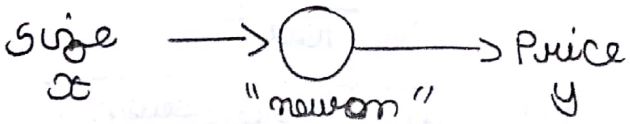
Deep Learning. ai

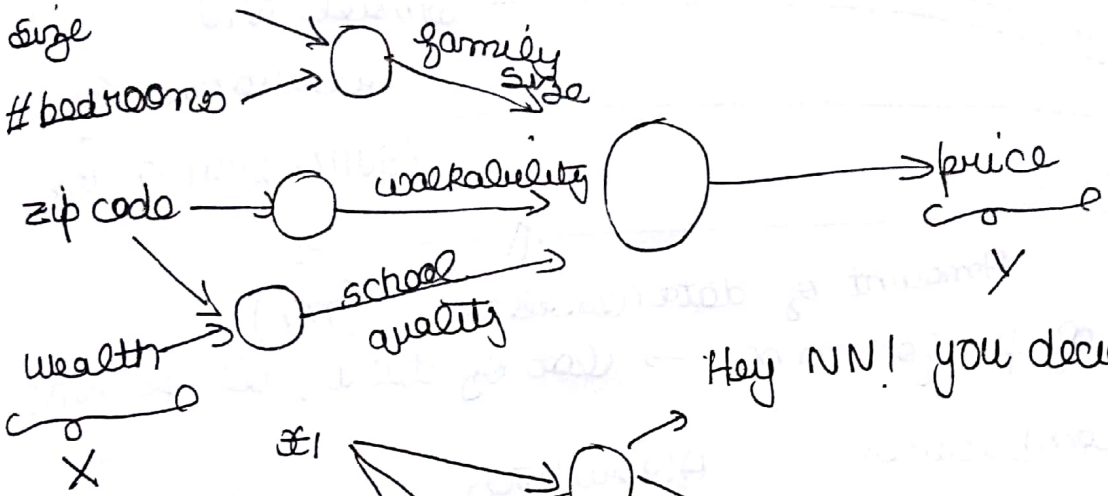
. AI is new electricity

→ Deep learning highly sought after.

→ Training Neural network.



RELU → Rectified Linear Unit.



Given (x, y) NNs can do well in mapping x → y.

Supervised learning

	Input (x)	Output (y)	Application
standard NN	Home features	Price	Real Estate
	Ad, user info	Click on ad?	Online Advert
CNN	Image	Object (1, ..1000)	Photo tagging
RNN	Audio	Text transcript	Speech
	(sequence)		

Structured Data

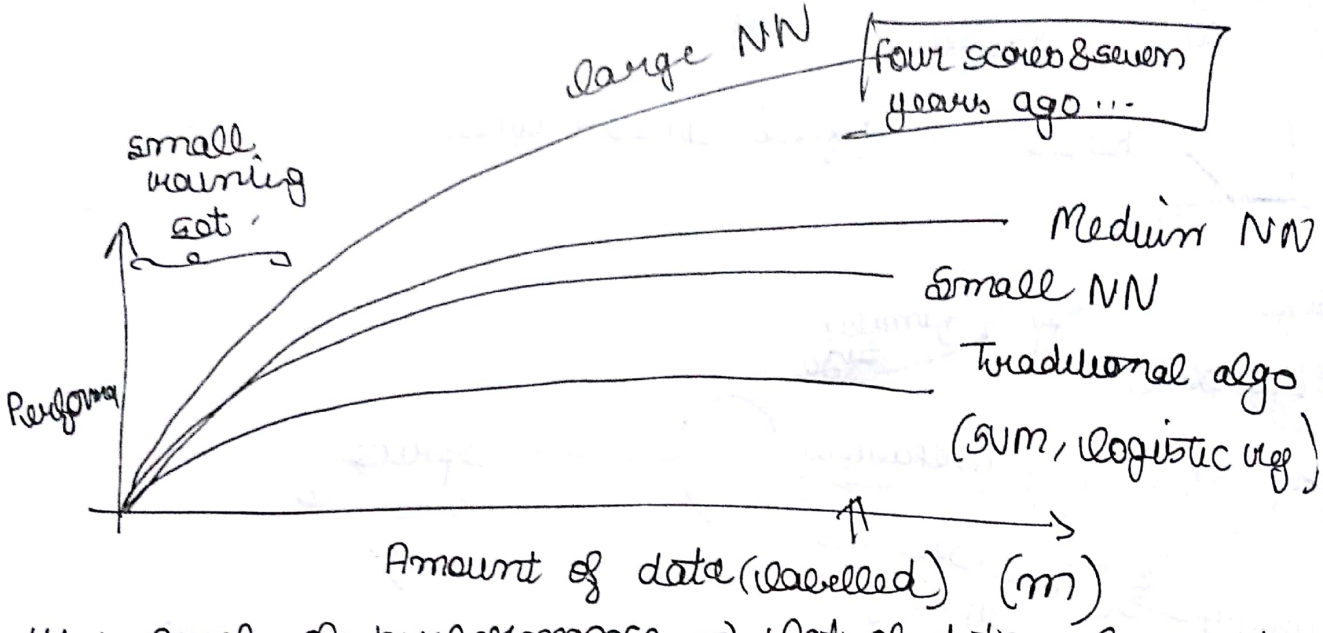
Size	#bedrooms	..	Price

Unstructured

Audio

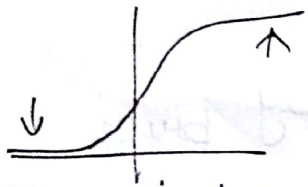
Image → pixel values

Text

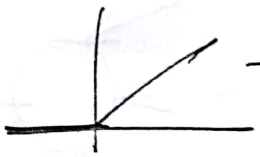


High level of performance → lot of data, large NN.

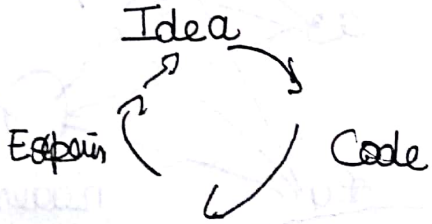
· Data · Computation · Algorithms



learning becomes slow as gradient becomes 0,



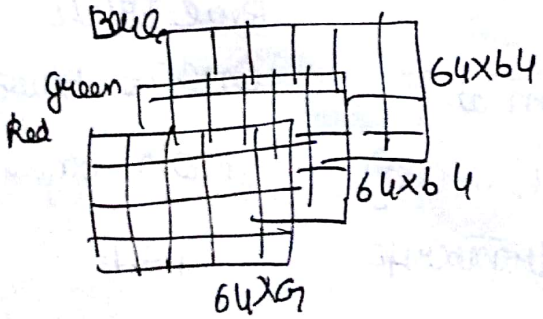
→ RELU gradient is 1.



Week-2

Logistic Regression – binary classification (1 (cat) vs 0 (non cat))

→ No for loop.



$$\rightarrow X = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

→ 64x64x3

$n_x = 12288$

Notation

(x, y) $x \in R^{n_x}$, $y \in \{0, 1\}$

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

$m = m_{train}$ m_{test} → test examples

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

$$X = \begin{bmatrix} - & x^{(1)T} & - \\ - & x^{(2)T} & - \\ & \vdots & \\ - & x^{(m)T} & - \end{bmatrix}$$

difficult

$X.shape = (n_x, m)$

$Y = [y^{(1)} \; y^{(2)} \; \ldots \; y^{(m)}]$ $\quad Y \in R^{1 \times m}$

$Y.shape = (1, m)$

Logistic Regression — chance to tell it is a cat picture

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

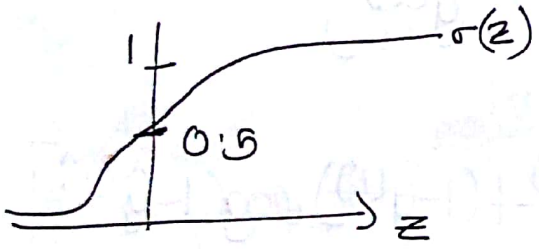
$x \in R^{n_x}$

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

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

$w^T x + b$ → Linear regression but (0 & 1) → we want

$\sigma(w^T x + b)$ → Logistic regression



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

if z is very very large $\sigma(z) = \frac{1}{1+0} = 1$

If z is very large $\sigma(z) = \frac{1}{1 + \text{bignumber}} = 0$

Learn w & b so that $\hat{y}$ becomes 1.

w & b are separate.

In some conventions $x_0 = 1$, $x \in R^{n_x+1}$

$$\hat{y} = \sigma(\theta^T x)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \\ \} w \end{matrix}$$ ↪ more tedious

Cost function

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

Given $\{(x^{(1)}, y^{(1)}) \ldots (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

Loss (error function):

$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ → results in non convex. ∪∩∪

(gradient descent gets stuck at local)

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

If $y=1$, $L(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large.

If $y=0$: $L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log 1-\hat{y}$ large, want $\hat{y}$ small.

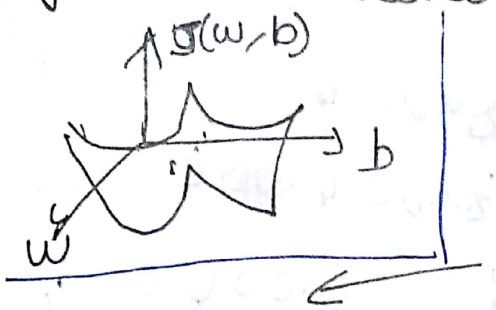
Cost function:

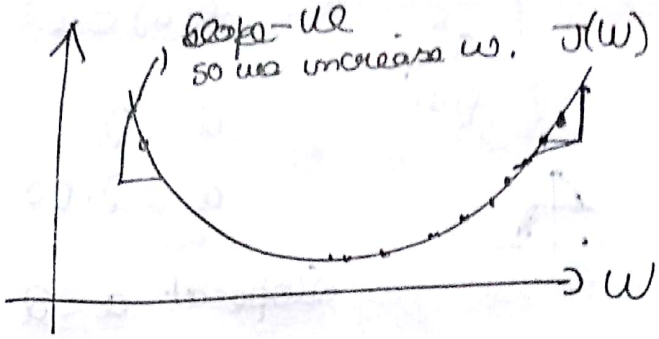
$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

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

$\frac{0.006}{0.002} = 3$

Gradient Descent





Repeat {

$$w := w - \alpha \frac{d J(w)}{d(w)} \quad \text{("dw")}$$

(derivate is positive)

$\frac{d J(w)}{d(w)}$ → Slope

α learning rate (control step)

(want to find w, b that minimize $J(w,b)$

→ Convex function ∪ not ∩

Initialize w, b to zero (that's what ppl do instead of random initialization)

Repeat {

$$w := w - \alpha\, dw$$

}

Sometimes represented as $\frac{\partial J(w,b)}{\partial w}$

$J(w,b)$

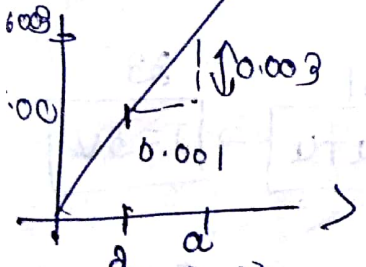
$$w := w - \alpha \frac{d J(w,b)}{dw} \rightarrow dw$$

$$b := b - \alpha \frac{d J(w,b)}{db} \rightarrow db$$

Derivatives

$a = 2 \quad f(a) = 6 \qquad a = 2.001 \quad f(a) = 6.003$

$f(a) = 3a$

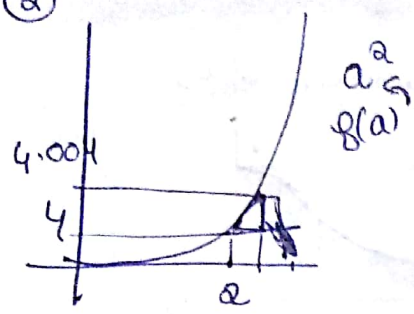


$$\frac{\text{height}}{\text{width}} = \frac{0.003}{0.001} = 3$$

$$\frac{d f(a)}{da} = 3 = \frac{d}{da} f(a)$$

(At any part of function slope is same)

(2)



$g(a) = a^2$

$a = 2$ $\quad g(a) = 4$

$a = 2.001$ $\quad g(a) = 4.004$

Slope at $a = 2$ is 4 $\quad \left(\frac{0.004}{0.001}\right) = 4$

Now at $a = 5$ $\quad g(a) = 25$, $\quad a = 5.001$ $\quad g(a) = 25.010$

$\frac{d}{da} g(a) = 10$ when $a = 5$.

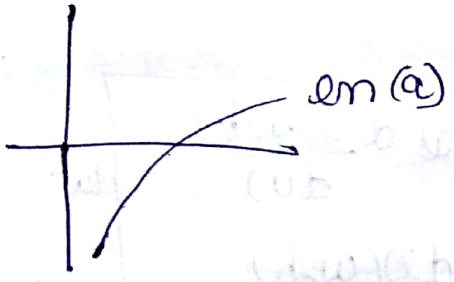
$$\boxed{\frac{d}{da} a^2 = 2a}$$

$g(a) = \log_e(a)$ $\quad \ln(a)$ $\quad \frac{d}{da} g(a) = \frac{1}{a}$

$a = 2$ $\quad g(a) = 0.6931$

$a = 2.001$

$g(a) = 0.69365$



$\frac{d g(a)}{da} = 0.005$

---

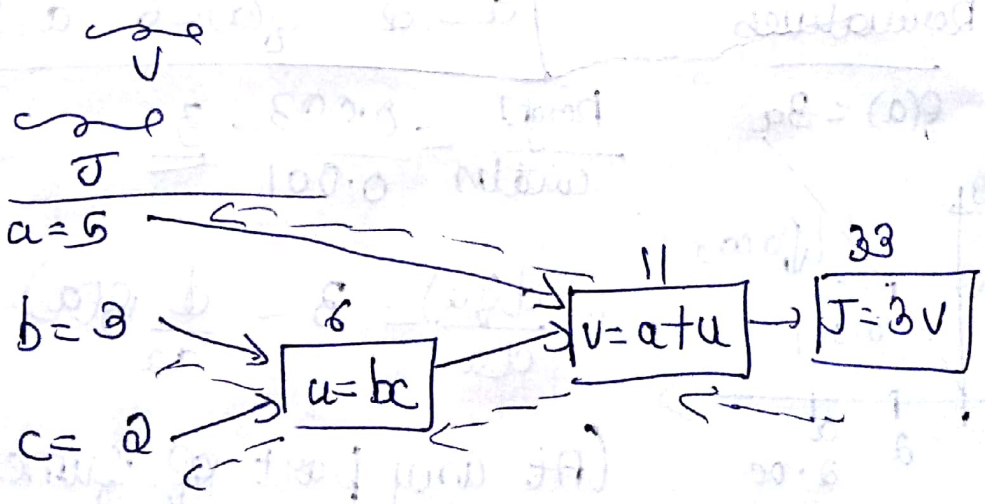
computation graph

$J(a,b,c) = 3(a + \underbrace{bc}_{u}) = 3(5 + 3*2) = 33$

$u = bc$

$v = a + u$

$J = 3v$



<--- backward right to left computation of derivatives

$\frac{dJ}{dV} = ? = 3$ Now $J = 3V$

& $V = 11 \longrightarrow V = 11.001$

$J = 33.003$

$g(a) = 3a$

$\frac{dg(a)}{da} = \frac{dg}{da} = 3 \qquad \frac{dJ}{dV} = 3$

Chain rule

$a \to V$

$V \to J$

$\frac{dJ}{da} = \frac{dV}{da} \frac{dJ}{dV}$

$= 3 * 1 = 3$

$\frac{dJ}{da} = 3$

$a = 5 \to 5.001$

$V = 11 \to 11.001$

$J = 33 \to 33.003$

$\frac{dV}{da} = 1$

J → Final output variable

d Final O/p

dvar → intermediate → "dvar"

"dv" = 3

"da" = 3

"db" = 6

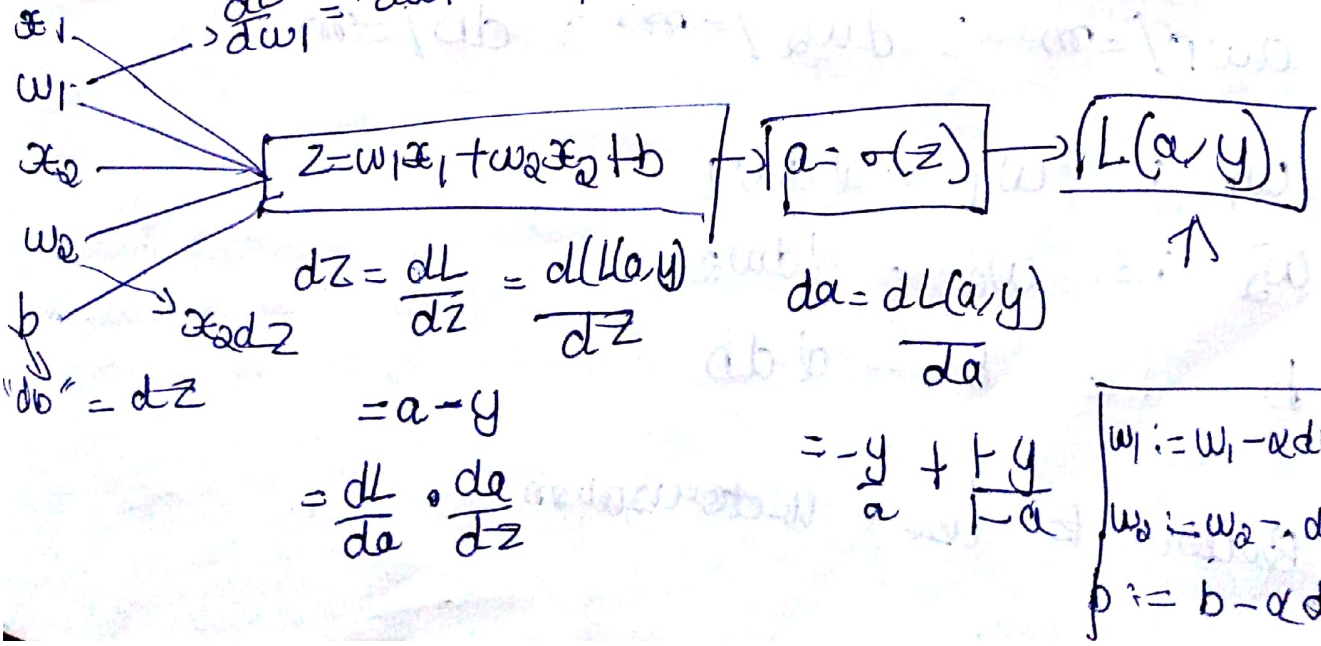
"dc" = 9

Logistic regression

$z = w^T x + b$

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

$L(a, y) = -(y \log(a) + (1-y)\log(1-a))$



$dz = \frac{dL}{dz} = \frac{dL(a,y)}{dz}$

$= a - y$

$= \frac{dL}{da} \cdot \frac{da}{dz}$

$da = \frac{dL(a,y)}{da}$

$= -\frac{y}{a} + \frac{1-y}{1-a}$

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

Gradient descent on m examples

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})$$

Compute $dw_1^{(i)}$ for $dw_1^{(i)}$ → b

all training examples & average it

$J=0;\ dw_1=0;\ dw_2=0;\ db=0$

For $i=1$ to m

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

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

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

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

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

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

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

$n=2$

$J\ /= m$

$dw_1\ /= m;\ dw_2\ /= m;\ db\ /= m$

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

Better to use vectorization

What is vectorization?

(Shift Enter to execute)

$z = w^T x + b$

vectorized $z = np.dot(w, x) + b$ ($w^T x$)

SIMD → Single Instruction Multiple data (parallelisation) Both CPU & GPU
Instructions.

→ $u = np.exp(v)$ $np.log(v)$ $np.abs(v)$
$np.maximum(v, 0)$ (element wise)

Vectorizing Logistic regression. $v^{**}2$ square $1/v$ → element wise

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

$$Z = [z^{(1)} \; z^{(2)} \dots z^{(m)}] = w^T X + [b \; b \; b]_{1 \times m}$$
$$= [w^T x^{(1)} + b \;\; w^T x^{(2)} + b \dots w^T x^{(m)} + b]$$
$$z^{(1)} \qquad z^{(2)}$$

$Z = np.dot(w.T, X) + b$ (a, 1) & auto converts b to row (broadcasting)

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

Vectorizing gradient Descent.

$dz = [a - y]_{1 \times m}$ $\quad dZ = A - Y.$

$dw = x^{(1)} dz^{(1)}$ | $db = 0$
$\vdots$ | $db += dz^{(1)}$
$dw += x^{(m)} dz^{(2)}$ | $\vdots$
$dw /= m$ | $db /= m$

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

$dw = \left[\frac{1}{m} X \cdot dz^T\right]_{n \times 1}$

$X \to n \times m$
$dz \to 1 \times m$
$dw = n \times m \times m \times 1$
$= n \times 1$

Put all boxes for multiple iterations in a for loop

for i in range(1000):

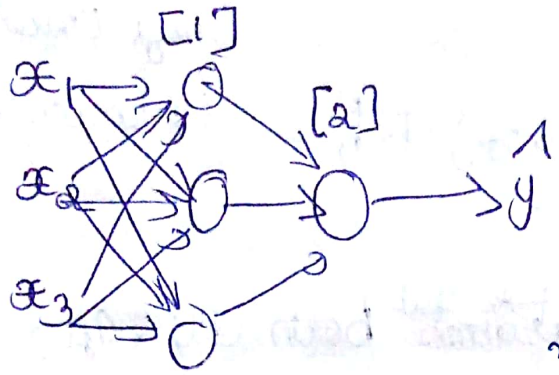
& then $w = w - \alpha dw$
$b = b - \alpha db$.

$(m, n) \pm (1, n)$ broadcasting
$* / (m, n)$

$(m, 1) \pm * / R \to (m, 1)$

$A.sum(axis = 0)$ → sum vertically

Neural Networks Overview



Never use rank 1 arrays

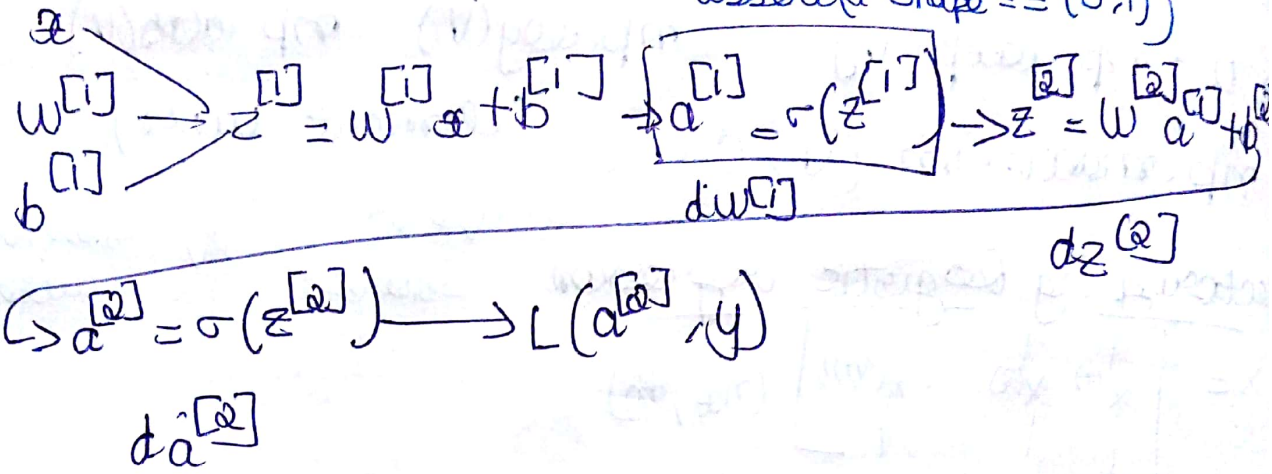
$a = np.random.randn(5)$

$a.shape = (5,)$

Not row, not column

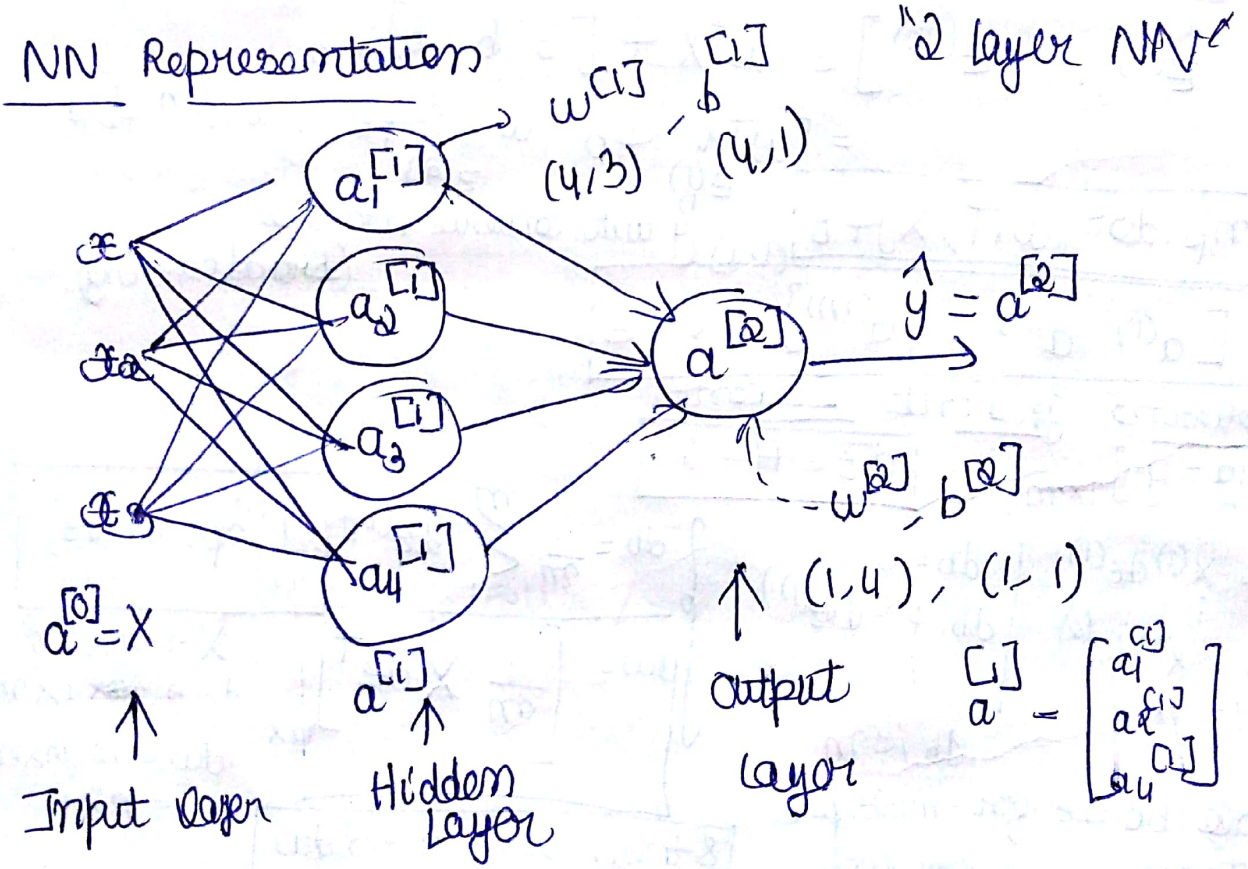
$np.dot(a, a.T) != 1$ for rank 1

assert(a.shape == (5,1))

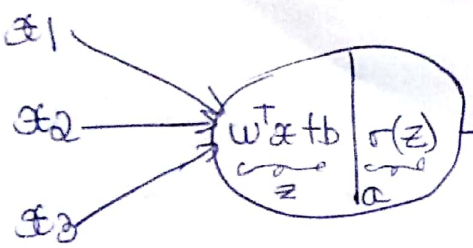


NN Representation

"2 layer NN"



Computing NN O/P



$z = w^T x + b$

$a = \sigma(z)$

z as a vector

$$z = \begin{bmatrix} -(w_1)^T- \\ -(w_2)^T- \\ -(w_3)^T- \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} w_1^T x + b \\ \vdots \end{bmatrix}$$

vectorized

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

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

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

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

Given input x:

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

$(4,1) \quad (4,3) \quad (3,1) \quad (4,1)$

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

$4\times1 \quad 4\times1$

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

$(1,1) \quad (1,4) \quad (4,1) \quad (1,1)$

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

$1\times1 \quad 1,1$

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

~~$z^{[1](i)} = w^{[1]}[i]$~~

unvectorized

for $i = 1$ to m,

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

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

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

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

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

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

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

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

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

$\vdots$

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

layer, example

Vectorized implementation

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

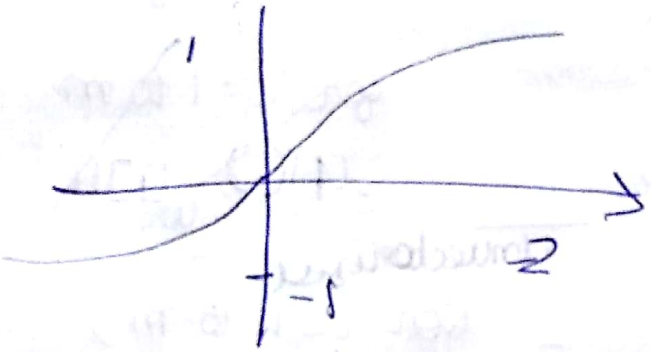
$$A^{[1]} = \sigma(z^{[1]})$$

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

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

Activation functions

tanh function hyperbolic tan



$$a = \tan(h) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

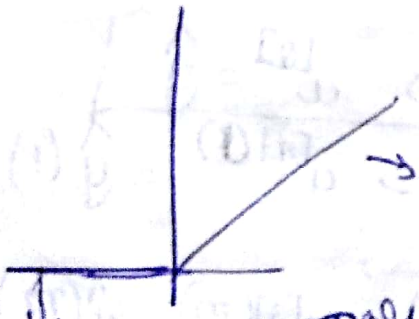
works better as mean will $b = 0$ (data is centred)

tanh → hidden layer

sigmoid → output layer as $\hat{y}$ should be $0 \le \hat{y} \le 1$

RELU

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



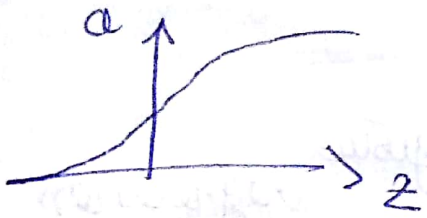
→ derivative 1 for here

→ for $z > 0$

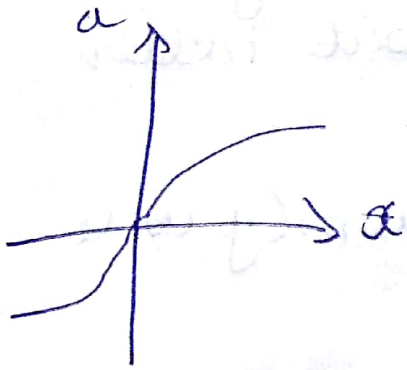
slope 0 (leaky ReLU better)

ReLU (tanh sometime) default for all hidden layers

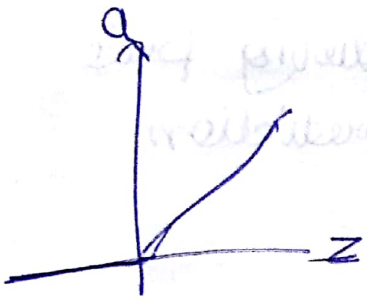
sigmoid → o/p layer



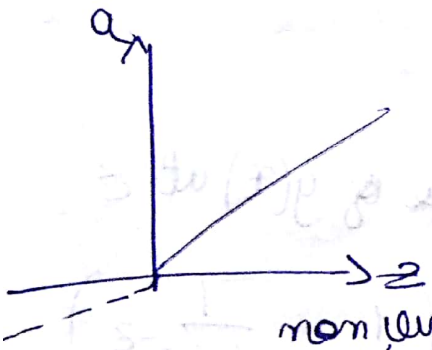
sigmoid : $a = \frac{1}{1+e^{-z}}$ } use only for o/p



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



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



Leaky RELU

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

Why use activation function

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

$a^{[1]} = z^{[1]}$ $\quad g(z) = z$ "linear activation function"

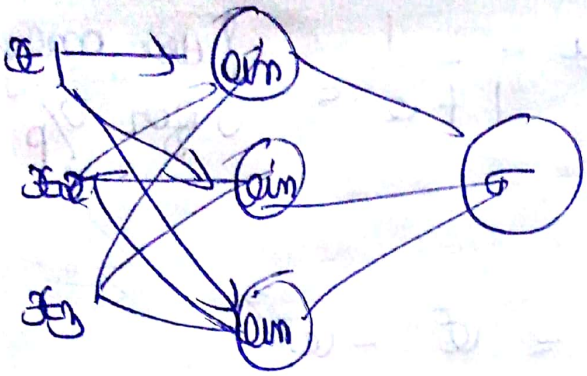
$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$

$a^{[2]} = z^{[2]}$

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

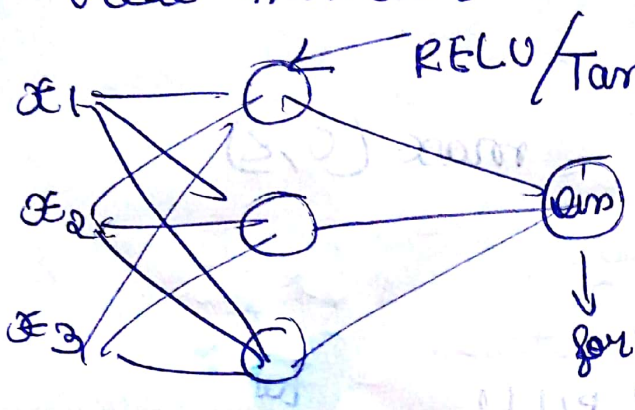
$= \underbrace{(w^{[2]} w^{[1]})}_{w'} x + \underbrace{(w^{[2]} b^{[1]} + b^{[2]})}_{b'}$

$= w' x + b'$ } linear regression



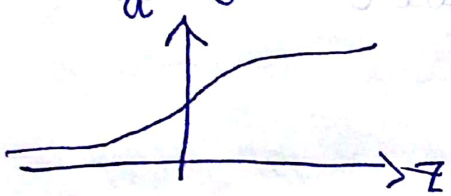
} Logistic regression without hidden layer.

$g(z) = z$ for regression problem (y is a real number)



} Having price prediction

Slope of activation functions



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

$z = 10$, $g(z) = \frac{1}{1+e^{-10}} = 1$

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

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

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

$z = -10$ $g(z) = \frac{1}{1+e^{10}} \approx 0$ $z = 0$ $g(z) = \frac{1}{2}$

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

$= \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$

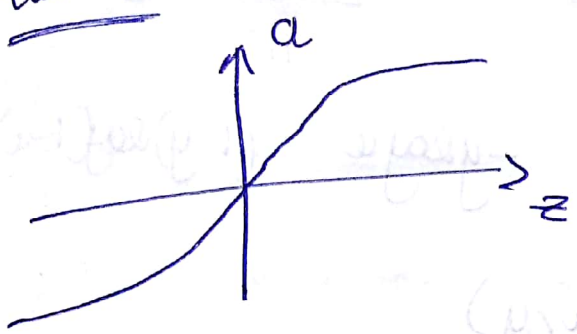
$$g'(z) = \frac{d}{dz}(g(z))$$

In NN's

$g'(z) = a(1-a)$ as $a^{[0]} = x$

$a^{[1]}$ next layer etc

tan h



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

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

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

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

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

$g'(z) = 1 - 1^2 = \underline{0}$

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

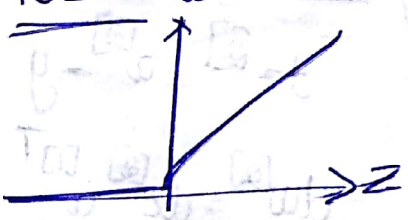
$g'(z) \quad 1 - (-1)^2 = \underline{0}$

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

$g'(z) = 1$

$\boxed{a = g(z) \Rightarrow g'(z) = 1 - a^2}$

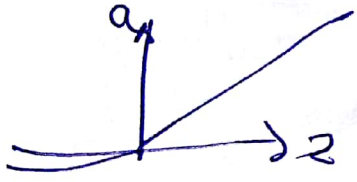
ReLU



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

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

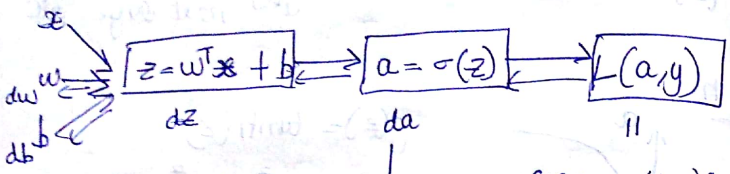
Leaky ReLU



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

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

Logistic gradients



$dz = a - y$

$-y\log a - (1-y)\log(1-a)$

$dz = da \cdot g'(z)$

$g(z) = \sigma(z)$

$da = \frac{d}{da} L(a,y)$

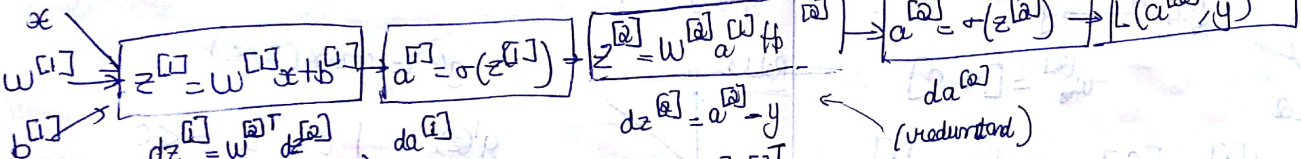
$= \frac{-y}{a} + \frac{1-y}{1-a}$

$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial z}$ } chain rule

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

$dw = dz \cdot x \qquad db = dz$

In NNs we do this twice



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

$da^{[1]}$

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

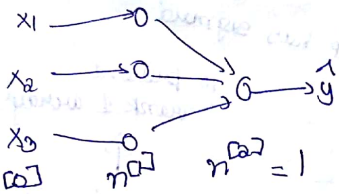
$dw^{[2]} = dz^{[2]} a^{[1]T}$

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

$da^{[2]}$ (redundant)

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

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



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

$z^{[2]}, dz^{[2]} \quad (n^{[2]}, 1) \quad - \quad (1,1)$

$z^{[1]}, dz^{[1]} \quad (n^{[1]}, 1)$

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

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

dimensions

$dw = dz \cdot x$ ← plays role of x

Vectorized implementation

$dz^{[2]} = A^{[2]} - Y$ $y \rightarrow$ o/p has sigmoid

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

To prevent rank 1 arrays

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

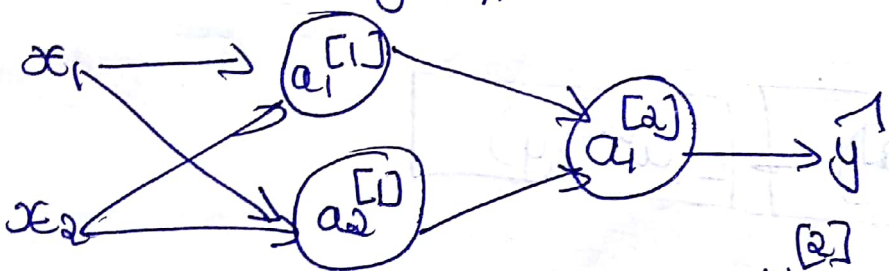
$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$

$n^{[1]}, m \quad n^{[1]}, m$ $\quad$ $n^{[1]}, m$ → derivate of activation

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

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

Random initialization



$n^{[0]} = 2 \quad n^{[1]} = 2$

$w_2^{[2]} = [0\ 0]$

$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow a_1^{[1]} = a_2^{[1]}$

In backprop $dz_1^{[1]} = dz_2^{[1]}$ [Symmetric]

$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\quad w^{[1]} = w^{[1]} - \alpha\, dw$

$w^{[1]}$ 1st row = 2nd row $\begin{bmatrix} \cdots \\ \cdots \end{bmatrix}$

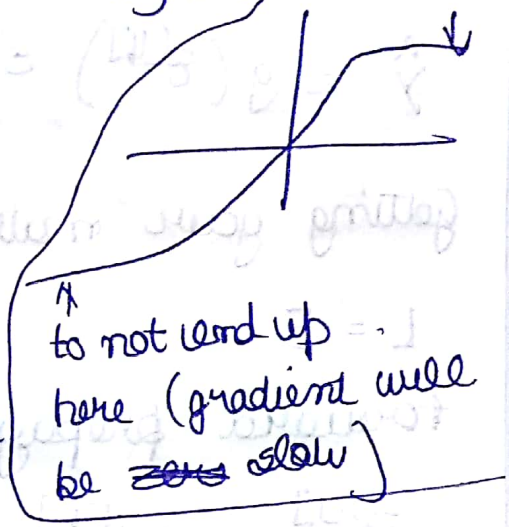
Hidden units calculate same thing in all iterations

$W^{[1]} = np.random.randn((2,2)) * 0.01$ — to make weights small

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

$W^{[2]} = np.random.randn((2,2)) * 0.0$

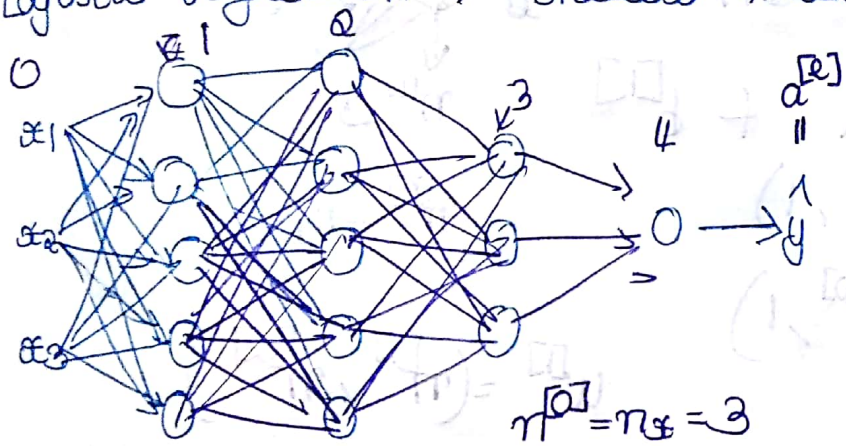
$b^{[2]} = 0$



Week - 4

Deep Neural Networks

Logistic regression → "shallow" model



$n^{[0]} = n_x = 3$

$n^{[1]} = 5$
$n^{[2]} = 5$
$n^{[3]} = 3$
$n^{[4]} = n^{[L]} = 1$

$L = 4$

$n^{[l]}$ = # units in layer l $\quad X = a^{[0]}$

$a^{[l]}$ = activations in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$ $\quad W^{[l]}$ = weights for $z^{[l]}$

$b^{[l]}$ =

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

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

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

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

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

Vectorized

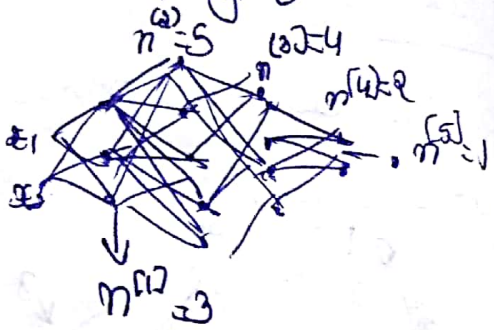
$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[1]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\hat{y} = g(Z^{[4]}) = A^{[4]}$$

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

to 4

Getting your matrix dimensions right.

$L = 5$



Forward propagation

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

$(3,1)$ $(3,2)$ $(2,1)$

[3 hidden layers

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

$n^{[0]} = n_x = 2$

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

$$W^{[2]} = (n^{[2]}, n^{[1]}) = (5,3)$$

$$Z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

$(5,1)$ $(5,3)$ $(3,1)$

$W^{[3]} = (4,5)$

$W^{[4]} = (2,4)$

$W^{[5]} = (1,2)$

$$W^{[l]} = (n^{[l]}, n^{[l-1]})$$

$$b^{[l]} = (n^{[l]}, 1)$$

$dw^{[l]} = (n^{[l]}, n^{[l-1]})$

$db^{[l]} = (n^{[l]}, 1)$

to 4 $z^{[l]} = g^{[l]}(a^{[l]})$ $\quad z^{[l]}, a^{[l]} = (n^{[l]}, 1)$

Same dimensions

Now for multiple examples.

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

$\begin{bmatrix} z_0 & \cdots & m \\ z_1 & \cdots & \\ z_n & \cdots & \end{bmatrix}$ $(n^{[1]}, n^{[0]})$ $(n^{[0]}, m)$ $(n^{[1]}, 1)$

broadcasting

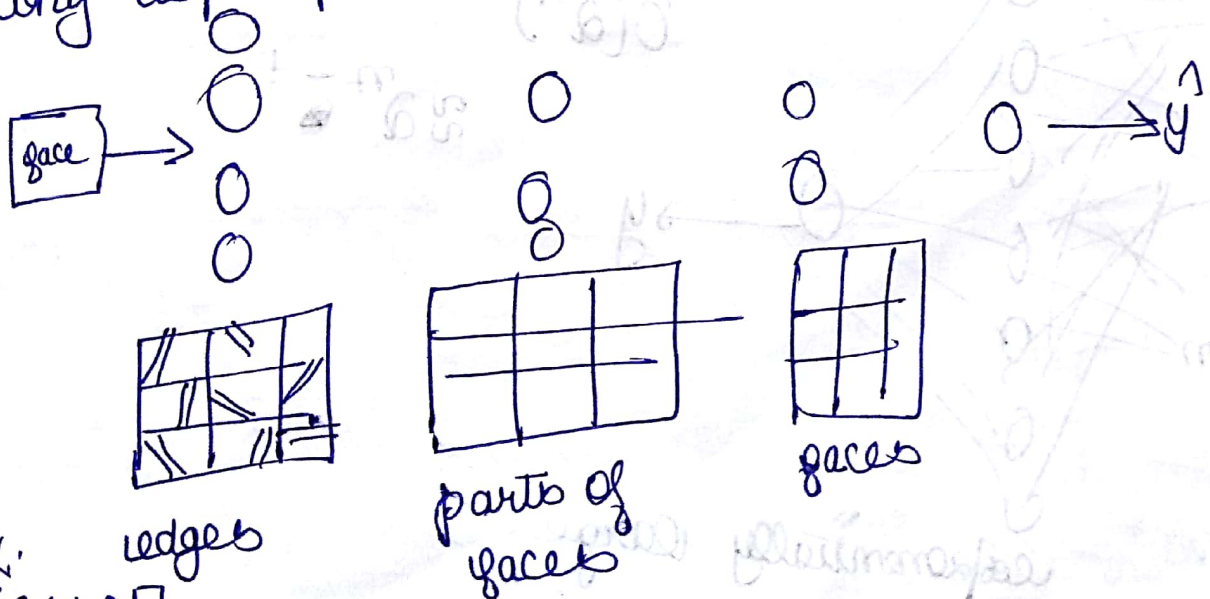
$n \times m$

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

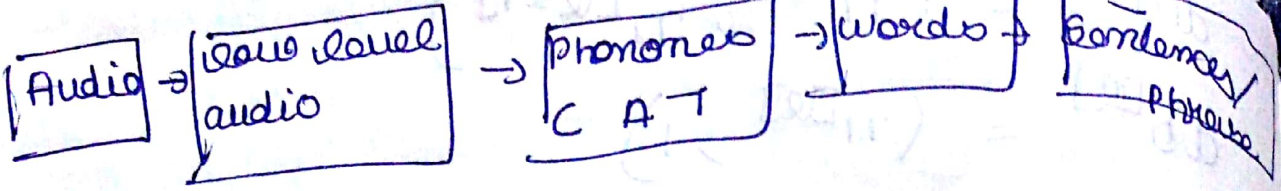
$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$l = 0 \quad A^{[0]} = X = (n^{[0]}, m)$

Very deep representation

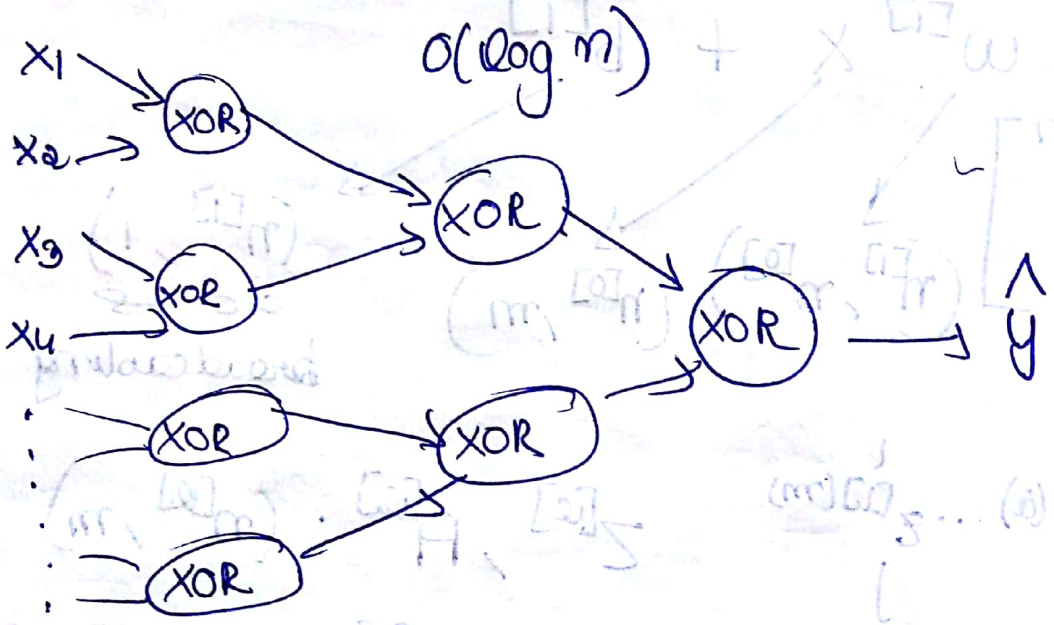


* [CNN]

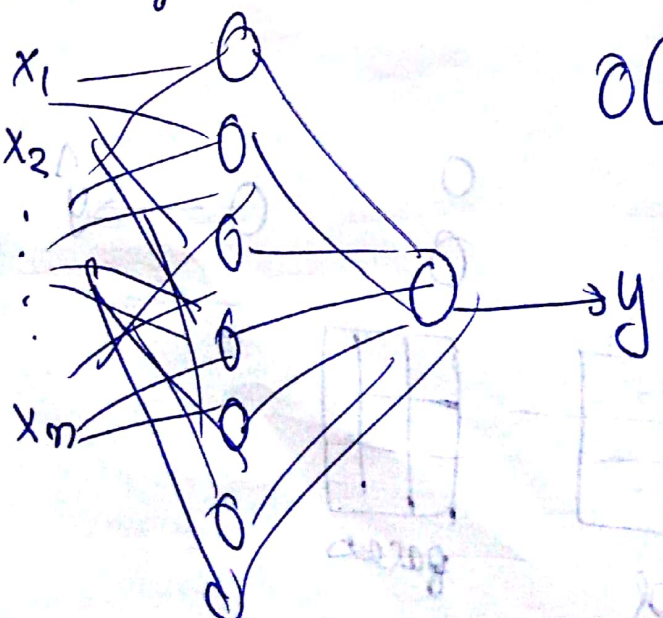


Circuit theory

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



But if were not allowed 'L-Layer



$O(2^n)$

$\approx 2^{n-1}$

exponentially large.

Building blocks of deep neural nets

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

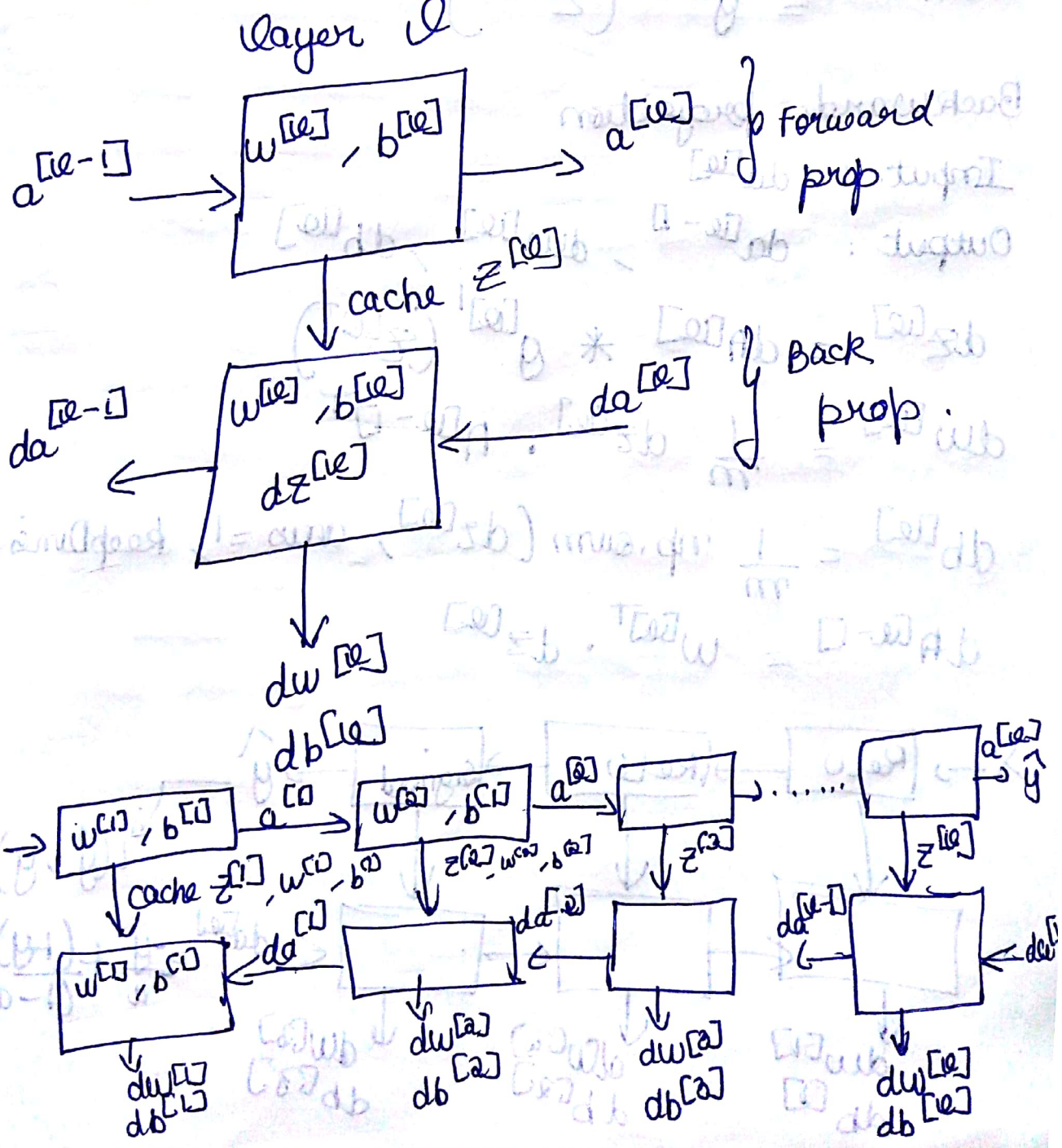
Forward Input: $a^{[l-1]}$, output $a^{[l]}$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ [cache $z^{[l]}$]

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

Backward: Input $da^{[l]}$, output $da^{[l-1]}$

(cache $z^{[l]}$). $dw^{[l]}$ $db^{[l]}$



$$w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$$
$$b^{[l]} = b^{[l]} - \alpha\, db^{[l]}$$

Forward propagation.

Input : $a^{[l-1]}$

output $a^{[l]}$, cache ($z^{[l]}$), $w^{[l]}$, $b^{[l]}$

$$z^{[l]} = w^{[l]} \cdot A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g^{[l]}(z^{[l]}) \qquad X = A^{[0]}$$

Backward projection

Input : $da^{[l]}$

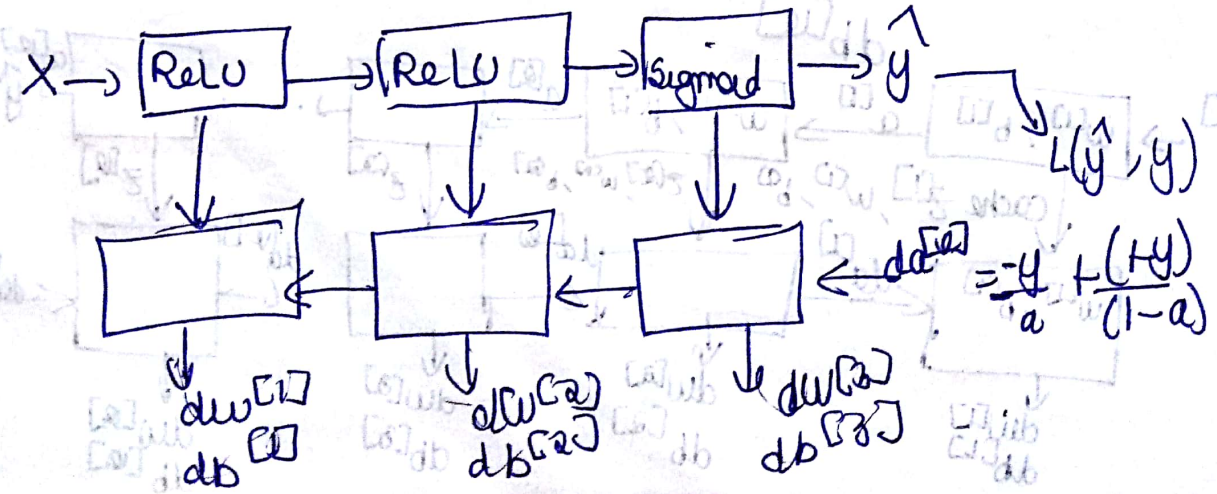
Output : $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$

$$dz^{[l]} = dA^{[l]} * g^{[l]'}(z^{[l]})$$

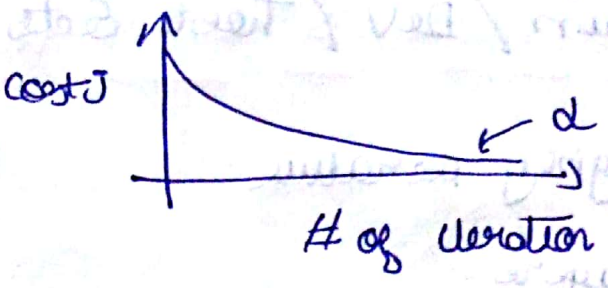
$$dw^{[l]} = \frac{1}{m}\, dz^{[l]} \cdot A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m}\, np.sum(dz^{[l]}, axis=1, keepDims=true)$$

$$dA^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$



Hyperparameters.



Learning rate α

iterations

hidden layer L

hidden units $n^{[1]}, n^{[2]}$

Choice of actuation functions

"It's like the brain"