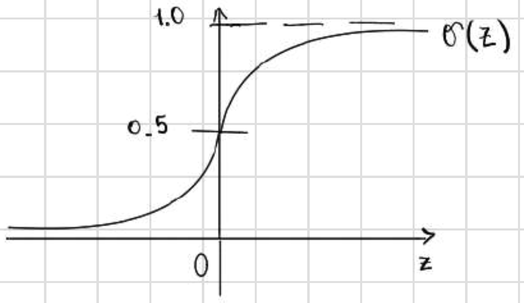
#Neural Networks and Deep Learning

Logistic Regression

- Input: $x \in \mathbb{R}^{nx}$ ("n" dimensional vector) • Parâmeters: $w \in \mathbb{R}^{nx}$, $b \in \mathbb{R}$.
- Output : $\hat{y} = P(y=1 \mid x)$, $0 \leq \hat{y} \leq 1$.

$\hat{y} = w^T x + b$ doesnt make sense for logistic regression because $\hat{y}$ can't be a large number.
So we use the sigmoid function in this case: $\hat{y} = \sigma(w^T x + b) \longrightarrow w^T x + b = z \longrightarrow \hat{y} = \sigma(z)$



$$\sigma(z) = \frac{1}{1+e^{-z}} ; \quad \lim_{z \to \infty} \sigma(z) = 1 ; \quad \lim_{z \to -\infty} \sigma(z) = 0 .$$

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

Loss Function For Logistic Regression (single error)

We dont use $L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$ in L.R. because the optmization will not converge

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

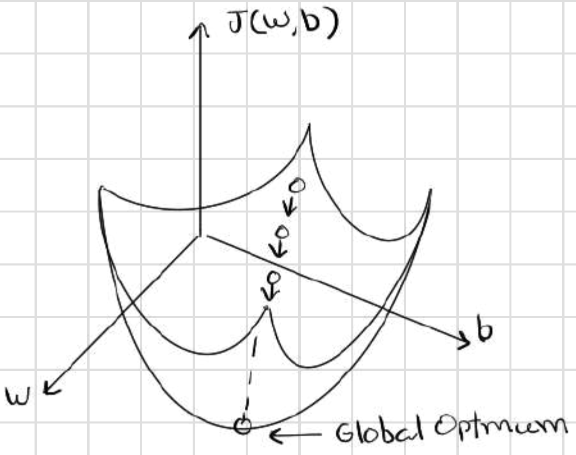
- If $y = 0$: $L(\hat{y}, y) = -\log \hat{y}$ ← want $\log(\hat{y})$ large, so want $\hat{y}$ large
- If $y = 1$: $L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ large, so want $\hat{y}$ small

Cost Function (average error)

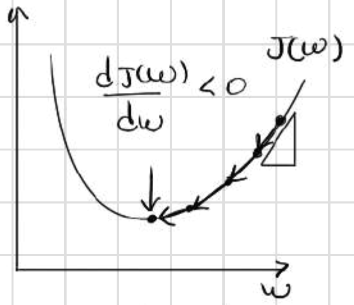
$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \cdot \log \hat{y}^{(i)} + (1-y^{(i)}) \cdot \log(1-\hat{y}^{(i)}) \right]$$

Gradient Descent

Want to find w, b to minimize $J(w,b)$.



Considering only w



Reapeat until converged:

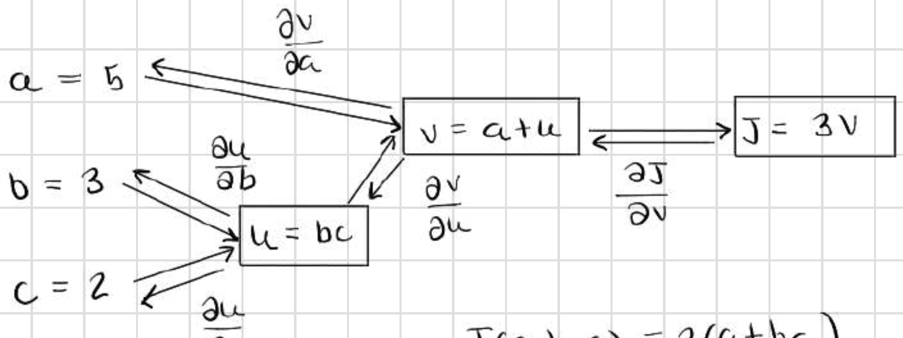
$$\{ w := w - \alpha \frac{dJ(w)}{w} \}$$

Considering multiple paramoters, we use "partial derivative".

$$\left[\begin{array}{l} w := w - \alpha \dfrac{\partial J(w,b)}{\partial w} \\ b := b - \alpha \dfrac{\partial J(w,b)}{\partial b} \end{array} \right]$$

Computation Graph

To find partial derivatives we're going to consider the propagation thrognt the computational graph.



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

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

$$\frac{\partial J}{\partial u} = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial u} = 3 \times 1 = 3$$

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial a} = 3 \times 1 = 3$$

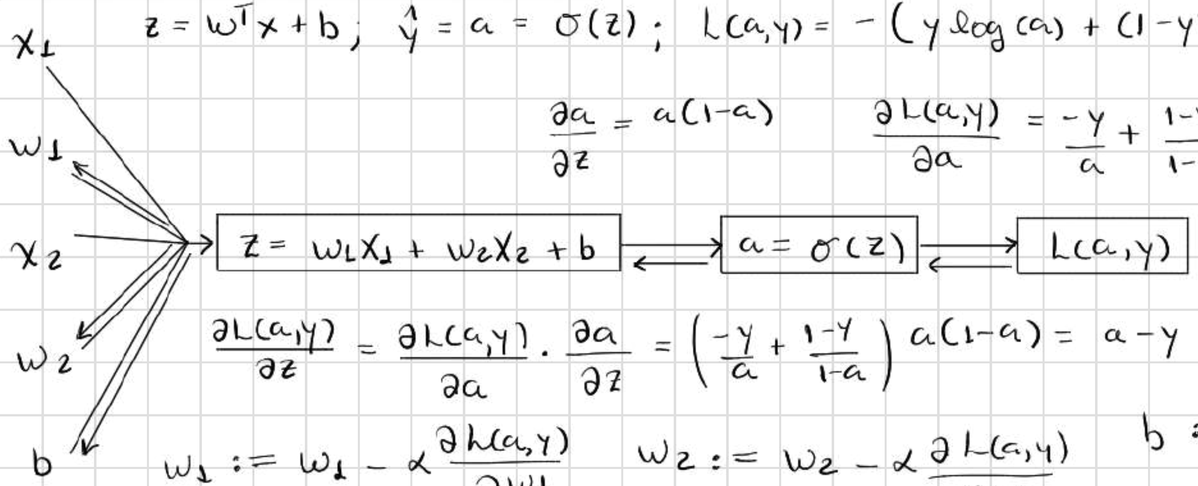
$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial b} = 3 \times c = 6$$

$$\frac{\partial J}{\partial c} = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial u} \cdot \frac{\partial u}{\partial c} = 3 \times b = 9$$

Neural Networks and Deep Learning

Gradient Descent on "m" Exemple

$$z = w^T x + b; \quad \hat{y} = a = \sigma(z); \quad L(a,y) = -\left(y \log(a) + (1-y)\log(1-a)\right)$$



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

$$\frac{\partial L(a,y)}{\partial w_1} = x_1 \cdot \frac{\partial L(a,y)}{\partial z}$$

$$\frac{\partial L(a,y)}{\partial w_2} = x_2 \cdot \frac{\partial L(a,y)}{\partial z}$$

$$\frac{\partial L(a,y)}{\partial b} = \frac{\partial L(a,y)}{\partial z}$$

$$\frac{\partial L(a,y)}{\partial z} = \frac{\partial L(a,y)}{\partial a} \cdot \frac{\partial a}{\partial z} = \left(\frac{-y}{a} + \frac{1-y}{1-a}\right) a(1-a) = a - y$$

$$w_1 := w_1 - \alpha \frac{\partial L(a,y)}{\partial w_1} \qquad w_2 := w_2 - \alpha \frac{\partial L(a,y)}{\partial w_2} \qquad b := b - \alpha \frac{\partial L(a,y)}{\partial b}$$

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

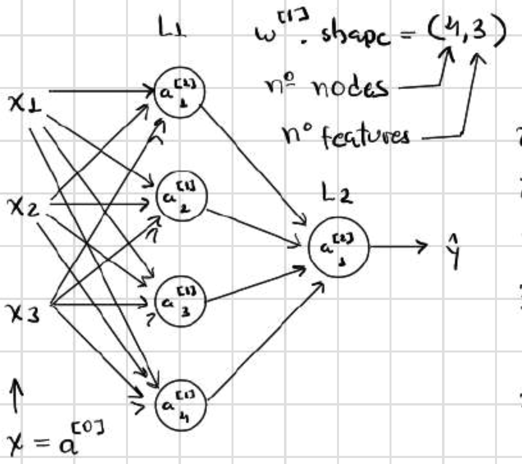
Algorithm :

```
J=0; dw1 = 0; dw2 = 0; db = 0
For i = 1 to m :
    $z^{(i)} = w^T x^{(i)} + b$
    $a^{(i)} = \sigma(z^{(i)})$
    $J += -[y^{(i)} \log(a^{(i)}) + (1-a^{(i)}) \log(1-a^{(i)}))$
    $dz = a^{(i)} - y^{(i)}$
    $dw_1 += x_1^{(i)} \cdot dz^{(i)}$
    $dw_2 += x_2^{(i)} \cdot dz^{(i)}$
    $db += dz^{(i)}$
J = J/m; dw1 = dw1/m; dw2 = dw2/m; db = db/m;
```

Vectorized Implementation

$$Z = w^T X + b$$
$$A = \sigma(Z)$$
$$dZ = A - Y$$
$$dw = \frac{1}{m} X dZ^T$$
$$db = \frac{1}{m} dZ$$
$$w := w - \alpha dw$$
$$b := b - \alpha db$$

Neural Network



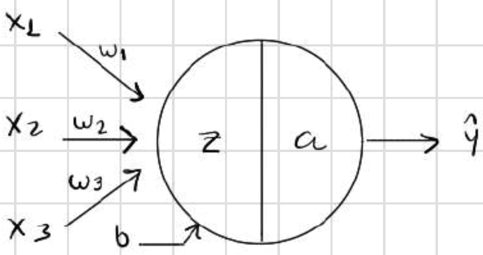
$w^{[1]}$. shape = (4,3)

nº nodes

nº features

$$z_1^{[1]} = w_1^{[1]T} X + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} X + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} X + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} X + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

$$z_1^{[2]} = w_1^{[2]T} X + b_1^{[2]}, \quad a_1^{[2]} = \sigma(z_1^{[2]})$$



$$\boxed{z = w^T . x + b} \; \boxed{a = \sigma(z)}$$

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

$$z^{[2]} = \begin{bmatrix} w_1^{[2]T} \end{bmatrix} \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[2]} \end{bmatrix} = \begin{bmatrix} w_1^{[2]} \cdot a^{[1]} + b_1^{[2]} \end{bmatrix} = \begin{bmatrix} z_1^{[2]} \end{bmatrix}$$

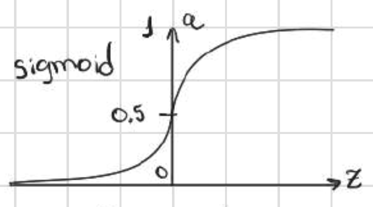
vectorizing across multiple exemples

```
for i = 1 to m:
    $z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$
    $a^{[1](i)} = \sigma(z^{[1](i)})$
    $z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$
    $a^{[2](i)} = \sigma(z^{[2](i)})$
```

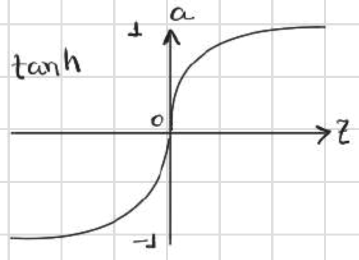
$\downarrow$

$$Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

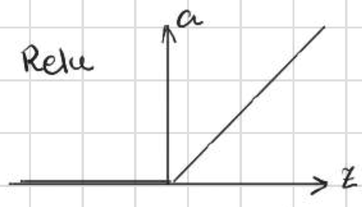
Activation Functions



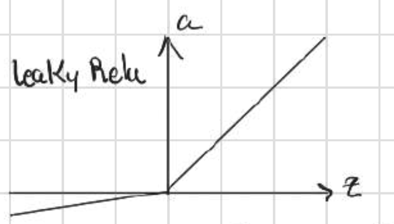
$$a = \left(\frac{1}{1+e^{-z}}\right)$$



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



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



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

Neural Networks and Deep Learning

Gradient Descent For Neural Networks

In this case, $g(Z)$ represents a generic activation function.

Forward propagation

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}X + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$

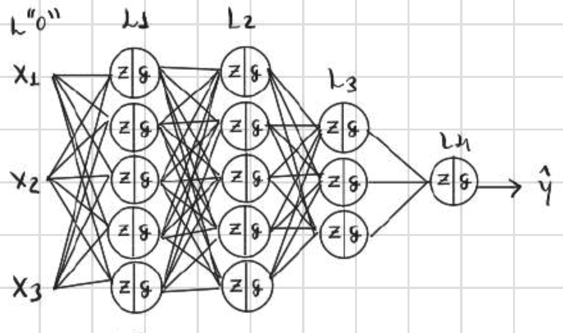
Back propagation

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

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$
$$db^{[2]} = \frac{1}{m} \cdot \text{np.sum}(dZ^{[2]}, \text{axis}=1, \text{Keepdims}=\text{True})$$
$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} \cdot \text{np.sum}(dZ^{[1]}, \text{axis}=1, \text{Keepdims}=\text{True})$$

$*$ → element wise product

Deep Learning Notation



$X = a^{[0]}$

L = number of layers

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

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

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

$W^{[l]}$ $b^{[l]}$ = weights for $Z^{[l]}$

Forward propagation

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

Vectorized vectorization

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

Parameters Dimensions

$Z^{[l]} \rightarrow (n^{[l]},1)$

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

$A^{[l]} \rightarrow (n^{[l]},1)$

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

$dW^{[l]} \rightarrow (n^{[l]},n^{[l-1]})$

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

$$\begin{tikzcd} Z^{[1]} \arrow[d] & = W^{[1]} \arrow[d] & \cdot X \arrow[d] & + b^{[1]} \arrow[r] & (n^{[1]},1) \arrow[d, bend left, "\text{python broadcast}"] \\ (n^{[1]},m) & (n^{[1]},n^{[0]}) & (n^{[0]},m) & & (n^{[1]},m) \end{tikzcd}$$

Notes that:

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

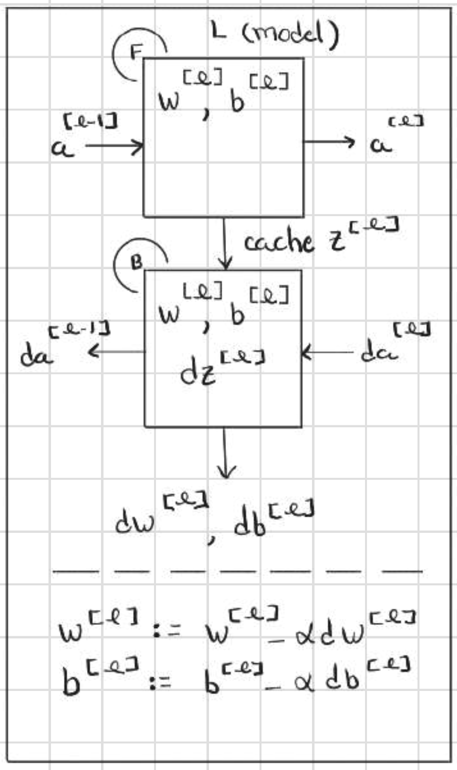
$Z^{[l]}, A^{[l]} \rightarrow (n^{[l]},m)$

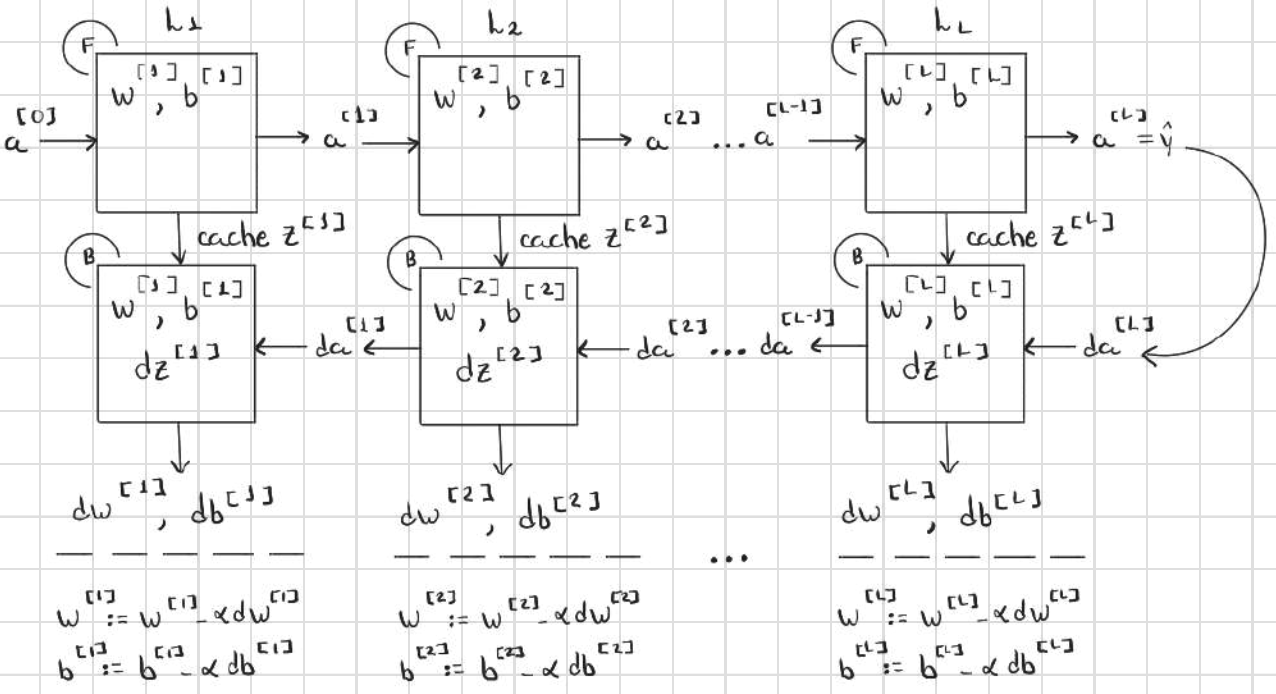
for $l=0$, $A^{[0]} = X \rightarrow (n^{[0]},m)$

$dZ^{[l]}, dA^{[l]} \rightarrow (n^{[l]},m)$

Forward and Backward function blocks

Forward: Input $a^{[l-1]}$, output $a^{[l]}$, cache $z^{[l]}$. Backward: Input $da^{[l]}$, output $da^{[l-1]}$, $dw^{[l]}$, $db^{[l]}$.





ANNs and the human neuron