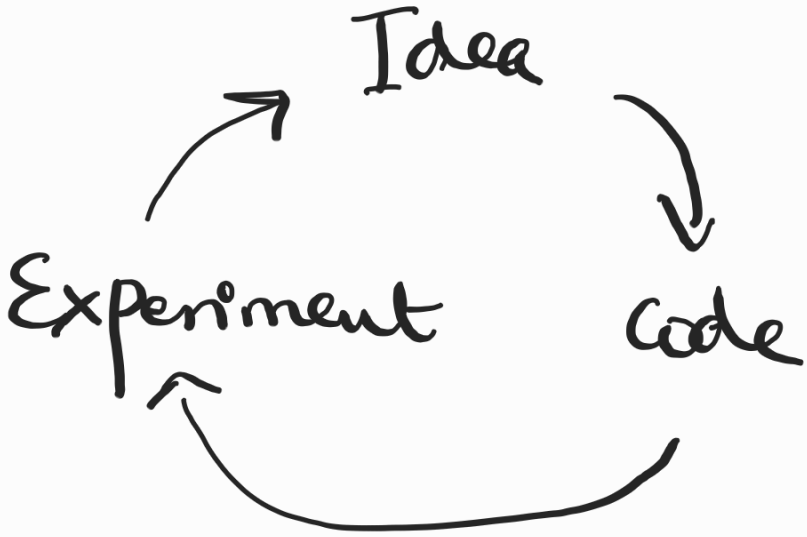


Sigmoid



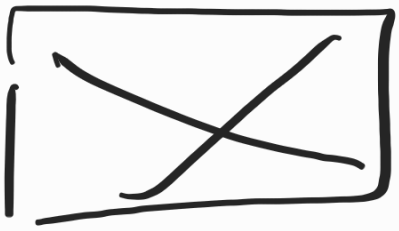
ReLU

→



Logistic models as NN

(i) Binary Classification :



$\longrightarrow$ 1/0
(cat/not cat)

Notation:

$(x, y) \rightarrow$ Single training ex.

$x \in \mathbb{R}^{n_x}$ (x dim. vector)

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

m : training example

$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$$M = M_{train}, \quad M_{test}$$

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

$$\longleftarrow m \longrightarrow$$

$$X.shape = (n_x, m)$$

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

$$Y.shape = (1, m)$$

Logistic Regression

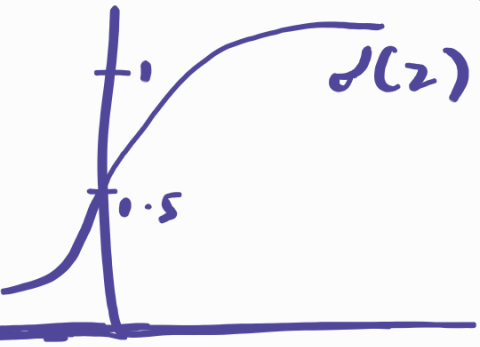
Given x, want $\hat{y} = P(y=1 \mid x)$

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\cancel{\hat{y} = w^T x + b}$

linear models (not binary)!!

$$y = \sigma(w^T x + b)$$



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

for large z, $e^{-z} \approx 0$

Cost function :

$$\hat{y} = \sigma(w^T x + b), \quad \sigma(z) = \frac{1}{e^{-z}+1}$$

$$\cancel{\rightarrow} \; L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

non convex function

gradient descent does not work.

$$L(\hat{y}, y) = -\left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right)$$

If $y = 1$, $L = -\log \hat{y}$

$\log \hat{y}$ large $= \hat{y}$ large

but $\hat{y} \leq 1$, so in bounds

else if $y=0$, $L = -\log(1-\hat{y})$

$\log(1-\hat{y}) = (1-\hat{y})$ Large

loss will make $\hat{y}$ small.

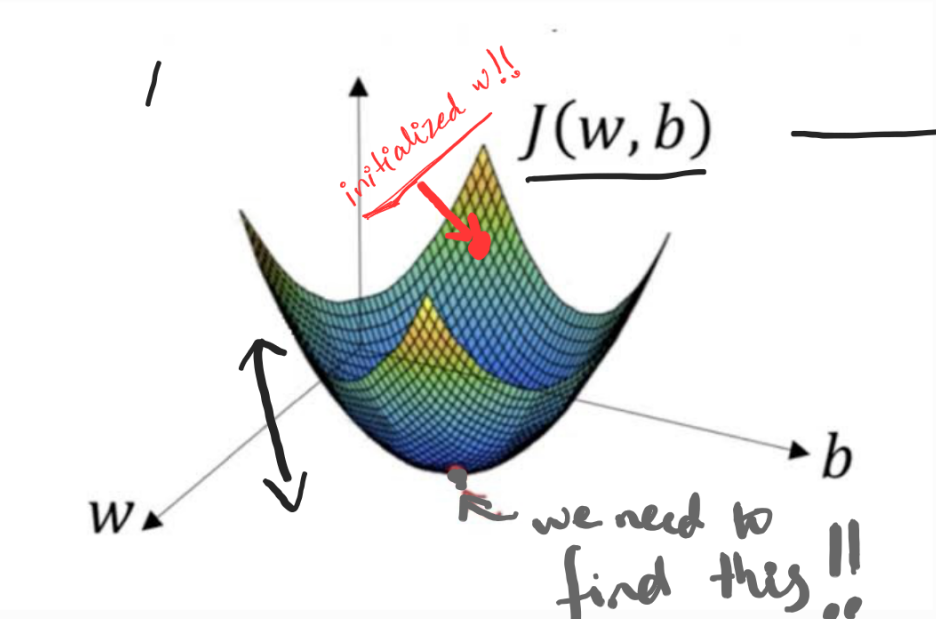
$$\text{Cost } J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right)$$

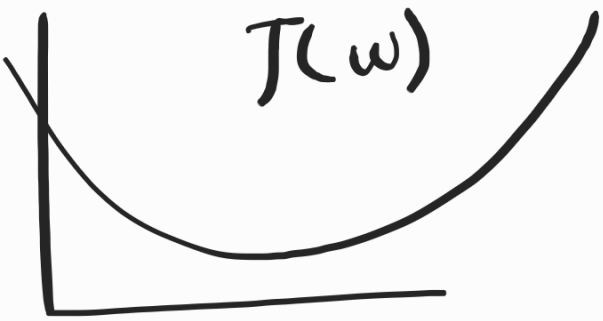
Gradient Descent

problem:

for, $J(w, b)$, loss is minimum



→ some surface above w, b



w (ignore b)

Repeat {

rate of change

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

} until convergence

Computation Graph (v. imp)

Simple example:

$$J(a,b,c) = 3(a+bc)$$

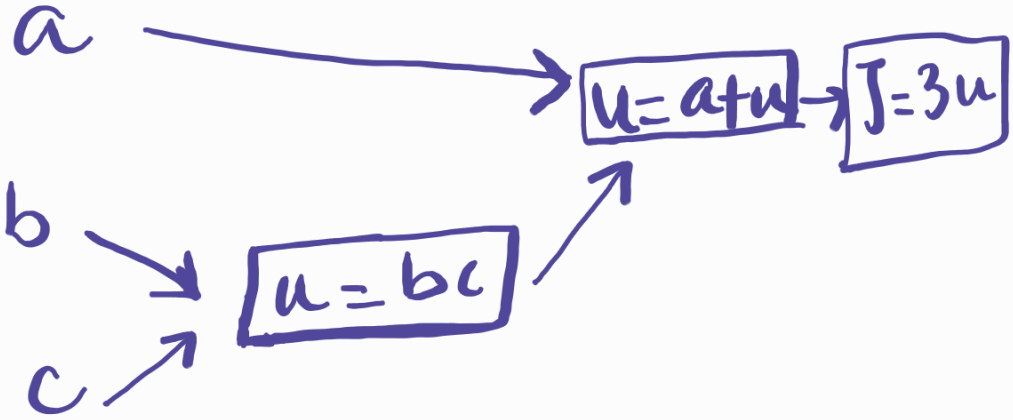
Three steps:

$$u = bc$$

$$v = a+u$$

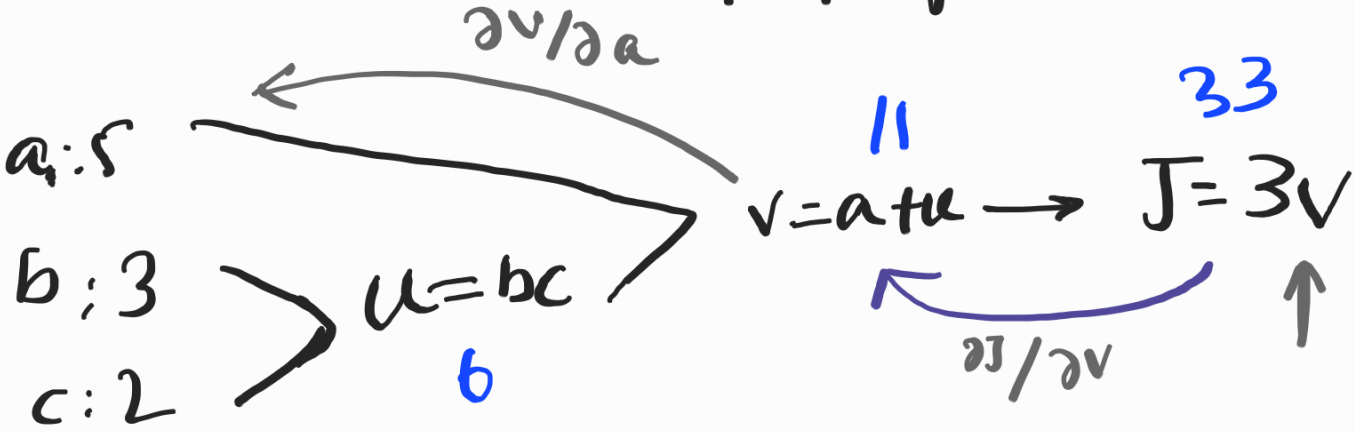
$$J = 3v$$

Computation graph:



left → right pass : forward pass

Derivatives with Computation Graph: (back propagation)



$$\frac{\partial J}{\partial v} = \frac{\partial}{\partial v}(3v) = 3$$

$$f(a) = 3a$$

$a = 5 \rightarrow 5.001$

what is net impact on v?

$v = 11 \longrightarrow 11.\underbrace{001}$

$J = 33 \rightarrow 33.\underbrace{003}$

change in "a" propagates to the <u>right</u>.

$a \rightarrow$ changes $V \longrightarrow$ changes J

Basically:

$$\frac{\partial J}{\partial a} = \underbrace{\frac{\partial J}{\partial V} \times \frac{\partial V}{\partial a}}_{\text{chain rule!}}$$

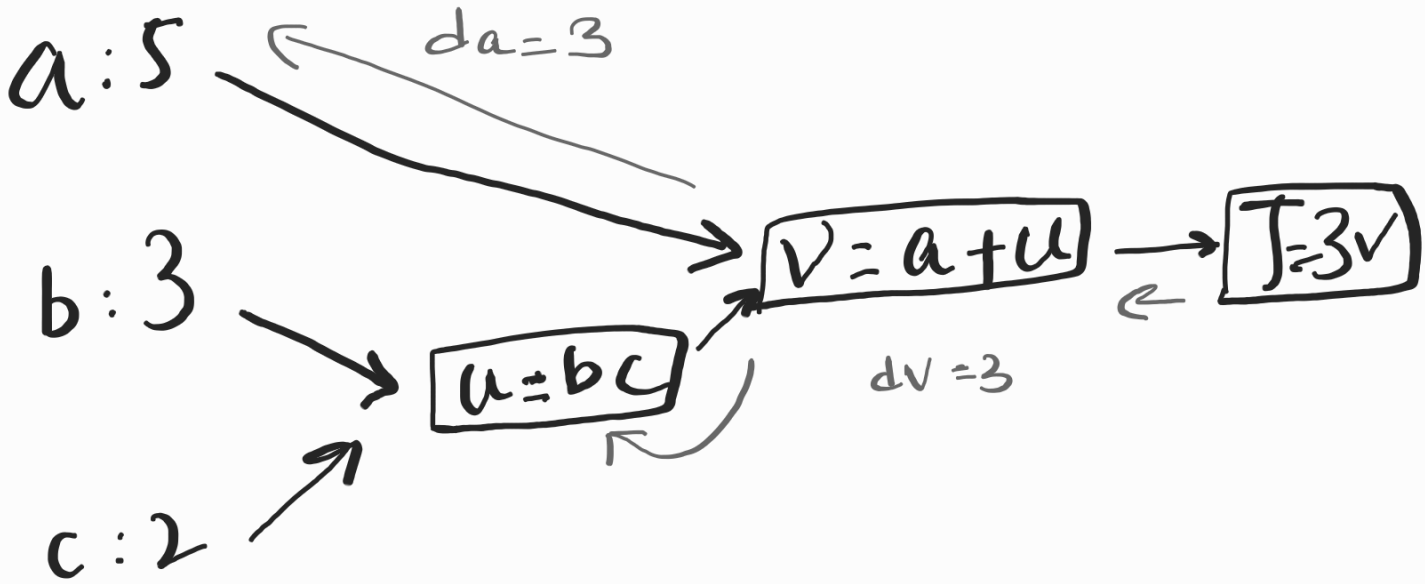
For any final output:

$$\frac{\partial(\text{Final Output var})}{\partial \text{ var}} \longrightarrow \text{in code}$$

$$\downarrow$$

$$\boxed{\text{dvar}}, \quad dV = 3, \quad da = 3$$

Key Takeaway: partially back prop. done



$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial v} \times \frac{\partial v}{\partial u} \times \frac{\partial u}{\partial b}$$

$$= 3 \times 1 \times c$$

$$= 3 \times 1 \times 2$$

$$\frac{\partial J}{\partial b} = 6$$

db = 6

python code name

$$\frac{\partial J}{\partial u} = ? = 3 = \underbrace{\frac{\partial J}{\partial v}}_{3} \times \underbrace{\frac{\partial v}{\partial u}}_{1}$$

$b = 3 \rightarrow 3.001$

$u = b \cdot c = 6.002$

$v = u + a = 11.002$

$J = 3v = 33.006$

6 × much!

just what the derivative predicts!

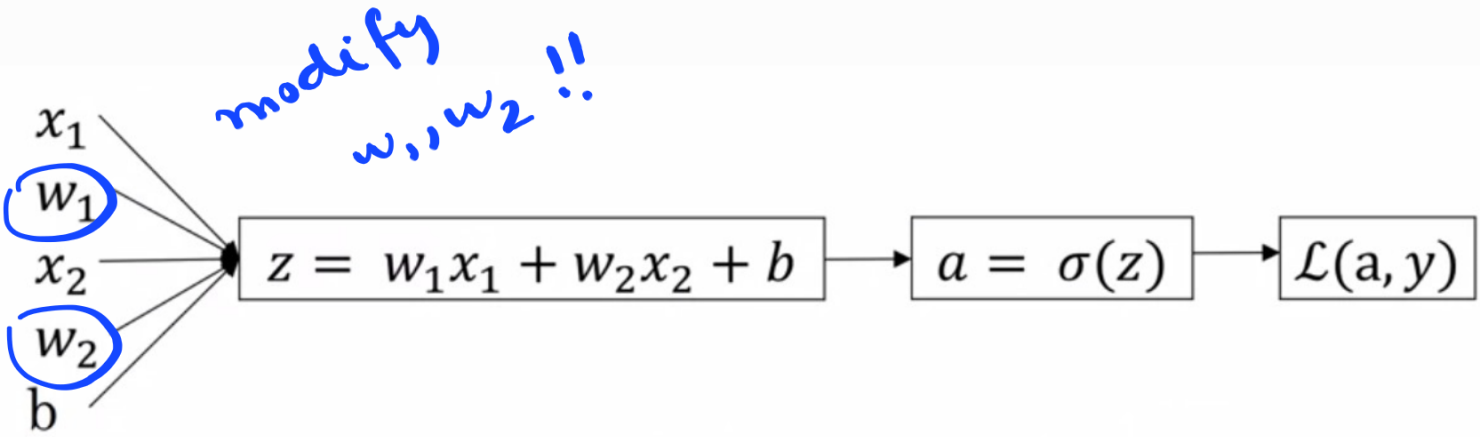
Note : What is $\partial J / \partial c$? $\boxed{\text{ANS} = 9}$

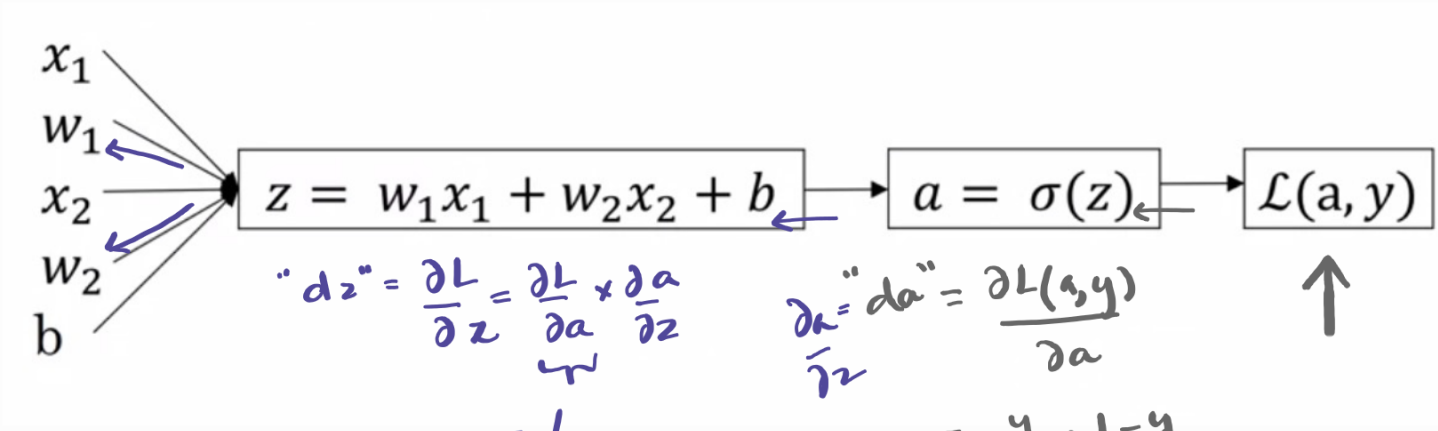
Computation Graph for Logistic Regression Models:

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

$$\hat{y} = a = \sigma(z)$$

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





$$\text{"}dz\text{"} = \frac{\partial L}{\partial z} = \underbrace{\frac{\partial L}{\partial a}}_{-\frac{1}{1-a}} \times \frac{\partial a}{\partial z}$$

$$\frac{\partial a}{\partial z} = \text{"}da\text{"} = \frac{\partial L(a,y)}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a}$$

(Basic calc)

$$\boxed{\frac{\partial L}{\partial z} = a - y}$$

$$\frac{\partial L}{\partial w_1} = \text{"}dw_1\text{"} = x_1 dz$$

$$\frac{\partial L}{\partial w_2} = \text{"}dw_2\text{"} = x_2 dz$$

This is simple example!!

Algorithm for "m" examples?

$J = 0$

$dw_1 = 0$

$dw_2 = 0$

$db = 0$

for $i=1$ to m:

$$z^{(i)} = w^T x^{(i)} + b$$

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

$$J \mathrel{+}= -\left[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})\right]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$

$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$

$$db \mathrel{+}= dz^{(i)}$$

Just one step of Gradient Descent!!

$J /= m$

$dw_1 /= m$

$dw_2 /= m$

$db /= m$

explicit for loops = slow

use vectori-zation!!

Vectorization

Traditional for loops are slow.

Solution: vectorization!

In logistic model:

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

$$w = \begin{bmatrix} \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \end{bmatrix}$$

very large vectors!

Non-vectorized:

```
z = 0
for i in range(n-x):
    z += w[i] * x[i]
z += b
```

Vectorized:

$$z = \underbrace{np.dot(w, x) + b}$$

$\Rightarrow$ much much faster!!

for $|\text{vector}| = 10$ million

vectorized $= 1.5$ ms

Non-vectorized $= 500$ ms

Key Takeaway:

whenever possible, avoid an explicit for loop.

But how does it work?

for any $u = A\vec{v}$

$\because$ A is a matrix

$$u = np.dot(A, v)$$

non-vectorized

```
for i ......
    for j ....
u_i += (A[i][j] + v[j])
```

$O(n)^2$ eliminated!!

Two for loops.

```
J = 0, dw1 = 0, dw2 = 0, db = 0
(1) for i = 1 to m:
        z^(i) = w^T x^(i) + b
        a^(i) = σ(z^(i))
        J += -[y^(i) log a^(i) + (1 - y^(i)) log(1 - a^(i))]
        dz^(i) = a^(i) - y^(i)
(2)     dw_1 += x_1^(i) dz^(i)
        dw_2 += x_2^(i) dz^(i)
        db += dz^(i)
J = J/m, dw_1 = dw_1/m, dw_2 = dw_2/m, db = db/m
```

$n_x = 2$
for j = 1 .... n_x do:
dwj += ..

How do we eliminate them?

$J = 0$, $\boxed{\cancel{dw1 = 0, dw2 = 0}}$, $db = 0$

$dw = np.zeros((n-x,1))$

for i = 1 to m:

$z^{(i)} = w^T x^{(i)} + b$

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

$J += -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$\boxed{\cancel{dw_1 += x_1^{(i)} dz^{(i)}} \quad \cancel{dw_2 += x_2^{(i)} dz^{(i)}}}$

$dw += x^{(i)} dz^{(i)}$

$db += dz^{(i)}$

$J = J/m$, $\boxed{dw_1 = \cancel{dw_1/m, dw_2} = dw_2/m}$, $db = db/m$

$dw /= m$

Vectorizing Logistic Regression

① Carry out forward prop. through vectorization.

$$(n_x, m) \quad X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix}$$

X is a matrix of training examples

$$\left[z^{(1)}\ z^{(2)} \cdots z^{(m)}\right] = \underline{\underline{w^T X}} + \underbrace{[b\ b\ b \cdots b]}_{\text{shape } (1,m)}$$

$$w^T \left[x^{(1)} \cdots x^{(2)} \cdots x^{(3)} \cdots x^{(m)}\right] + [b \cdots]$$

$$\left[w^T x^{(1)}\ w^T x^{(2)} \cdots w^T x^{(m)}\right] (b \cdots)$$

$$\underbrace{\left[w^T x^{(1)} + b\ \ w^T x^{(2)}\ b \cdots\right]}_{\text{shape } (1,m)}$$

exactly the same as required!

$$\boxed{Z = np.dot(w.T, X) + b}$$

"Broadcasting"

$$A = [a^{(1)}\ a^{(2)}\ \cdots\ a^{(m)}] = \sigma(z)$$

Vectorizing Logistic Regression's Gradient Output:

$$dz^{(1)} = a^{(1)} - y^{(1)} \qquad dz^{(2)} = a^{(2)} - y^{(2)}$$

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

$$A = [a^{(1)} \cdots a^{(n)}] \quad Y = [y^{(1)} \cdots y^{(m)}]$$

$$dZ = A - Y = [a^{(1)} - y^{(1)}\ a^{(2)} - y^{(2)} \cdots]$$

for Logistic Regression:

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

$$db = np.sum(dz)$$

$$dw = \frac{1}{m} \underline{\underline{Xdz^T}}$$

Original implementation:

$J = 0,\ dw1 = 0,\ dw2 = 0,\ db = 0$

~~for i = 1 to m:~~

$z^{(i)} = w^T x^{(i)} + b$

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

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$\cancel{dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}}$

$\cancel{dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}}$

$db \mathrel{+}= dz^{(i)}$

$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m,\ db = db/m$

$dw \mathrel{+}= x^{(i)} * dz^{(i)}$

Vectorized:

$Z = w^T X + b$

$= np.dot(w^T, X) + b$

$A = \sigma(Z)$

$dZ = A - Y$

$dw = \frac{1}{m} X dz^T$

$db = \frac{1}{m} np.sum(dz)$

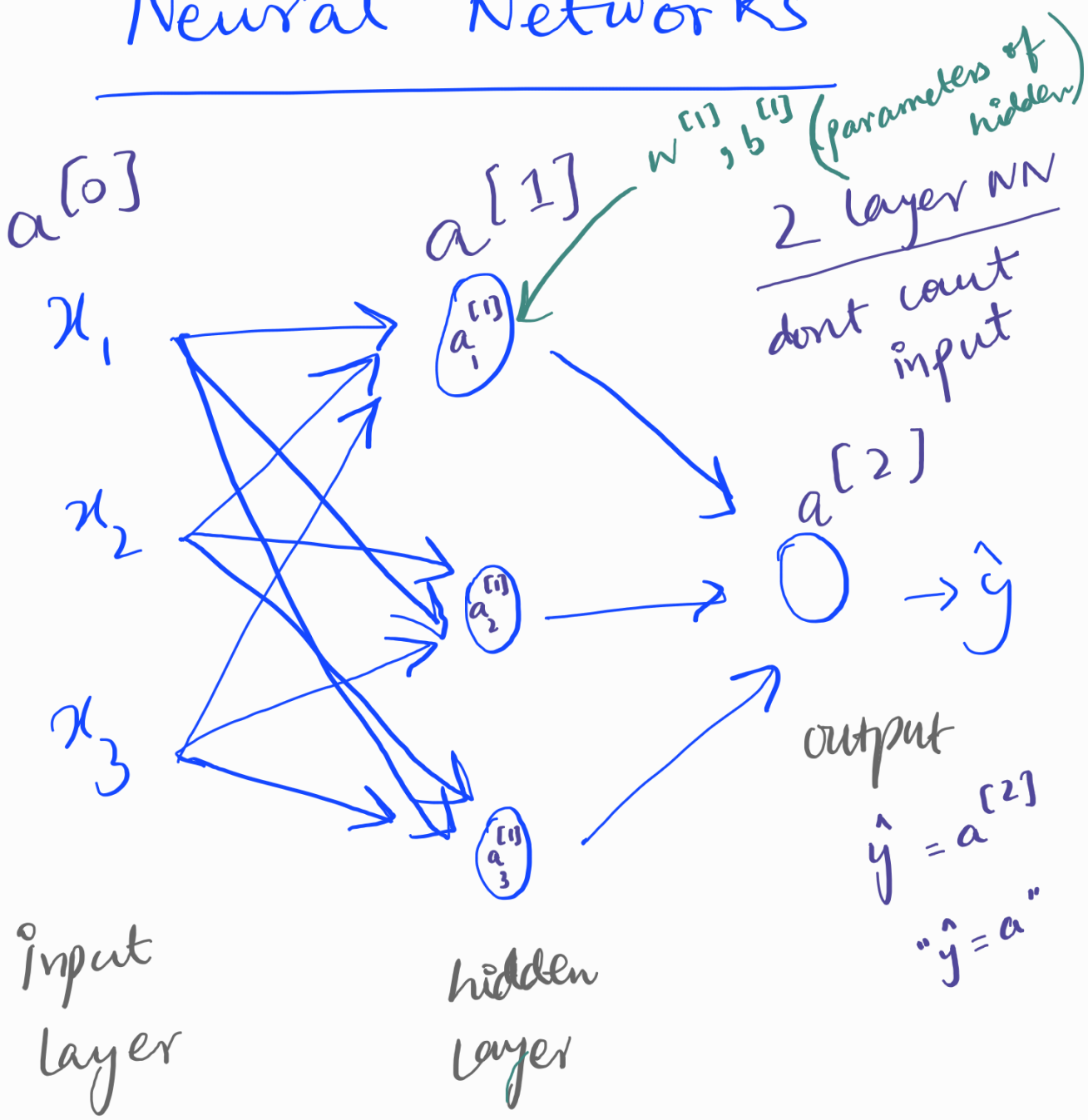
$w := w - \alpha dw$

$b := b - \alpha db$

The lines in blue implement both forward and back prop without for loop!

Only use for loop for iterations now:

Neural Networks



Notations:

Layer No.

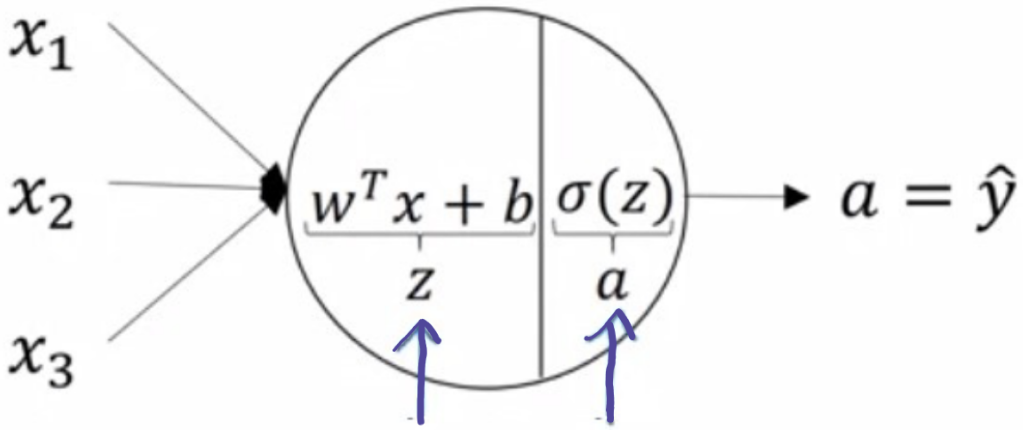
$$a^{[0]} = X$$

activations

input feature

Computing a Neural Network's Output:

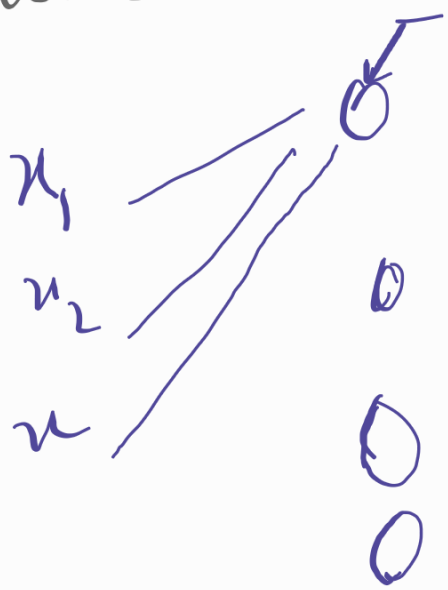
Logistic regression:



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

$$a = \sigma(z)$$

A neural network does this a lot more times.



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

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

$a_i^{[l]}$ ← layer

← node in layer

$(w_1^{[1]})^T x$ $\quad a^{[1]}$ (stack them!)

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

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

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

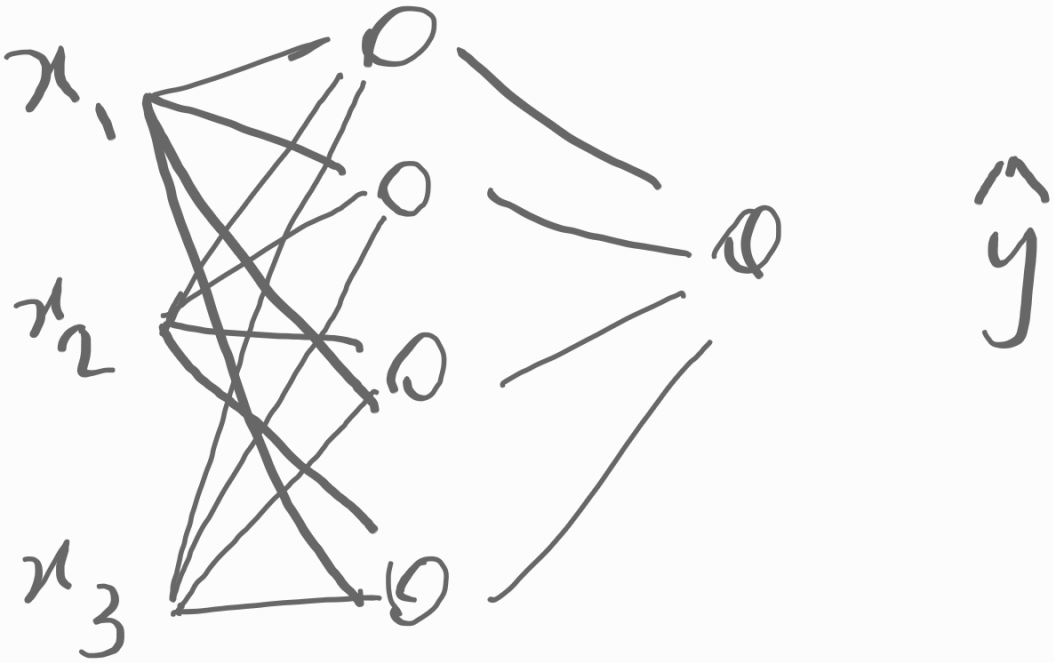
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

How to vectorize?

$$\begin{bmatrix} w_1^{[1]} \\ w_2^{[1]} \\ w_3^{[1]} \\ w_4^{[1]} \end{bmatrix}^T X = \begin{bmatrix} w_1^{[1]T} X + b_1^{[1]} \\ w_2^{[1]T} X + b_2^{[1]} \\ w_3^{[1]T} X + b_3^{[1]} \\ w_4^{[1]T} X + b_4^{[1]} \end{bmatrix} \quad Z^{[1]}$$

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

Vectorization across Multiple Training examples:



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

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

$$\vdots \qquad\qquad \vdots$$

$$a^{[2](i)}$$

a ← layer

(i) ← example i

pseudocode:

for $i=1$ to m:

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

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

get rid of this!

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

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

Stack z's into cols:

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

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

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

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

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

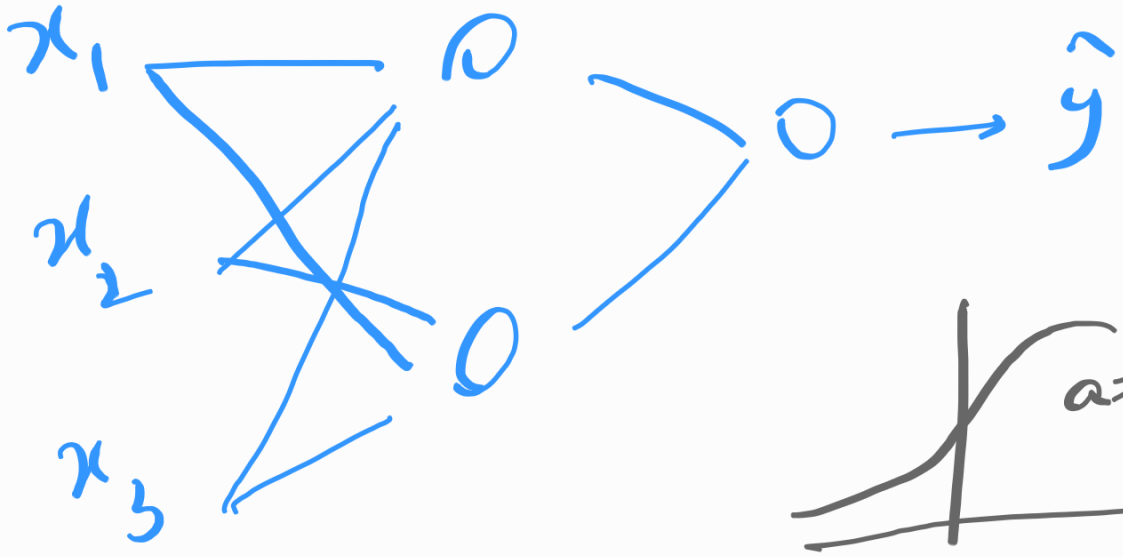
$$A^{[1]} = \begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \cdots \end{bmatrix}$$

An important reminder!

$$\begin{array}{c} \text{hidden unit traversal} \\ \updownarrow \\ \end{array} \begin{bmatrix} a^{[1](1)} & a^{[1](2)} & \cdots \end{bmatrix}$$

$\longleftrightarrow$ training example

Activation functions



given X:

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

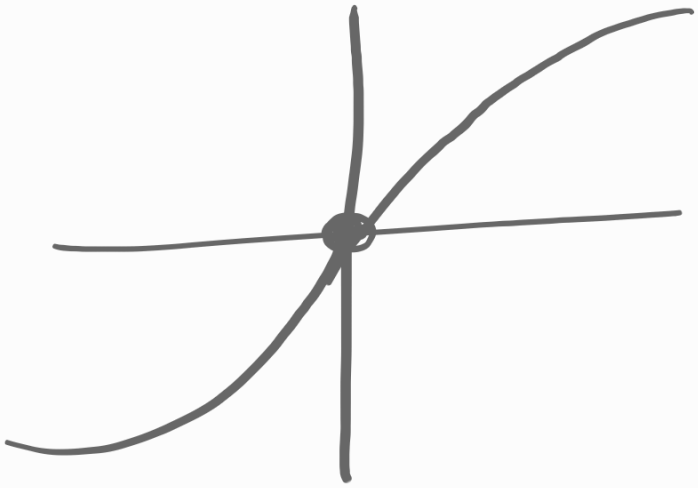
$$a^{[1]} = \cancel{\sigma(z^{[1]})}\ g(z^{[1]})$$

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

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

Choices for $g(z^{[i]})$:

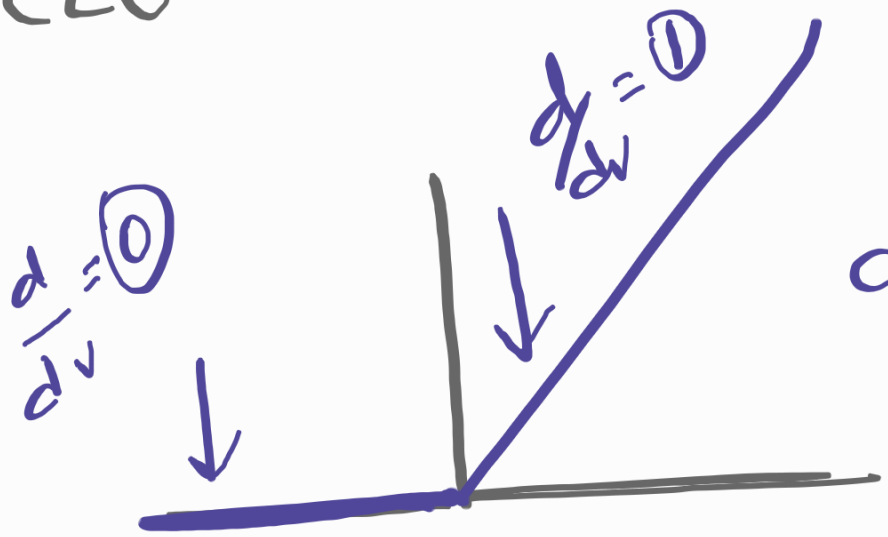
tanh x :



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

except output : tanh x is preferred

ReLU :

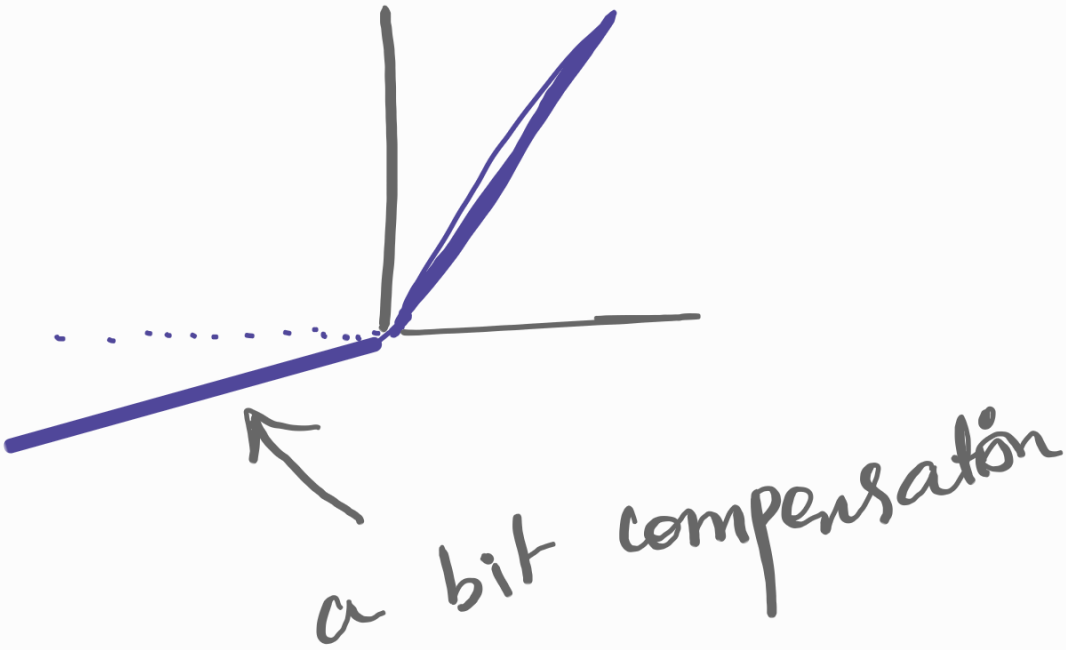


$a = \max(0, z)$

ReLU is the go-to choice for hidden layer.

Leaky ReLU:

for -ve values in ReLU.



Derivatives of Activation functions:

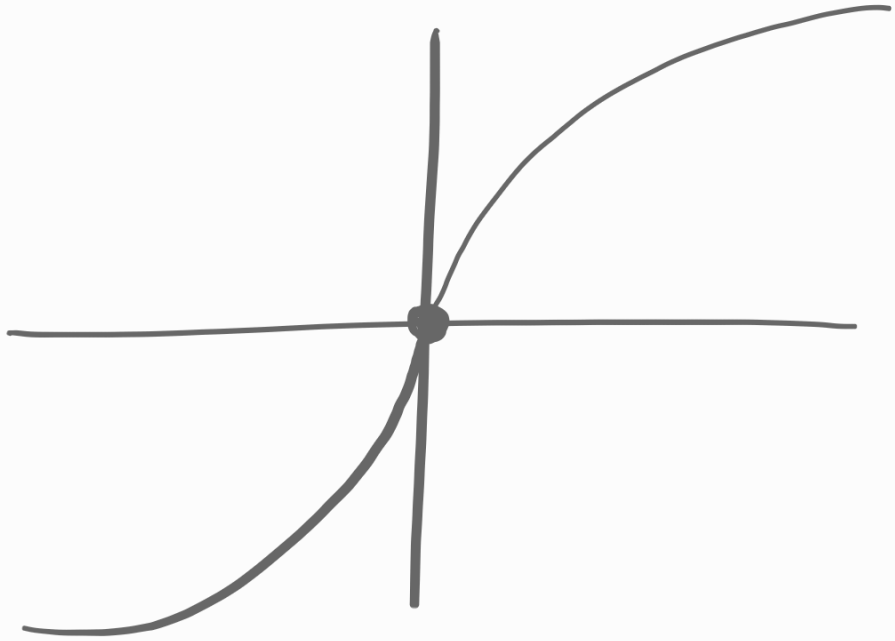
$$g(z) = \frac{1}{1+e^{-z}}$$

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

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

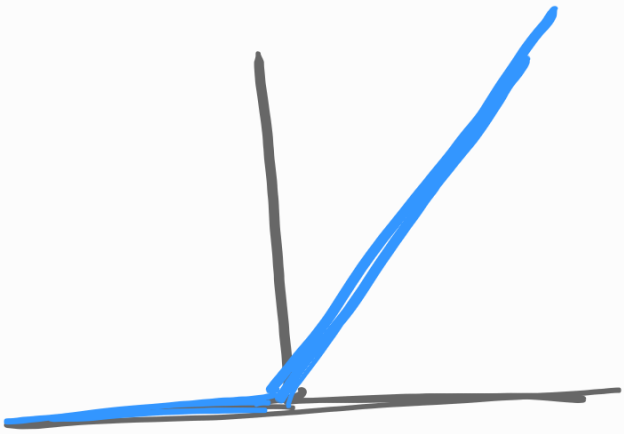
$$g'(z) = \frac{d}{dz} g(z) = g(z)\left(1 - g(z)\right)$$

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



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

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



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

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

Gradient Descent for NN

Parameters:

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

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

Cost: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]})$

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

Parameters = RANDOM

Gradient Descent:

Repeat {

Compute pred. $(\hat{y}^{(i)}, i = 1 \dots m)$

$$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}, \quad db^{[1]} = \frac{\partial J}{\partial w^{(1)}} \dots$$

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

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

} until Convergence

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:

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

$$dw^{(2)} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

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

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

element wise product

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

$$db^{[1]} = \frac{1}{m} \, np.sum\left(dZ^{[1]}, ax=1, keepTrue\right)$$