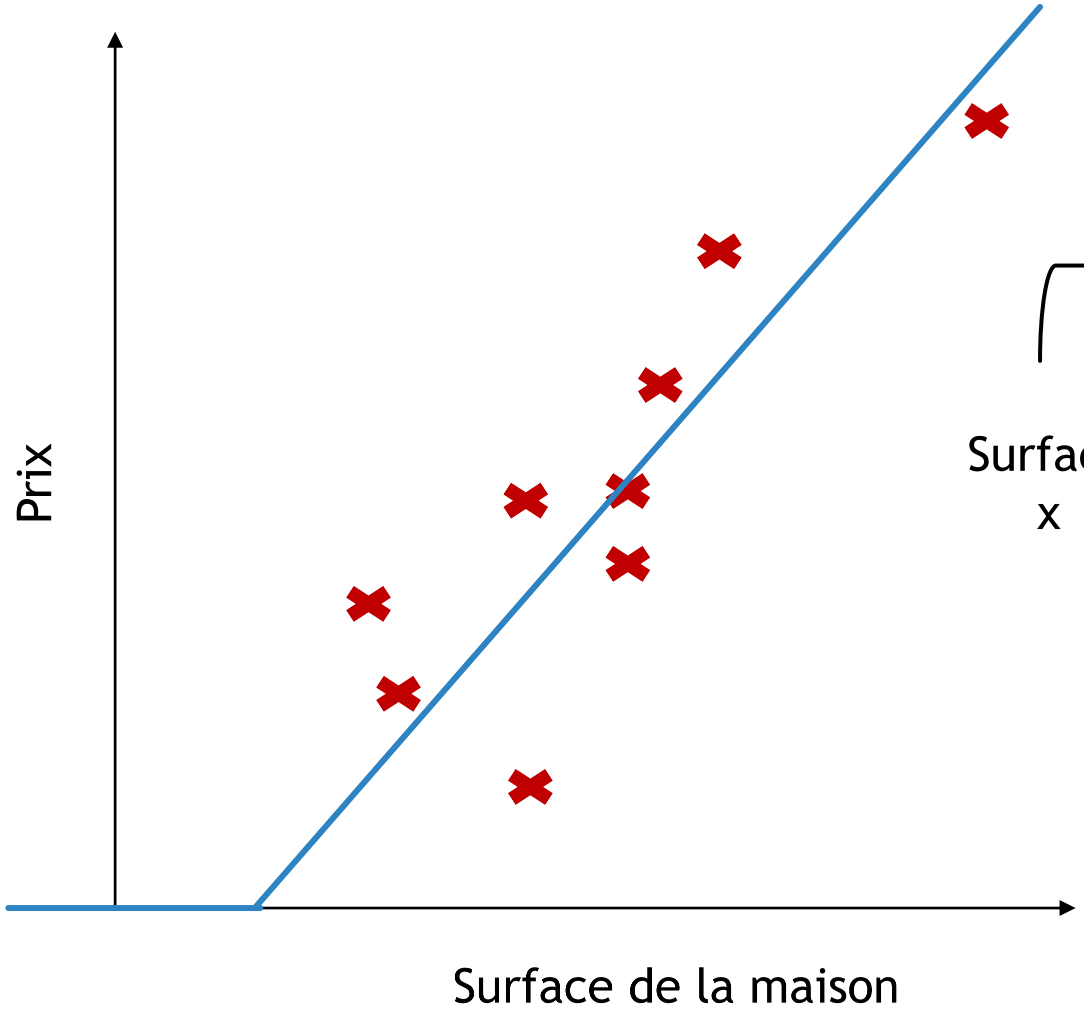
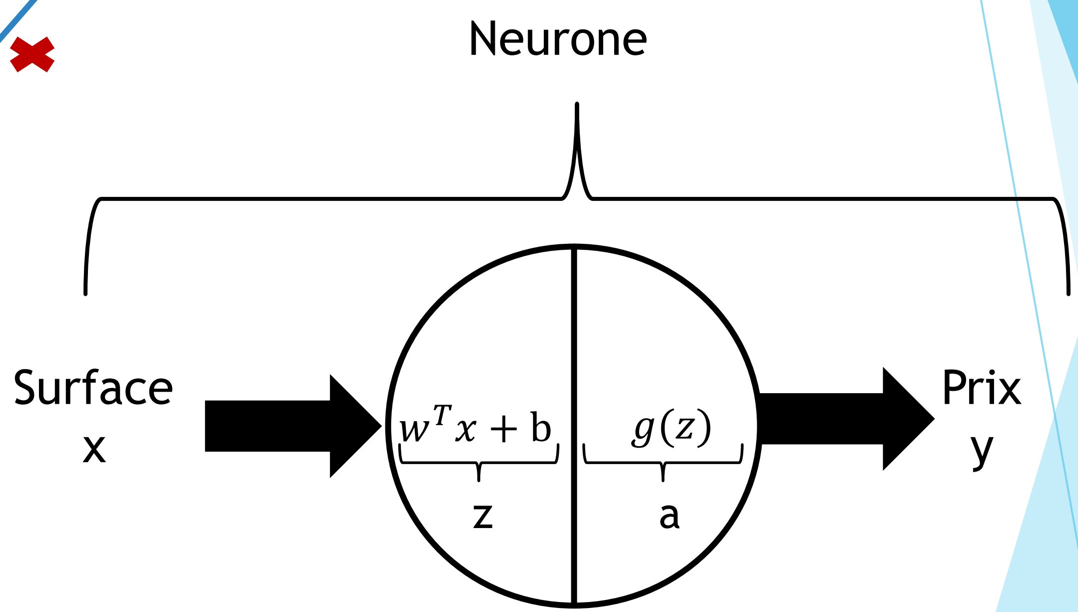
Neural Network

Deep learning par la pratique

Qu'est-ce qu'un réseau neuronal ?

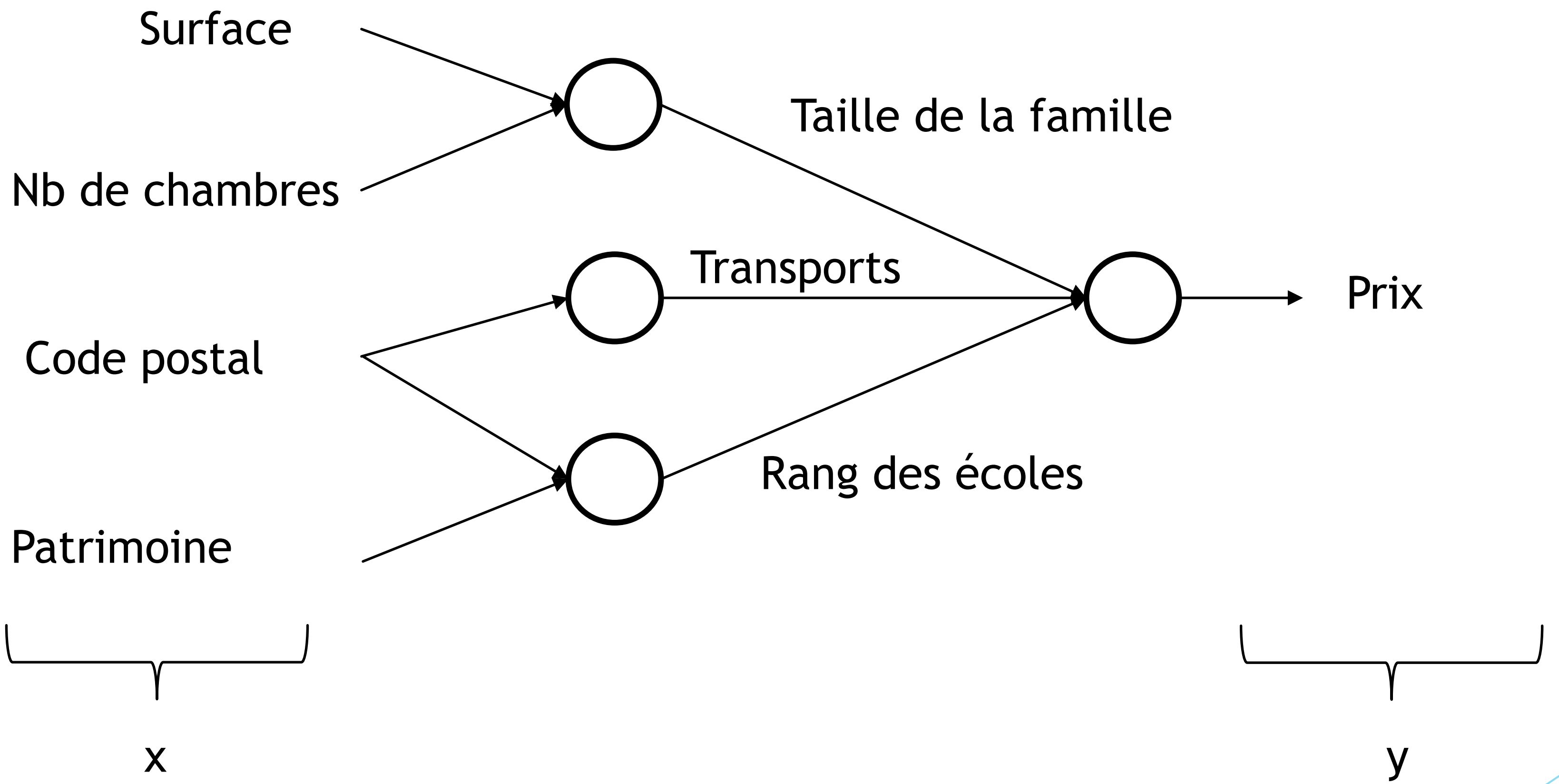




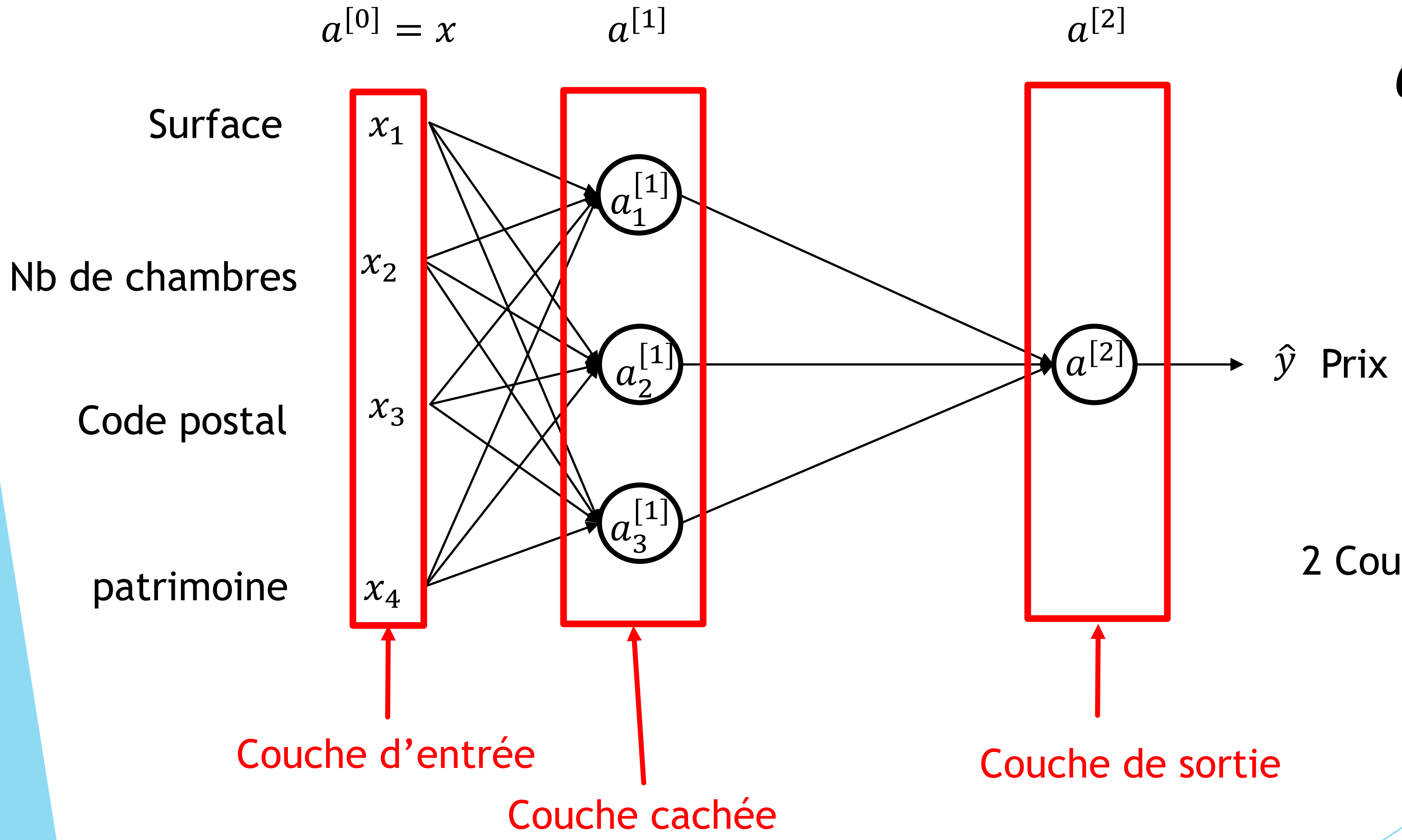
Un réseau neuronal est construit
en connectant de nombreux neurones

Fonction d’activation : $\mathrm{g}(z) = \max(0, z)$

Votre premier réseau neuronal



Représentation des réseaux neuronaux

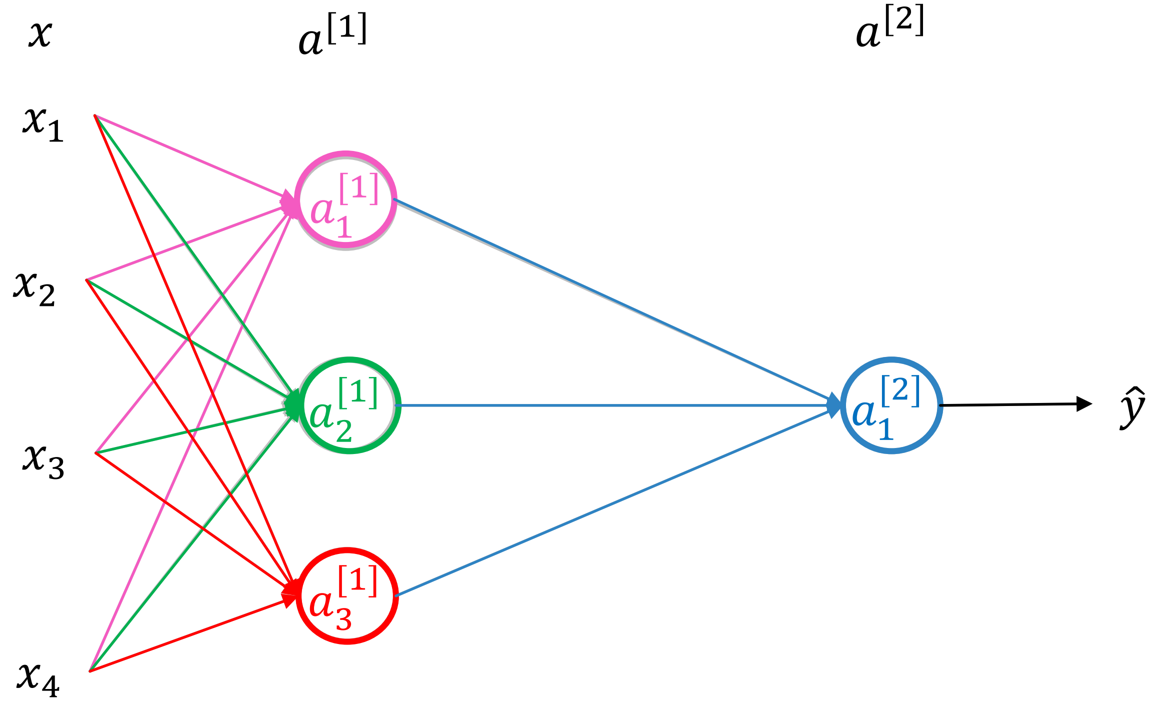


$a_1^{[1]}$ ← Couche

Neurone de la couche

2 Couches de neurones

Représentation des réseaux neuronaux



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

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

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

$$z_1^{[2]} = w_1^{[2]^T} a^{[1]} + b_1^{[2]} ; \; a_1^{[2]} = g(z_1^{[2]})$$

Vectorisation

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]^T} x + b_1^{[1]} \\ w_2^{[1]^T} x + b_2^{[1]} \\ w_3^{[1]^T} x + b_3^{[1]} \end{bmatrix} ; \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} = g\left(\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix}\right)$$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[1]} & w_{1,2}^{[1]} & w_{1,3}^{[1]} & w_{1,4}^{[1]} \\ w_{2,1}^{[1]} & w_{2,2}^{[1]} & w_{2,3}^{[1]} & w_{2,4}^{[1]} \\ w_{3,1}^{[1]} & w_{3,2}^{[1]} & w_{3,3}^{[1]} & w_{3,4}^{[1]} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} \quad ; \quad \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} = g\left(\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{bmatrix}\right)$$

$$\underset{(3,1)}{z^{[1]}} = \underset{(3,4)}{W^{[1]^T}} \underset{(4,1)}{x} + \underset{(3,1)}{b^{[1]}} \quad ; \quad a^{[1]} = g(z^{[1]})$$

$$\underset{(1,1)}{z^{[2]}} = \underset{(1,3)}{w^{[2]^T}} \underset{(3,1)}{a^{[1]}} + \underset{(1,1)}{b^{[2]}} \quad ; \quad a^{[2]} = g(z^{[2]})$$

Vectorisation sur plusieurs exemples

~~for i = 0 to m :~~

$$\overset{(3,1)}{a^{[1](i)}} = g(\overset{(3,4)}{w^{[1]^T}} \overset{(4,1)}{x^{(i)}} + \overset{(3,1)}{b})$$

$$\overset{(1,1)}{\hat{y}^{(i)}} = a^{[2](i)} = g(\overset{(1,3)}{w^{[2]^T}} \overset{(3,1)}{a^{[1](i)}} + \overset{(1,1)}{b})$$

$$\overset{(3,m)}{a^{[1]}} = g(\overset{(3,4)}{w^{[1]^T}} \overset{(4,m)}{X} + \overset{(3,1)}{b})$$

$$\overset{(1,m)}{\hat{y}} = a^{[2]} = g(\overset{(1,3)}{w^{[2]^T}} \overset{(3,m)}{a^{[1]}} + \overset{(1,1)}{b})$$

$x \in \mathbb{R}^{n_x}$ $\quad x^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

$\hat{y} \in \mathbb{R}$

$X \in \mathbb{R}^{(n_x, m)}$ $\quad X = \begin{bmatrix} {x_1}^{(1)} & {x_1}^{(2)} & {x_1}^{(3)} & \dots & {x_1}^{(m)} \\ {x_2}^{(1)} & {x_2}^{(2)} & {x_2}^{(3)} & \dots & {x_2}^{(m)} \\ {x_3}^{(1)} & {x_3}^{(2)} & {x_3}^{(3)} & \dots & {x_3}^{(m)} \\ {x_4}^{(1)} & {x_4}^{(2)} & {x_4}^{(3)} & \dots & {x_4}^{(m)} \end{bmatrix}$

$\hat{y} \in \mathbb{R}^m$

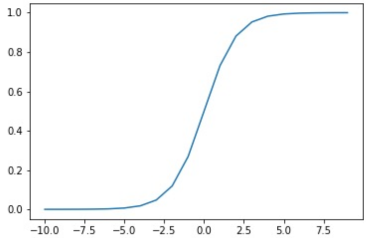
Fonction d'activation

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

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

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

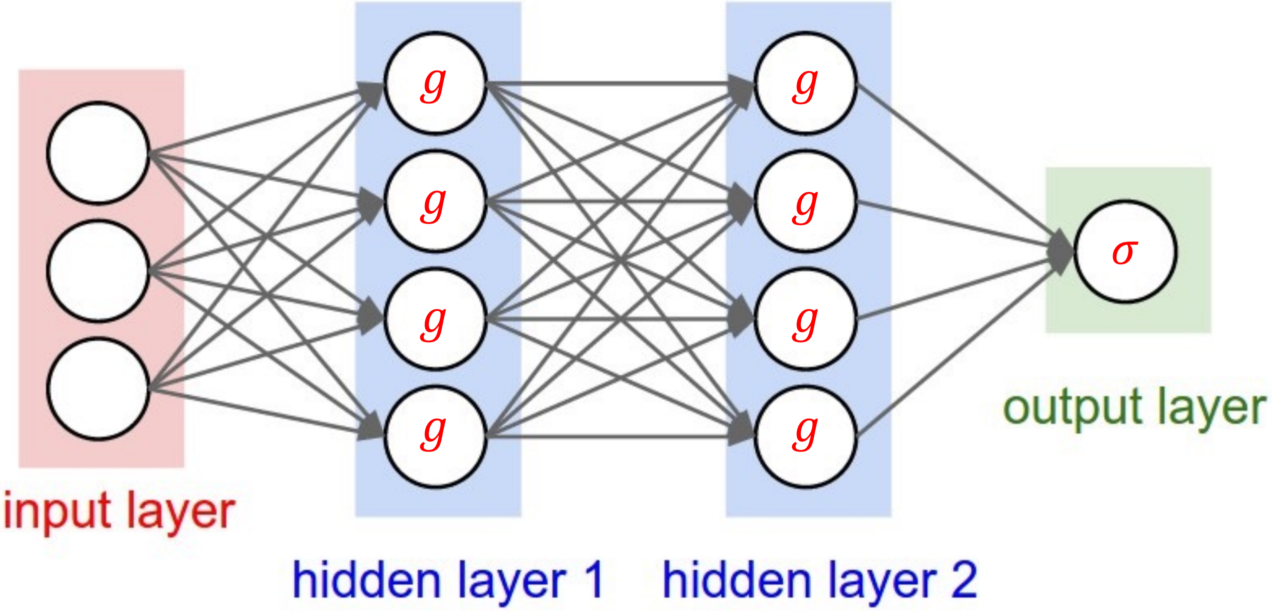
$$a^{[2]} = \cancel{\sigma(z^{[2]})} \quad g(z^{[2]})$$



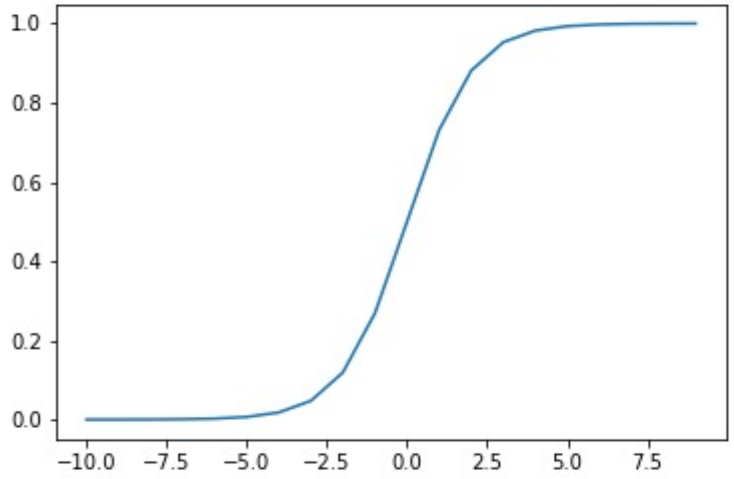
Saturation de la sigmoïde pour les grandes valeurs

Ne pas utiliser de Sigmoïd dans l'activation dans un réseau de neurones

Dans le cas d'une classification binaire utilisation de la Sigmoïd pour la couche de sortie



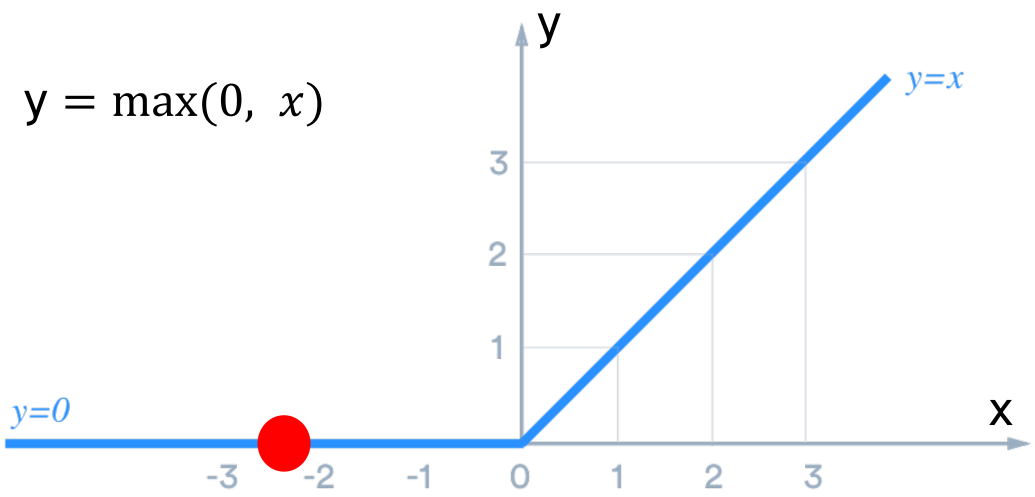
Sigmoïd



- La sigmoïde peut saturer et conduire à des gradients évanescents.
- Non centré sur le zéro
- e^X est coûteux en termes de calcul.

Rectified Linear Unit (ReLU)

$$y = \max(0, \ x)$$



Avantages :

ReLU ne sature pas pour les valeurs positives

ReLU est assez rapide à calculer

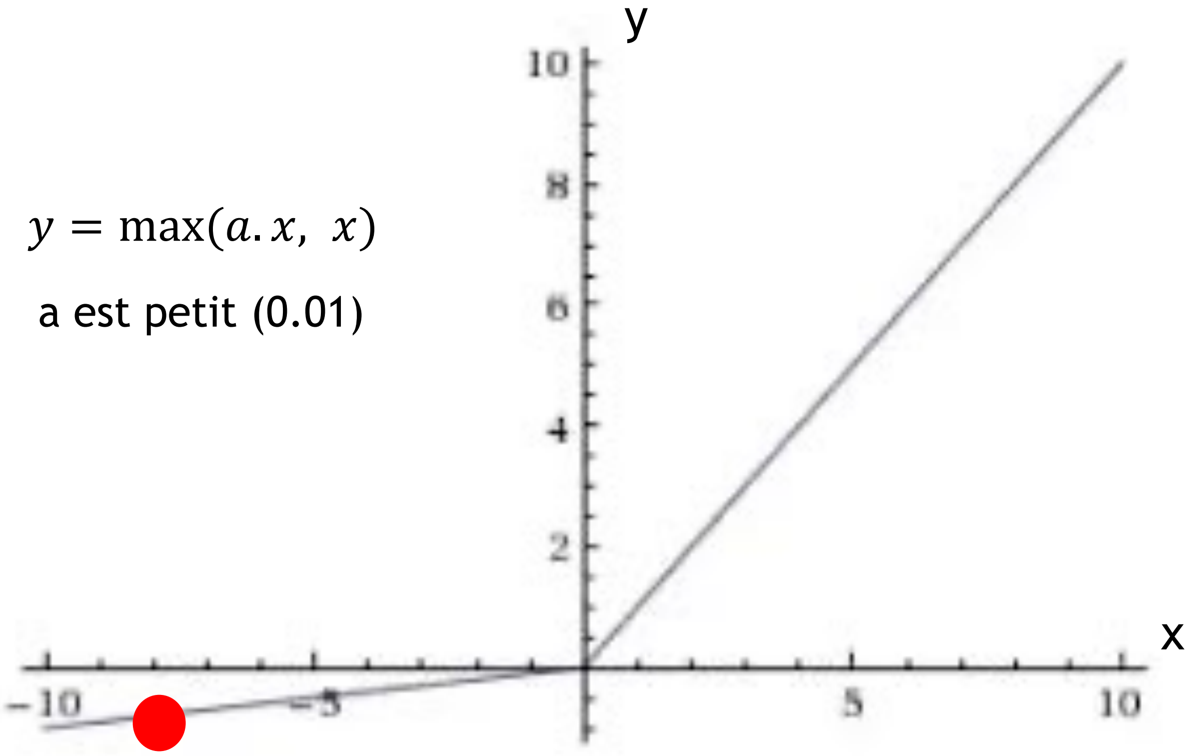
Inconvénients :

ReLU souffre d'un problème connu sous le nom de "dying ReLU".

Leaky ReLU

$$y = \max(a.x,\ x)$$

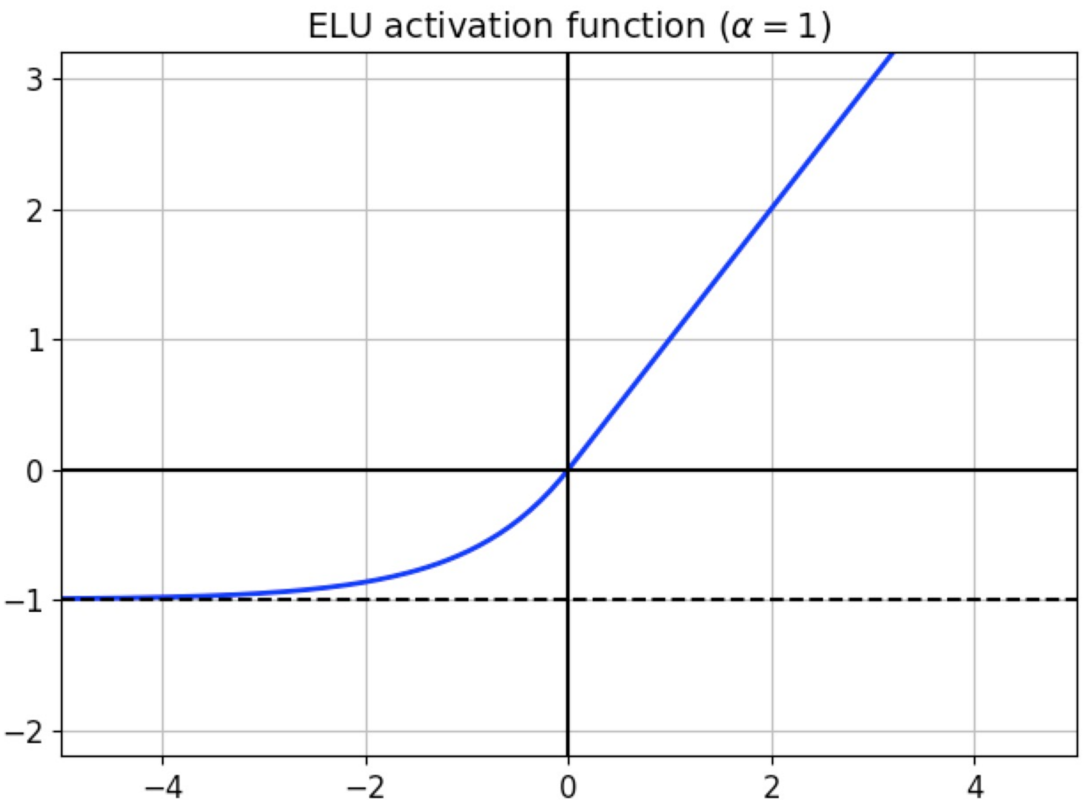
a est petit (0.01)



Variantes :

- Parametric leaky Relu (PReLU), si vous disposez de beaucoup de données.
- Randomized leaky Relu (RReLU), si votre réseau neuronal est surentraîné.

Exponential Linear Unit (ELU)



$$ELU_\alpha(z) = \begin{cases} \alpha(\exp(z) - 1) & if\ z < 0 \\ z & if\ z \geq 0 \end{cases}$$

Avantages :

- Il prend des valeurs négatives, ce qui permet au neurone d'avoir une moyenne plus proche de 0

- Il a un gradient non nul pour z < 0, ce qui évite le problème des ‘dying ReLU'.

Inconvénients

- Il est plus lent à calculer que ReLU.

Fonctions d’activations conseils

- En général pour la performance :

ELU > Leaky ReLU > ReLU

- Si vous vous souciez du temps d'exécution :

Leaky ReLU > ELU

- Par défaut :

ReLU

Descente de gradient pour les réseaux de neurones

Paramètres : $\overset{(n^{[1]},n^{[0]})}{w^{[1]}}$, $\overset{(n^{[1]},1)}{b^{[1]}}$, $\overset{(n^{[2]},n^{[1]})}{w^{[2]}}$, $\overset{(n^{[2]},1)}{b^{[2]}}$ $\quad n_x = n^{[0]}$, $n^{[1]}$, $n^{[2]} = 1$

Fonction de coût : $J(w^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{n} \mathcal{L}(\overset{a^{[2]}}{\hat{y}}, y)$

Descente de gradient :

Répéter {

Calculer la prédiction $(\hat{y}^{(i)}, i = (1, \ldots, m))$

$\boxed{dw^{[1]}} = \frac{dJ}{dw^{[1]}}, \boxed{db^{[1]}} = \frac{dJ}{db^{[1]}}, \boxed{dw^{[2]}} = \frac{dJ}{dw^{[2]}}, \boxed{db^{[2]}} = \frac{dJ}{db^{[2]}}$

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

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

$w^{[2]} := w^{[2]} - \alpha\, dw^{[2]}$

$b^{[2]} := b^{[2]} - \alpha\, db^{[2]}$}

Forward propagation

$$x,\ w^{[1]},\ b^{[1]} \rightarrow \boxed{z^{[1]} = w^{[1]^T} X + b^{[1]}} \rightarrow \boxed{a^{[1]} = g(z^{[1]})} \xrightarrow{w^{[2]},\ b^{[2]}} \boxed{z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}} \rightarrow \boxed{a^{[2]} = g(z^{[2]})} \rightarrow \boxed{\mathcal{L}(a^{[2]}, y)}$$

Back propagation

$w^{[2]}$

$b^{[2]}$

x

$w^{[1]}$ $\boxed{z^{[1]} = w^{[1]^T} X + b^{[1]}}$ $\boxed{a^{[1]} = g(z^{[1]})}$ $\boxed{z^{[2]} = w^{[2]^T} a^{[1]} + b^{[2]}}$ $\boxed{a^{[2]} = g(z^{[2]})}$ $\boxed{\mathcal{L}(a^{[2]}, y)}$

$b^{[1]}$

$\mathrm{d}z^{[1]} = W^{[2]} dz^{[2]} * g^{[1]'}(z^{[1]})$

$\mathrm{d}w^{[1]} = dz^{[1]} . x^T$

$a_0{}^T$

$\mathrm{d}b^{[1]} = \mathrm{d}z^{[1]}$

$\mathrm{d}a^{[1]} = \cdots$

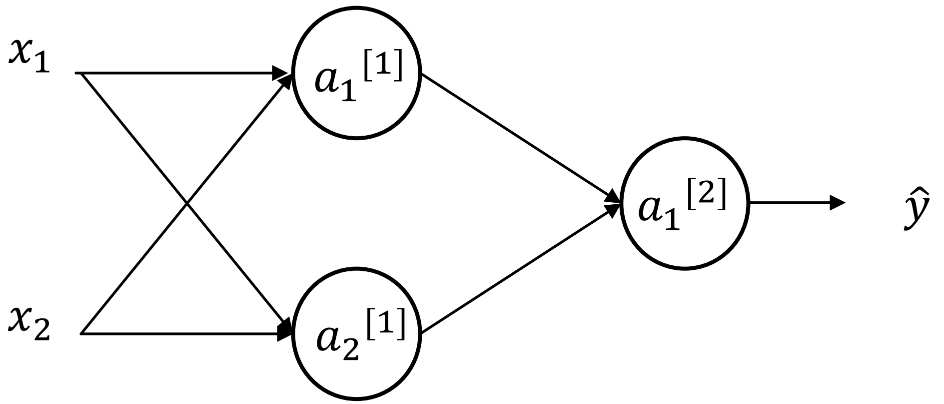
$$\mathrm{d}z^{[2]} = da \,.\, g'(z) = a - y$$

$$\mathrm{d}w^{[2]} = dz^{[2]} a^{[1]^T}$$

$$\mathrm{d}b^{[2]} = dz^{[2]}$$

$$\mathrm{da} = \mathcal{L}(a^{[2]}, y) = -y\, log(a) - (1-y)\log(1-a) = -\frac{y}{a} + \frac{1-y}{1-a}$$

Initialiser les poids à zéro ?

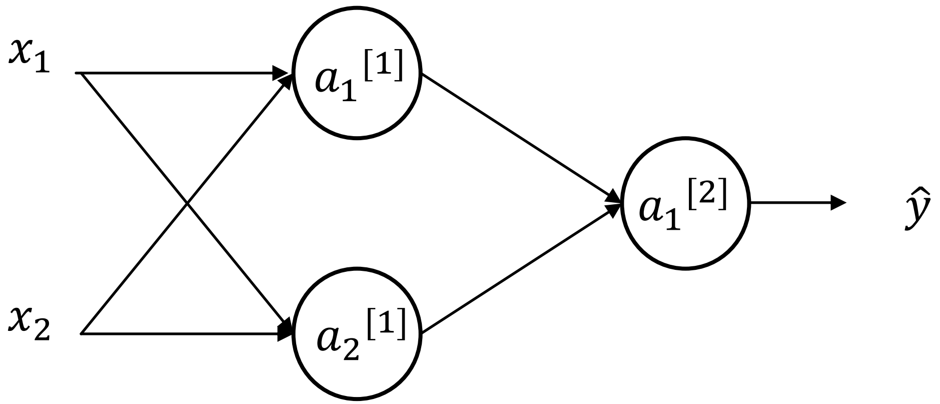


$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$w^{[1]} = \begin{bmatrix} p & l \\ p & l \end{bmatrix}$$

$$a_1^{[1]} = a_2^{[1]} \longrightarrow dz_1^{[1]} = dz_2^{[1]} \longrightarrow dw^{[1]} = \begin{bmatrix} u & v \\ u & v \end{bmatrix} \longrightarrow w^{[1]} := w^{[1]} - \alpha dw$$

Initialiser le poids de façon aléatoire



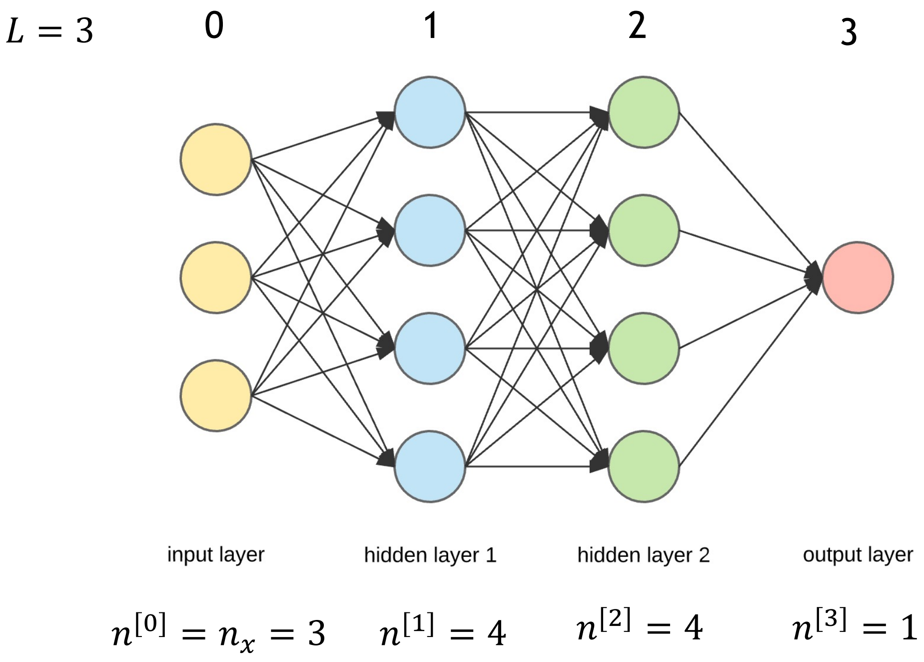
$$w^{[1]} = np.random.rand((2{,}2))\,.\overbrace{0.01}^{\text{Petit nombre}}$$

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

$$w^{[1]} = np.random.rand((1{,}2))\,.\overbrace{0.01}^{\text{Petit nombre}}$$

$$b^{[1]} = 0$$

Bien choisir les dimensions de la matrice



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

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

x

$$\underset{(n^{[1]},m)}{Z^{[1]}} = \underset{(n^{[1]},n^{[0]})}{w^{[1]^T}} \; \underset{(n^{[0]},m)}{A^{[0]}} + \underset{(n^{[1]},1)}{b^{[1]}}$$

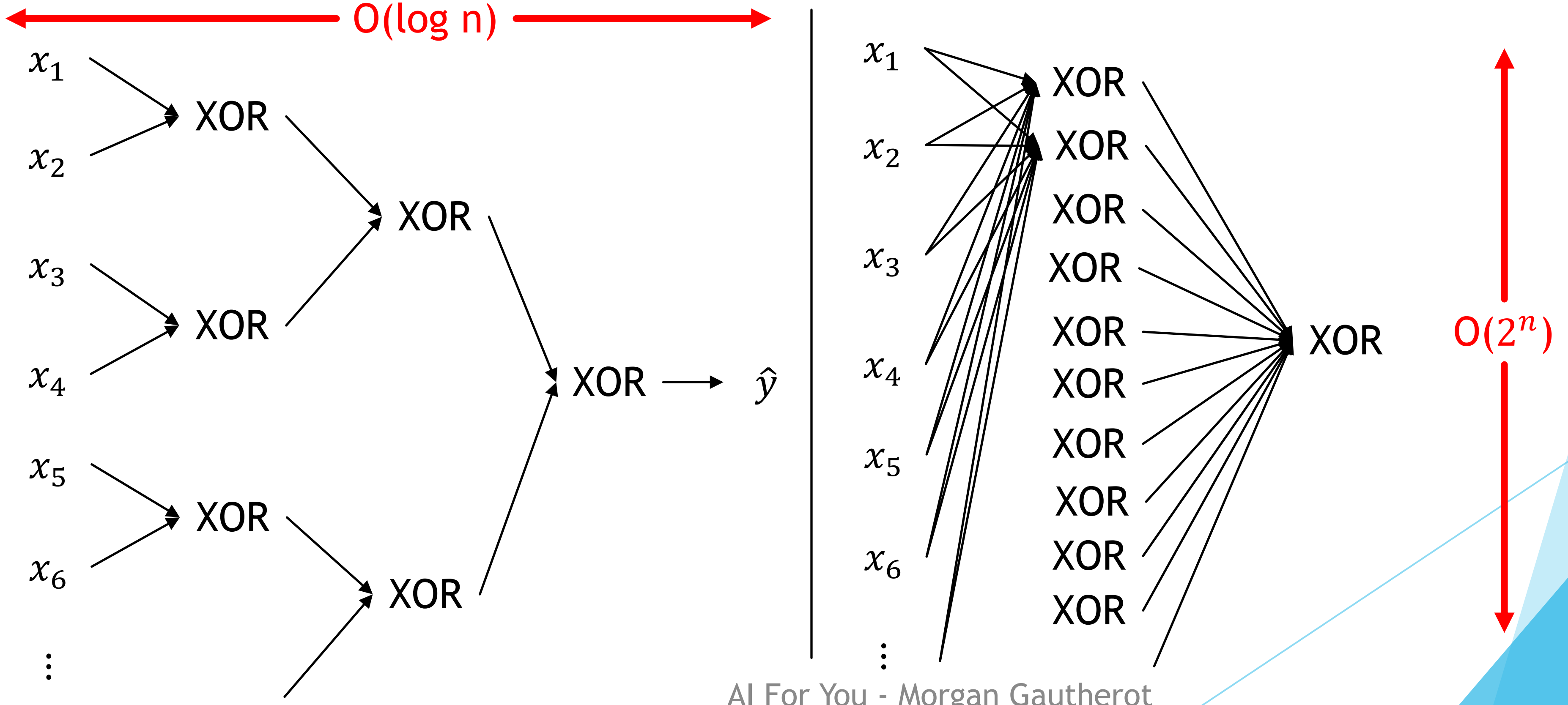
$$\underset{(n^{[2]},m)}{Z^{[2]}} = \underset{(n^{[2]},n^{[1]})}{w^{[2]^T}} \; \underset{(n^{[1]},m)}{A^{[1]}} + \underset{(n^{[2]},1)}{b^{[2]}}$$

$$\underset{(n^{[3]},m)}{Z^{[3]}} = \underset{(n^{[3]},n^{[2]})}{w^{[3]^T}} \; \underset{(n^{[2]},m)}{A^{[2]}} + \underset{(n^{[3]},1)}{b^{[3]}}$$

Pourquoi des représentations profondes ?

Pour approximer une fonction, une architecture de réseau neuronal plus profonde nécessite moins de paramètres pour être entraînée qu'une architecture moins profonde.

$$y = x_1 XOR\ x_2\ XOR\ x_3\ XOR\ x_4\ XOR\ x_5\ XOR\ x_6\ XOR\ \ldots$$



Résumé de l'apprentissage profond