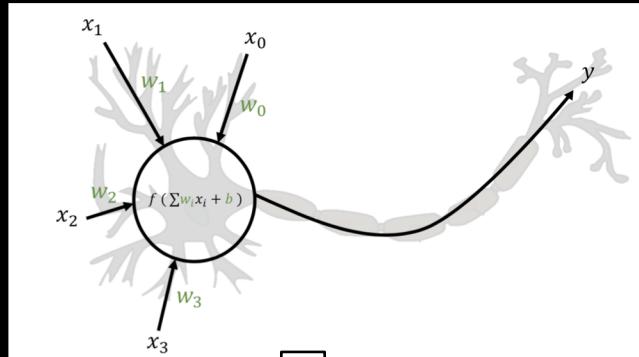


# Introduction à l'apprentissage profond ( Deep Learning)

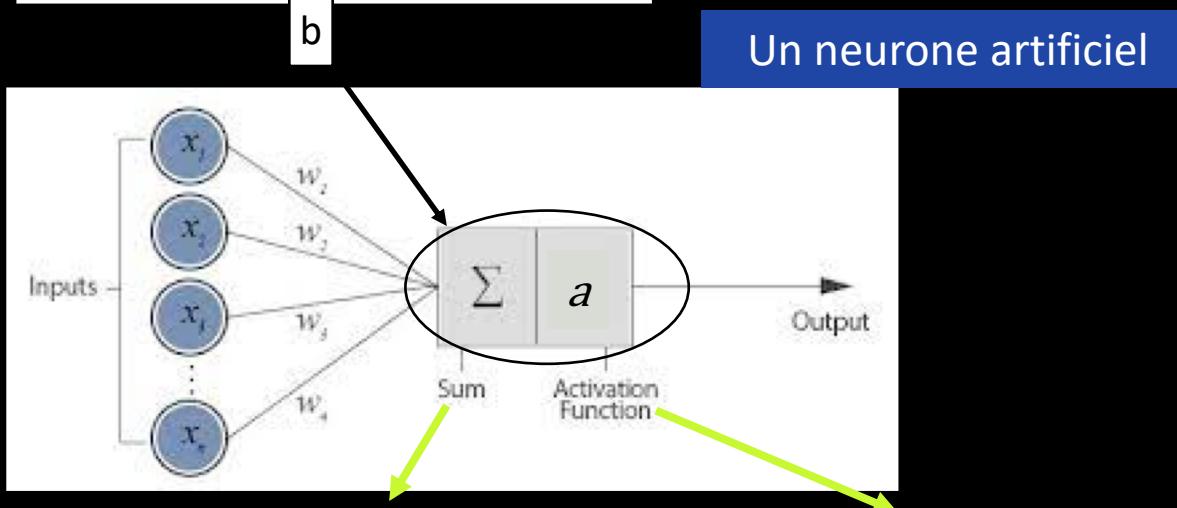
# Aapprentissage profond (Deep learning) : Définition

- Apprentissage profond (Deep learning) :
  - Apprentissage basé sur des réseaux de neurones
  - Objectif apprendre des représentations abstraites des données
- Au cœur de cet apprentissage : → Les Réseaux de Neurones

# Un neurone artificiel → le perceptron



Un neurone reçoit des signaux à travers ses synapses, puis calcule son propre signal qu'il transmet aux autres neurones (ses voisins)



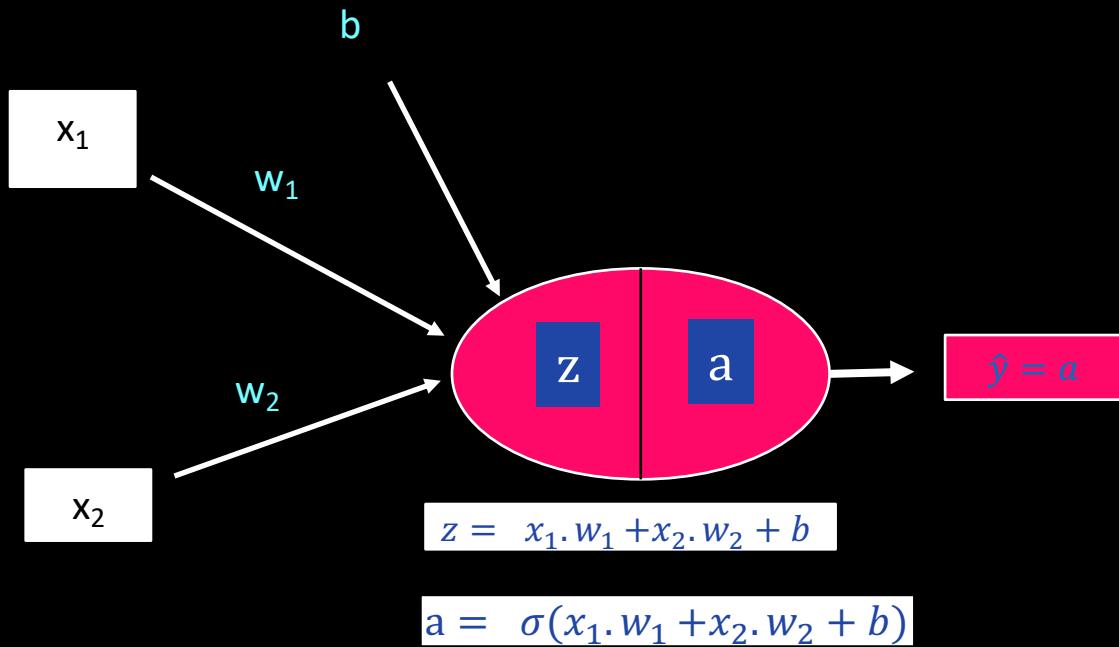
$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + \cdots x_n \cdot w_n + b$$

$$a = \sigma(z)$$

$\sigma$  Une sigmoïde, Relu, identité ...

# Un neurone artificiel → le perceptron

| x <sub>1</sub> : Surface (m <sup>2</sup> ) | x <sub>2</sub> : Nb Chambres | y: prix (1000€) |
|--|------------------------------|-----------------|
| 523  | 1                            | 100             |
| 645  | 1                            | 150             |
| 708  | 2                            | 200             |
| 1034                                       | 3                            | 300             |
| 2290                                       | 4                            | 350             |
| 2545                                       | 5                            | 440             |
| 2410                                       | 4                            | 380             |
| 540  | 2                            | 220             |
| 2610                                       | 4                            | 680             |

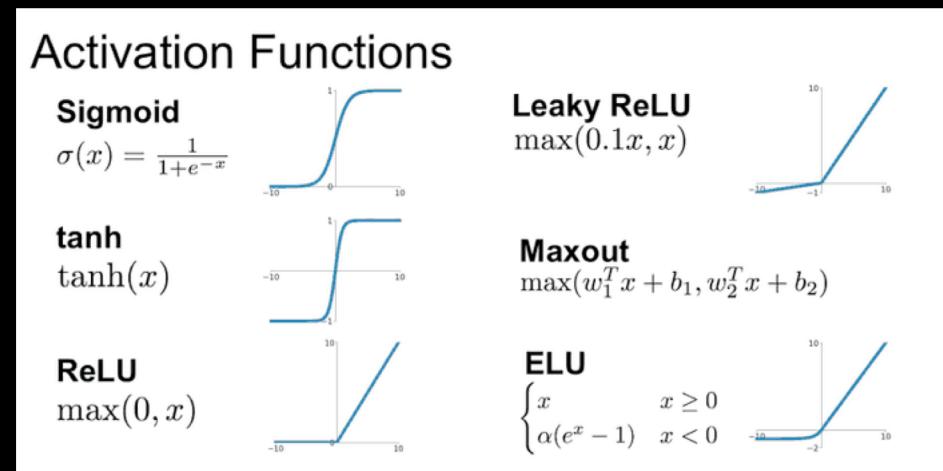


Les paramètres du modèle(du neurone):  $w_1$ ,  $w_2$ , et  $b$

# Un neurone artificiel → fonction d'activation

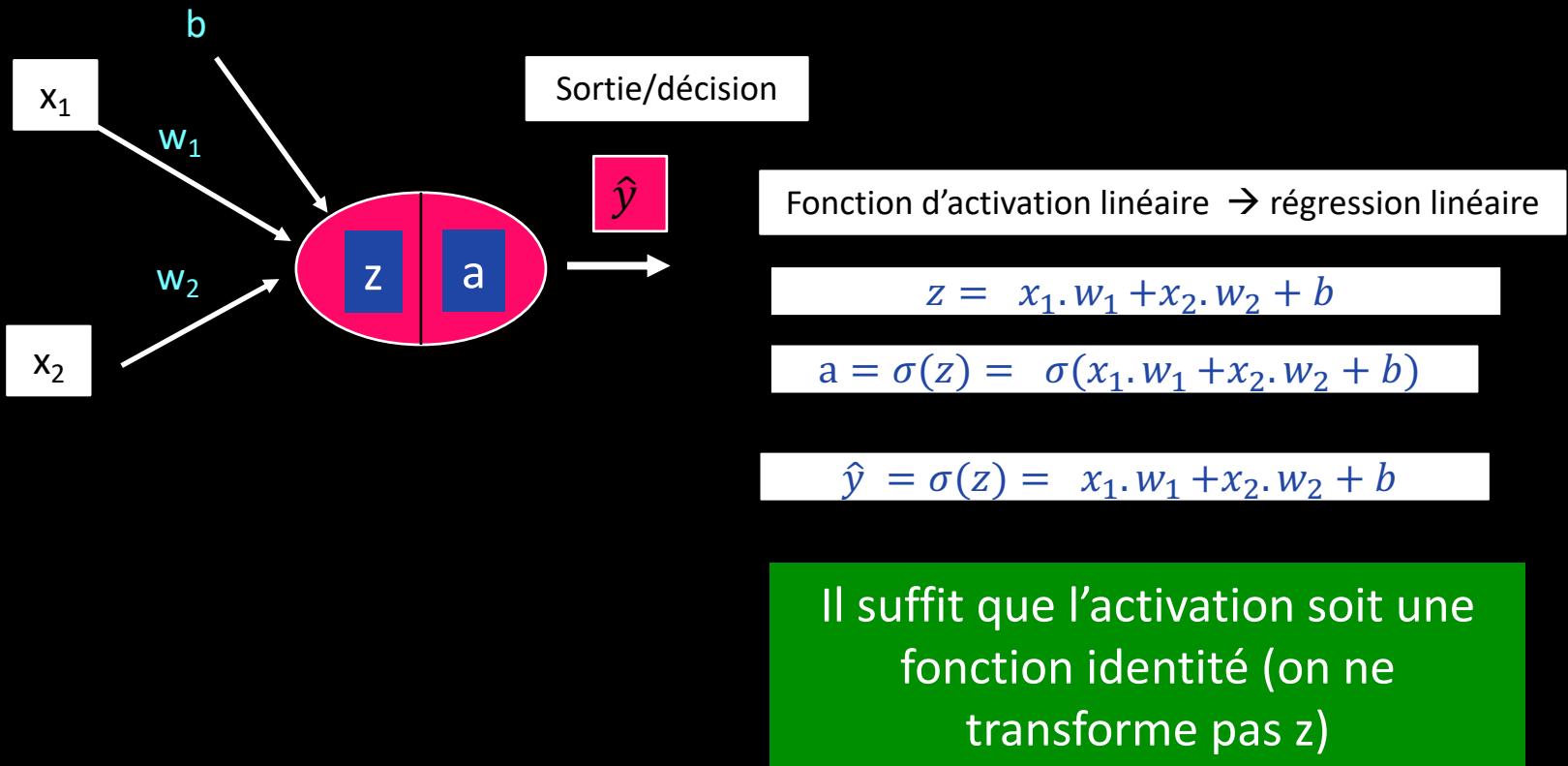
- Une panoplie de fonctions d'activation
- Dépend du label (sortie) à prédire
  - Identité  $\sigma(z) = z$  → Régression
  - Logistique (sigmoïde)  $\sigma(z) = \frac{1}{1+\exp(-z)}$  → Classification binaire
  - Relu :  $\sigma(z) = \max(0, z)$  → Classification binaire
  - Softmax  $\sigma(z) = \frac{e^{-z}}{\sum_{z_0 \in outputs} e^{-z_0}}$  → Classification multilabels

- tanh,
- ...



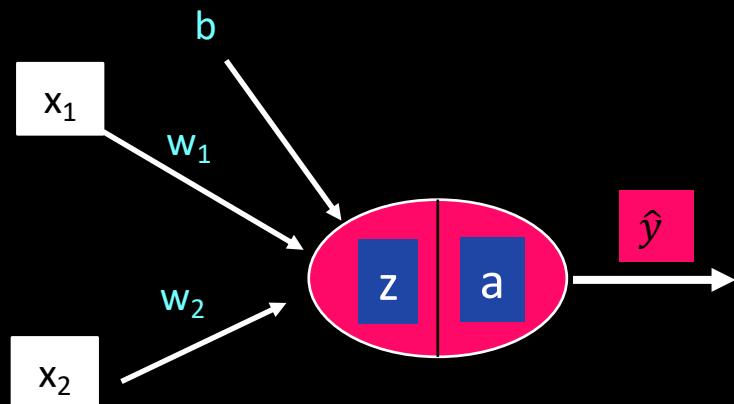
# Le perceptron : régression linéaire et classification linéaire

- Ce simple neurone permet de réaliser une simple régression



# Le perceptron : régression linéaire et classification linéaire

- ... et aussi une simple régression logistique (classification)



Fonction d'activation de type logistique →  
Classification linéaire

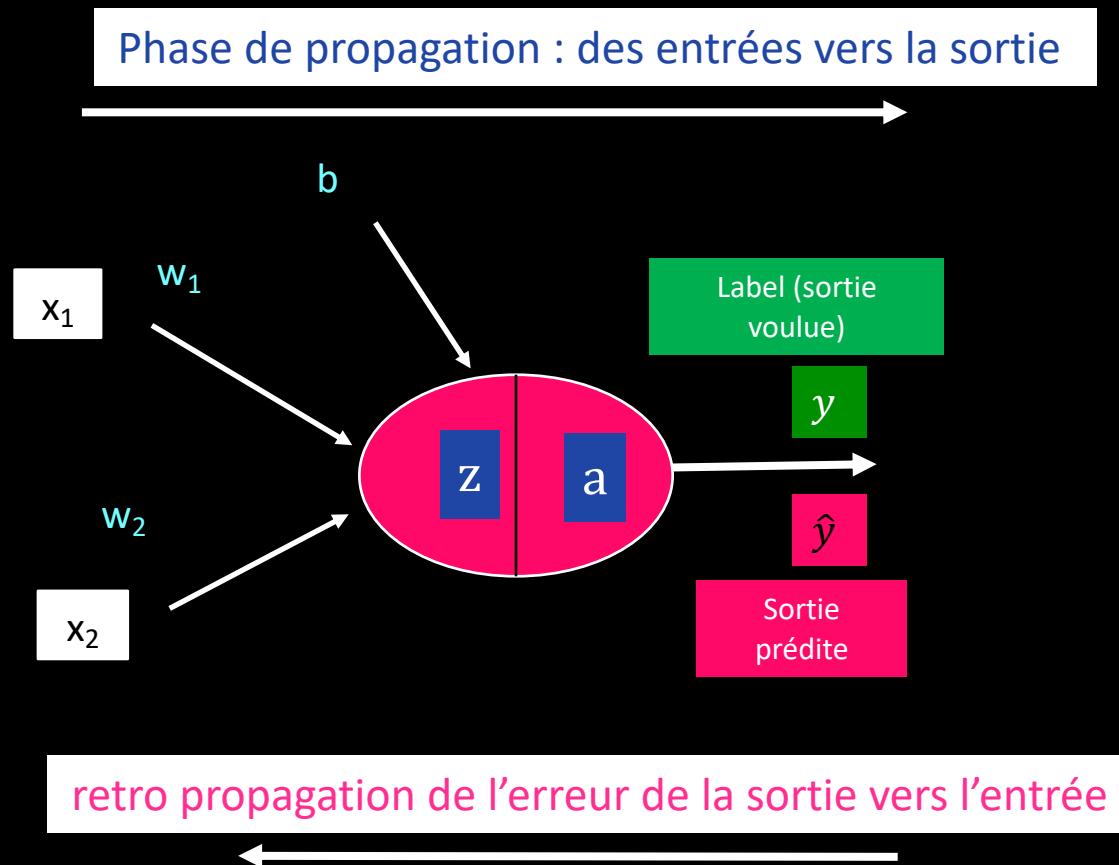
$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + b$$

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

$$\hat{y} = \begin{cases} 1 & \text{si } a > 0.5 \\ 0 & \text{sinon} \end{cases}$$

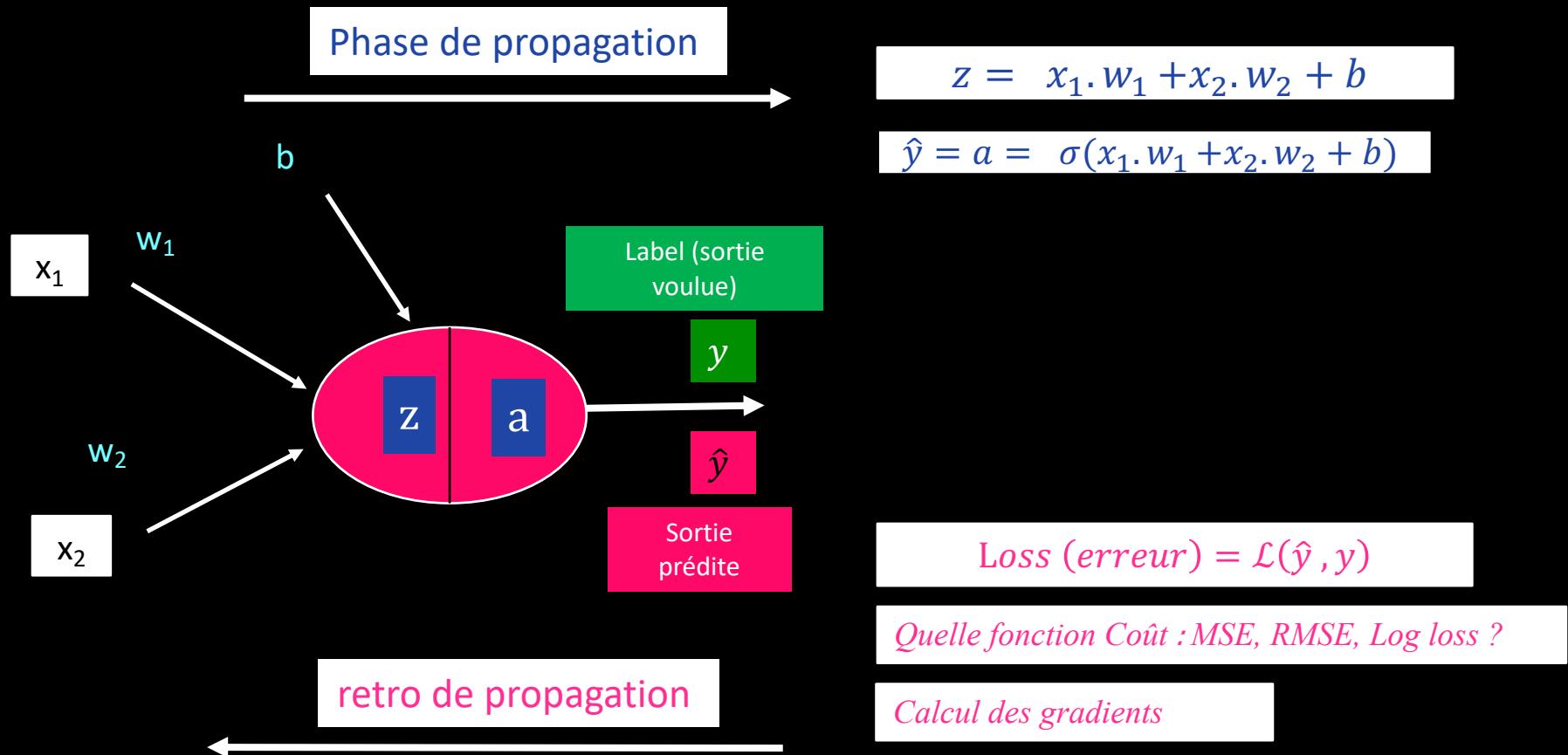
# Perceptron : apprentissage

- Méthode descente du gradient : 2 phases (étapes)



# Perceptron : apprentissage

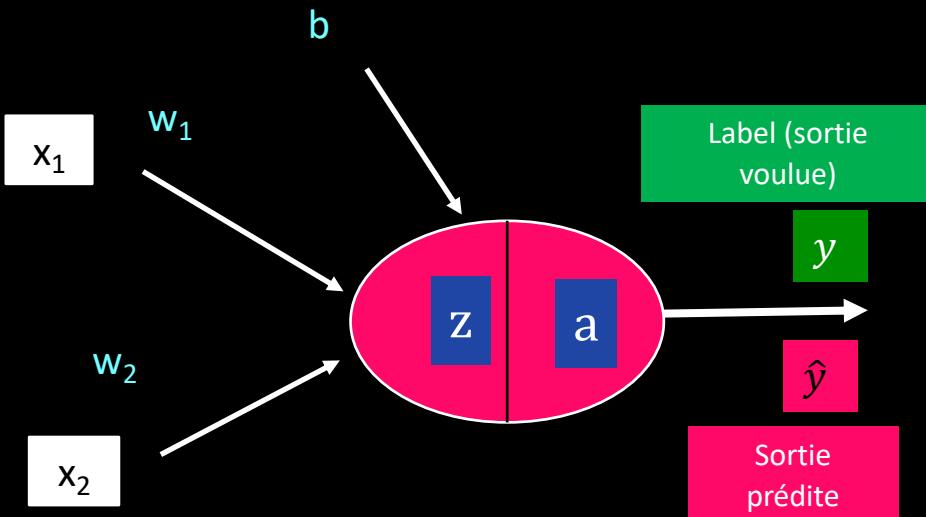
## • Méthode descente du gradient



Objectif de l'apprentissage : ajuster les paramètres du modèle,  $w_1$ ,  $w_2$  et  $b$  de façon à minimiser l'erreur.

# Perceptron : apprentissage

- Méthode de la descente du gradient



Minimiser la « Loss » (le coût)  
MSE, RMSE, Log loss ?

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2m} \sum ((y^{(i)} - \hat{y}^{(i)})^2)$$

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

Du déjà vu (slides sur la régression et la classification) : ajuster les paramètres  $w$  et  $b$  qui permettront de ramener la Loss à zéro

Comment : Calculer les gradients (les dérivées partielles) de la fonction coût par rapport à ses paramètres ( $w, b$ ), puis modifier ces paramètres

$$w_j = w_j - \alpha \frac{\delta \mathcal{L}(\hat{y}, y)}{\delta w_j}$$

$$b = b - \alpha \frac{\delta \mathcal{L}(\hat{y}, y)}{\delta b}$$

# Perceptron : apprentissage

- Méthode du gradient : calculer les gradients



$$z = x_1 \cdot w_1 + x_2 \cdot w_2 + b$$

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

Prenons par exemple une Log Loss

$$\mathcal{L}(\hat{y}, y) = \frac{-1}{m} \sum_{i=1}^m [y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})]$$

$$\frac{\delta \mathcal{L}(\hat{y}, y)}{\delta w_1} = \frac{\delta \mathcal{L}}{\delta a} * \frac{\delta a}{\delta z} * \frac{\delta z}{\delta w_1}$$

$$\frac{\delta \mathcal{L}(\hat{y}, y)}{\delta w_2} = \frac{\delta \mathcal{L}}{\delta a} * \frac{\delta a}{\delta z} * \frac{\delta z}{\delta w_2}$$

$$\frac{\delta \mathcal{L}(\hat{y}, y)}{\delta b} = \frac{\delta \mathcal{L}}{\delta a} * \frac{\delta a}{\delta z} * \frac{\delta z}{\delta b}$$



$$\frac{\delta \mathcal{L}(\hat{y}, y)}{\delta w_j} = \frac{1}{m} \sum_{i=1}^m (a^{(i)} - y^{(i)}) x_j^{(i)}$$

$$\frac{\delta \mathcal{L}(\hat{y}, y)}{\delta b} = \frac{1}{m} \sum_{i=1}^m (a^{(i)} - y^{(i)})$$

## Le perceptron : récap.

- Chaque neurone reproduit (agit) exactement les mêmes fonctionnalités qu'un algorithme de régression linéaire ou logistique
- A l'issue de l'apprentissage, le neurone (le perceptron) apprend un modèle linéaire

# Le perceptron : Mise en oeuvre.

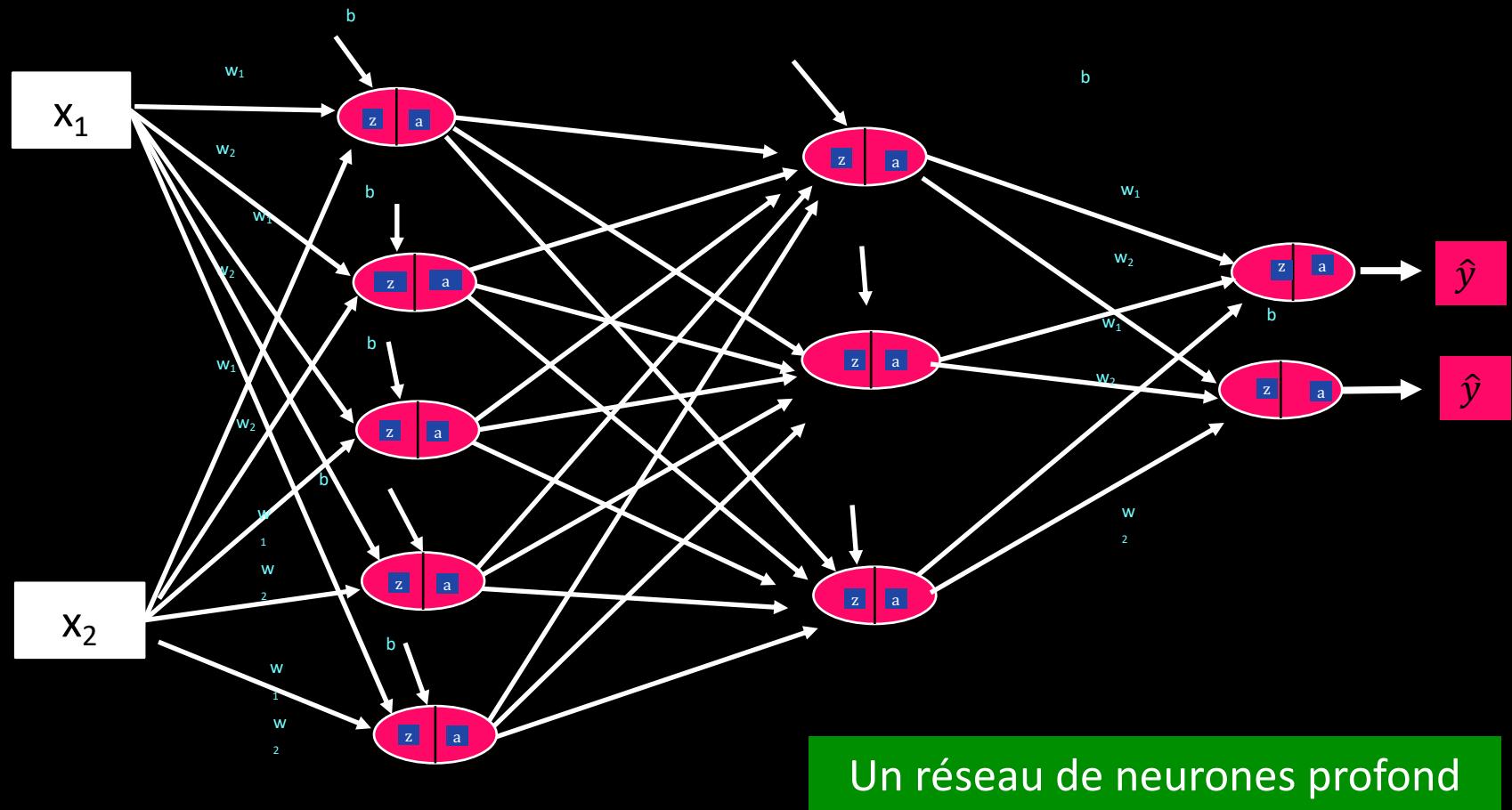
- Paramètres importants :
  - Architecture (rien à faire) : un neurone
  - Quelle activation? : dépend du type problème (identifié, relu, sigmoïde)
  - Quelle fonction coût(Erreur/loss)? : MSE, BinaryCrossEntropy, ??
  - Quelle métrique ? : mse, f1score, précision, ???
  - Quel algorithme d'apprentissage (optimiseur)?
    - Mais c'est la descente du gradient non ? Oui mais, ...
    - → il existe différents algos qui proposent différente optimisation de la descente du gradient (nous les verrons plus loin)

# Réseaux de neurones profonds

## Deep neural Network

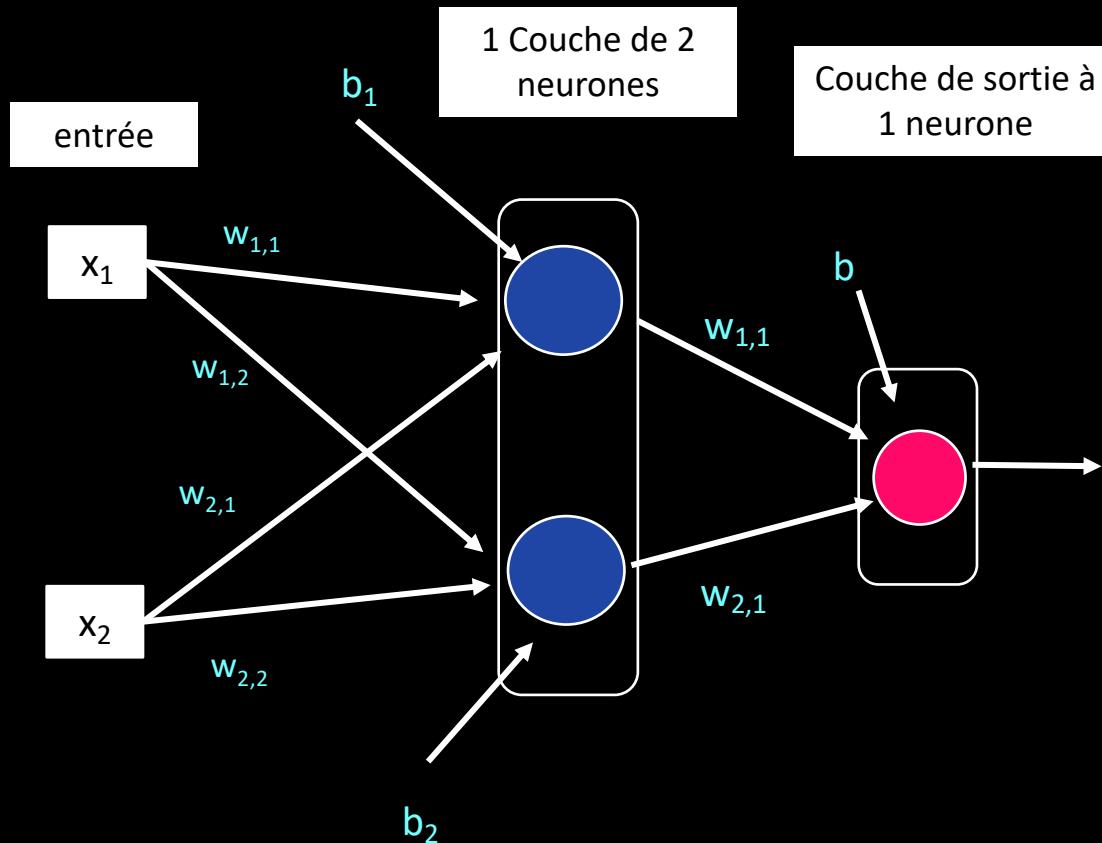
# Modèle non linéaire (complexe)

- Si on utilise plusieurs neurones (plusieurs modèles linéaires) en cascade, on apprendra un modèle non linéaire capable de faire des combinaisons complexes des « features » (combinaison de combinaisons et de combinaisons)



# Réseaux de neurones multicouches

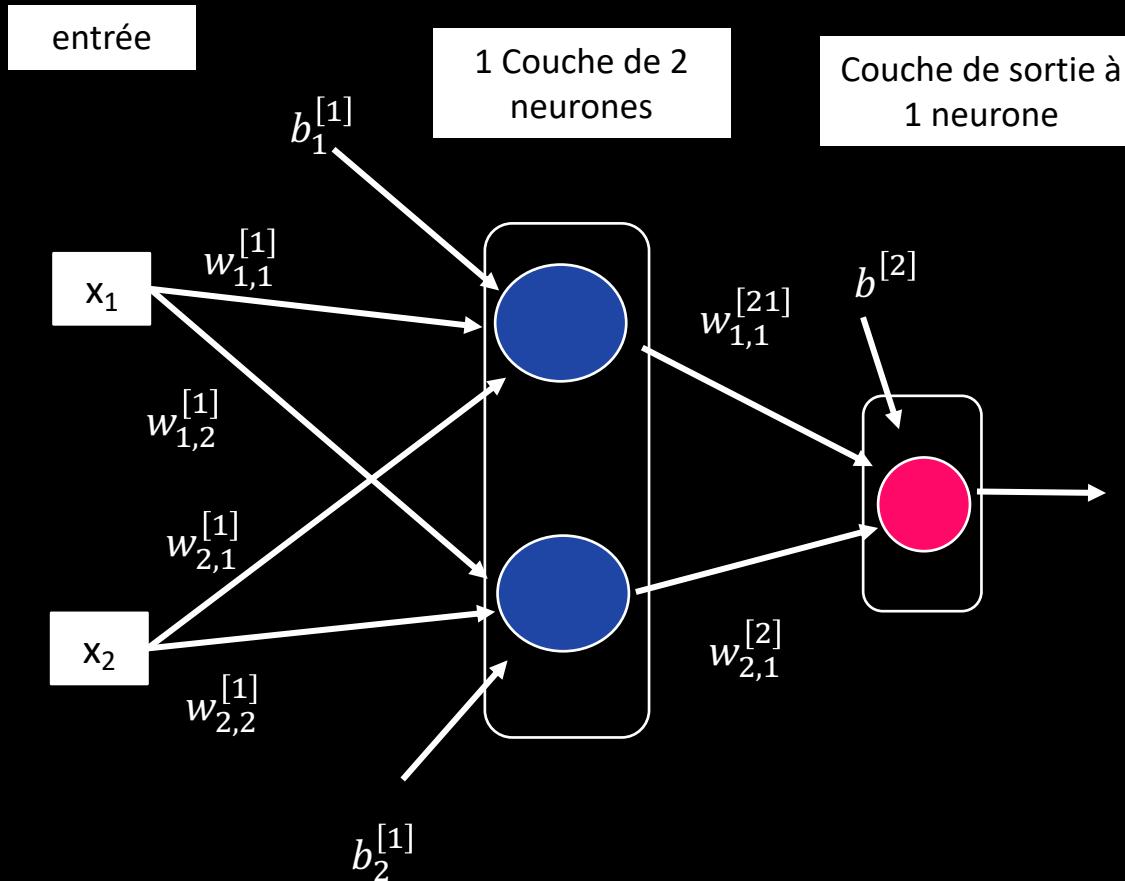
- Rajouter des neurones sous forme de couches



Confusion entre les  $w$  et  $b$  des différentes couches → expliciter la couche

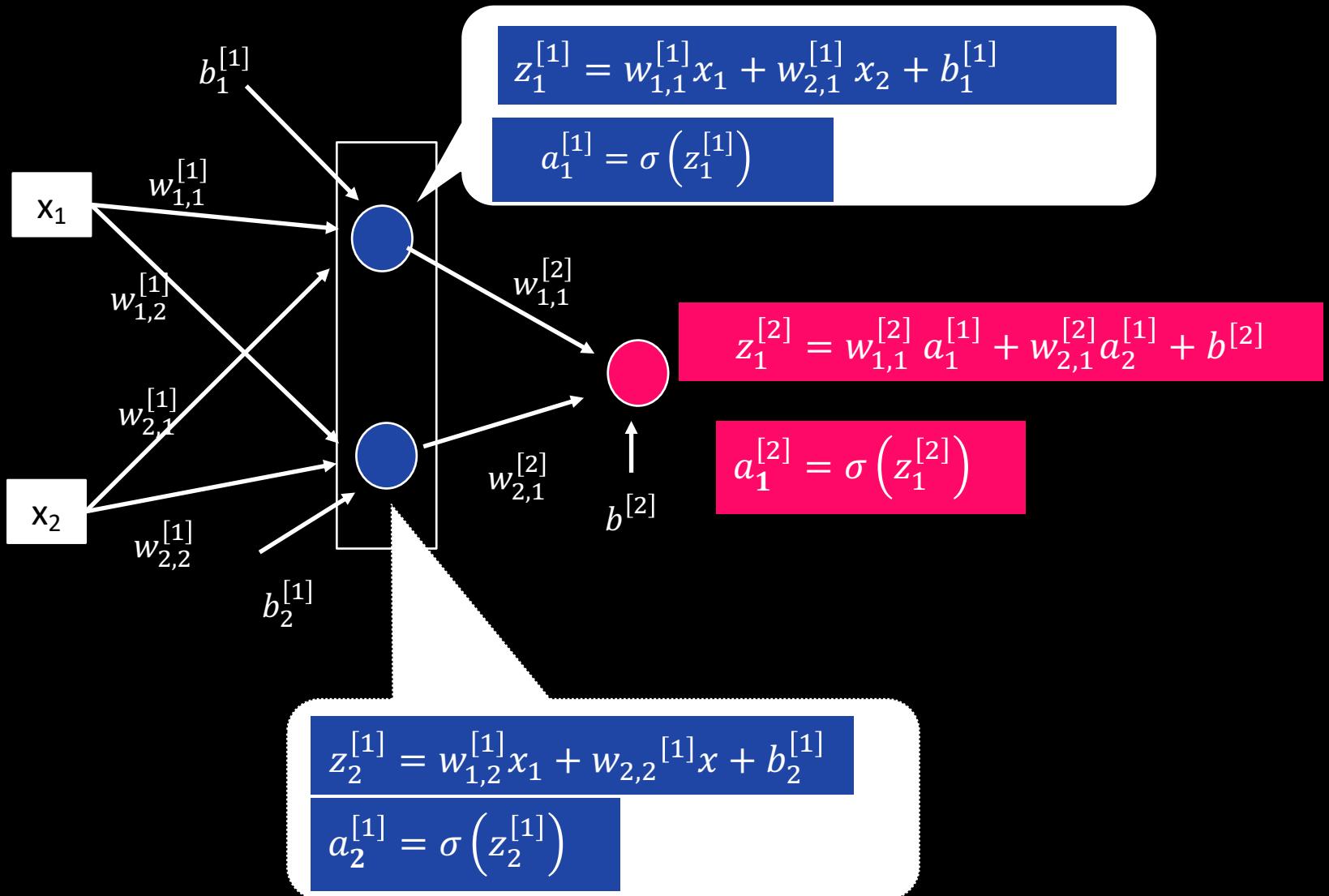
# Réseaux de neurones multicouches

- Rajouter des neurones sous forme de couches et notations



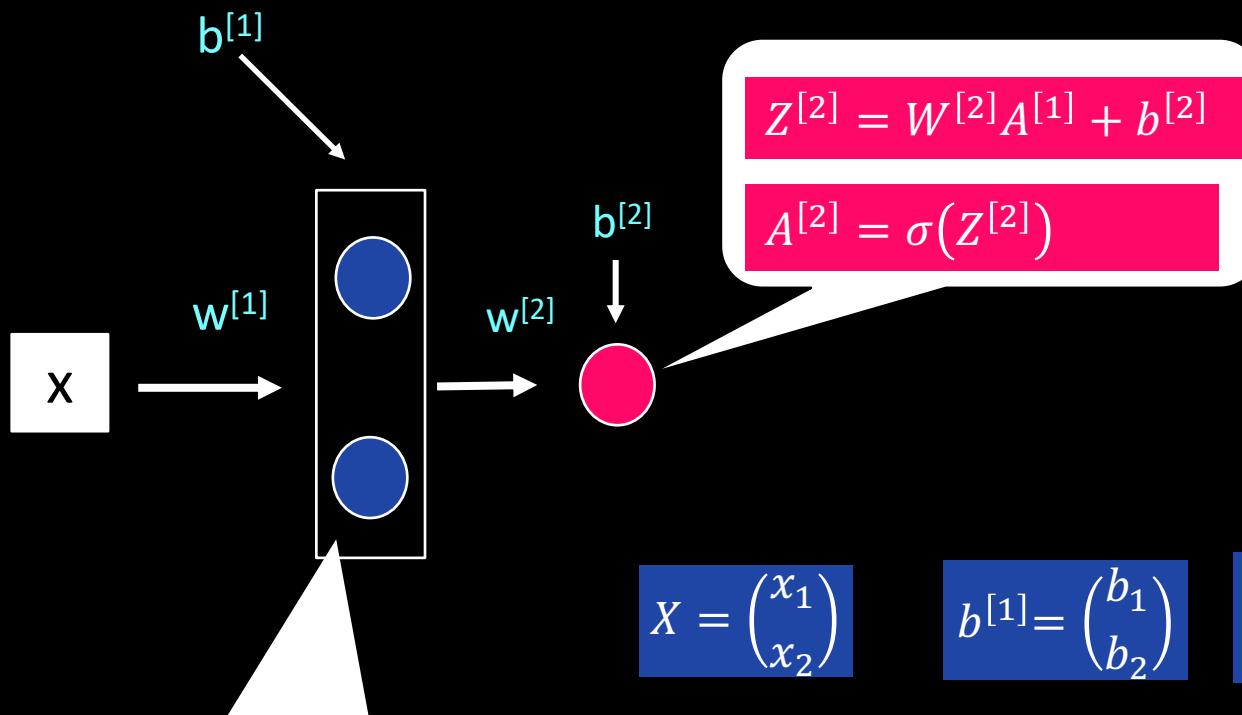
# Réseaux de neurones multicouches

- Rajouter des neurones sous forme de couches



# Réseaux de neurones multicouches

- Ecriture matricielle : vectorisation



$$X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad b^{[1]} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad W^{[1]} = \begin{pmatrix} w_{1,1}, w_{1,2} \\ w_{2,1}, w_{2,2} \end{pmatrix}$$

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

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

$$Z^{[1]} = \begin{pmatrix} w_{1,1}x_1 + w_{2,1}x_2 + b_1 \\ w_{1,2}x_1 + w_{2,2}x_2 + b_2 \end{pmatrix}$$

$$A^{[1]} = \begin{pmatrix} \sigma(w_{1,1}x_1 + w_{2,1}x_2 + b_1) \\ \sigma(w_{1,2}x_1 + w_{2,2}x_2 + b_2) \end{pmatrix}$$

# Réseaux de neurones multicouches

- Ecriture matricielle : vectorisation (Attention à la manière dont sont représentées vos données)

$$X = (x_1, x_2)$$

$$b^{[1]} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$W^{[1]} = \begin{pmatrix} w_{1,1}, w_{1,2} \\ w_{2,1}, w_{2,2} \end{pmatrix}$$

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

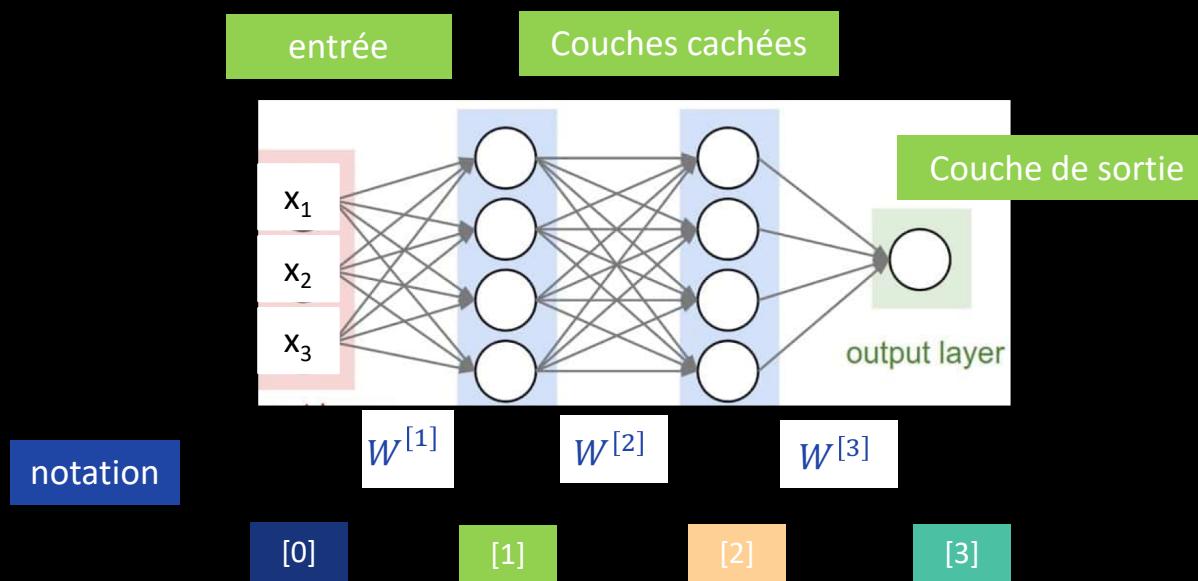
$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[1]} = \begin{pmatrix} w_{1,1}x_1 + w_{2,1}x_2 + b_1 \\ w_{1,2}x_1 + w_{2,2}x_2 + b_2 \end{pmatrix}$$

$$A^{[1]} = \begin{pmatrix} \sigma(w_{1,1}x_1 + w_{2,1}x_2 + b_1) \\ \sigma(w_{1,2}x_1 + w_{2,2}x_2 + b_2) \end{pmatrix}$$

# Un réseau multicouches : MLP

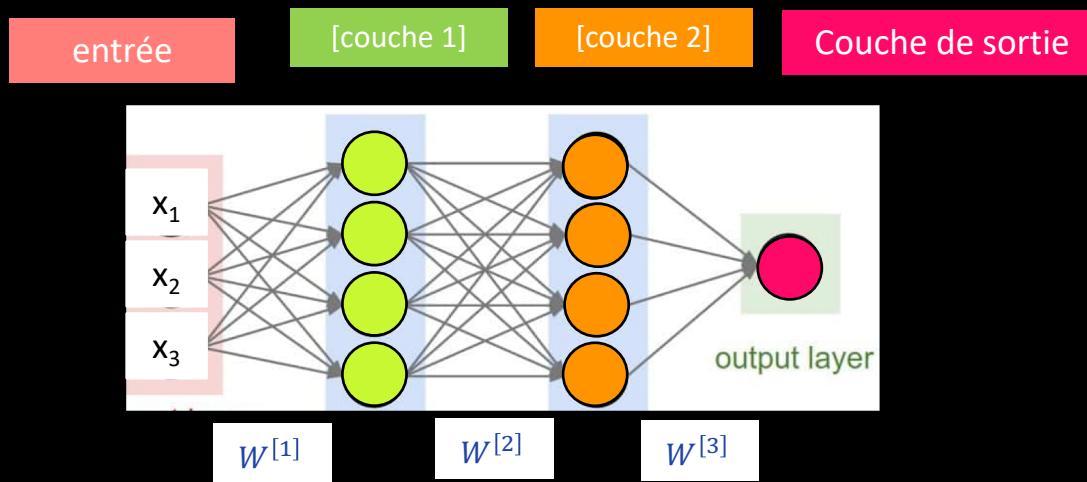
- Un réseau de neurones est une succession de couches de neurones, reliées entre elles, comportant
  - une entrée ( $X$ ),
  - des couches cachées de neurones
  - une ou plusieurs sorties ( $Y$ )
  - chaque neurone se comporte comme → un perceptron



# Un réseau multicouches : MLP

- Phase de propagation de signaux de l'entrée vers la sortie

| $x_1$ | $x_2$ | $x_3$ | $y$ |
|-------|-------|-------|-----|
| 3     | 1     | 22.0  | 0   |
| 1     | 0     | 38.0  | 1   |
| 3     | 0     | 26.0  | 1   |
| 1     | 0     | 35.0  | 1   |
| 3     | 1     | 35.0  | 0   |



$x_1$  : pclass  
 $x_2$  : Sex  
 $x_3$  : Age

Y: survived

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

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

$$Z^{[3]} = A^{[2]} \cdot W^{[3]} + b^{[3]}$$

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

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

$$A^{[3]} = \sigma(Z^{[3]})$$

Pour une couche  $[l]$

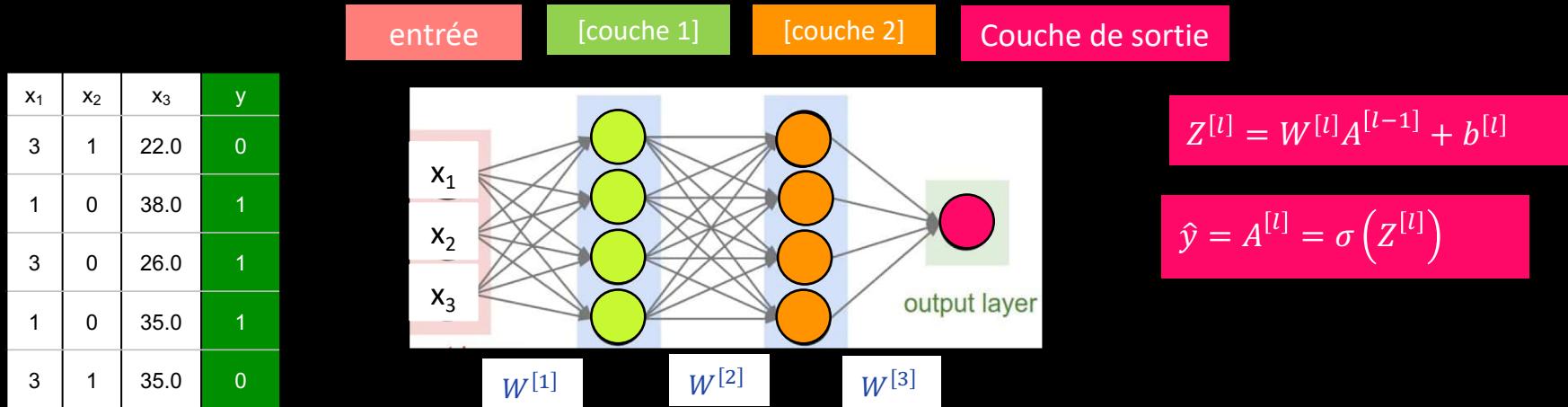
$$Z = A^{[l-1]} \cdot W^{[l]} + b^{[l]}$$

$$a^{[l]} = \sigma(Z^{[l]})$$

En python, le calcul de  $Z$  s'écrit  
 $Z = X @ W + b$

# Un réseau multicouches : Descente de gradient

- La phase d'apprentissage : rétro propagation des gradients (de l'erreur)



x<sub>1</sub>: Pclass  
 x<sub>2</sub>: Sex  
 x<sub>3</sub>: Age

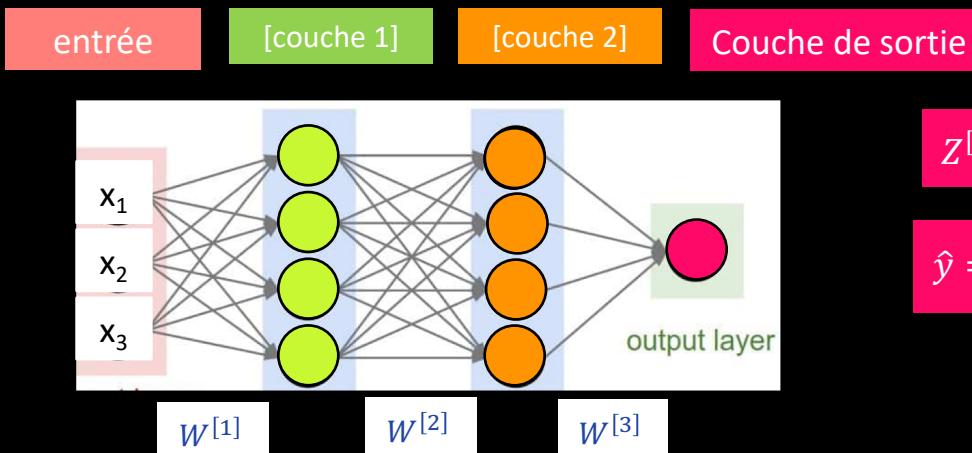
Y: survived

1. Définir une fonction coût
  - $\mathcal{L} = \frac{1}{m} \sum (\mathcal{L}(y^{(i)}, a^{(i)})$
1. Calculer les gradients au niveau de chaque couche (l'impact de chaque paramètre d'une couche sur l'erreur)
  - $\frac{\partial \mathcal{L}}{\partial W^{[l]}}$  et  $\frac{\partial \mathcal{L}}{\partial b^{[l]}}$
2. Mettre à jour les paramètres
  - $W^{[l]} = W^{[l]} - \alpha \frac{\delta \mathcal{L}}{\delta W^{[l]}}$  et  $b^{[l]} = b^{[l]} - \alpha \frac{\delta \mathcal{L}}{\delta b^{[l]}}$

# Un réseau multicouches : Régression

- La phase d'apprentissage : rétro propagation des gradients (de l'erreur)

| x <sub>1</sub> | x <sub>2</sub> | x <sub>3</sub> | y   |
|----------------|----------------|----------------|-----|
| 2              | 1              | 1              | 220 |
| 1.2            | 1              | 1              | 120 |
| 1.5            | 2              | 2              | 230 |
| 2.5            | 2              | 2              | 320 |
| 2              | 3              | 2              | 300 |



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

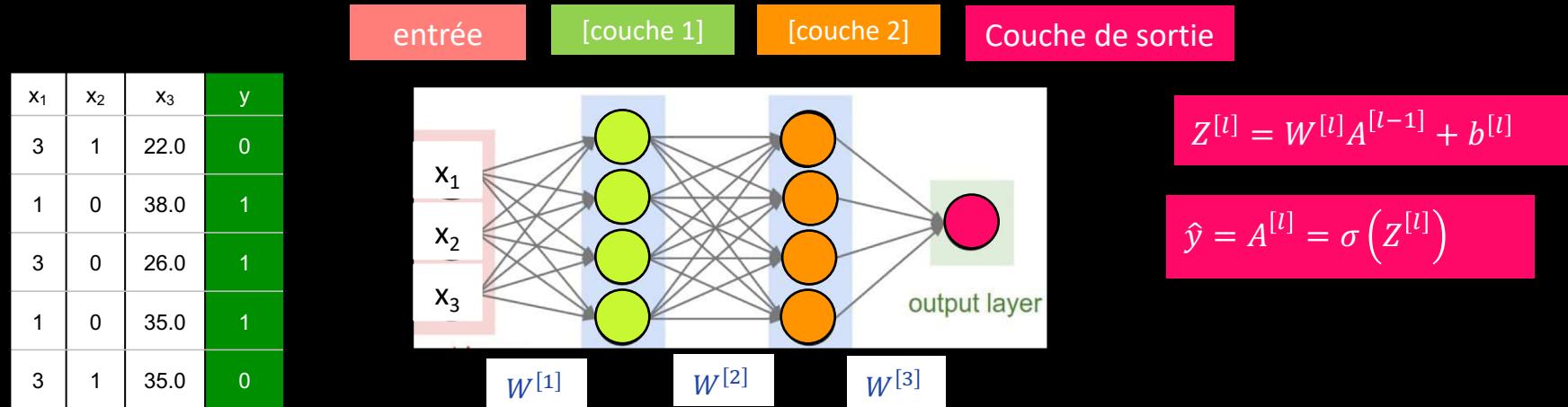
$$\hat{y} = A^{[l]} = \sigma(Z^{[l]})$$

$x_1$ : Surface  
 $x_2$ : NbSDB  
 $x_3$ : Etage  
Y: Prix

1. Un neurone de sortie avec une activation identité

# Un réseau multicouches : Classification binaire

- La phase d'apprentissage : rétro propagation des gradients (de l'erreur)



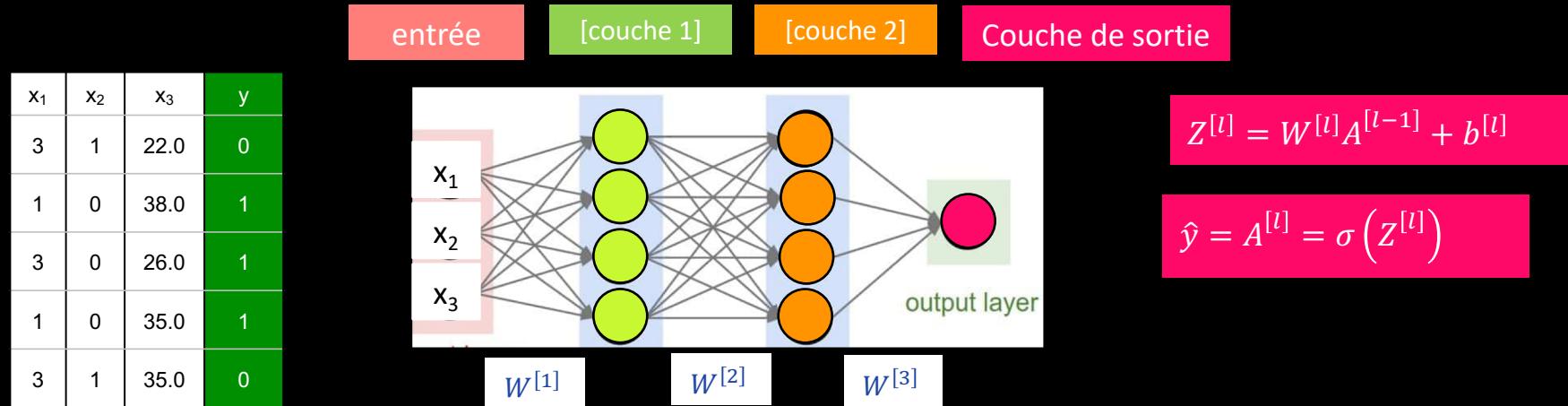
$x_1$ : Pclass  
 $x_2$ : Sex  
 $x_3$ : Age  
Y: survived

- La sortie un neurone simple avec une activation de type sigmoïde

$$\hat{y} = \begin{cases} 1 & \text{si } a > 0.5 \\ 0 & \text{sinon} \end{cases}$$

# Un réseau multicouches : Multiclasses

- La phase d'apprentissage : rétro propagation des gradients (de l'erreur)

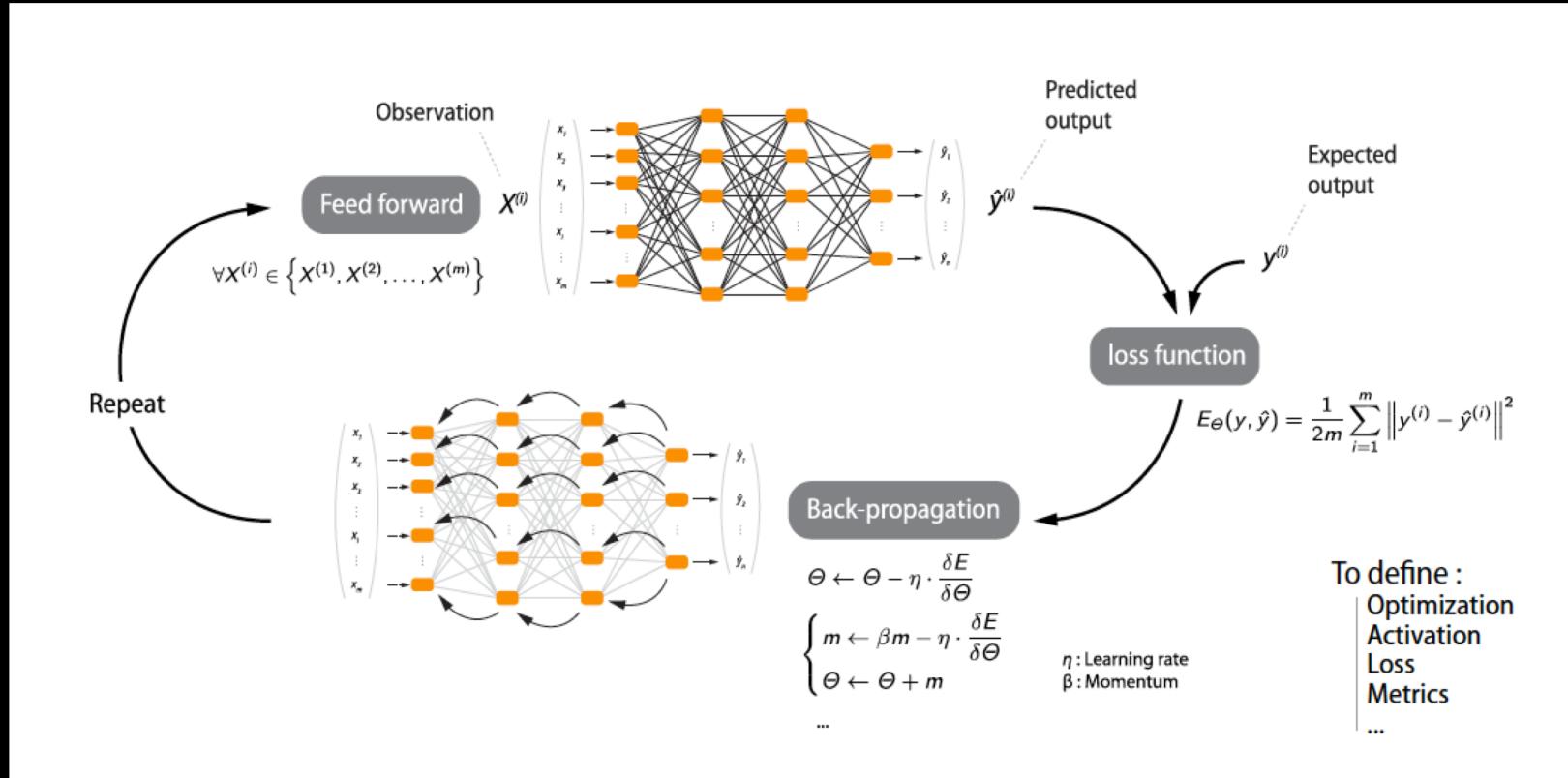


x<sub>1</sub>: Pclass  
x<sub>2</sub>: Sex  
x<sub>3</sub>: Age  
  
Y: survived

- La sortie autant de neurones que de classes on utilise un SoftMax

# Un réseau multicouches : Récap processus global

- Propagation/ rétropropagation



# Les librairies (environnements) pour Deep learning

- Trois Librairies importantes : Keras, Pytorch et Tensorflow



# Mise en œuvre sous tensorflow et Keras

---

- Préparation des données (du Dataset)
  - Sélection de variables, Normalisation, Imputation
- Définir un modèle de réseaux de neurones
  - Définir son architecture, nombre de couches, nombre de neurones par couche, l'activation des neurones par couche
- Compiler le modèle
  - Définir l'optimiseur (l'algo de descente de gradient à utiliser), la fonction cout, les métriques de de performances,....
- Entrainer le modèle (le fit des cours précédents)
  - fournir les données d'entraînement, le nombre d'époques, la taille du lot (batch), les données de test
- Evaluation du modèle sur des données de test
- Prédiction (utilisation) du modèle appris

# Mise en œuvre sous tensorflow et Keras

- Définir un modèle de réseaux de neurones
  - Importer tensorflow (import tensorflow as tf)
  - On peut importer des bibliothèques spécifiques de tensorflow
    - from tensorflow.keras import datasets,
    - ....
  - Définition d'un modèle séquentiel (couches)

## Option 1

```
model = tf.keras.Sequential ([  
    tf.keras.layers.Input(10),  
    tf.keras.layers.Dense(units=1, activation = 'linear')  
])
```

## Option 2

```
model = tf.keras.Sequential ()  
model.add(tf.keras.layers.Input(10, name="InputLayer"))  
model.add(tf.keras.layers.Dense(32, activation='relu'))  
model.add(tf.keras.layers.Dense(units=1, activation = 'sigmoid'))
```

# Mise en œuvre sous tensorflow et Keras

- Définir un modèle de réseaux de neurones
  - Importer tensorflow (import tensorflow as tf)
  - On peut importer des bibliothèques spécifiques de tensorflow
    - From tensorflow.keras import datasets,
    - .....
  - Définition d'un modèle séquentiel (couches)

Il existe d'autres façons de définir un réseau de neurones

# Mise en œuvre sous Tensorflow et Keras

- Définir un modèle de réseaux de neurones
- Compiler le modèle (**compile**), revient à instancier
  - l'optimiseur (**Optimizer**) : Adam, SGD, RMSprop, ....
  - La fonction cout (**Loss**): BinaryCrossentropy, CategoricalCrossentropy, *MeanSquaredError*
  - Les métriques (**Metrics**) : Accuracy, Precision, Recall

```
model.compile (      optimizer='adam',
                    loss= 'BinaryCrossentropy',
                    metrics=['accuracy'])
```

# Mise en œuvre sous Tensorflow et Keras

- Définir un modèle de réseaux de neurones
- Compiler le modèle (**compile**)
  - Définir l'optimiseur,
  - loss,
  - Metrics

```
model.compile (      optimizer='adam',
                     loss= 'BinaryCrossentropy',
                     metrics=['accuracy', 'precision'])
```

- Entraînement du modèle (**fit**)

```
history = model.fit(X_train, y_train,
                     epochs=100,
                     batch_size= 10,
                     verbose=False,
                     validation_data = (X_test, y_test)
                    )
```

# Mise en œuvre sous tensorflow et Keras

- Définir un modèle de réseaux de neurones
- Compiler le modèle
- Entraînement du modèle
- Prédiction (inférence) (**evaluate**)

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test accuracy :', score[0])
print('Test precision :', score[1])
```

Fin

---