Binary Classificat°

Cat (1) v/s Non-Cat (0)

$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ \vdots \end{bmatrix}$ Image - $\underbrace{64 \times 64}_{\text{Image dimen}^n} \times \overset{R,G,B}{3} = 12288$ $\quad n = n_x = 12288$

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

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

$m = m_{train} \qquad m_{test}$ = #tst examples

$$X = \begin{bmatrix} | & | & & | \\ x^1 & x^2 & \cdots & x^m \\ | & | & & | \end{bmatrix} \begin{matrix} \uparrow \\ n_x \\ \downarrow \end{matrix} \qquad X \in \mathbb{R}^{n_x \times m}$$

$\leftarrow m \rightarrow$

[python: X.shape = n_x, m]

$$Y = [y^1, y^2 \ldots y^m] \qquad Y \in \mathbb{R}^{1 \times m}$$

[python: Y.shape = $(1, m)$]

LOGISTIC REGRESSION

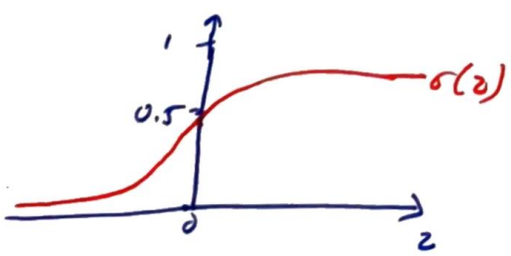
Given x, want $\hat{y} = P(y=1 \mid x)$ where $0 \le \hat{y} \le 1$

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

Output $\hat{y} = w^T x + b$ } {-> it would be difficult to predict ∵ here $\hat{y}$ might be >1 or <0 }

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

↓ sigmoid



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

If z large $\Rightarrow \quad \sigma(z) \approx \frac{1}{1+0} = 1$

If z large -ve no. $\Rightarrow \quad \sigma(z) = \frac{1}{1 + \text{Big no.}} \approx 0$

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

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

Loss (error) f^:

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

If $y=1$, then $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ $\Rightarrow$ want $\log \hat{y}$ to be large, $\therefore$ want $\hat{y}$ large but $\because$ it is binary classificat^ problem, $\hat{y}$ cannot be >1 $\Rightarrow$ we want $\hat{y}$ to be as close as 1

If $y=0$, then $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y}) \Rightarrow$ want $\log(1-\hat{y})$ to be large, n'ly want $\hat{y}$ small n'ly want $\hat{y}$ to be as close to 0

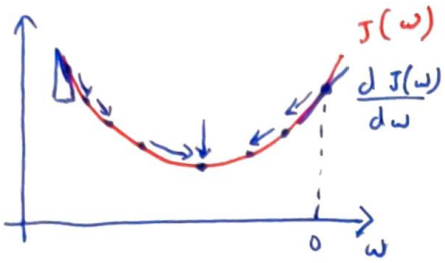
Cost f^:

$$J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^i, y^i) = \frac{-1}{m} \sum_{i=1}^{m} \left[y^i \log \hat{y}^i + (1-y^i) \log(1-\hat{y}^i)\right]$$

or $$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^i \log \hat{y}^i + (1-y) \log(1-\hat{y}^i)\right]$$

Gradient Descent:

Want to find w, b that minimize $J(w,b)$ → convex f^



Repeat : {

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

}

wly : $J(w,b)$

Repeat {

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

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

}

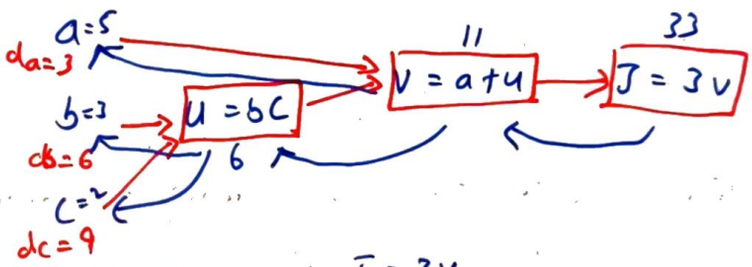
Computation Graph

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

$u = bc$

$v = a + u$

$J = 3v$



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

$J = 3v$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

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

$a = 5 \rightarrow 5.001$

$v = a + u = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

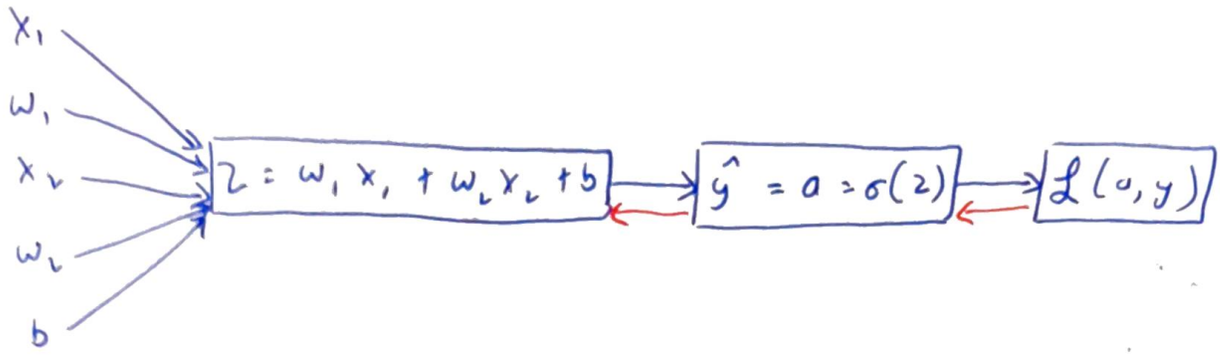
$\Rightarrow \frac{dJ}{da} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{da}$ → "da"

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

$\frac{d(\text{Final Output Variable})}{d(\text{var})}$ } ⇒ in code "d var"

wly $\frac{dJ}{du} = 3$ {"du"}, $\frac{dJ}{db} = \frac{dJ}{du}\frac{du}{db} = 3 \times 2 = 6$ {"db"}

wly "dc" = 9



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

$$\text{"}dz\text{"} = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) \cdot \left(a(1-a)\right)$$

$$= a - y$$

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

$$\text{"}dw_2\text{"} = x_2 \cdot dz$$

$$\text{"}db\text{"} = dz$$

$$\Rightarrow \begin{cases} w_1 := w_1 - \alpha\, dw_1 \\ w_2 := w_2 - \alpha\, dw_2 \\ b := b - \alpha\, db \end{cases}$$

Gradient Descent on m training examples:

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

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

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{d}{dw_1} \mathcal{L}(a^i, y^i)}_{dw_1^i - (x^i, y^i)}$$

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

For $i = 1$ to m

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

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

$$J_t = -\left[y^i \log a^i + (1-y^i) \log(1-a^i)\right]$$

$$dz^i = a^i - y^i$$

$$dw_1 \; t = x_1^i dz^i$$

$$dw_2 \; t = x_2^i dz^i$$

$$db \; t = dz^i$$

$\uparrow n = 2$

$J /= m$

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

$$\Rightarrow dw_1 = \frac{dJ}{dw_1}$$

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

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

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

Issue : :: 2 for loops < for m training sets, for each feature ⇒ making computation complex

⇒ Solution

VECTORIZATION

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

Non Vectorized:

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

Vectorized:

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

($w^T x$)

Logistic regression derivatives (Vectorized)

```
J = 0 , dw = np.zeros(n_x, 1) , db = 0
for i = 1 to m :
    z^i = w^T x^i + b
    a^i = σ(z^i)
    J += -[y^i log ŷ^i + (1-y)(log(1-ŷ))]
    dz^i = a^i (1 - a^i)
    dw += x^i dz^i
    db += dz^i
J = J/m , dw /= m , db = db/m
```

VECTORIZING LOGISTIC REGRESSION

$z^{(1)} = w^T x^{(1)} + b$ $\quad z^{(2)} = w^T x^{(2)} + b$ $\quad z^{(3)} = w^T x^{(3)} + b$

$a^{(1)} = \sigma(z^{(1)})$ $\quad a^{(2)} = \sigma(z^{(2)})$ $\quad a^{(3)} = \sigma(z^{(3)})$

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

$$\begin{bmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(m)} \end{bmatrix} = w^T X_T \begin{bmatrix} b & b & \cdots & b \end{bmatrix}_{1\times m}$$

$$= w^T \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}_{n\times m} \begin{bmatrix} b & b & \cdots & b \end{bmatrix}_{1\times m}$$

$$Z = \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \cdots & w^T x^{(m)} + b \end{bmatrix}$$

{Python: `Z = np.dot(w.T, x) + b`} $\Rightarrow$ "Broadcasting"

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \cdots & a^{(m)} \end{bmatrix} = \sigma(Z)$$

VECTORIZING LOGISTIC REGRESSION'S GRADIENT DESCENT

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

$$dZ = \begin{bmatrix} dz^{(1)} & dz^{(2)} & \cdots & dz^{(m)} \end{bmatrix}_{1\times m}$$

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \cdots & a^{(m)} \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} & \cdots & y^{(m)} \end{bmatrix}$$

$$dZ = A - Y = \begin{bmatrix} a^{(1)} - y^{(1)} & a^{(2)} - y^{(2)} & \cdots & a^{(m)} - y^{(m)} \end{bmatrix}$$

```
dw = 0
dw += X(1) dz(1)
dw += X(2) dz(2)
  :
dw /= m
```

```
db = 0
db += dz(1)
db += dz(2)
  :
db += dz(m)
db /= m
```

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

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

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

$= \frac{1}{m} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{pmatrix} dz^{(1)} \\ dz^{(2)} \\ \vdots \\ dz^{(m)} \end{pmatrix}$

$= \frac{1}{m} \left[x^{(1)} dz^{(1)} + \cdots x^{(m)} dz^{(m)} \right]_{n \times 1}$

Implementing Logistic Regression

$z = w^T x + b$

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

$A = \sigma(z)$

$dz = A - Y$

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

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

$w = w - \alpha\, dw$

$b = b - \alpha\, db$

Broadcasting Example

Calories from Carbs, Proteins, Fats in 100 g of diff. foods:

	Apples	Beef	Eggs	Potatoes
Carbs	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$= A_{3\times4}$ (3, 4)

Calculate % of calories from Carbs, Potatoes, Fat. Can you do this without explicit for-loop!

```
A = np.array([[ ... ],
              [ ... ],
              [ ... ]])
```

```
cal = A.sum(axis=0)
~~percentage = A/cal.reshape(1,4) * 100~~
percentage = A/cal * 100
(3,4)  (1,4)
```

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

3

$m \times n$

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{m \times n} + \begin{bmatrix} 100 \\ 200 \end{bmatrix} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix}$

General Principle:

(m,n) matrix $\pm$ $(1,n) \rightsquigarrow (m,n)$

$*$ $(m,1) \rightsquigarrow (m,n)$

$/$

$(m,1) \pm \mathbb{R}_{(1,1)} \rightsquigarrow (m,1)$

$(1,n) * \mathbb{R}_{(1,1)} \rightsquigarrow (1,n)$

$(1,n) / \mathbb{R}_{(1,1)} \rightsquigarrow (1,n)$

Note:

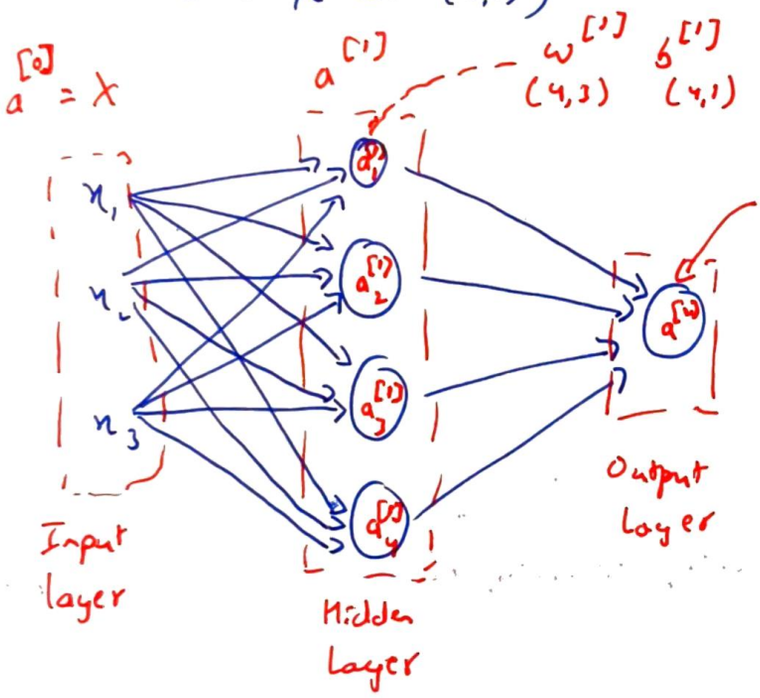
i) `a = np.random.randn(5)`

`a.shape : (5,)` => "rank 1 array" } - Don't use

`a = np.random.randn(5,1)` -> shape (5,1) => column vector

`a = np.random.randn(1,5)` -> shape (1,5) => row vector

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

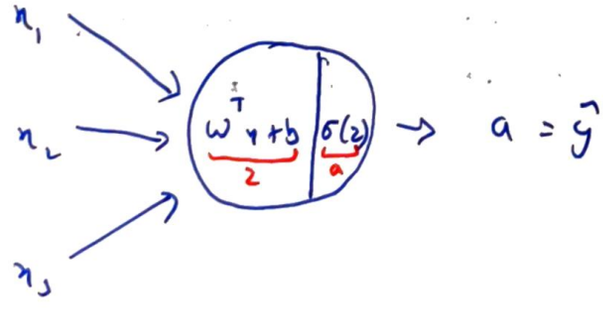


"2 layer NN"

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

"$\hat{y} = a$"

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix}$$



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

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

$a_i^{[1]} \rightarrow$ layer

$a_i \rightarrow$ node in layer

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

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

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

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

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

$$z^{[1]} = \underbrace{\begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ — & w_3^{[1]T} & — \\ — & w_4^{[1]T} & — \end{bmatrix}}_{W^{[1]}}_{4\times 3} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}_{4\times 1} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

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

Given Input x ($a^{[0]}$):

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

$$\rightarrow a^{[1]}_{4\times 1} = \sigma(z^{[1]})_{4\times 1}$$

$$\rightarrow z^{[2]}_{1\times 1} = w^{[2]}_{1\times 4} a^{[1]}_{4\times 1} + b^{[2]}_{1\times 1}$$

$$\rightarrow a^{[2]}_{1\times 1} = \sigma(z^{[2]})_{1\times 1}$$

Vectorizing across multiple examples:

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

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

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

$\vdots$

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

$a^{[2](i)}$: layer 2; $\rightarrow$ training eg. i

for $i = 1$ to m:

$$\begin{cases} z^{[1](i)} = w^{[1]} x^{(i)} + b^{[1]} \\ a^{[1](i)} = \sigma(z^{[1](i)}) \\ z^{[2](i)} = w^{[2]} a^{[1](i)} + b^{[2]} \\ a^{[2](i)} = \sigma(z^{[2](i)}) \end{cases}$$

To compute:

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

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

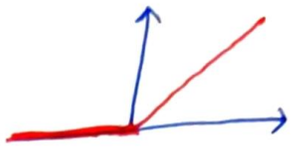
$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

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

hidden units. (↕) training egs. (↔)

ReLU Rectified Linear Unit

$a = \max(0, z)$



Note:

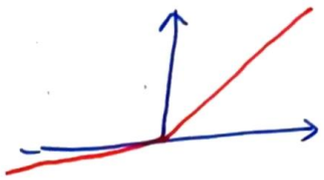
If the value of output is (0,1) → sigmoid activation in output unit

ReLU - default

Advantage → derivative = 0 when -ve value

Leaky ReLU:

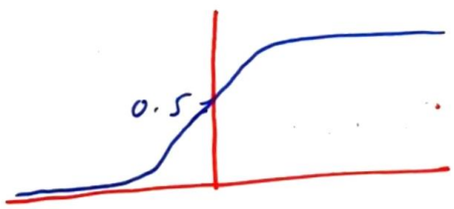
$a = \max(0.012, z)$



Derivations of Activation Fn:

1) Sigmoid activation fn:

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



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

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

If $z = 10$, $g(z) \simeq 1$

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

when $z = -10$ $\quad g(z) \simeq 0$

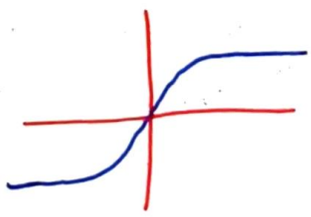
$$\frac{d\,g(z)}{dz} \simeq 0\,(1-0) \simeq 0$$

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

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

2) Tanh activation fn:

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



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

If $z = 10$, $\tanh(z) \simeq 1$

$g'(z) \simeq 0$

If $z = -10$, $\tanh(z) \simeq -1$

$g'(z) \simeq 0$

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

$g'(z) = 1$

3) ReLU

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

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

4) 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 FOR NEURAL NETWORKS

Parameters: $w^{[1]}$ $(n^{[1]}, n^{[0]})$, $b^{[1]}$ $(n^{[1]}, 1)$, $w^{[2]}$ $(n^{[2]}, n^{[1]})$, $b^{[2]}$ $(n^{[2]}, 1)$

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

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

Gradient descent:

Repeat {

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

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

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

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

⋮ ⋮ ⋮

Forward Propogation:

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

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

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

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

Back propogation:

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

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

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

keep dims = True

↓

prevents from ... list ... may like (y

$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[1]}, m)} * g^{[1]'}(z^{[1]}$$

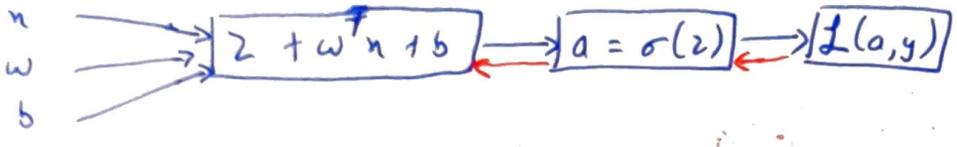
* by element wise product

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

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

Backpropagation Intuition

Logistic regression:



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

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

$$dz = a - y$$

$$\boxed{dz = da \cdot g'(z)} \qquad \begin{cases} \text{where,} \\ g(z) = \sigma(z) \end{cases}$$

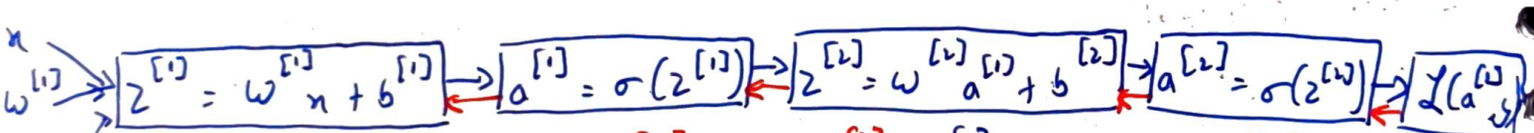
$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{da} \cdot \boxed{\frac{da}{dz}}$$

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

$$dw = dz \cdot x$$

$$db = dz$$

Neural Network gradients:



$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$ $\quad da^{[1]}$ $\quad dz^{[2]} = a^{[2]} - y$ $\quad da^{[2]}$

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

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

dimensions: $w^{[2]} \rightarrow (n^{[2]}, n^{[1]})$

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

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

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

$$dw^{[1]} = d_2^{[1]} \cdot x^T \rightarrow a^{[0]T}$$
$$db^{[1]} = d_2^{[1]}$$

Summary:

$$d_2^{[2]} = a^{[2]} - y$$
$$dw^{[2]} = d_2^{[2]} a^{[1]T}$$
$$db^{[2]} = d_2^{[2]}$$
$$d_2^{[1]} = w^{[2]T} d_2^{[2]} * g^{[1]'}(z^{[1]})$$
$$dw^{[1]} = d_2^{[1]} x^T$$
$$db^{[1]} = d_2^{[1]}$$

Vectorized implementation:

$$z^{[1]} = w^{[1]} x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$
$$z^{[1]} = w^{[1]} X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(z^{[1]})$$

Vectorized equations:

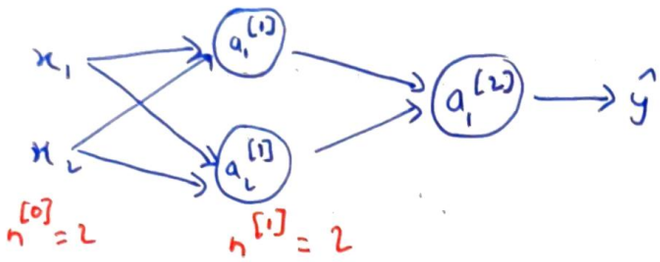
$$d_2^{[2]} = A^{[2]} - Y$$
$$dw^{[2]} = \frac{1}{m} d_2^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} np.sum(d_2^{[2]}, axis=1, keepdims=True)$$
$$d_2^{[1]} = \underbrace{w^{[2]T} d_2^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]'}(z^{[1]})}_{(n^{[1]}, m)} \quad (n^{[1]}, m)$$

→ element wise product

$$dw^{[1]} = \frac{1}{m} d_2^{[1]} x^T$$
$$db^{[1]} = \frac{1}{m} np.sum(d_2^{[1]}, axis=1, keepdims=True)$$

Random Initialization:

What happens if you initialize weights to zeros?



$$w^{[1]}_{2\times 2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$\Rightarrow a_1^{[1]} = a_2^{[1]}$, & when done back propagat°, then

$$d_{2_1}^{[1]} = d_{2_2}^{[1]}$$

$\Rightarrow$ Symmetric neuron (Identical)

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

Then ∵ both wts. are same, d_2 are same, ∴ weights will be updated in the same way $\Rightarrow$ act as single some unit

What should be done?

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

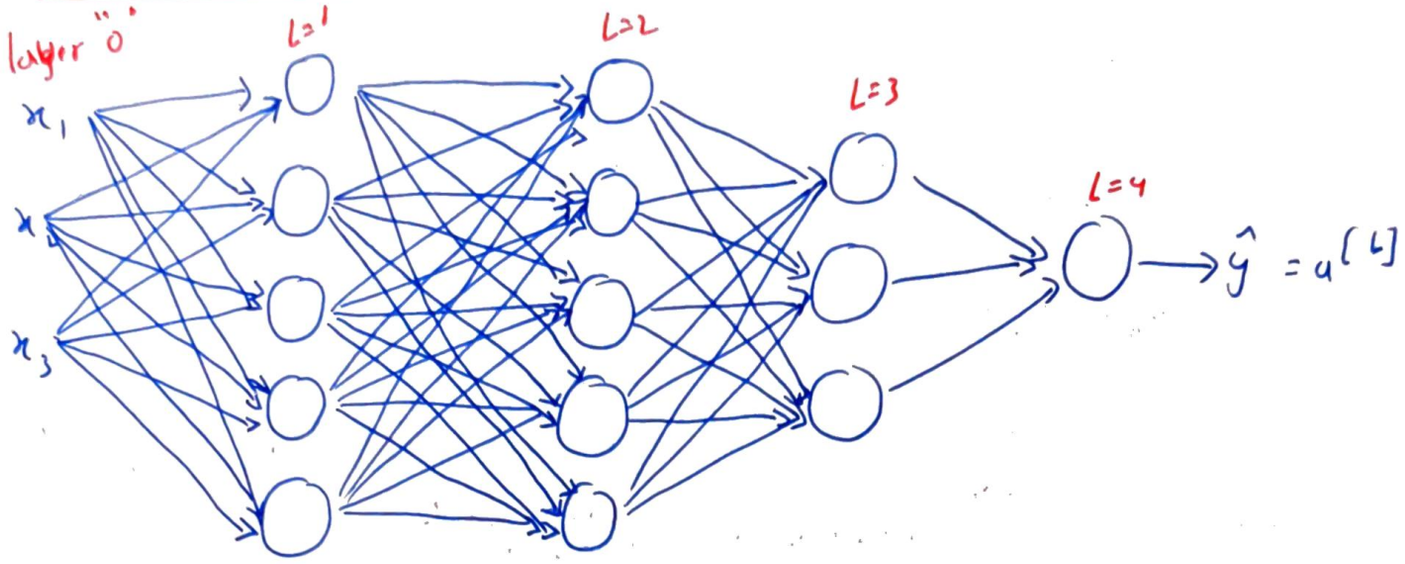
$$b^{[1]} = np.zero((2,1))$$

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

$$b^{[2]} = 0$$

0.01 ∵ should start with very small values & if it is very big, the values after activat° will be either very large or very small making learning slowdown (∵ if sigmoid, on the flatter side of graph)

DEEP NEURAL NETWORK NOTATION:



$X = a^{[0]}$ $\quad L = 4$ (no. of layers)

$n^{[l]}$ = no. of units in layer l

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

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

$b^{[l]}$ = biases for $z^{[l]}$

Forward Propogation in Deep Neural Network:

$X: z^{[1]} = w^{[1]} \underbrace{X}_{a^{[0]}} + b^{[1]}$

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

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

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

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

$z^{[4]} = w^{[4]} a^{[3]} + b^{[4]}$

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

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

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

Vectorized:

$Z^{[1]} = w^{[1]} \underbrace{X}_{A^{[0]}} + b^{[1]}$

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

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

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

$\hat{Y} = g^{[4]}(Z^{[4]}) = A^{[4]}$

for loop

Getting your Matrix dimensions right:

Parameters $w^{[l]}$ & $b^{[l]}$ $\quad L = 5, \; n^{[1]} = 3, \; n^{[2]} = 5, \; n^{[3]} = 4, \; n^{[4]} = 2$

$n^{[5]} = 1$

$$z^{[1]} = w^{[1]} \cdot x + b^{[1]} \qquad n^{[0]} = n_x = 2$$

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

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

$\Rightarrow w^{[1]}$ dimensions: $(n^{[1]}, n^{[0]})$, only $w^{[2]}$: $(5,3)$: $(n^{[2]}, n^{[1]})$

Generally: $w^{[l]}$ dimensions: $(n^{[l]}, n^{[l-1]})$

$$\Rightarrow \quad z^{[1]} = \underbrace{w^{[1]} \cdot x}_{(3,1)} + b^{[1]} \Rightarrow (3,1)$$

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

$\Rightarrow$ Generally $b^{[l]} : (n^{[l]}, 1)$

So when backpropogat^n:

$$dw^{[l]} = w^{[l]} \Rightarrow : (n^{[l]}, n^{[l-1]})$$

$$db^{[l]} = b^{[l]} \Rightarrow : (n^{[l]}, 1)$$

Vectorized Implementation:

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

$$\Rightarrow \quad Z^{[1]} = w^{[1]} \cdot X + b^{[1]}$$

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

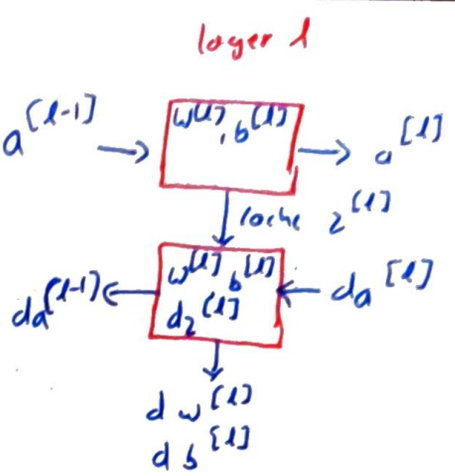
$\therefore$ Generally: $Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

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

& $dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

Forward & backword function:

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



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

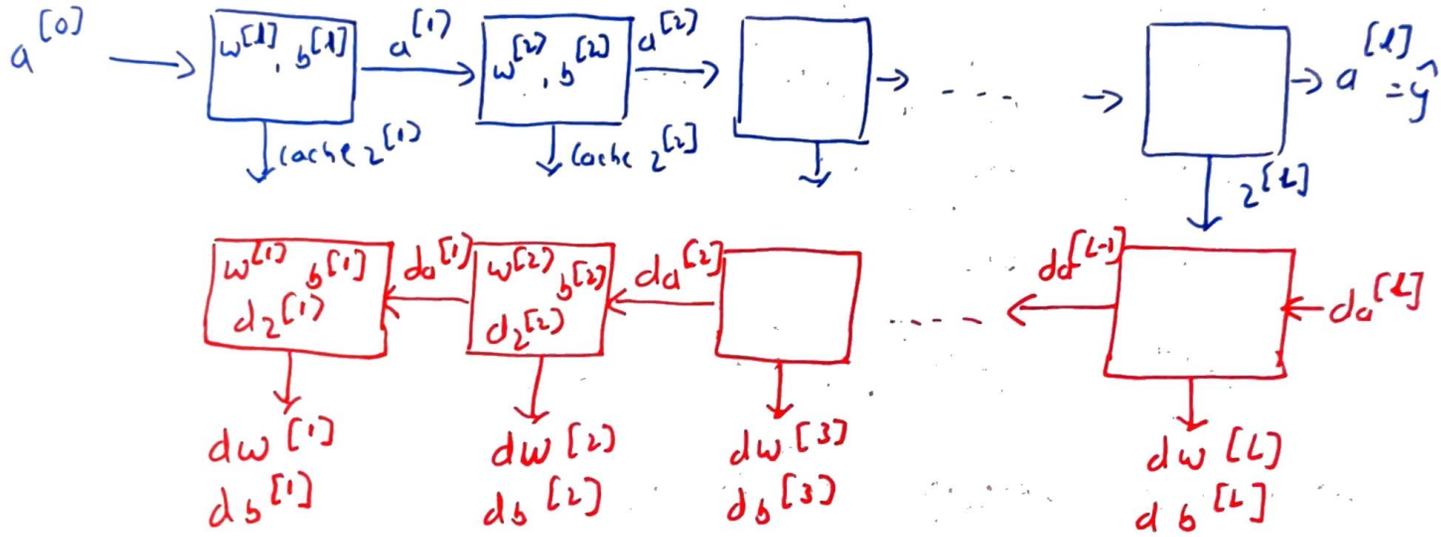
$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ Cache : $z^{[l]}$

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

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

cache $z^{[l]}$ $dw^{[l]}$

$db^{[l]}$



$\Rightarrow$ Update : $w^{[l]} = w^{[l]} - \alpha\, dw^{[l]}$

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

Forward propogat° for layer l:

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

Output : $a^{[l]}$, cache ($z^{[l]}$) → $w^{[l]}, b^{[l]}$

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

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

Backword propogat°:

Input : $da^{[l]}$

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

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]}) \quad -①$$

$$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]}$$

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

$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]} \quad -②$$

Using ① & ② [plugging da in ①]

$$\Rightarrow dz^{[l]} = w^{[l+1]T} dz^{[l+1]} * g^{[l]'}(z^{[l]})$$

Vectorized (Backword):

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

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

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

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

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

↳ for last step

PARAMETERS VS HYPERPARAMETERS

Parameters : $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, w^{[3]}, b^{[3]}, \ldots$

Hyperparameters :
- Learning rate → α
- Iterations
- # no. of hidden layers → L
- no. of hidden units → $n^{[1]}, n^{[2]}, \ldots$
- Choice of activat° function

Momentum, mini-batch size, regularizat°