Deep Learning

Notes : Perceptron , Activation Function , Gradient Descent , backpropagation , Loss Function , Metrics , Optimizer , Learning Rate , CNN , RNN , Structured Data , Unstructured Data , Research Papers , History

COURSE 1 : NN AND DEEPLEARNING

Notes

→ ReLu : Rectified Linear Unit

LOGISTIC REGRESSION AS AN NN

1/ Notation

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

Training exp : $m = \{(x_1, y_1), \dots (x_m, y_m)\}$

m_{train} m_{test}

$$X = \begin{bmatrix} \vdots \\ x_1, \dots, x_m \\ \vdots \end{bmatrix} \quad \uparrow n_x \downarrow \qquad \leftarrow m \rightarrow$$

$X \in \mathbb{R}^{n_x \times m}$

$X.shape = (n_x, m)$

$$Y = [y_1, \dots, y_m] \qquad Y \in \mathbb{R}^{1 \times m} \qquad Y.shape = (1, m)$$

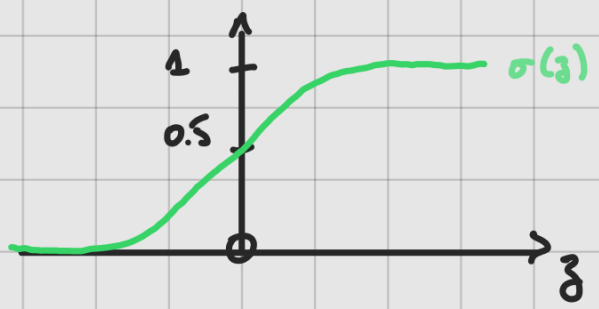
Logistic Reg

Given x, want $\hat{y} = P(y=1 \mid x)$ $\qquad x \in \mathbb{R}^{n_x}$

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

output : $\hat{y} = \sigma(\underbrace{w^T x + b}_{\text{Reg lin}})$ (sigm F)

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



In some convention Define new extra feature :

$x_0 = 1 \qquad x \in \mathbb{R}^{n_x + 1}$

$\hat{y} = \sigma(\theta^T x)$ avec $\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \\ \} w \end{matrix}$

=> more easier to implemat in NN

Logistic Regression Cost function

$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ (MSE) → Quadratic

! Non convex

Want convex fonction (for Gradient descent) Loss F

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

If $y = 1$: $\mathcal{L}(\hat{y}, 1) = -\log \hat{y}$ (want $\log \hat{y}$ large → $\hat{y}$ large)

$y = 0$: $\mathcal{L}(\hat{y}, 0) = -\log(1-\hat{y})$ want $\hat{y}$ small

Cost F: $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}_i, y_i)$

Gradient Descent

Repeat

$$w := w - \alpha \frac{dJ(w)}{dw}$$

lr

actualise

In code $dw =$ $\frac{dJ(w)}{dw}$

↳ $w := w - \alpha\, dw$

$J(w,b)$

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

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

In code dw, db

Computation Graph

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

$$v = a + u$$

$$J = 3v$$

$u = bc$

$v = a + u$

$J = 3v$

$a = 5$

$b = 3$

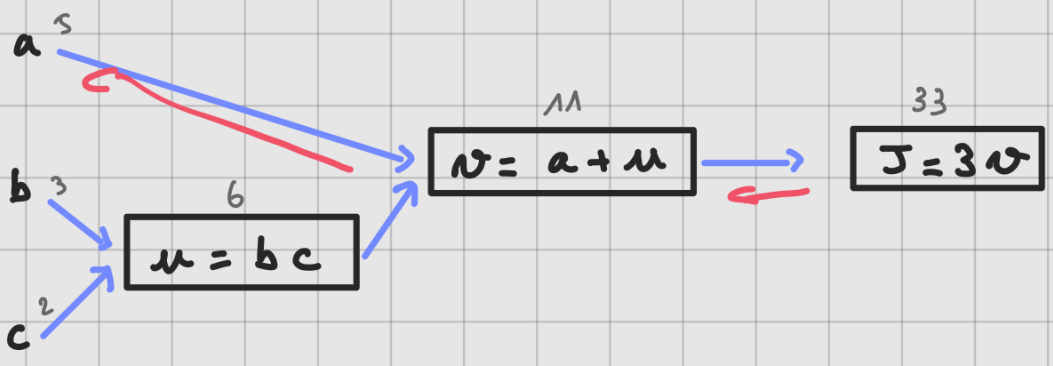
$c = 2$

$u = 6$

$v = 11$

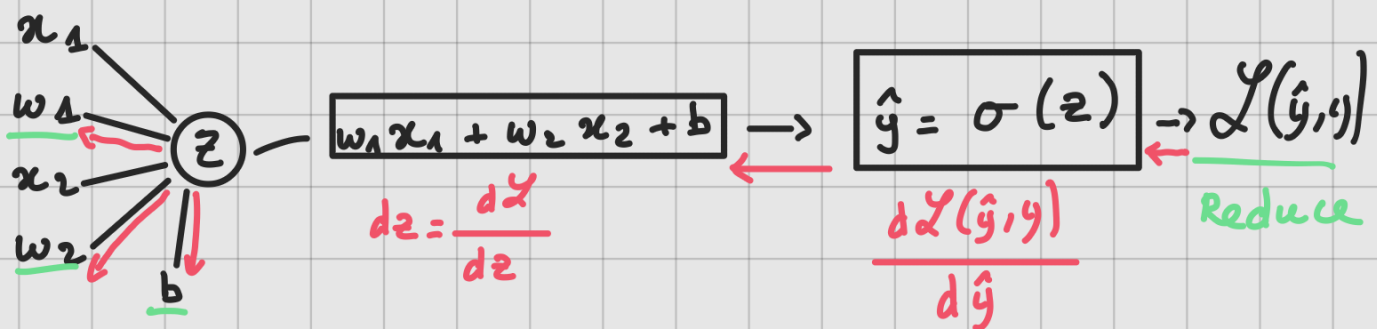
$J = 33$

$$\frac{dJ}{dv} = ? = 3$$



Logistic Regression Gradient Descent

with 2 var : x_1 , x_2



$$dz = \frac{dL}{dz} = \hat{y} - y$$

$$\frac{dL(\hat{y},y)}{d\hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

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

$$"dw_2" = x_2\, dz \qquad "db" = dz$$

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

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

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

Logistic Regression on m examples

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

$$\hat{y}_i = \sigma(z_i) = \sigma(w^T x_i + b)$$

$$\frac{d}{dw_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_i} \mathcal{L}(\hat{y}_i, y_i)}_{dw_1^{(i)}}$$

In also:

$J = 0$, $dw_1 = 0$, $dw_2 = 0$, $db = 0$

For $i = 1$ to m:

$$z_i = w^T x_i + b$$

$$\hat{y}_i = \sigma(z_i)$$

$$J \mathrel{+}= -\left[y_i \log \hat{y}_i + (1 - y_i) \log (1 - \hat{y}_i)\right]$$

$$dz_i = \hat{y}_i - y_i$$

$$\left.\begin{aligned} dw_1 &\mathrel{+}= x_1^{(i)} dz_i \\ dw_2 &\mathrel{+}= x_2^{(i)} dz_i \end{aligned}\right\} n = 2$$

$db \mathrel{+}= dz_i$

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

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

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

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

Weakness : 2 for loop ⚠

↳ have to vectorize

Vectorization

$$z = w^T x + b \qquad w = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad w \in \mathbb{R}^{n_x} \quad x \in \mathbb{R}^{n_x}$$

non vectorize :

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

Vec :

```
z = np.dot(w, x) + b
```

much faster

Vectorizing Logistic Regression

$z^{(1)} = w^T x^{(1)} + b$ $\quad z^{(2)} = w^T x^{(2)} + b \quad \dots$

$\hat{y}^{(1)} = \sigma(z^{(1)})$ $\quad \hat{y}^{(2)} = w^T x^{(2)} + b \quad \dots$

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

$$Z = [z^{(1)}, \dots z^{(m)}] = w^T X + \underbrace{[b, b, \dots b]}_{1 \times m}$$

$$= [w^T x^{(1)} + b, \dots, w^T x^{(m)} + b]$$

↳ np. dot $(w^T X) + b$

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

$dZ = \hat{Y} - Y$

$dw = 0$	$db = 0$
$dw += x^{(1)} dz^{(1)}$	$db += dz^{(1)}$
$dw += \dots$	⋮
⋮	
$dw /= m$	$db /= m$

} For loop

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

$$= \frac{1}{m} \; np.sum(dZ)$$

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

$$= \frac{1}{m} \begin{bmatrix} \vdots & & \vdots \\ x^{(1)} & \cdots & x^{(m)} \\ \vdots & & \vdots \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \left[x^{(1)} dz^{(1)} , \ldots , x^{(m)} dz^{(m)} \right]$$

h algo:

$J=0$, $dw_1 = 0$, $dw_2 = 0$, $db = 0$

For $i = 1$ to m:

$z_i = w^T x_i + b$

$\hat{y}_i = \sigma(z_i)$

$J \mathrel{+}= -\left[y_i \log \hat{y}_i + (1-y_i) \log (1-\hat{y}_i) \right]$

$dz_i = \hat{y}_i - y_i$

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

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

} $n=2$

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

$db \mathrel{+}= dz_i$

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

$Z = w^T X + b$

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

$\hat{Y} = \sigma(Z)$

$dZ = \hat{Y} - Y$

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

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

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

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

Broadcasting in Python

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100 \rightarrow \begin{bmatrix}100\\100\\100\\100\end{bmatrix} = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}_{(2,3)} + \begin{bmatrix}100 & 200 & 300\end{bmatrix} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$$

$$\begin{bmatrix}100 & 200 & 300\end{bmatrix} \rightarrow \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix}_{(2,3)}$$

$$+ \begin{bmatrix}100\\200\end{bmatrix} \rightarrow \begin{bmatrix}100 & 100\\200 & 200\end{bmatrix} = \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$$

$r + s$

$(u \times v) + (w \times x)$

$(a+b)(a-b) + (b+c)(b-c)$

$a^2 - ab + ab - b^2 + b^2 - c^2$

Algo

Forward and Backward Propagation

Inputs :

- w : weights
- b : biais
- X , Y

Outputs :

- Cost
- Grads

Forward

$$A = \sigma(w^T X + b)$$

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

En numpy : `A = sigmoid(np.dot(w.T, X) + b)`

`J = (1/m) * np.sum(Y * np.log(A) + (1-Y) * np.log(1-A))`

Backward

$$\frac{\partial J}{\partial w} = \frac{1}{m} X (A-Y)^T$$

`dw = 1/m * np.dot(X, (A-Y).T)`

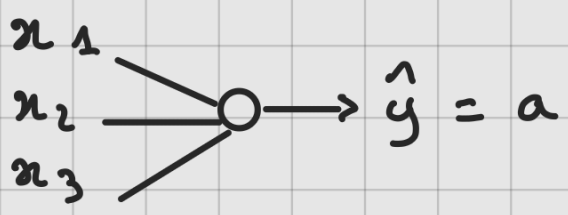
$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

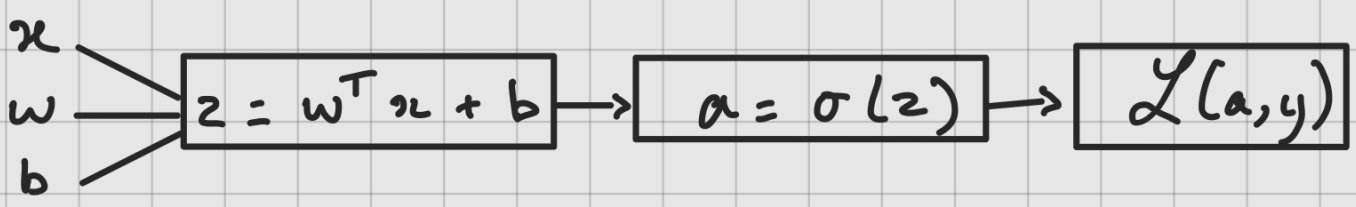
`db = 1/m * np.sum(A-Y)`

Return : Cost (J), Grads (dw , db)

NEURAL NETWORK

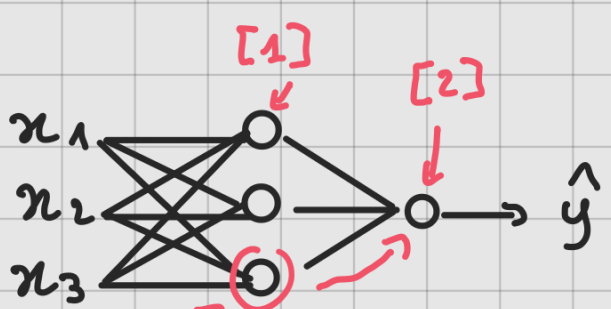
Logistic :

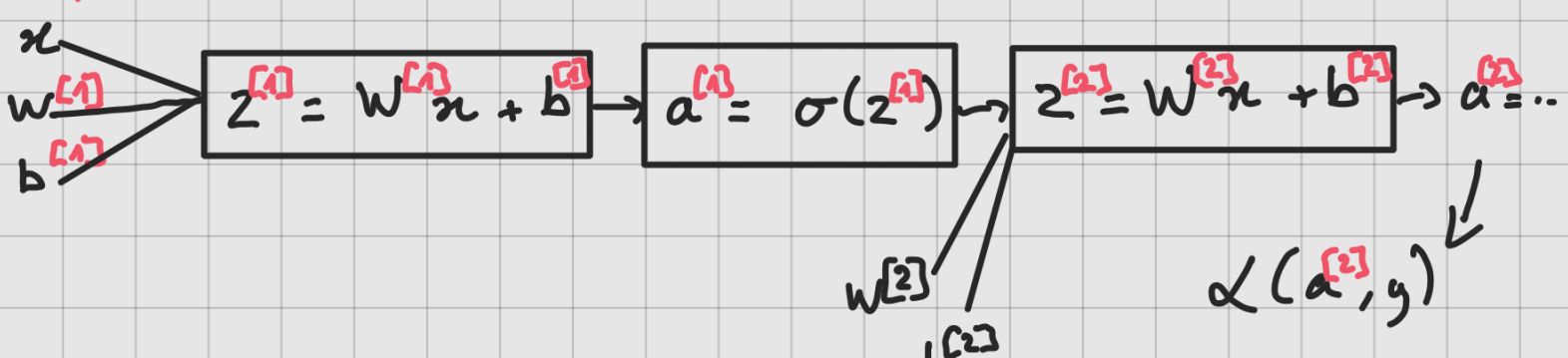




Neural Network :

$[i]$ → layers
(i) → exemple



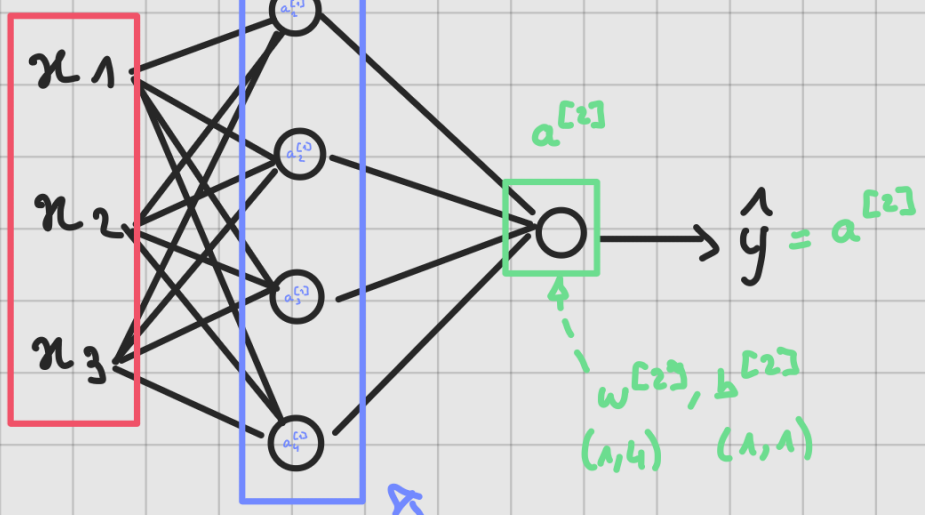


Notation :

$a^{[0]} = X$

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

2 layers NN (ø Input)



input Layer

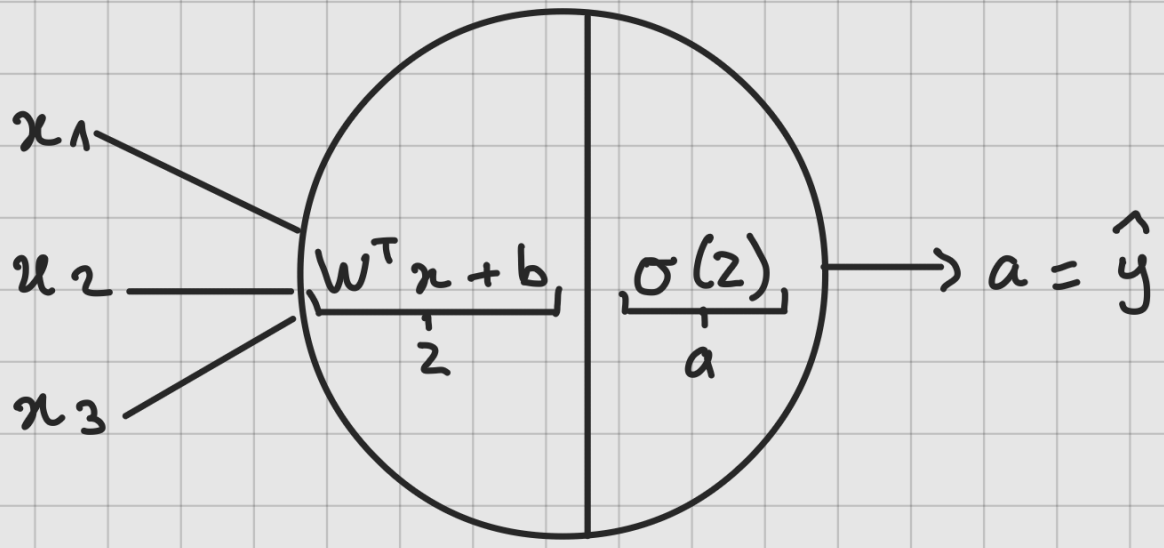
hidden layer

Output layer

$w^{[1]}, b^{[1]}$

(4,3) (4,1)

One nodes :



first N

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

fourth

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

$W^{[1]}$ $\quad$ $b^{[1]}$

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

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

$$z^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

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

Given an input x :

$$z^{[1]} = W^{[1]} \underbrace{x}_{a^{[0]}} + b^{[1]}$$

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

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

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

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

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

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

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

Vectorizing

$$\left.\begin{aligned} z^{[1]} &= W^{[1]} x + b^{[1]} \\ a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \end{aligned}\right\}$$

For $i = 1$ to m

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

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

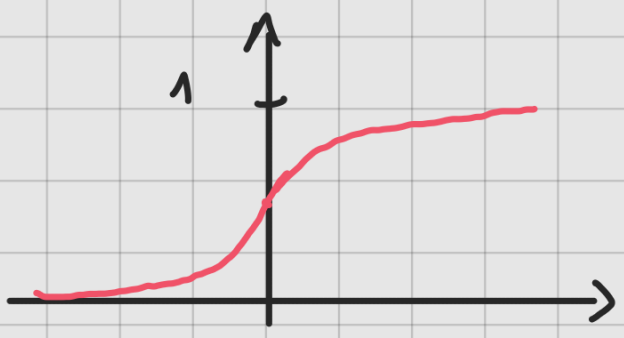
$\vdots$

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

$x^{(i)} \longrightarrow a^{[2](i)}$

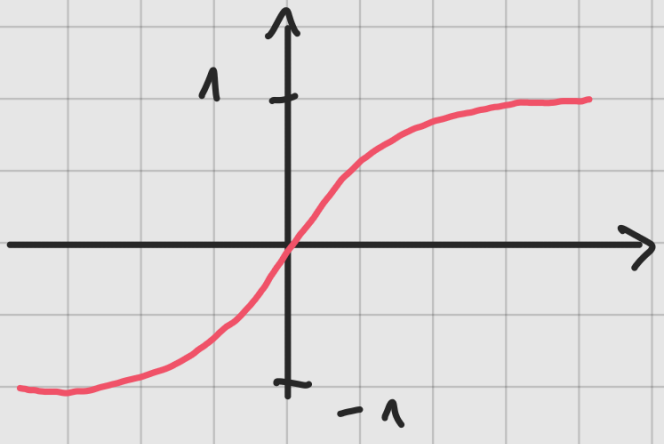
Activation Function

Sigmoid (used in output layer binary class)



$$a = \frac{1}{1 + e^{-x}}$$

Tanh : (almost always better) (exept output layer)



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

Relu :

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

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

Derivatives

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

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

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

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

Relu

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

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

Leaky Relu

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

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

Gradient Descent

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

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

$n_x = n^{[0]}$: nb input ; $n^{[1]}$: nb hidden ; $n^{[2]} = 1$: output

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

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

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

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

Cost function :

$$J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\underbrace{\hat{y}}_{a^{[2]}}, y)$$

Repeat :

- Compute pred $(\hat{y}^{(1)}, \dots, \hat{y}^{(m)})$
- $dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...
- $W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$
- $b^{[1]} := b^{[1]} - \alpha \, db^{[1]}$
- ⋮

Forward Propagation :

$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$

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

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

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

Back Propagation :

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

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

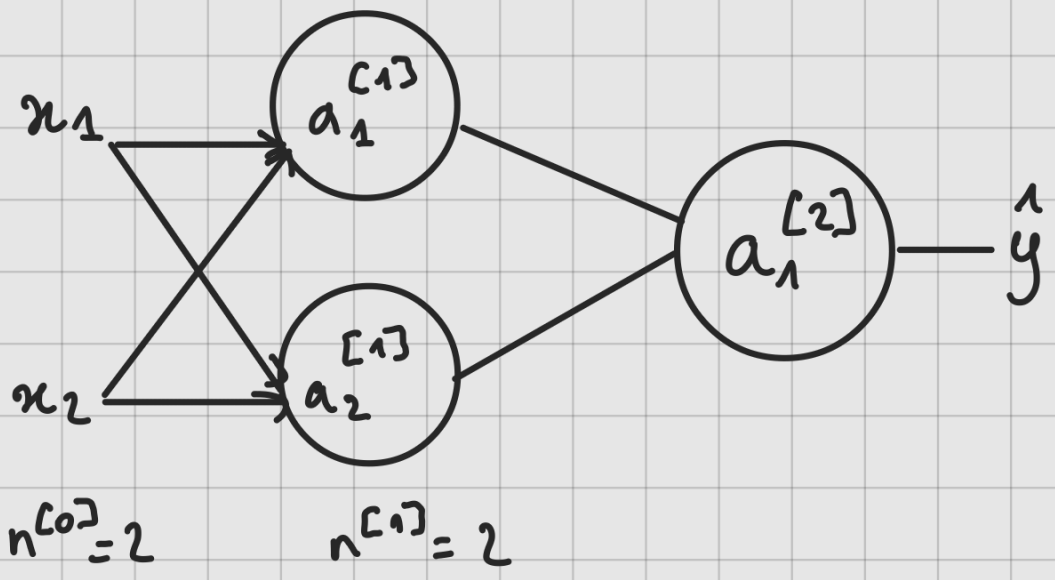
$$db^{[2]} = \frac{1}{m} \text{np.sum}(dZ^{[2]}, \text{axis}=1, \text{keepdims}=T)$$

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'}(Z^{[1]})}_{(n^{[1]}, m)}$$

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

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

Random Initialization:



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

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

$\downarrow$

Symetric $\rightarrow$ do exatly the same thing

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

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

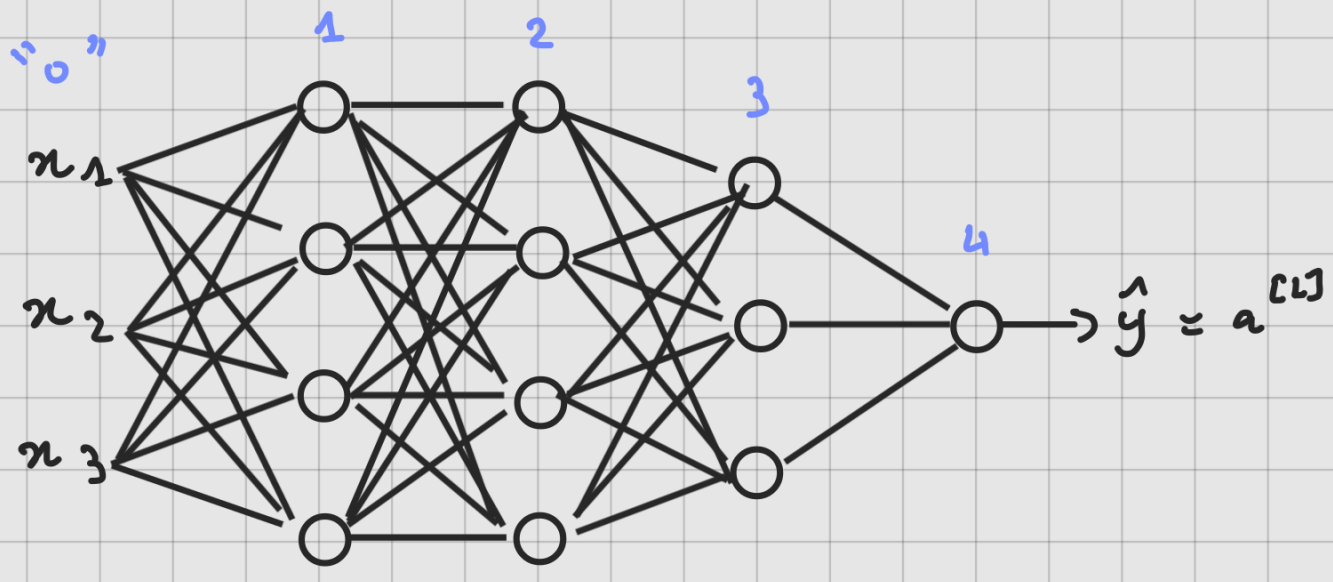
to have small values

$\searrow$ 2

GD faster

Deep NN

4 layer NN:



$L = 4$ (nb of layers)

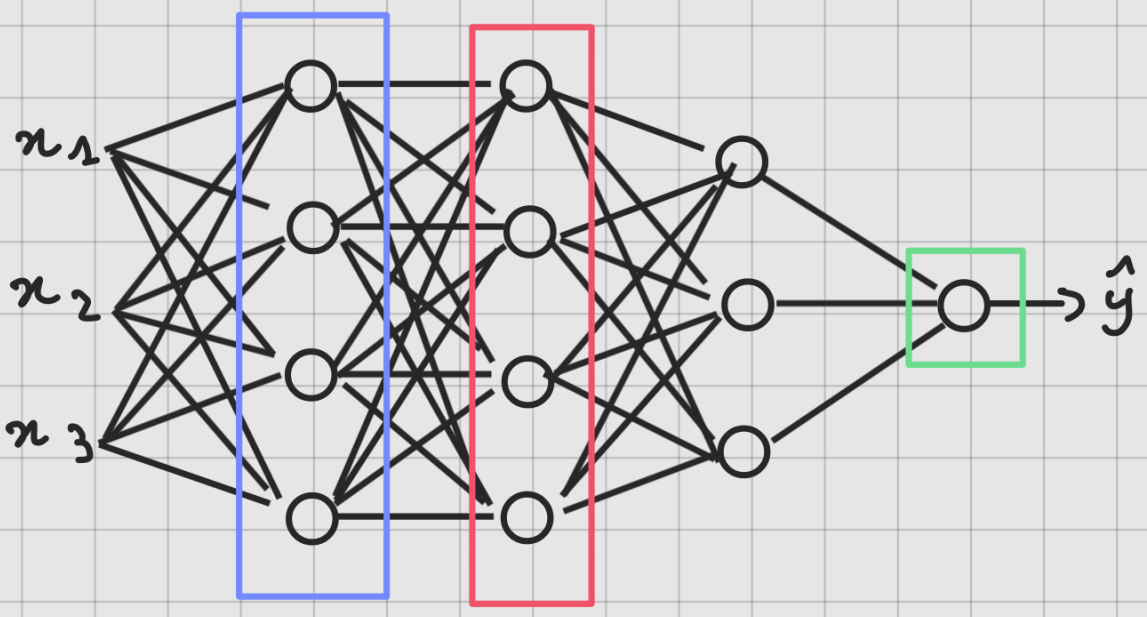
$n^{[l]}$ = nb of units in layer l

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

$n^{[0]} = n_x = 3$ $\quad X = a^{[0]}$

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

Forward Propagation:



One output X:

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

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

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

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

$$\vdots$$

$$Z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}$$

$$a^{[4]} = g^{[4]}(Z^{[4]}) = \hat{y}$$

$$Z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}$$

$$a^{[\ell]} = g^{[\ell]}(Z^{[\ell]})$$

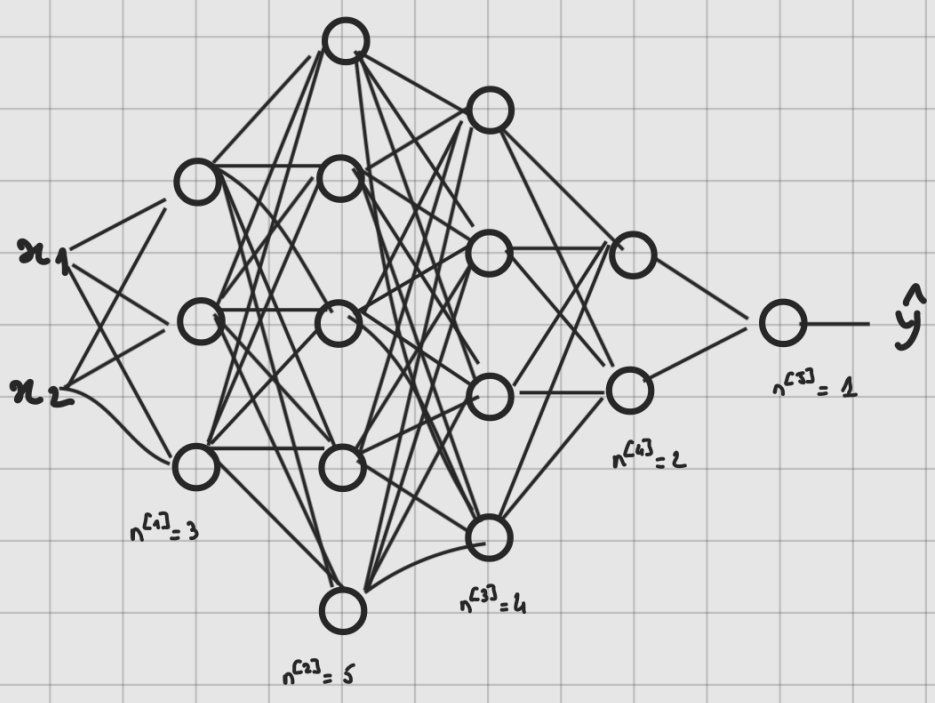
Vectorize :

$$z^{[1]} = w^{[1]} X + b^{[1]} \quad \dots \quad \hat{y} = g^{[4]}(z^{[4]}) = A^{[4]}$$

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

↳ loop 1 to 4 (layers) For loop

Getting Matrix Dimensions Right



$L = 5$

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

$(3,1) \quad (3,2) \quad (2,1)$

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

$$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

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

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

Vectorize

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

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

Python broadcast → $(n^{[1]}, m)$

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

Why deep representations ?

take a look at Circuit theory and deep Learning

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

Building Blocks of DNN :

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

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

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

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

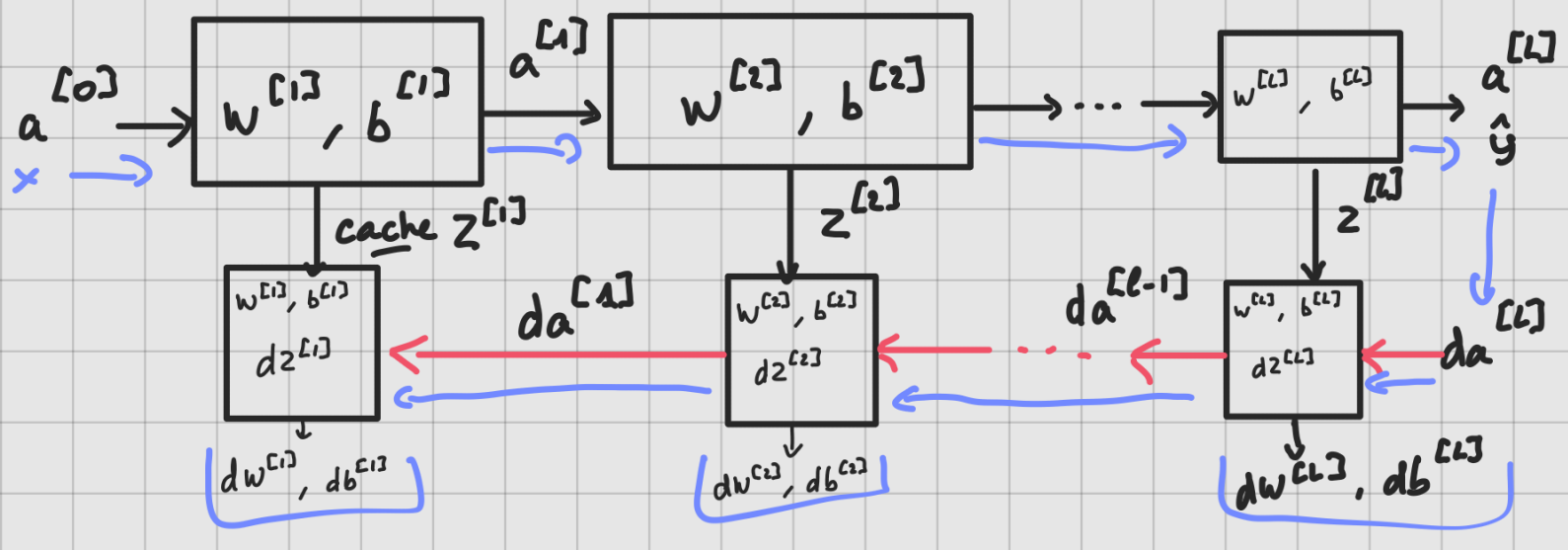
Backward: Input $da^{[l]}$ (cache $z^{[l]}$), output $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$

layer l: $a^{[l-1]} \rightarrow [\, w^{[l]}, b^{[l]} \,] \rightarrow a^{[l]}$ Fwd

cache $z^{[l]}$

$da^{[l-1]} \leftarrow [\, w^{[l]}, b^{[l]}, dz^{[l]} \,] \leftarrow da^{[l]}$ Bwd

$dw^{[l]}$, $db^{[l]}$



$$W^{[l]} := W^{[l]} - \alpha\, dW^{[l]}$$
$$b^{[l]} := b^{[l]} - \alpha\, db^{[l]}$$

Forward Propagation:

$w^{[l]}, b^{[l]}$ ↙

Input: $a^{[l-1]}$ Output: $a^{[l]}$, cache $(z^{[l]})$

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

Vectorize

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

Backward Propagation:

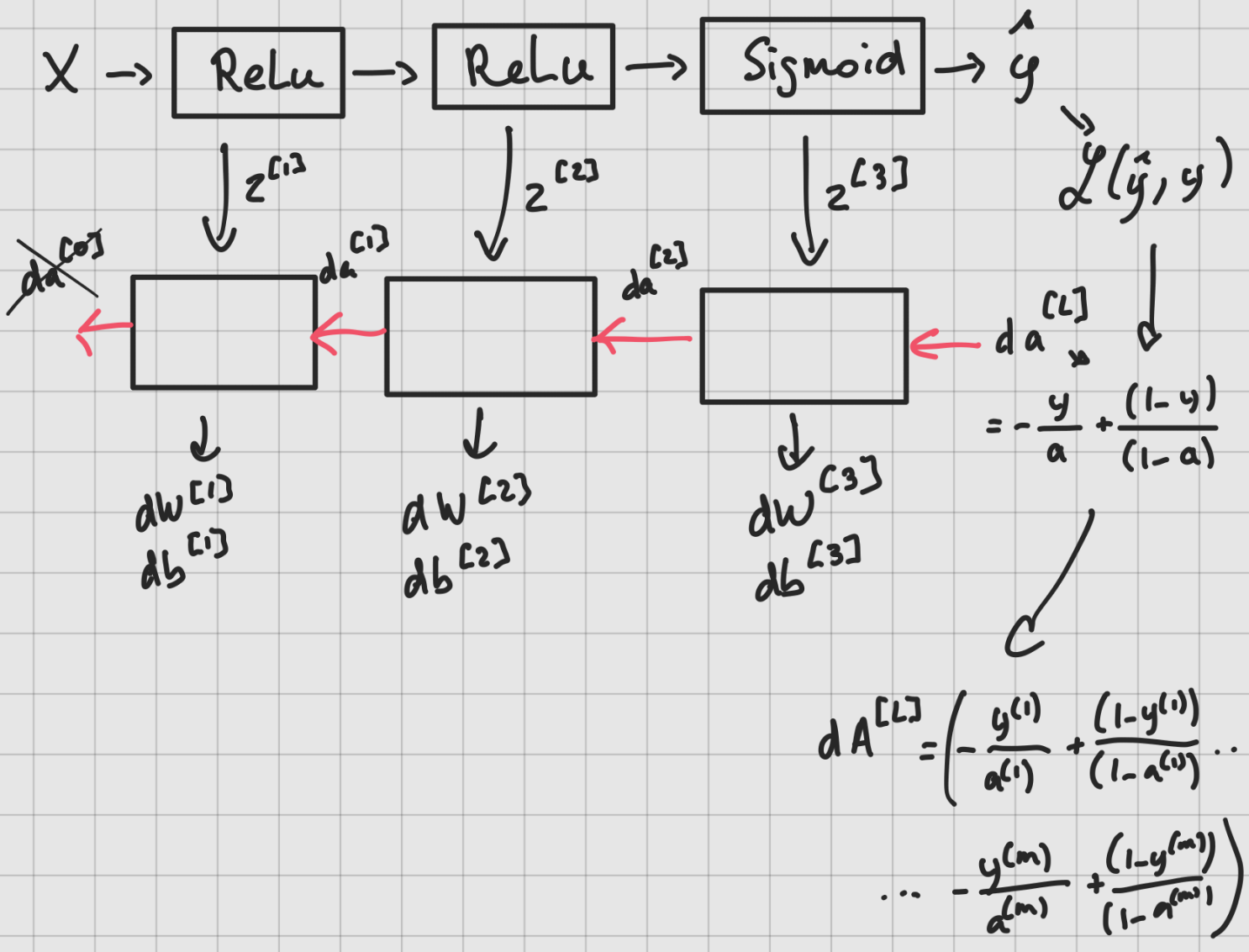
Input: $da^{[l]}$ Output: $da^{[l-1]}, dw^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$$
$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}_{\text{cache}}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

Vec

$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, ax=1, keepDims)$$
$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

Summarise :



$$dA^{[L]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots \right.$$

$$\left. \cdots - \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})} \right)$$

Parameters vs Hyperparameters :

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

Hyperparameters : - Learning rate α ~

- Nb of iterations
- Nb of Hidden Layers L

- Nb of Hidden units $n^{[1]}$, $n^{[2]}$, ...

- Choice of Activation func

Other hyperparams : Momentum, minibatch size, regularization, ...