

# INF8175 - Intelligence artificielle

*Méthodes et algorithmes*

## Module 7: Réseaux de neurones et apprentissage profond



POLYTECHNIQUE  
MONTRÉAL

Quentin Cappart

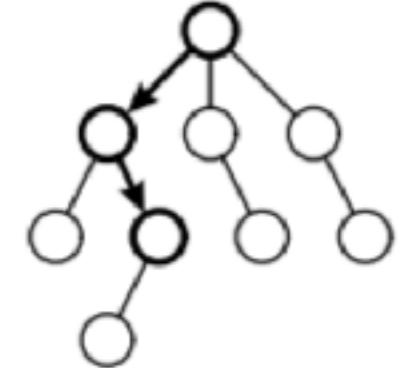
# Contenu du cours

## Raisonnement par recherche (essais-erreurs avec de l'intuition)

**Module 1:** Stratégies de recherche

**Module 2:** Recherche en présence d'adversaires

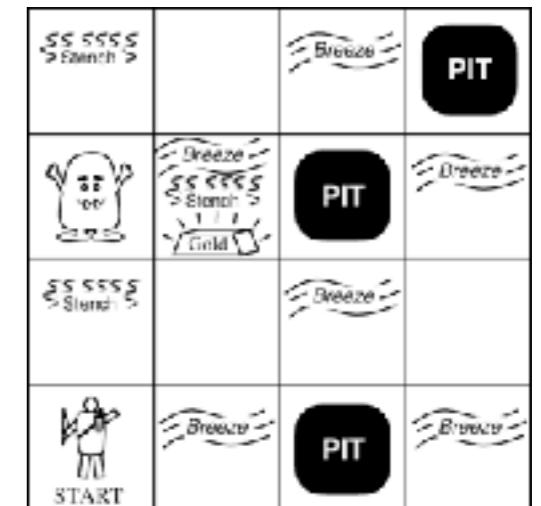
**Module 3:** Recherche locale



## Raisonnement logique

**Module 4:** Programmation par contraintes

**Module 5:** Agents logiques



## Raisonnement par apprentissage

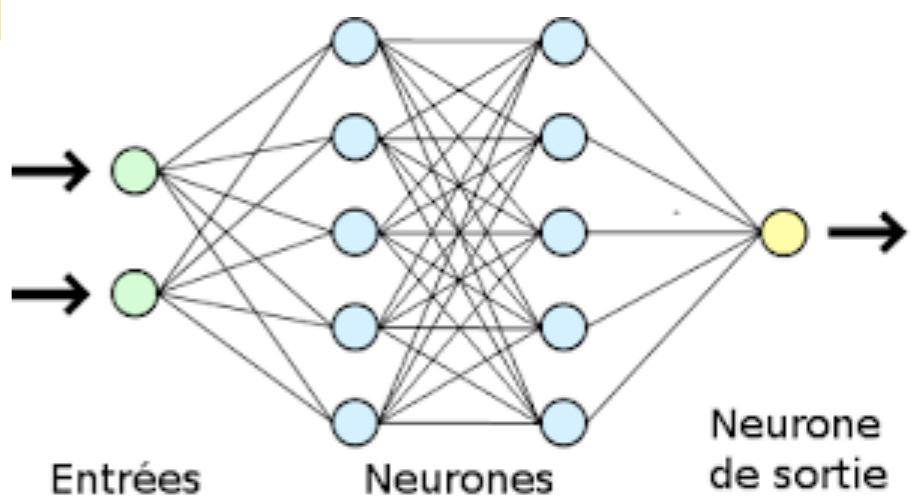
**Module 6:** Apprentissage supervisé

**Module 7:** Réseaux de neurones et apprentissage profond



**Module 8:** Apprentissage non-supervisé

**Module 9:** Apprentissage par renforcement



## Considérations pratiques et sociétales

**Module 10:** Utilisation en industrie, éthique, et philosophie

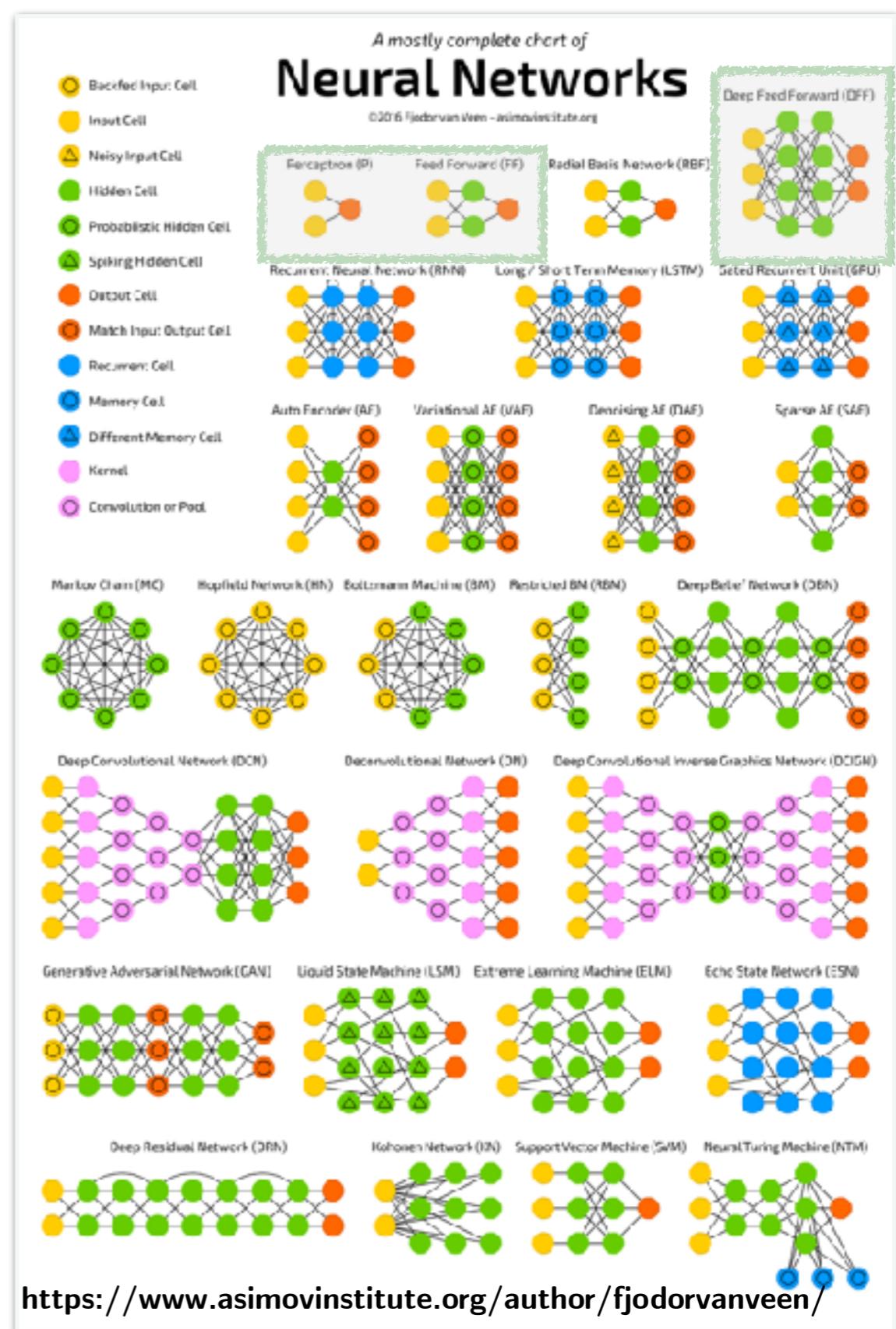
# Table des matières

## Réseaux de neurones

1. Concepts principaux des réseaux de neurones
2. Extension de la régression logistique
3. Notations standards des réseaux de neurones
4. Exécution de la *backpropagation* du gradient
5. Principes des fonctions d'activation
6. Initialisation des paramètres

## Apprentissage profond

1. Limitation des petits réseaux de neurones
2. Principes de l'apprentissage profond
3. Profondeur d'un réseau de neurones
4. Introduction aux hyper-paramètres
5. Conseils pratiques

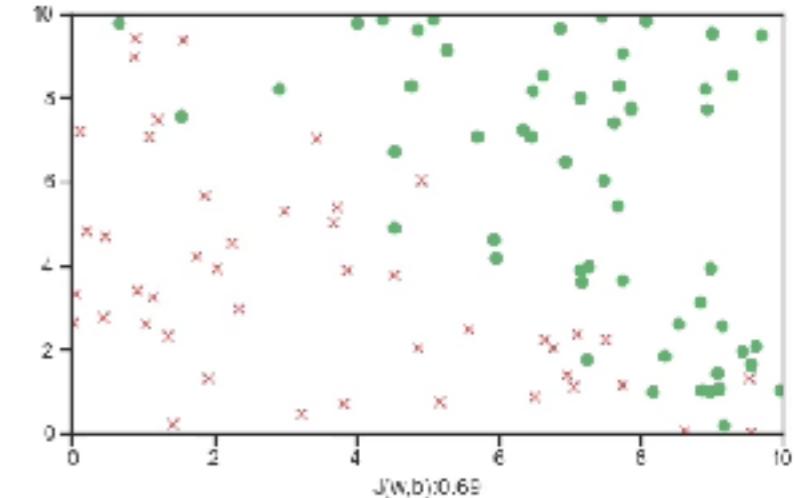
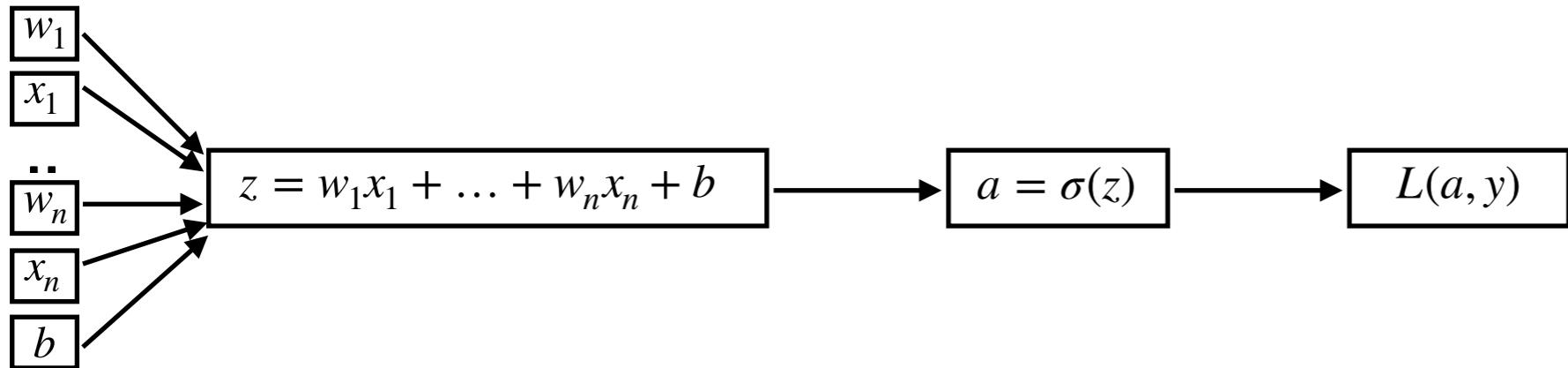


# Rappel de la régression logistique

Une régression logistique construit d'une fonction prédisant une valeur entre 0 et 1 (module précédent)

**Etape 1:** on prend une combinaison linéaire des caractéristiques

**Etape 2:** on applique une sigmoïde sur cette combinaison



**Intérêt:** permet de créer une frontière de décision linéaire

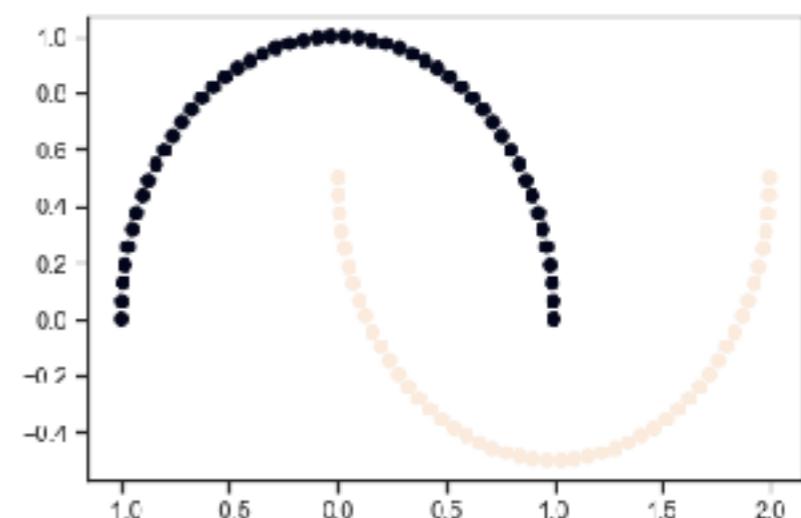
**Algorithme d'apprentissage:** descente de gradient avec une passe en avant et en arrière

**Hypothèse:** une tendance linéaire entre les données peut être établie

**Ce n'est malheureusement pas souvent le cas en pratique**

**Besoin:** on doit construire une fonction (hypothèse) plus expressive

Les réseaux de neurones permettent de combler ce manque



# Aperçu d'un réseau de neurones



## Réseau de neurones

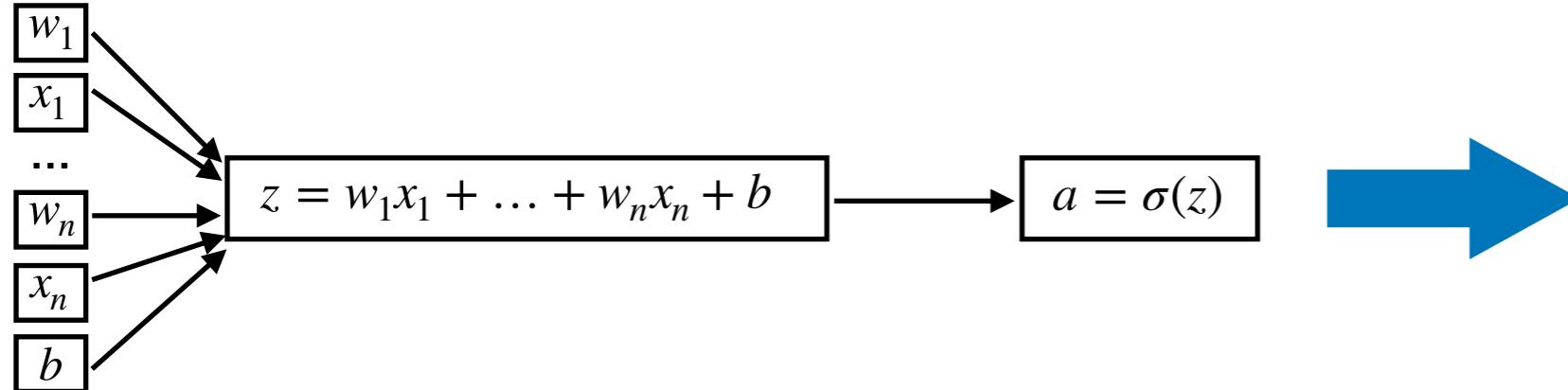
Ensemble de neurones organisés en un réseau de plusieurs couches



Qu'est-ce qu'un neurone dans ce contexte ?

Sans le savoir, on a déjà construit nos premiers neurones !

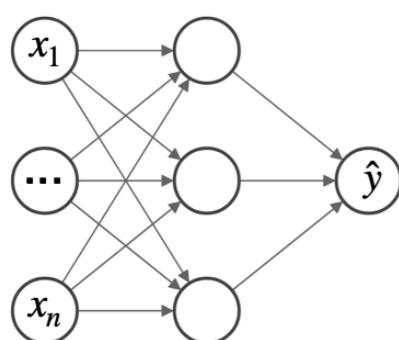
**Neurone actuel:** combinaison linéaire de variables, à laquelle est appliquée une sigmoïde



Exemple de neurone

**Par la suite:** on verra comment créer d'autres types de neurones (non basé sur une sigmoïde)

**Représentation visuelle:** les paramètres à apprendre ( $w_1, \dots, w_n, b$ ) sont souvent implicites



**Réseau:** les neurones sont organisés en un réseau de plusieurs couches (*layers*)

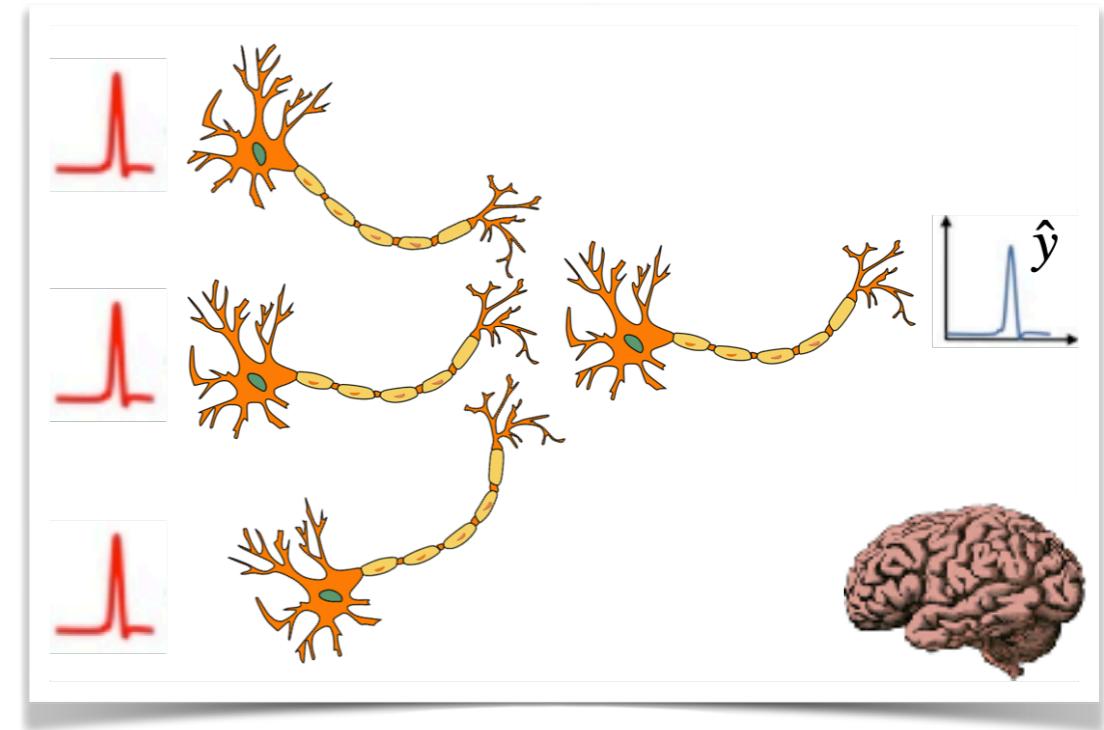
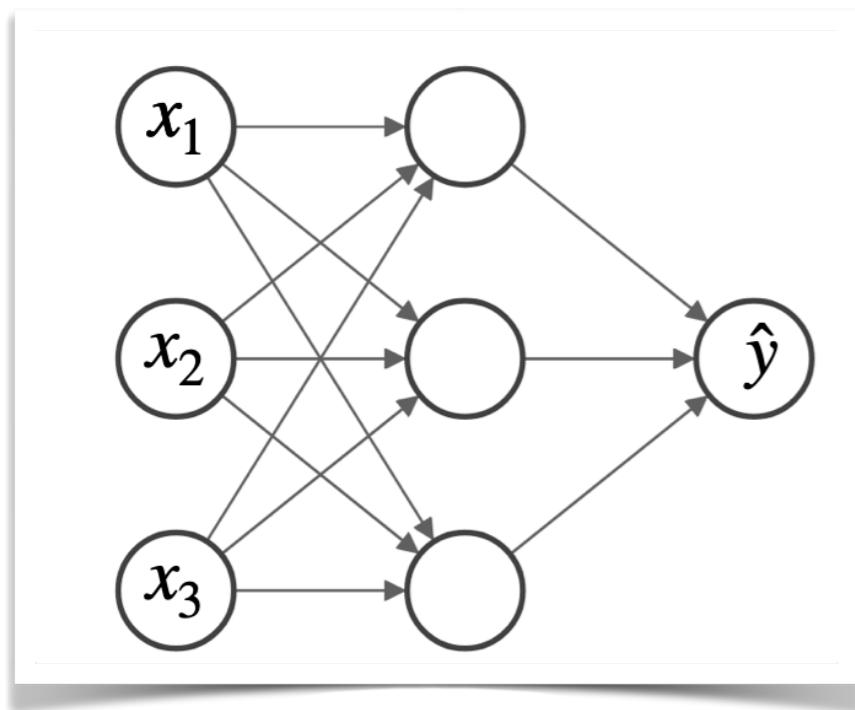
**Paramètres:** chaque neurone a ses propres paramètres à apprendre ( $w$  et  $b$ )

**Intuition:** chaque neurone apporte sa propre contribution pour la prédiction

**Conséquence:** on a un modèle plus expressif qu'une régression logistique

L'idée est d'apprendre une fonction différente avec chaque neurone, qui sont ensuite composées

# Métaphore biologique



**Cerveau humain:** organisé en un énorme réseau de neurones (86 milliards de neurones)

**Transmission de l'information:** sous forme de signal électrique, en passant de neurone en neurone

**Attention:** il s'agit simplement une métaphore, et non une correspondance exacte

**Il y a de nombreuses différences entre le fonctionnement d'un cerveau et ce qu'on va voir**

**Attention:** les inspirations naturelles (même existantes) sont plus de nature marketing que scientifique

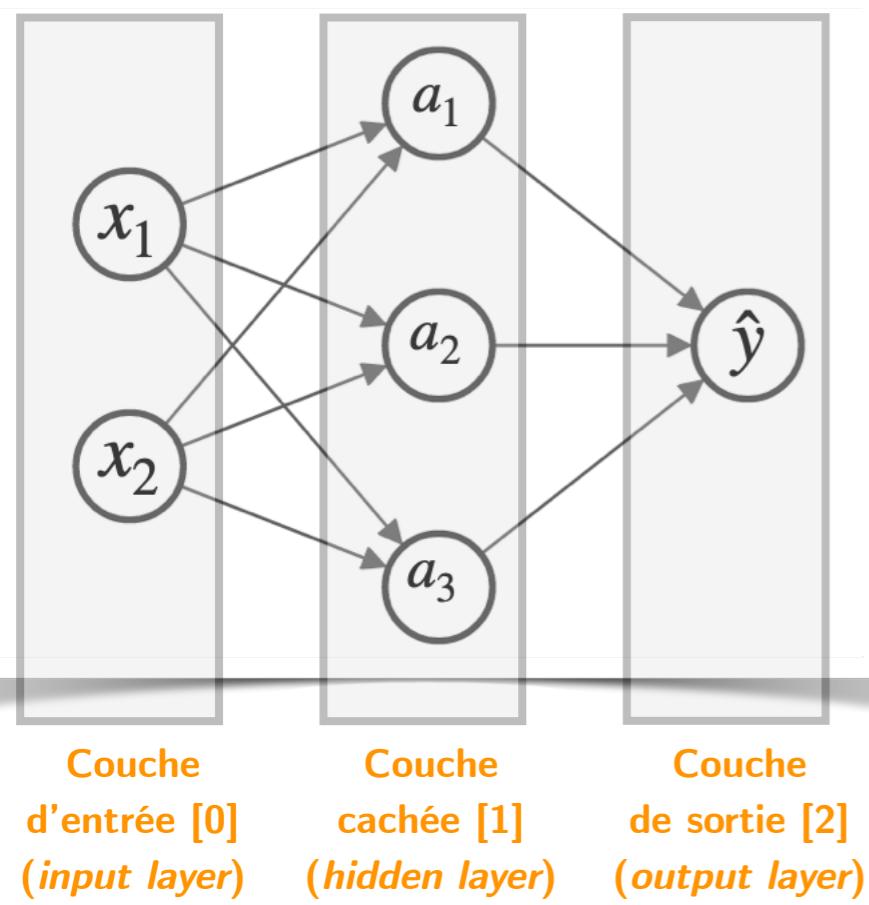


*"C'est un abus de langage que de parler de neurones !  
De la même façon qu'on parle d'aile pour un avion mais aussi pour un oiseau,  
le neurone artificiel est un modèle extrêmement simplifié de la réalité biologique."*

[https://www.sciencesetavenir.fr/high-tech/intelligence-artificielle/  
selon-yann-lecun-l-intelligence-artificielle-a-20-ans-pour-faire-ses-preuves\\_120121](https://www.sciencesetavenir.fr/high-tech/intelligence-artificielle/selon-yann-lecun-l-intelligence-artificielle-a-20-ans-pour-faire-ses-preuves_120121)

Yann LeCun

# Réseau de neurones à deux couches: notations



**Couche d'entrée:** couche reprenant les caractéristiques de la donnée

**La couche d'entrée n'a ainsi aucun paramètres à apprendre**

$x_1, x_2$

**Couche cachée:** couche dédiée aux calculs intermédiaires

$a_1, a_2, a_3$

**Couche de sortie:** couche dédiée à la prédiction finale

$\hat{y}$

**Convention:** la couche d'entrée est la couche 0

**Notation:** on va étendre nos notations précédentes

$x_j$ : caractéristique  $j$  pour une donnée spécifique

$a_j$ : sortie du neurone  $j$  à la couche cachée

$\hat{y}$ : prédiction pour une donnée selon ses caractéristiques  $x$

$n^{[i]}$ : nombre de neurones à la couche  $i$

$w_j^{[i]}, b_j^{[i]}$ : paramètres du neurone  $j$  à la couche  $i$

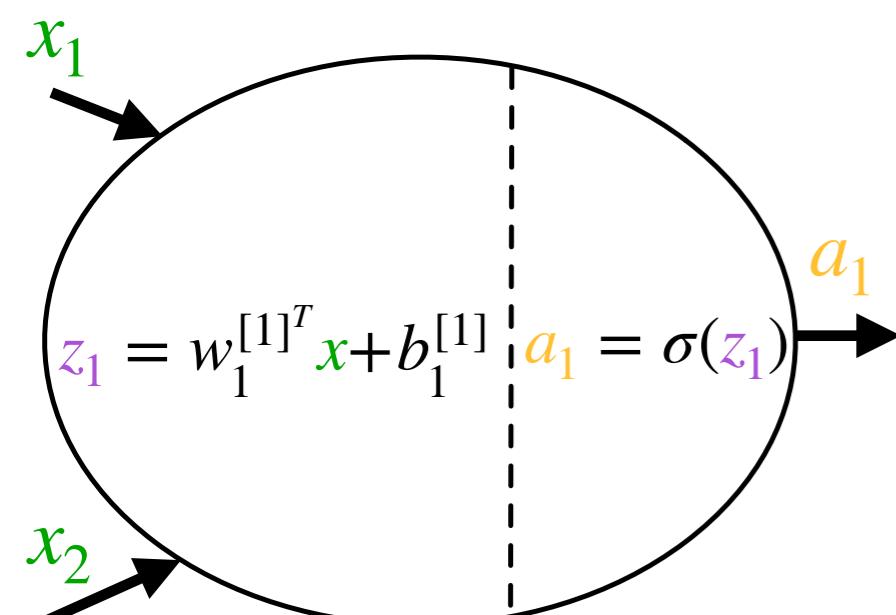
**Intuition:** un neurone est une régression logistique avec ses paramètres

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

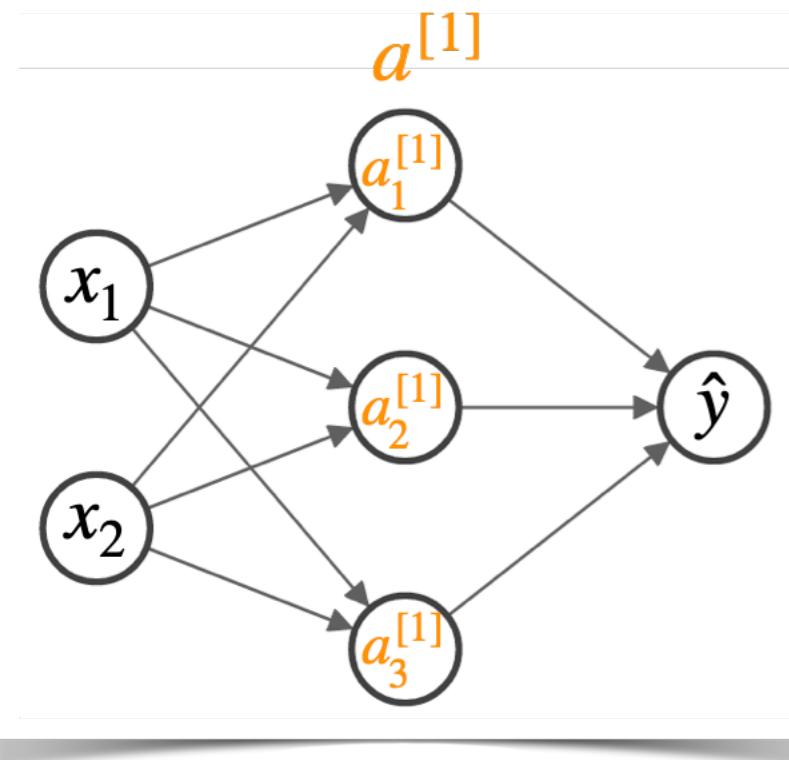
$$a_2 = \sigma(w_2^{[1]T}x + b_2^{[1]})$$

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

**Note:**  $w_j^{[i]}$  est de dimension  $n^{[i-1]} \times 1$  (nombre de flèches entrantes)



# Réseau de neurones: forme matricielle



**Notation matricielle:** regrouper toutes les valeurs en vecteurs et matrices

**Intérêt:** donne une expression compacte et plus efficace à calculer

$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$  vecteur des caractéristiques d'une instance spécifique

$$W^{[1]} = \begin{pmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ - & w_3^{[1]T} & - \end{pmatrix} \quad \text{Matrice des poids à la couche cachée}$$

$$b^{[1]} = \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} \quad \text{Vecteur des biais à la couche cachée}$$

**Etape de la combinaison linéaire:**  $z^{[1]} = W^{[1]}x + b^{[1]}$

$z_j^{[i]}$  : calcul intermédiaire du neurone  $j$  à la couche  $i$

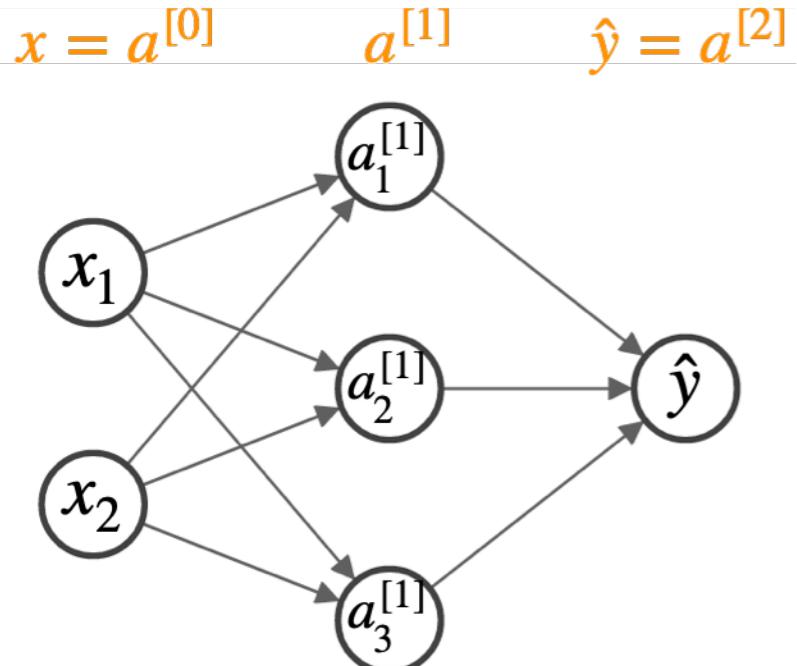
$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} = \begin{pmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ - & w_3^{[1]T} & - \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]T}x + b_1^{[1]} \\ w_2^{[1]T}x + b_2^{[1]} \\ w_3^{[1]T}x + b_3^{[1]} \end{pmatrix}$$

**Etape de l'application de la sigmoïde:**  $a^{[1]} = \sigma(z^{[1]})$

$a_j^{[i]}$  : sortie du neurone  $j$  à la couche  $i$  (suite de la notation précédente)

$$a^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} = \begin{pmatrix} \sigma(z_1^{[1]}) \\ \sigma(z_2^{[1]}) \\ \sigma(z_3^{[1]}) \end{pmatrix}$$

# Réseau de neurones: équations fondamentales



**Principe:** obtenir la prédition à partir des caractéristiques en entrée

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



$$\begin{aligned}z^{[1]} &= W^{[1]}a^{[0]} + b^{[1]} \\a^{[1]} &= \sigma(z^{[1]}) \\z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\a^{[2]} &= \sigma(z^{[2]})\end{aligned}$$

**Expression similaire:** notation  $a$  pour la couche d'entrée et de sortie

**Avantage:** donne une expression plus générique

Ces équations sont les équations fondamentales des réseaux de neurones

**Note:** appliquer ces équations revient à faire la passe en avant dans un réseau de neurones



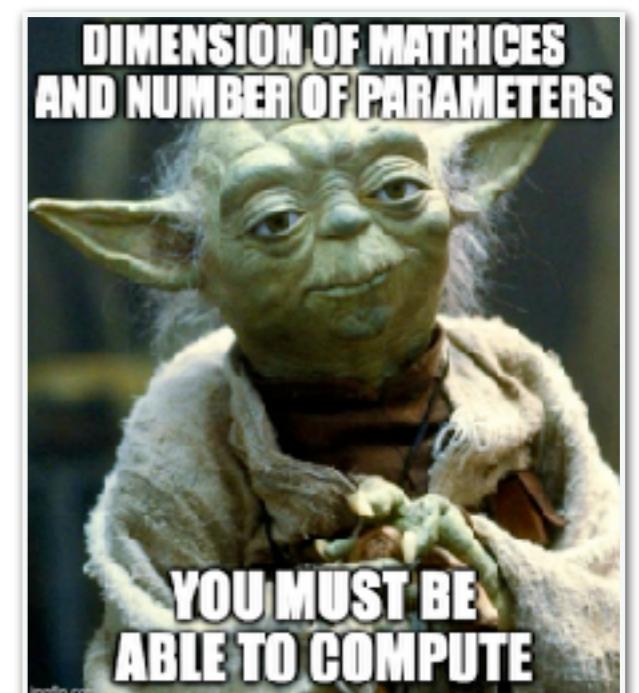
Combien de paramètres à apprendre a t-on dans ce réseau ?

**Principe:** calculer la taille de chaque matrice de paramètres, et sommer le tout

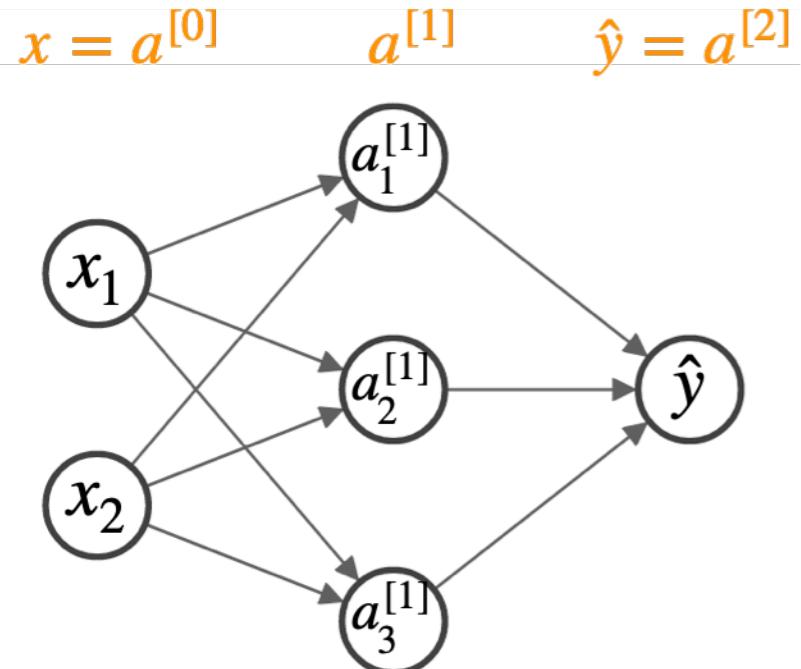
$$W^{[1]} : n^{[1]} \times n^{[0]} = 3 \times 2 \rightarrow 6 \quad b^{[1]} : n^{[1]} \times 1 = 3 \times 1 \rightarrow 3$$

$$W^{[2]} : n^{[2]} \times n^{[1]} = 1 \times 3 \rightarrow 3 \quad b^{[2]} : n^{[2]} \times 1 = 1 \times 1 \rightarrow 1$$

**Total :** 13 paramètres devant être entraînés



# Réseau de neurones : situation avec plusieurs données



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



$$\begin{aligned} z^{[1]} &= W^{[1]}a^{[0]} + b^{[1]} \\ a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \end{aligned}$$

**Limitation:** ces équations ne considèrent qu'une seule donnée en entrée

**Amélioration:** adapter ces équations à un ensemble de données

$$D : \left\{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)}) \right\}$$

Notation introduite dans le module précédent

Objectif: appliquer ces équations sur chaque donnée

$$\forall i \in \{1, \dots, m\}$$

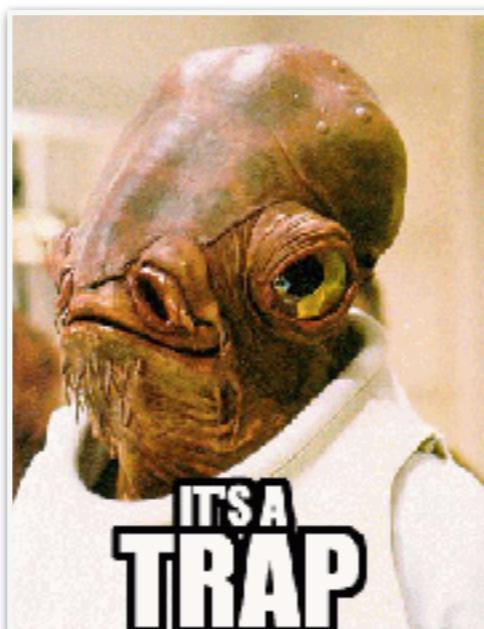
$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

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

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

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

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



La notation  $[i]$  réfère à une couche  $i$

La notation  $(i)$  réfère à une donnée  $i$

**Erreur fréquente:** confusion avec les indices

**Conseil:** comprenez ce que chaque indice signifie

# Réseau de neurones : calcul de la passe en avant itérative pour plusieurs données

```
forwardPassIterative( $D, W^{[1]}, b^{[1]}, \dots$ ) :
```

```
for  $i$  in 1 to  $m$  :
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

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

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

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

```
return  $a^{[2](1)}, \dots, a^{[2](m)}$ 
```

**Algorithme naïf:** prédire itérativement le résultat pour chaque donnée

?

Est-ce que c'est correct ?

Oui !

?

Est-ce que c'est efficace ?

Non ! Il y a moyen de faire mieux

?

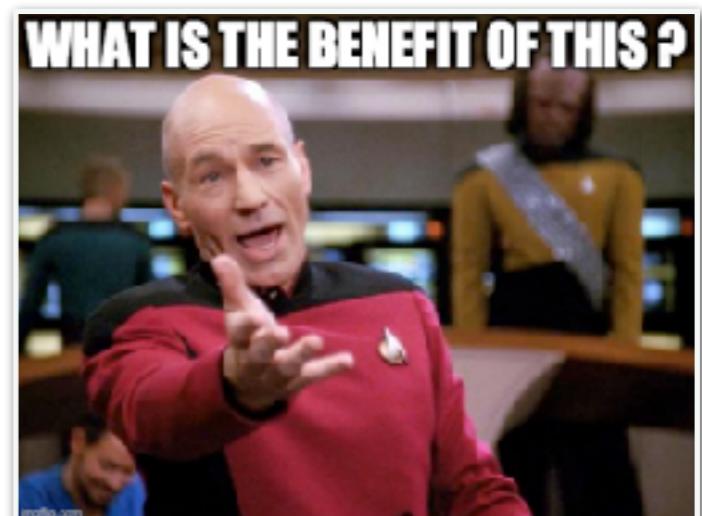
Avez-vous une autre alternative ?

**Idée:** on peut regrouper toutes les données dans une matrice

$$\begin{pmatrix} | \\ x^{(1)} \\ | \end{pmatrix}, \dots, \begin{pmatrix} | \\ x^{(m)} \\ | \end{pmatrix} \rightarrow \begin{pmatrix} | & \dots & | \\ x^{(1)} & \dots & x^{(m)} \\ | & \dots & | \end{pmatrix} = X$$

$n^{[0]} \times 1 \quad n^{[0]} \times 1 \quad n^{[0]} \times m$

#caractéristiques  $\times$  #données



**Intérêt:** permet la conception d'algorithmes plus efficaces, opérant sur des matrices

**Intérêt supplémentaire:** cela est d'autant plus vrai lorsqu'on utilise des GPUs

**Graphical processor unit (GPU):** processeur spécialisé pour réaliser des tâches similaires en parallèle

**Forme matricielle:** représentation des données en matrice à des fins de performances

# Forme matricielle pour plusieurs prédictions

**Forme matricielle:** une colonne contient les informations spécifiques pour une instance

$$X = \begin{pmatrix} | & \dots & | \\ x^{(1)} & \dots & x^{(m)} \\ | & \dots & | \end{pmatrix}$$

Dimension :  $n^{[0]} \times m$

Dimension : #caractéristiques × #données

**Conséquence:** les valeurs  $z$ , et  $a$  de chaque neurone deviennent une matrice

$$Z^{[1]} = \begin{pmatrix} - & w_1^{[1]T} & - \\ - & \dots & - \\ - & w_{n^{[1]}}^{[1]T} & - \end{pmatrix} \begin{pmatrix} | & \dots & | \\ x^{(1)} & \dots & x^{(m)} \\ | & \dots & | \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ \dots \\ b_{n^{[1]}}^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]T} x^{(1)} + b_1^{[1]} & \dots & w_1^{[1]T} x^{(m)} + b_1^{[1]} \\ \dots & \dots & \dots \\ w_{n^{[1]}}^{[1]T} x^{(1)} + b_{n^{[1]}}^{[1]} & \dots & w_{n^{[1]}}^{[1]T} x^{(m)} + b_{n^{[1]}}^{[1]} \end{pmatrix}$$

$$= \begin{pmatrix} | & \dots & | \\ z^{[1](1)} & \dots & z^{[1](m)} \\ | & \dots & | \end{pmatrix}$$

Dimension :  $n^{[1]} \times m$

Dimension : #neurones × #données

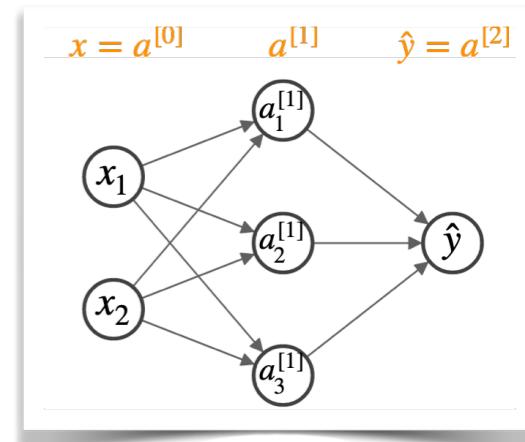
$$A^{[1]} = \begin{pmatrix} | & \dots & | \\ \sigma(z^{[1](1)}) & \dots & \sigma(z^{[1](m)}) \\ | & \dots & | \end{pmatrix}$$

Application de la sigmoïde à chaque valeur (*element-wise*)

$$= \begin{pmatrix} | & \dots & | \\ a^{[1](1)} & \dots & a^{[1](m)} \\ | & \dots & | \end{pmatrix}$$

Dimension :  $n^{[1]} \times m$

Dimension : #neurones × #données

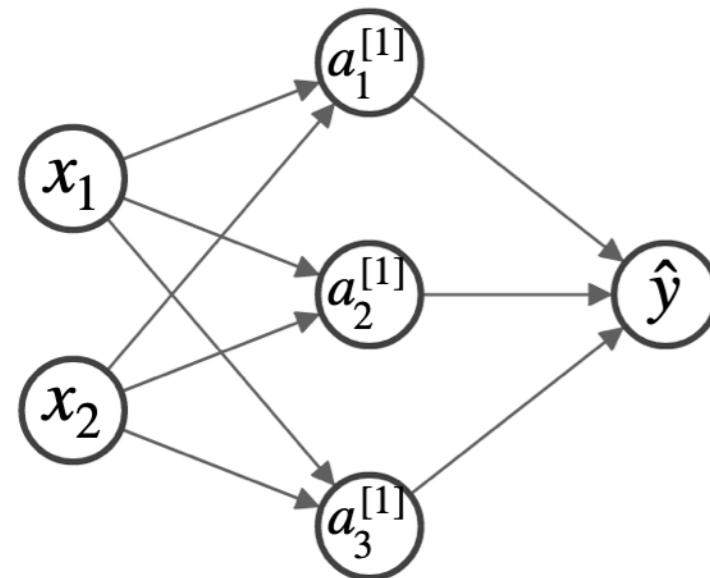


**Broadcasting de  $b$**

Ces opérations reviennent à calculer la passe en avant en même temps pour un ensemble de données

# Equations fondamentales sous forme matricielle

$$x = a^{[0]} \quad a^{[1]} \quad \hat{y} = a^{[2]}$$



**Illustration: passage à une forme matricielle (équations de droite)**

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



$$\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \\ \hat{Y} &= A^{[2]} \end{aligned}$$

**Convention:** on utilise des lettres majuscules pour les matrices

**Application:** problème de l'étudiant avec deux caractéristiques

# heures d'étude :  $x_1$

# heures de repos :  $x_2$

**Données:** quatre données d'entraînement

$$D : \left\{ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), (x^{(4)}, y^{(4)}) \right\}$$

**Représentation:** forme matricielle

$$X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{pmatrix}$$

**?** Que représente  $x_2^{(3)}$  ?

**Sortie de la première couche:**

$$A^{[1]} = \begin{pmatrix} a_1^{[1](1)} & a_1^{[1](2)} & a_1^{[1](3)} & a_1^{[1](4)} \\ a_2^{[1](1)} & a_2^{[1](2)} & a_2^{[1](3)} & a_2^{[1](4)} \\ a_3^{[1](1)} & a_3^{[1](2)} & a_3^{[1](3)} & a_3^{[1](4)} \end{pmatrix}$$

**?** Que représente  $a_3^{[1](2)}$  ?

**Réponse:**  
activation du 3e neurone  
de la couche 1 pour le  
2e étudiant

**Prédiction finale:** une valeur pour chaque donnée

$$A^{[2]} = (a_1^{[2](1)} \ a_1^{[2](2)} \ a_1^{[2](3)} \ a_1^{[2](4)})$$

$$\hat{Y} = (\hat{y}^{(1)} \ \hat{y}^{(2)} \ \hat{y}^{(3)} \ \hat{y}^{(4)})$$

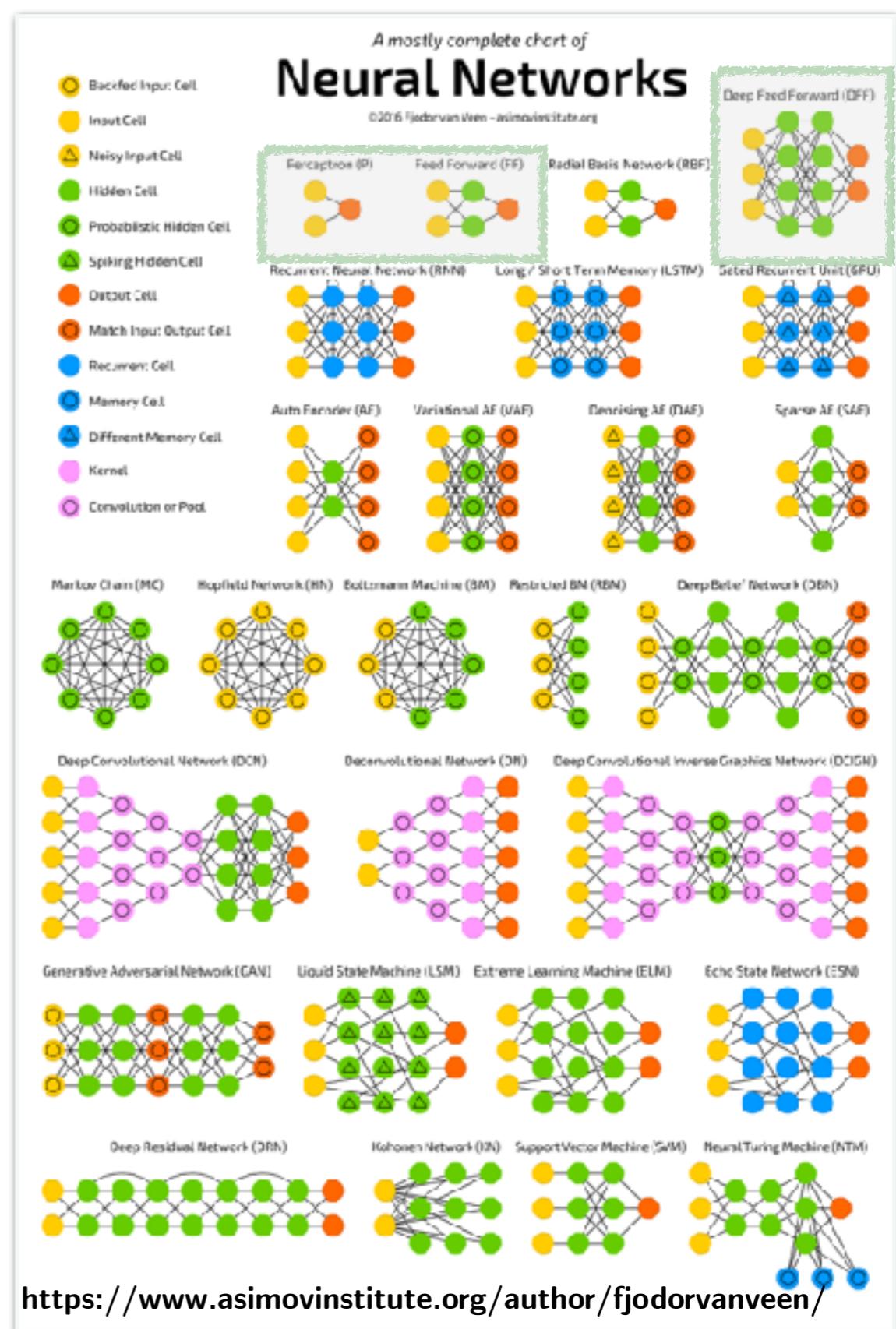
# Table des matières

## Réseaux de neurones

-  1. Concepts principaux des réseaux de neurones
-  2. Extension de la régression logistique
-  3. Notations standards des réseaux de neurones
- 4. Exécution de la *backpropagation* du gradient
- 5. Principes des fonctions d'activation
- 6. Initialisation des paramètres

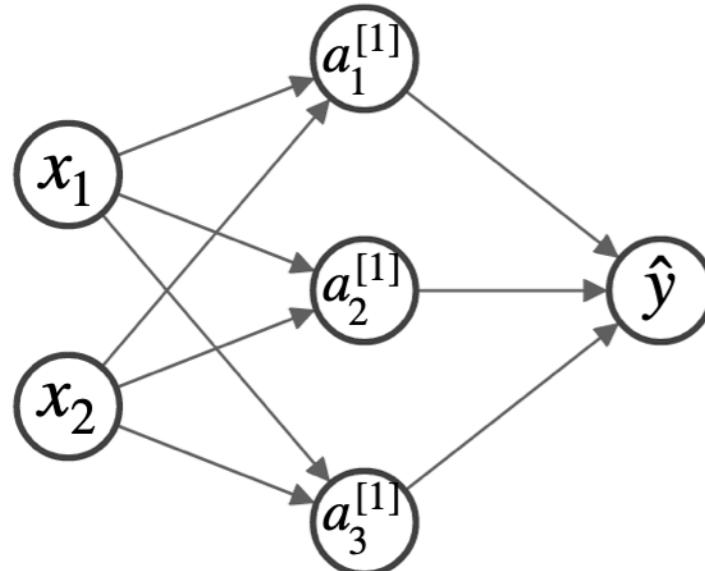
## Apprentissage profond

- 1. Limitation des petits réseaux de neurones
- 2. Principes de l'apprentissage profond
- 3. Profondeur d'un réseau de neurones
- 4. Introduction aux hyper-paramètres
- 5. Conseils pratiques



# Apprentissage supervisé par un réseau de neurones

$$x = a^{[0]} \quad a^{[1]} \quad \hat{y} = a^{[2]}$$



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

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

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

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

$$\hat{Y} = A^{[2]}$$

$$W^{[1]} : n^{[1]} \times n^{[0]} = 3 \times 2$$

$$b^{[1]} : n^{[1]} \times 1 = 3 \times 1$$

$$W^{[2]} : n^{[2]} \times n^{[1]} = 1 \times 3$$

$$b^{[2]} : n^{[2]} \times 1 = 1 \times 1$$

13 paramètres à déterminer

?

Comment déterminer la valeur de ces paramètres ?

**Objectif:** déterminer les paramètres qui minimisent une fonction de coût liée à la tâche de prédiction

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

Expression générique, qui ne diffère que par la fonction d'écart choisie

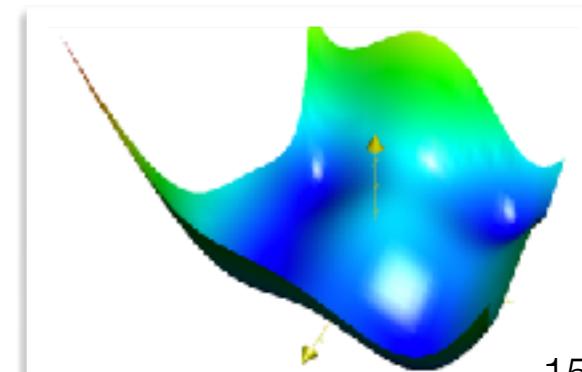
**Principe:** identique à une régression logistique, sauf que la fonction contient plus de paramètres

**Bonne nouvelle:** tout le réseau est différentiable si la fonction d'écart l'est également

**Conséquence:** la fonction peut être minimisée par une descente de gradient

**Mauvaise nouvelle:** la fonction de coût n'est plus convexe

**Conséquence:** on risque de converger dans un minimum local



# Descente de gradient dans un réseau de neurones

gradientDescent( $D, \alpha$ ) :

$W^{[2]}, b^{[2]}, W^{[1]}, b^{[1]} = \text{initializeRandomly}()$

repeat until convergence :

compute  $\hat{Y}$

compute  $J(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$

compute  $\frac{\partial J}{\partial W^{[2]}}, \frac{\partial J}{\partial b^{[2]}}, \frac{\partial J}{\partial W^{[1]}}, \frac{\partial J}{\partial b^{[1]}}$

$W^{[2]} = W^{[2]} - \alpha \frac{\partial J}{\partial W^{[2]}}$

$b^{[2]} = b^{[2]} - \alpha \frac{\partial J}{\partial b^{[2]}}$

$W^{[1]} = W^{[1]} - \alpha \frac{\partial J}{\partial W^{[1]}}$

$b^{[1]} = b^{[1]} - \alpha \frac{\partial J}{\partial b^{[1]}}$

return  $W^{[1]}, W^{[2]}, b^{[1]}, b^{[2]}$

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

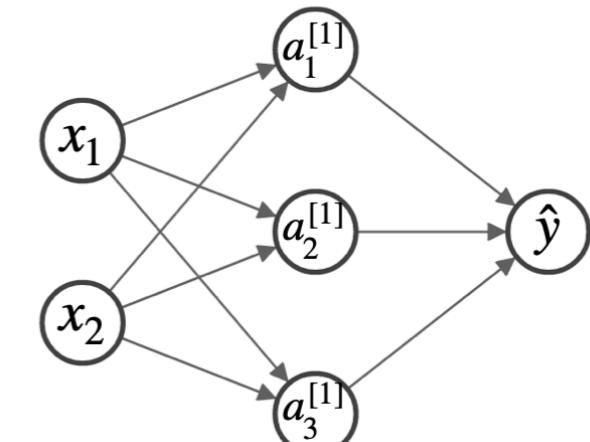
$$A^{[1]} = \sigma(Z^{[1]})$$

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

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

$$\hat{Y} = A^{[2]}$$

$$x = a^{[0]} \quad a^{[1]} \quad \hat{y} = a^{[2]}$$



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

$$\frac{\partial J}{\partial b^{[1]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial b^{[1]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial b^{[1]}} \right)$$

$$\frac{\partial J}{\partial b^{[2]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial b^{[2]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial b^{[2]}} \right)$$

$$\frac{\partial J}{\partial W^{[1]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial W^{[1]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial W^{[1]}} \right)$$

$$\frac{\partial J}{\partial W^{[2]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial W^{[2]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial W^{[2]}} \right)$$



Dérivation: [www.efavdb.com/backpropagation-in-neural-networks](http://www.efavdb.com/backpropagation-in-neural-networks)

Calcul du gradient: règle du chaînage sur un graphe de dépendances

# Conséquence de la non-convexité



Quelle est la conséquence majeure de la non-convexité de la fonction de coût ?

**Réalité:** on risque de converger dans un minimum local

**Conséquence:** on doit maintenant faire attention à la façon dont on entraîne le réseau

**Objectif:** faciliter l'apprentissage afin de converger vers un bon minimum local

**Analogie:** utilisation de métaheuristiques dans une recherche locale



Quelles sont les actions que nous pouvons prendre pour faciliter l'apprentissage ?



- (1) Remplacer la sigmoïde par une autre fonction (tanh, ReLU, LeakyReLU, etc.)
- (2) Initialiser adéquatement les paramètres (bonne solution initiale)
- (3) Utiliser un autre algorithme que la descente de gradient (SGD, Adam, etc.)
- (4) Rajouter des opérations complémentaires dans le réseau (normalisation, BatchNorm)
- (5) Sélectionner adéquatement certains hyper-paramètres (taux d'apprentissage, etc.)

Il y a encore continuellement des nouvelles idées qui sont introduites par la communauté scientifique

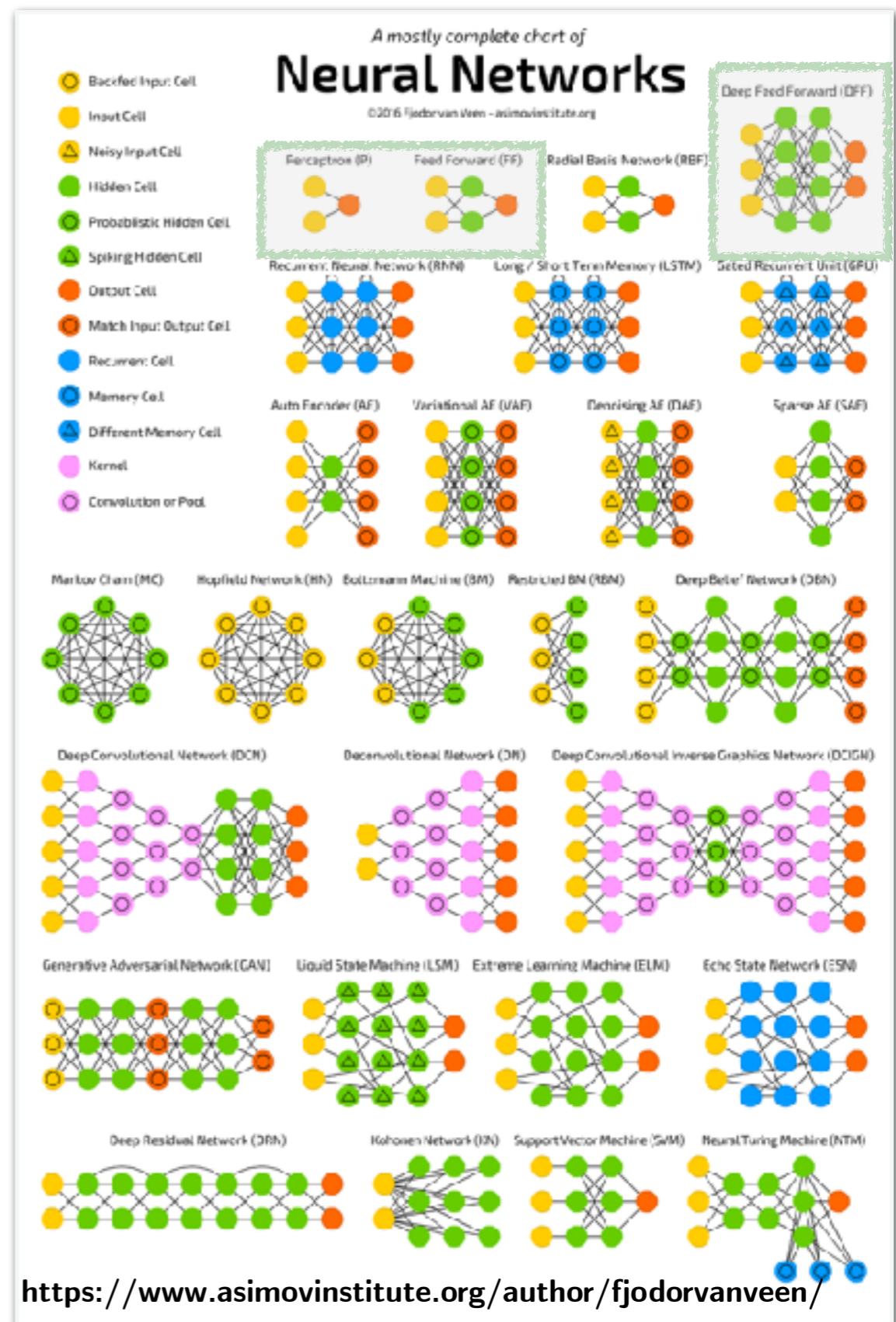
# Table des matières

## Réseaux de neurones

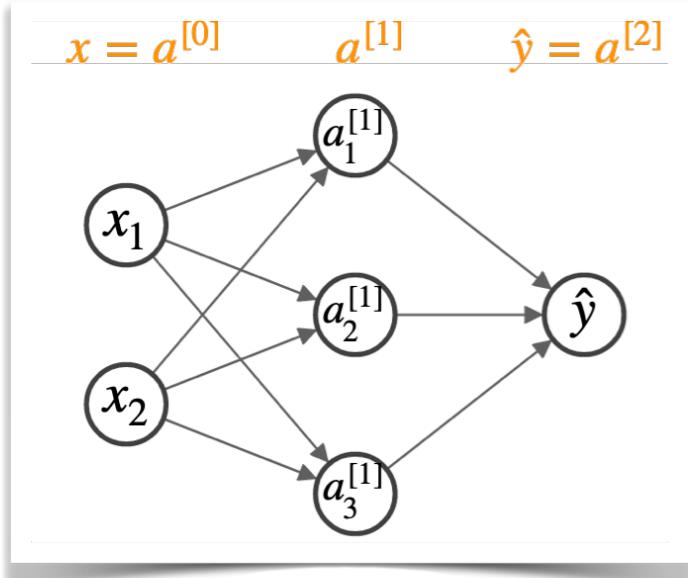
-  1. Concepts principaux des réseaux de neurones
-  2. Extension de la régression logistique
-  3. Notations standards des réseaux de neurones
-  4. Exécution de la *backpropagation* du gradient
- 5. Principes des fonctions d'activation
- 6. Initialisation des paramètres

## Apprentissage profond

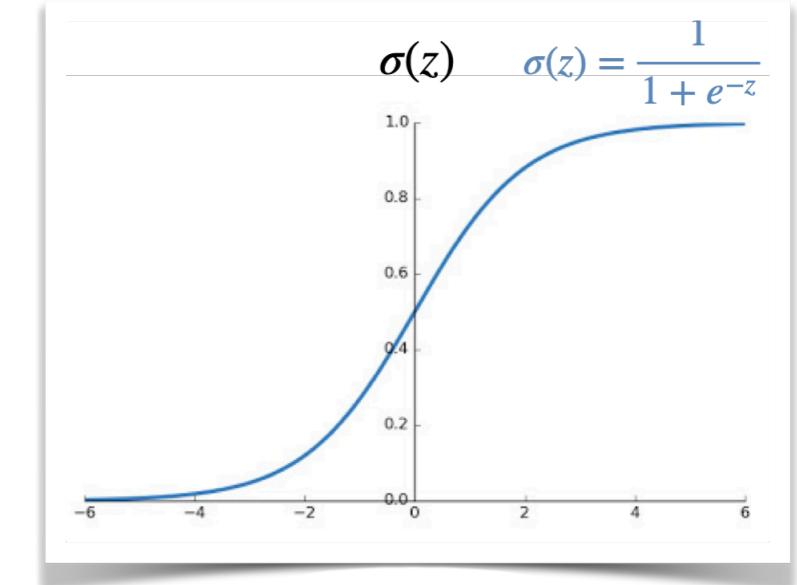
- 1. Limitation des petits réseaux de neurones
- 2. Principes de l'apprentissage profond
- 3. Profondeur d'un réseau de neurones
- 4. Introduction aux hyper-paramètres
- 5. Conseils pratiques



# Limitations de la sigmoïde



$$\begin{aligned}Z^{[1]} &= W^{[1]}X + b^{[1]} \\A^{[1]} &= \sigma(Z^{[1]}) \\Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\A^{[2]} &= \sigma(Z^{[2]}) \\\hat{Y} &= A^{[2]}\end{aligned}$$



**Situation actuelle:** une fonction sigmoïde est appliquée à la sortie de chaque neurone



**Objectif initial:** obtenir une valeur entre 0 et 1 (prédiction d'une probabilité)

**Pertinence du choix:** correct pour le neurone de la dernière couche

**Limitation:** ce choix est non justifié pour les neurones intermédiaires

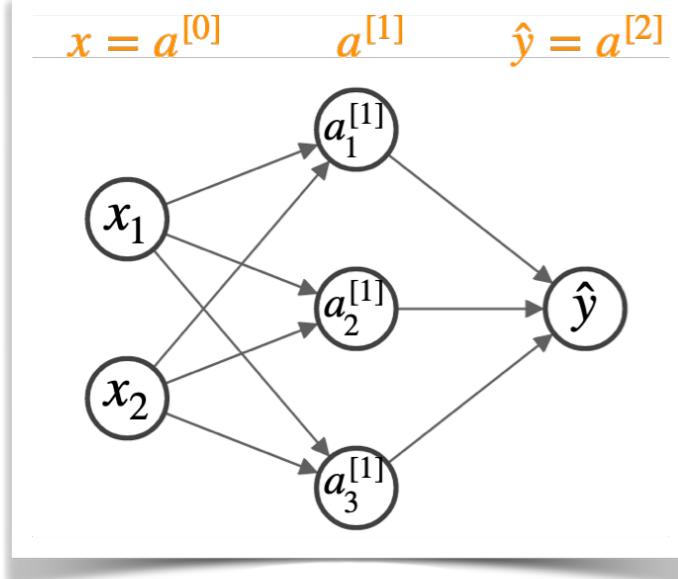
**Pire encore:** ce choix amène à des piétres performances

?

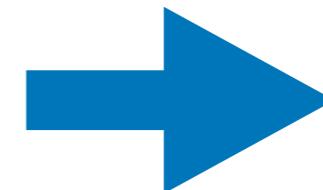
A t-on d'autres alternatives ?

 **Fonction d'activation**  
Transformation définissant la valeur de sortie un neurone après la combinaison des entrées

# Fonction d'activation



$$\begin{aligned}Z^{[1]} &= W^{[1]}X + b^{[1]} \\A^{[1]} &= \sigma(Z^{[1]}) \\Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\A^{[2]} &= \sigma(Z^{[2]}) \\\hat{Y} &= A^{[2]}\end{aligned}$$



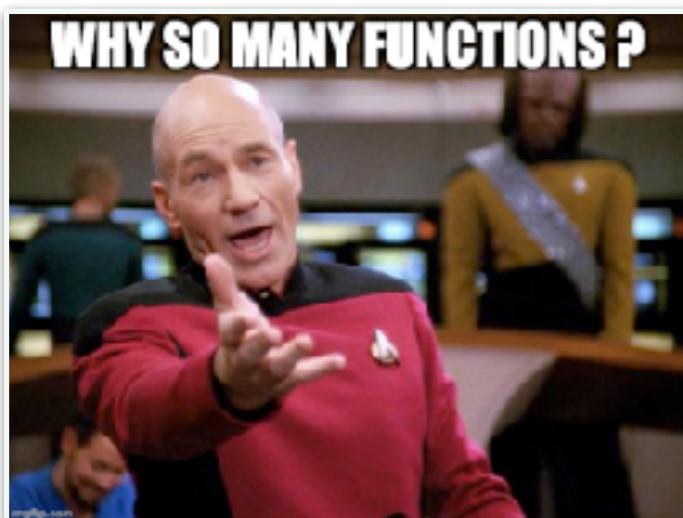
$$\begin{aligned}Z^{[1]} &= W^{[1]}X + b^{[1]} \\A^{[1]} &= g^{[1]}(Z^{[1]}) \\Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\A^{[2]} &= g^{[2]}(Z^{[2]}) \\\hat{Y} &= A^{[2]}\end{aligned}$$

**Fonction sigmoide:** fonction d'activation possible, mais il en existe plein d'autres

**Idée:** considérer une autre fonction que la sigmoide comme fonction activation

**Contrainte 1:** la fonction activation doit être **non-linéaire** (explication à venir)

**Contrainte 2:** la fonction doit être **principalement différentiable** (calcul de la descente de gradient)



Il existe une multitude de fonctions d'activation dans la littérature

Tangente hyperbolique (**tanh**)

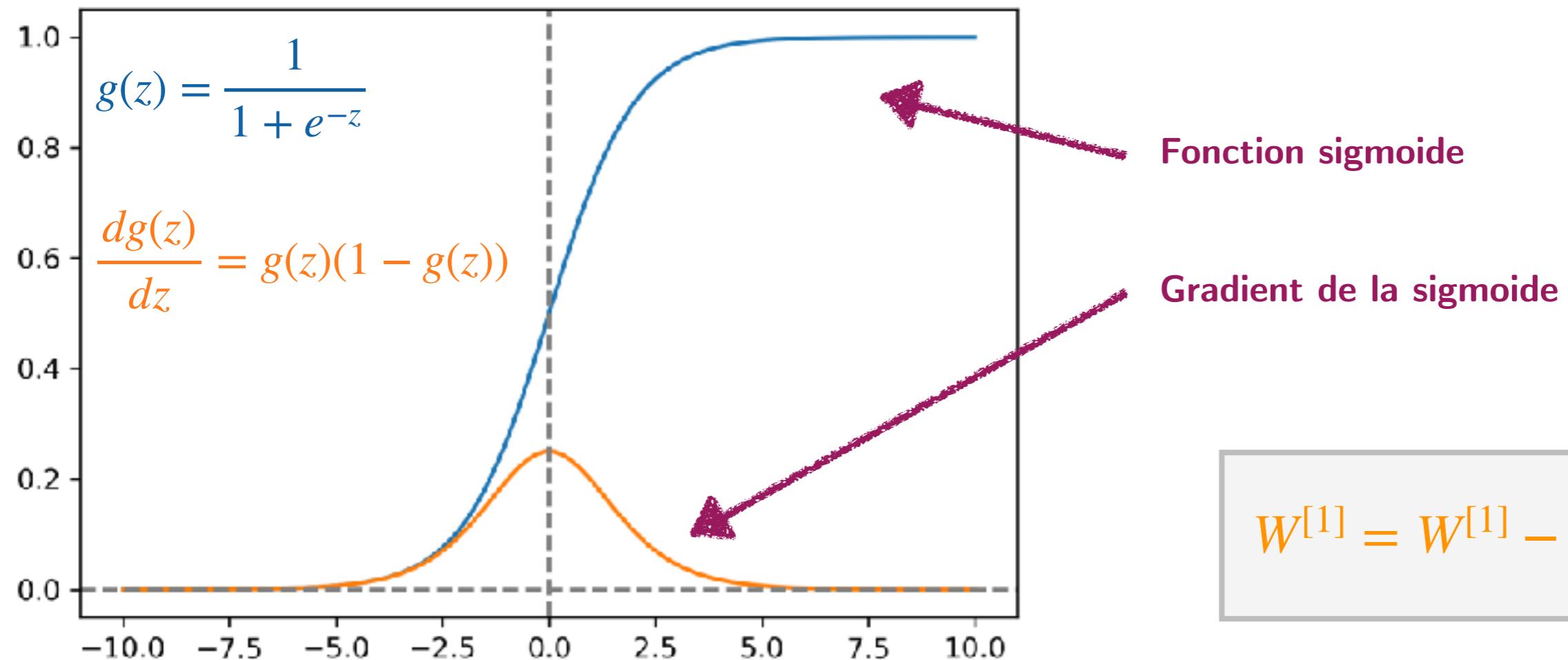
Rectified linear unit (**ReLU**)

Variantes du ReLU (Leaky ReLU, Exponential ReLU, etc.)

Et plein d'autres... (**Maxout**, etc.)

**Raison principale:** chacune a ses forces et faiblesses ainsi que leur champ d'application

# Fonction d'activation: la sigmoide



**Fonction sigmoide:** compresse n'importe quelle valeur dans un intervalle entre 0 et 1

**Application:** neurone de la dernière couche pour une classification binaire (prédiction d'une probabilité)

**Difficulté:** la fonction sature et le gradient quasi-nul pour des grosses valeurs (ralenti l'apprentissage)

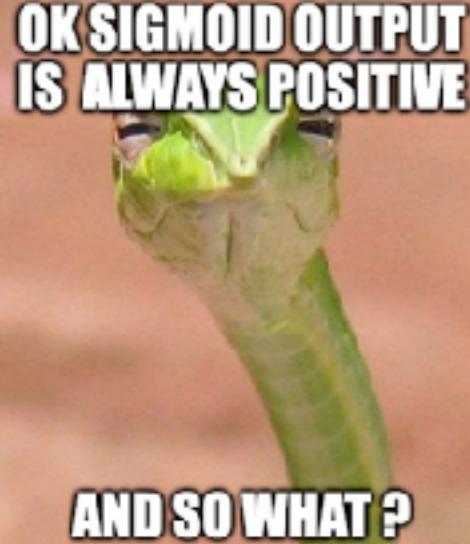
**Difficulté:** implique des opérations coûteuses (calcul d'exponentielles)

**Difficulté:** Les valeurs de sortie sont toujours positives

**Conséquence:** la descente de gradient aura tendance à zigzaguer (ralenti l'apprentissage)

**Recommendation:** utilisation non conseillée, sauf pour la sortie d'une classification binaire

# Valeurs positives de la sigmoide



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

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

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

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

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

Mise en situation: réseau à deux neurones cachés

Regardons l'impact sur le neurone de la dernière couche

$$z^{[2]} = w_1^{[2]}a_1^{[1]} + w_2^{[2]}a_2^{[1]} + b^{[2]}$$

Avec  $a_1^{[1]}$  et  $a_2^{[1]}$  positifs (sortie d'une sigmoïde)

Etape 1: analysons les dérivées partielles de ce neurone pour les différents poids ( $w$ )

$$\frac{\partial z^{[2]}}{\partial w_1^{[2]}} = a_1^{[1]}$$

$$\frac{\partial z^{[2]}}{\partial w_2^{[2]}} = a_2^{[1]}$$

A ce stade, on sait que ces 2 valeurs sont positives

Etape 2: analysons les dérivées partielles de la fonction d'écart

$$\frac{\partial L(\hat{y}, y)}{\partial w_1^{[2]}} = \frac{\partial L(\hat{y}, y)}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w_1^{[2]}} = a_1^{[1]} \frac{\partial L(\hat{y}, y)}{\partial z^{[2]}}$$

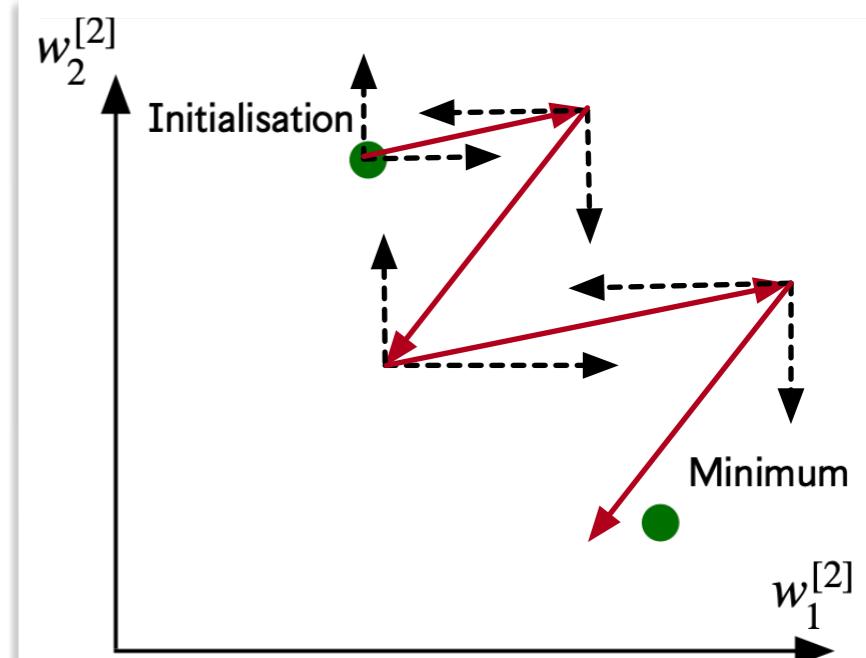
Valeur positive

Même valeur

$$w_1^{[2]} = w_1^{[2]} - \alpha \frac{\partial J}{\partial w_1^{[2]}}$$

Même signe

$$w_2^{[2]} = w_2^{[2]} - \alpha \frac{\partial J}{\partial w_2^{[2]}}$$

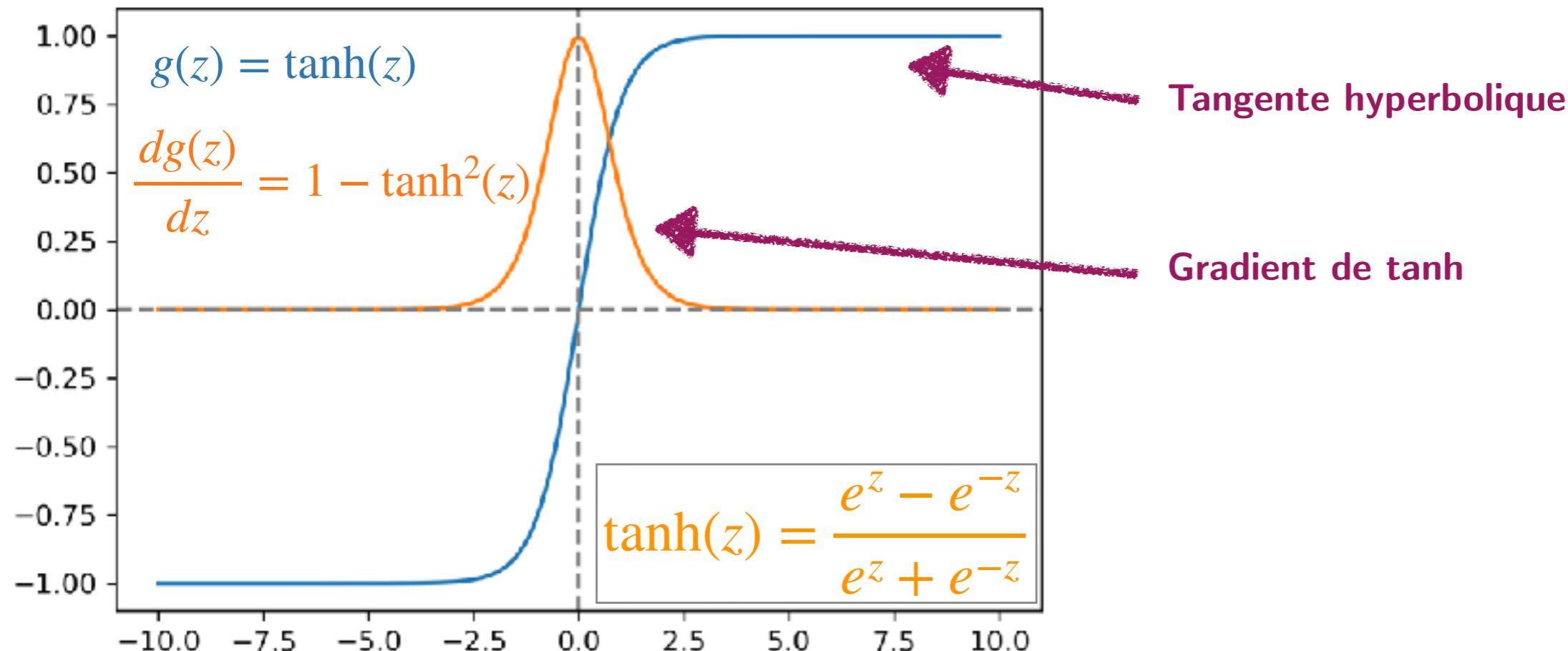


Observation: les deux dérivées partielles ont le même signe

Implication: on a le même sens de descente pour ces deux paramètres

Conséquence: donne un zigzag visuel dans la descente de gradient, ce qui ralenti l'apprentissage

# Fonction d'activation: tangente hyperbolique (tanh)



**Tangente hyperbolique:** compresse n'importe quelle valeur dans un intervalle entre -1 et 1

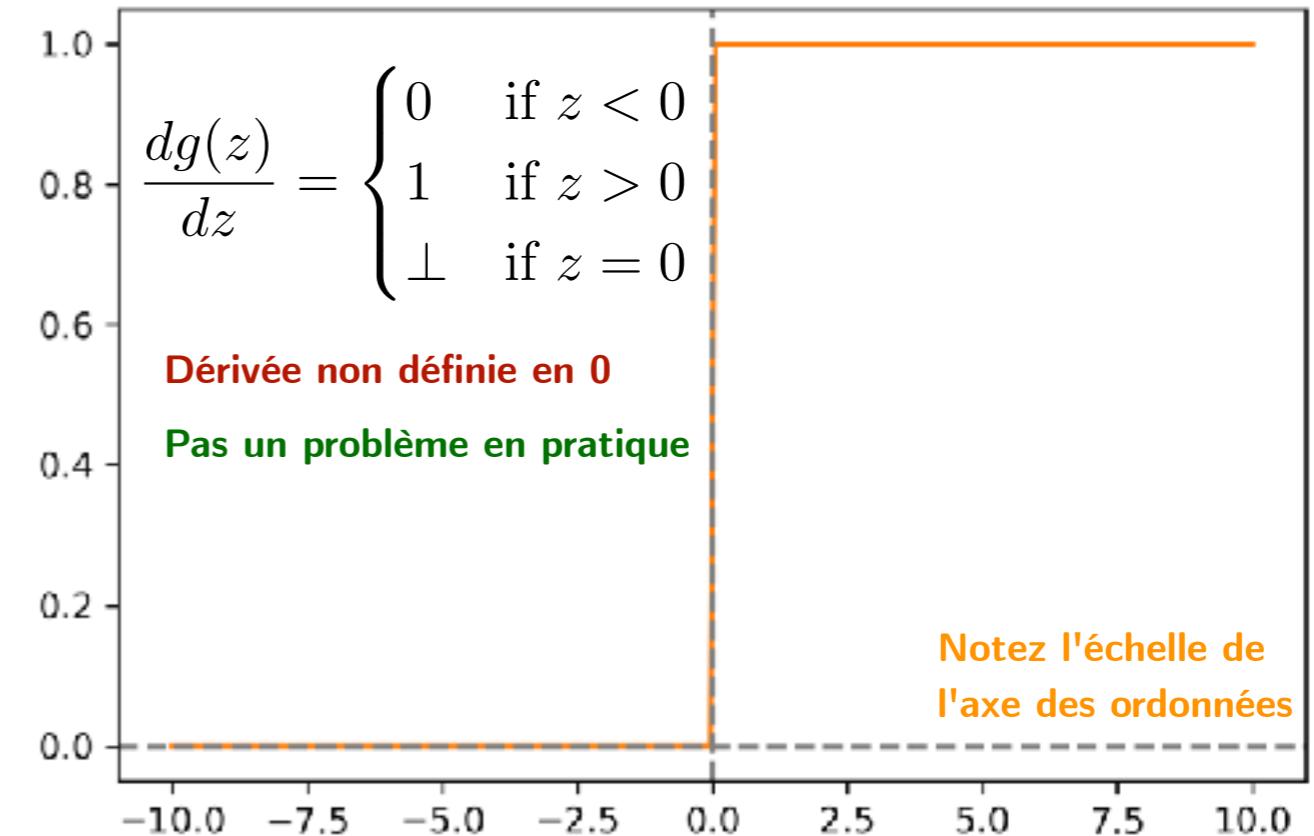
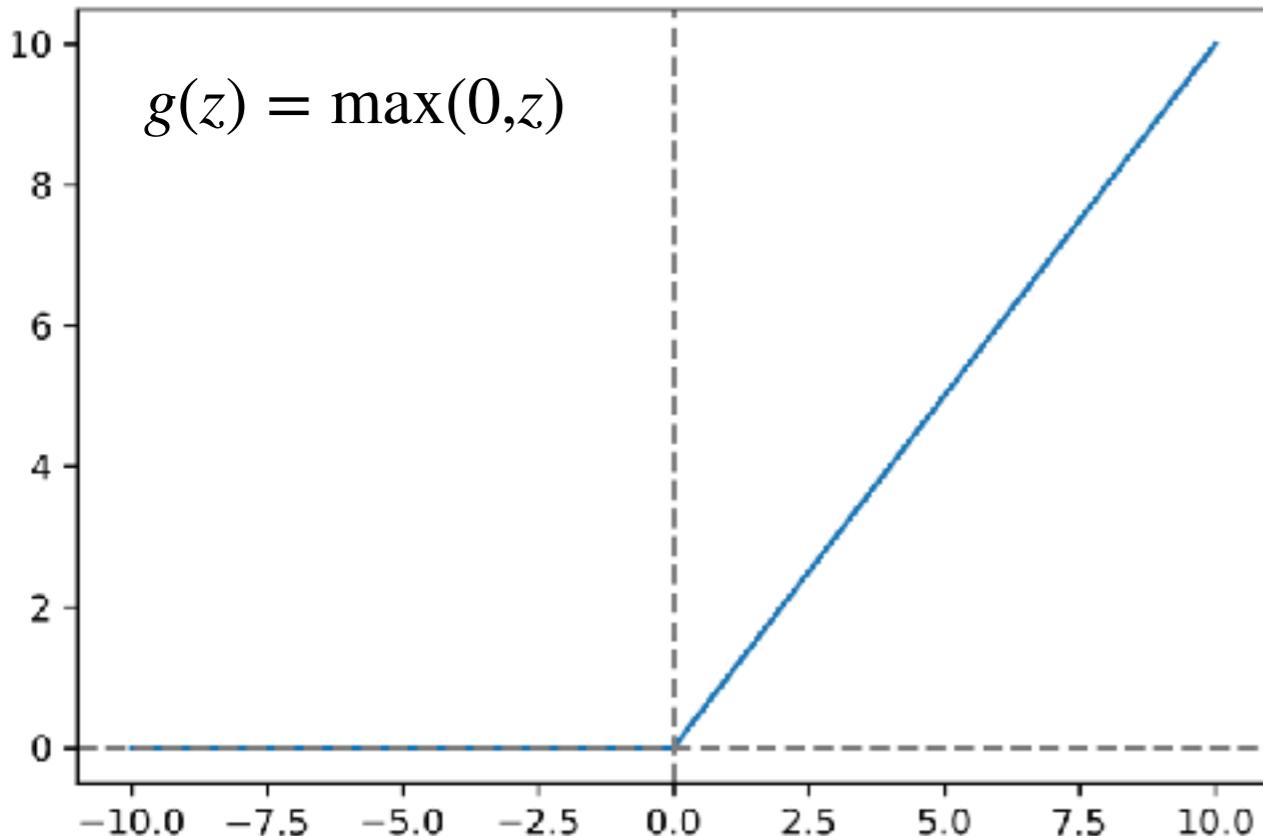
**Avantage:** les valeurs de la fonction sont centrées en 0 (mitige l'effet de zigzag)

**Difficulté:** la fonction sature et le gradient quasi-nul pour des grosses valeurs (ralenti l'apprentissage)

**Difficulté:** implique des opérations coûteuses (calcul d'exponentielles)

**Recommendation:** fonction quasi-toujours préférable à la fonction sigmoïde pour une couche cachée

# Fonction d'activation: *Rectified linear unit* (ReLU)



**ReLU:** fonction linéaire par morceaux qui met toutes les valeurs négatives à 0

**Avantage:** ne sature pas dans la région positive (convergence accélérée)

**Avantage:** opérations peu coûteuses

**Difficulté:** Les valeurs de sortie ne sont jamais négatives (effet de zigzag)

**Difficulté:** une descente trop forte peut amener une valeur dans la zone négative (**neurone mort**)

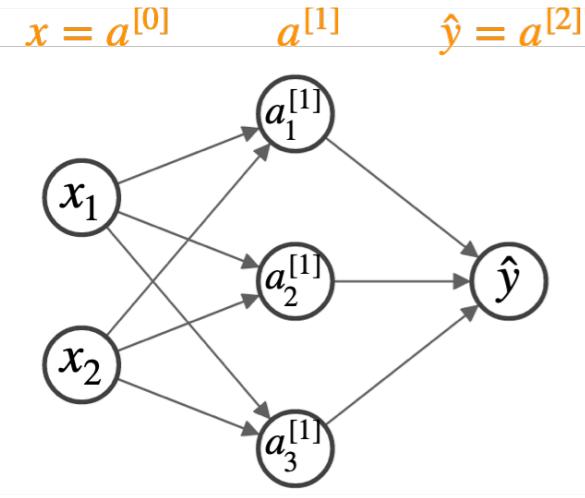
**Cause:** le gradient est nul dans la zone négative (rien ne sera appris pour ce neurone)

**Mitigation possible:** prendre un taux d'apprentissage pas trop grand

$$W^{[1]} = W^{[1]} - \alpha \frac{\partial J}{\partial W^{[1]}}$$

**Recommendation:** très bon choix de fonction d'activation (recommandé pour vos premiers modèles)

# Non-linéarité des fonctions d'activation



On souhaite avoir une activation non-linéaire. Pourquoi est-ce important ?

**Exemple:** considérons une activation linéaire pour les neurones cachés

**Fonction identité:**  $g(z) = z$

**Analyse:** regardons l'impact dans un réseau dédié à une prédiction de probabilité

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

$$A^{[1]} = Z^{[1]}$$

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

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

$$\hat{Y} = A^{[2]}$$

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

~~$$A^{[1]} = Z^{[1]}$$~~

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

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

$$\hat{Y} = A^{[2]}$$

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

~~$$A^{[1]} = Z^{[1]}$$~~

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

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

$$\hat{Y} = A^{[2]}$$

**Dimension de  $W^{[2]}W^{[1]}$ :**  $(1 \times 3) \cdot (3 \times 2) = 1 \times 2$

**Dimension de  $W^{[2]}b^{[1]} + b^{[2]}$ :**  $(1 \times 3) \cdot (3 \times 1) + (1 \times 1) = 1 \times 1$

$$Z^{[2]} = W'X + b'$$

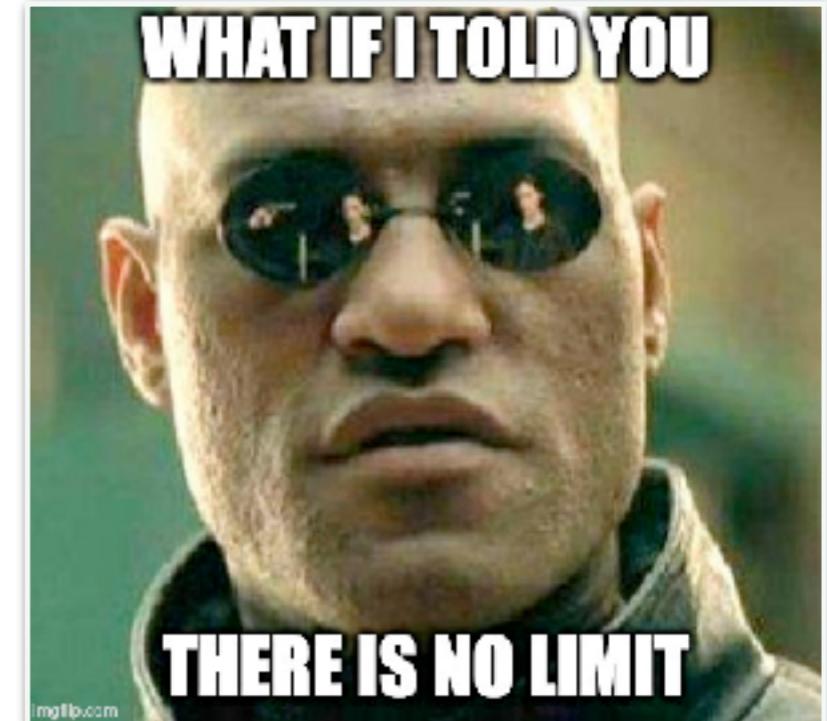
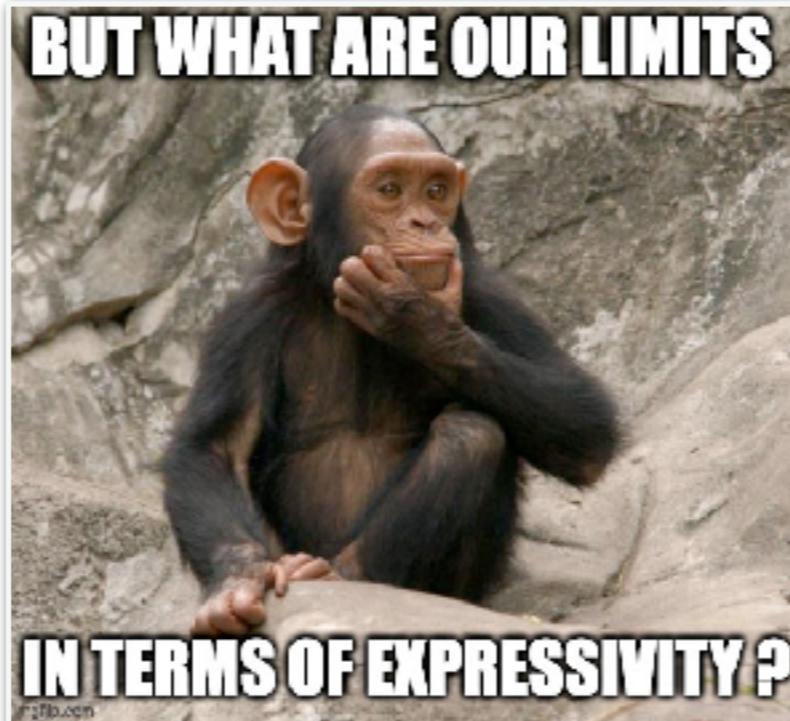
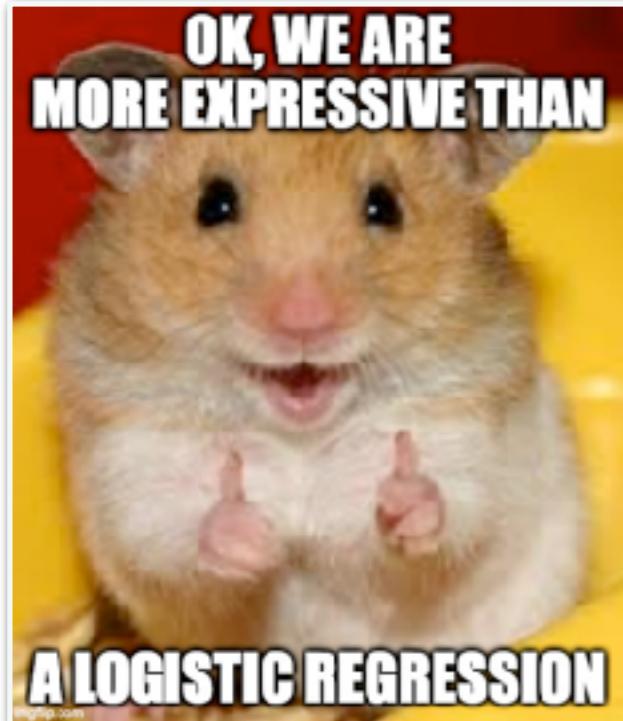
$$\hat{Y} = \sigma(Z^{[2]})$$

**Observation:** le réseau s'effondre en une combinaison linéaire des entrées  $X$

**Cause:** la composition de deux fonctions linéaires est aussi linéaire

Sans non-linéarité, le modèle se réduit à régression logistique

# Expressivité des réseaux de neurones



## Universalité des réseaux de neurones

Un réseau de neurones avec des fonctions d'activation non-linéaire peut théoriquement approximer avec une précision arbitraire n'importe quelle fonction réelle

Résultat théorique important: justifie le pouvoir expressif des réseaux de neurones (avec des non-linéarités)

Terminologie: on dit également qu'un réseau de neurones est un **approximateur universel**

Note: cela est également vrai avec une activation sigmoïde et seulement une couche cachée

Difficulté pratique: le réseau peut-être difficile à entraîner (d'où les mécanismes supplémentaires)

Difficulté pratique: ce résultat peut nécessiter un nombre immense de neurones

# Fonction d'activation de la dernière couche

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

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

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

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

$$\hat{Y} = A^{[2]}$$

?

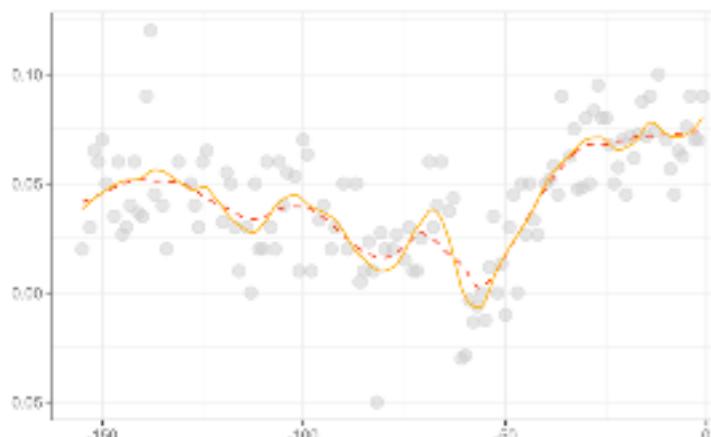
Peut-on utiliser la même fonction d'activation pour tous les neurones ?

En très grande majorité: oui, vous pouvez utiliser la même fonction partout

Exception: l'activation de la dernière couche a une importance particulière

Objectif de la dernière couche: définir le type de prédiction à obtenir

Règle d'or: la dernière activation doit retourner une valeur cohérente avec votre tâche de prédiction



Tâche 1: régression

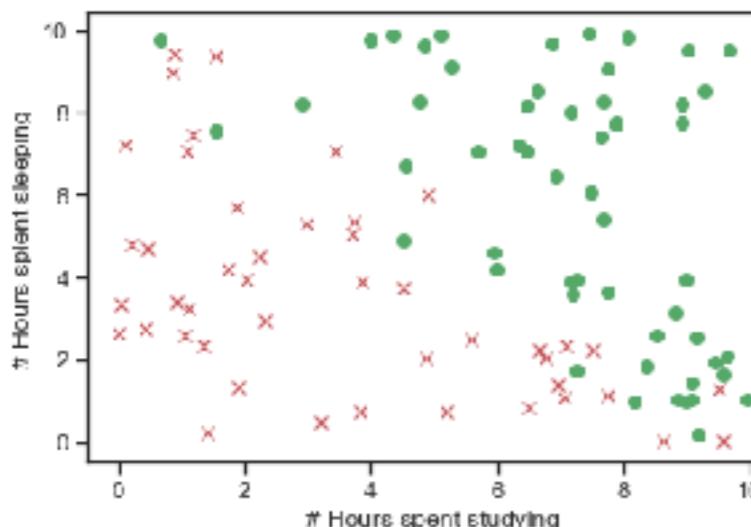
Objectif: obtenir une valeur réelle

Choix adapté: fonction identité (linéaire)

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

Fonction de perte liée: erreur quadratique

Note: il s'agit d'un des rares cas où vous devez utiliser une activation linéaire



Tâche 2: classification binaire

Objectif: obtenir une probabilité de succès (valeur entre 0 et 1)

Choix adapté: fonction sigmoïde

$$g^{[2]}(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Fonction de perte liée: entropie croisée binaire

# Fonction d'activation de la dernière couche



Chat  
Chien  
Oiseau



**Tâche 3: classification multi-classe**

**Objectif:** effectuer une prédiction entre plusieurs classes d'objets

**Extension naturelle de la classification binaire**



Comment effectuer une prédiction multi-classe ?

**Idée:** prédire une probabilité d'appartenance à chaque classe (de façon similaire au cas binaire)

**Etape initiale:** supposons que l'on ait  $C$  classes de prédiction

**Vraie valeur ( $y$ ):** correspond à une catégorie  $c \in C$



Chat

$$y = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Chat  
Chien  
Oiseau

**Encodage possible:** un vecteur de taille  $C \times 1$  avec un 1 à la bonne classe et 0 sinon (one-hot encoding)

**Objectif:** prédire la probabilité d'appartenance à chaque classe

**Prédiction:** correspond à un vecteur  $\hat{y} \in [0,1]^C$



Chat: 70%  
Chien: 26%  
Oiseau: 4%

$$\hat{y} = \begin{pmatrix} 0.70 \\ 0.26 \\ 0.04 \end{pmatrix}$$

**Remarque:** la somme des éléments de ce vecteur doit valoir 1 (ce sont des probabilités)



Comment obtenir un vecteur satisfaisant cette contrainte ?

# Fonction d'activation softmax



## Fonction softmax

Fonction qui transforme un vecteur de  $C$  réels en entrée, en un vecteur de  $C$  probabilités

Softmax :  $\mathbb{R}^C \rightarrow [0,1]^C$  (avec la somme des outputs qui vaut à 1)

$$\hat{y}_j = \text{Softmax}(z_j) = \frac{e^{z_j}}{\sum_{c=1}^C e^{z_c}}$$



**Exemple:**  $z = \begin{pmatrix} 5 \\ 4 \\ 2 \end{pmatrix} \rightarrow \hat{y} = \begin{pmatrix} \frac{e^5}{e^5 + e^4 + e^2} \\ \frac{e^4}{e^5 + e^4 + e^2} \\ \frac{e^2}{e^5 + e^4 + e^2} \end{pmatrix} = \begin{pmatrix} 0.70 \\ 0.26 \\ 0.04 \end{pmatrix}$

**Intérêt:** permet de réaliser des classifications avec plusieurs classes

**Utilisation:** à la dernière couche d'un réseau de neurones pour une tâche de classification multi-classe

**Impact sur le réseau:** la dernière couche aura un nombre de neurones qui équivaut au nombre de classes

**Intuition:** chaque neurone de la dernière couche va prédire une probabilité propre à la classe

**Classification finale:** effectuer la prédiction en choisissant la classe ayant la plus haute probabilité

# Fonction d'écart pour une classification multi-classe



Comment caractériser la qualité d'une classification multi-classe ?

Encore une fois, cela revient à définir une fonction d'écart appropriée à cette tâche

**Classification binaire:** utilisation de l'entropie croisée binaire

**Classification multi-classe:** utilisation de l'entropie croisée

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



**Erreur de l'entropie croisée (*categorical cross entropy loss*)**

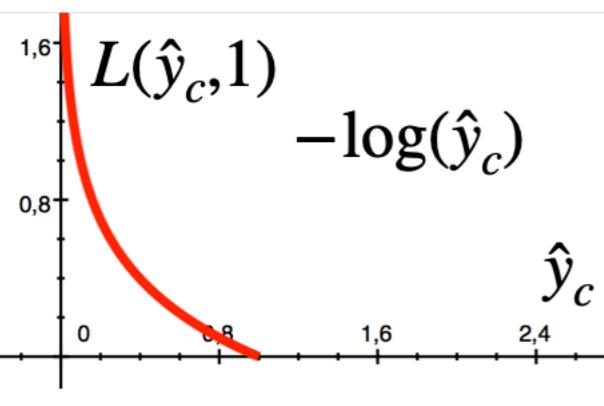
Fonction d'écart communément utilisée pour des tâches de classification multi-classe

$$L(\hat{y}, y) = - \sum_{c=1}^C y_c \log(\hat{y}_c) \text{ avec } \hat{y}_c \in [0,1] \text{ et } \sum_{c=1}^C \hat{y}_c = 1$$

$C$  : le nombre de classes présentes

$y_c$  : valeur du vecteur  $y$  qui vaut 1 si  $c$  correspond à la bonne classe (0 sinon)

$\hat{y}_c$  : valeur du vecteur  $\hat{y}$  qui correspond à la probabilité prédictive de la classe  $c$



**Intuition:** exactement le même principe que le cas binaire

**Principe:** minimiser la fonction pousse ( $\hat{y} \rightarrow 1$ ) pour la bonne classe

**Fonction de coût:** écart moyen de toutes les prédictions

$$\hat{y} = \begin{pmatrix} 0.70 \\ 0.26 \\ 0.04 \end{pmatrix} \approx y = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

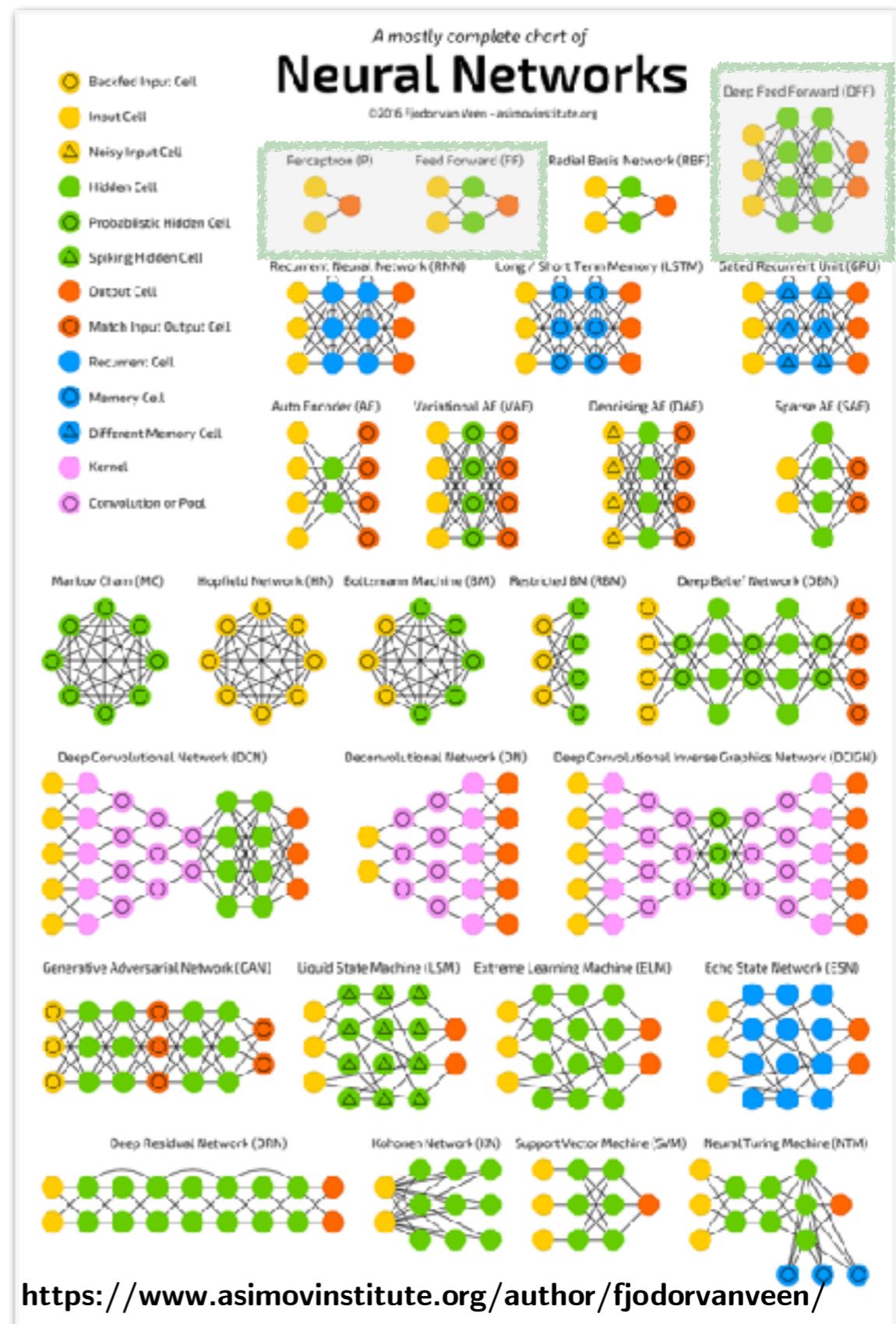
# Table des matières

## Réseaux de neurones

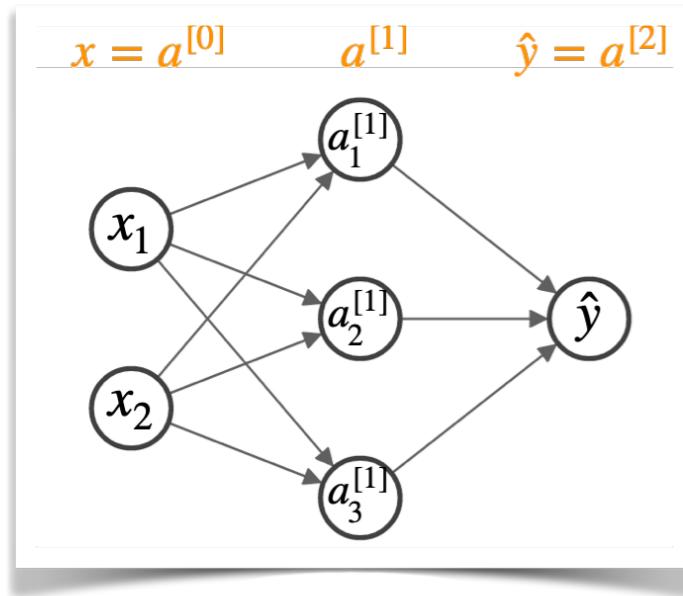
-  1. Concepts principaux des réseaux de neurones
-  2. Extension de la régression logistique
-  3. Notations standards des réseaux de neurones
-  4. Exécution de la *backpropagation* du gradient
-  5. Principes des fonctions d'activation
- 6. Initialisation des paramètres

## Apprentissage profond

- 1. Limitation des petits réseaux de neurones
- 2. Principes de l'apprentissage profond
- 3. Profondeur d'un réseau de neurones
- 4. Introduction aux hyper-paramètres
- 5. Conseils pratiques



# Initialisation des paramètres: valeur identique



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

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

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

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

$$\hat{Y} = A^{[2]}$$



**Idée 1:** initialiser tous les paramètres à la même valeur



Que pensez-vous de cette idée ?

**Contre-exemple:** essayons avec une valeur initiale de 0 pour tous les paramètres

Regardons ce qui se passe à la couche cachée

$$W^{[1]} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$b^{[1]} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$



Que peut-on dire de la valeur de sortie de ces 3 neurones ?

**Valeurs de sorties:** identiques pour les 3 neurones

$$a^{[1]} = \sigma(W^{[1]}x + b^{[1]})$$

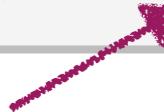
$$a_1^{[1]} = a_2^{[1]} = a_3^{[1]}$$

**Conséquence:** rien ne différencie les neurones, ils calculent la même fonction

**Passe en arrière:** les trois gradients vont par conséquent avoir la même valeur

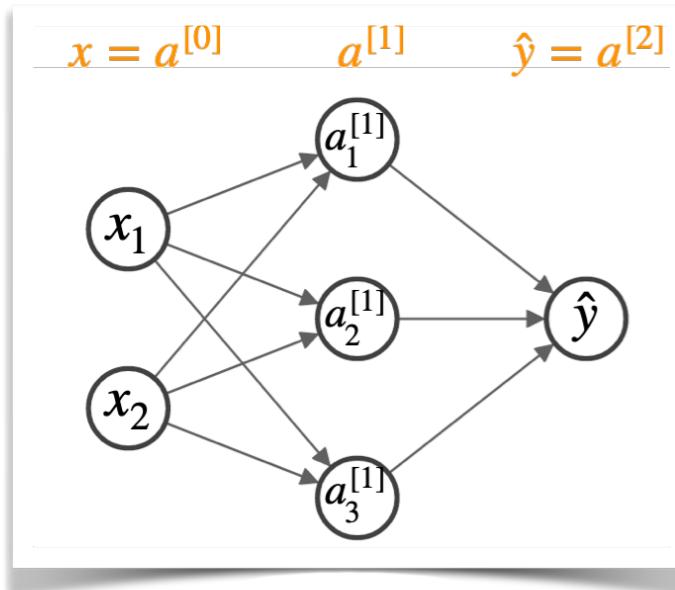
**Conclusion:** on n'est pas plus expressif qu'un seul neurone dans la couche

$$\frac{\partial J}{\partial W^{[1]}} = \begin{pmatrix} u & v \\ u & v \\ u & v \end{pmatrix}$$

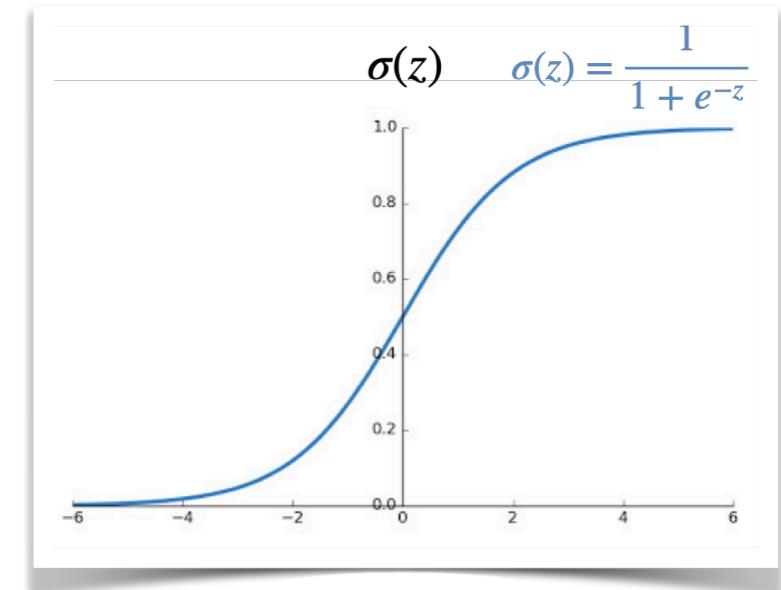


Valeurs du gradient liées au 3e neurone

# Initialisation des paramètres: valeur aléatoire quelconque



$$\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \\ \hat{Y} &= A^{[2]} \end{aligned}$$



**Idée 2:** initialiser tous les paramètres à une valeur aléatoire quelconque

**Intuition:** permet de corriger la difficulté précédente car le gradient lié à chaque neurone sera différent

?

Est-ce que n'importe quelle valeur est bonne ?

**Descente de gradient:** demande de calculer le gradient d'une fonction sigmoide (version de base)

?

Que se passe t-il si les paramètres ont une grande valeur (positive ou négative) ?

**Observation:** on se retrouve sur une région assez plate de la sigmoide (pente très faible)

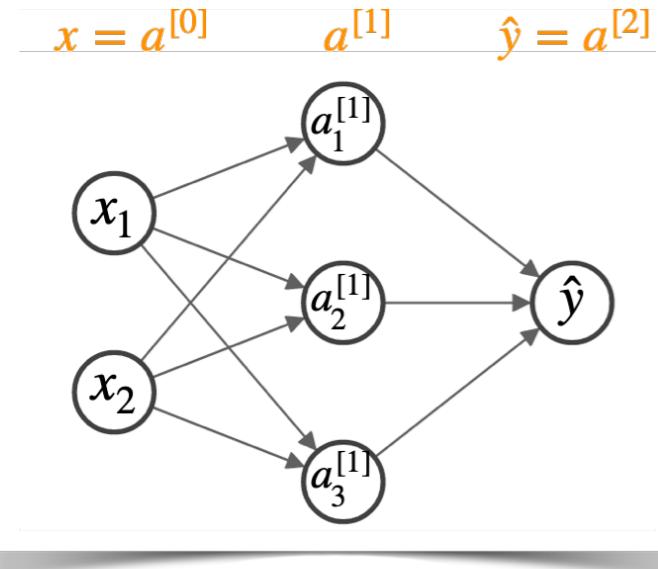
**Conséquence théorique:** la valeur du gradient sera par très petite

**Conséquence pratique:** l'entraînement sera lent

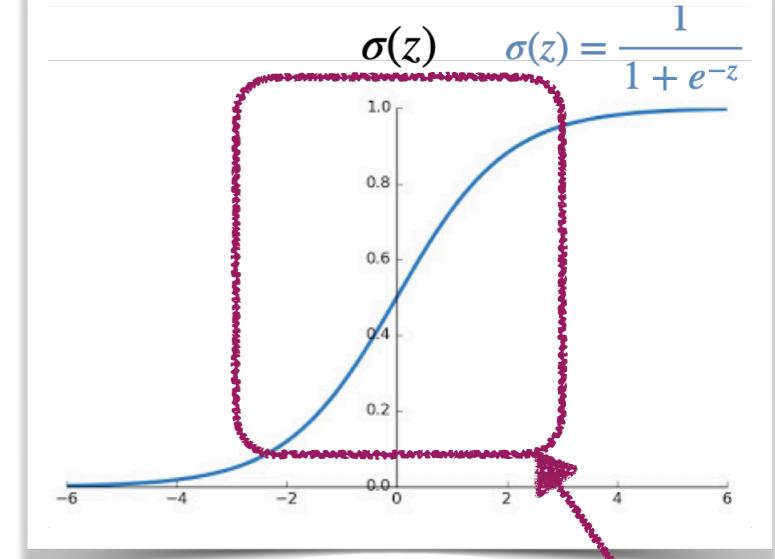
$$W^{[1]} = W^{[1]} - \alpha \frac{\partial J}{\partial W^{[1]}}$$

Très petite valeur

# Initialisation des paramètres: valeur aléatoire proche de zéro



$$\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \\ \hat{Y} &= A^{[2]} \end{aligned}$$



**Idée 3:** initialiser tous les paramètres à une valeur aléatoire proche de zéro

L'objectif est d'être dans cette zone

**Intuition:** permet de corriger la difficulté précédente en permettant des plus grands pas de gradient

**Approche simple:** initialiser les paramètres aléatoirement dans un intervalle entre -1 et 1

Cette initialisation est simple et donne de bons résultats en pratique pour de petits réseaux

**Note:** il n'y a aucun problème à initialiser les biais ( $b$ ) à zéro

## Autres fonctions d'initialisation (hors matière)

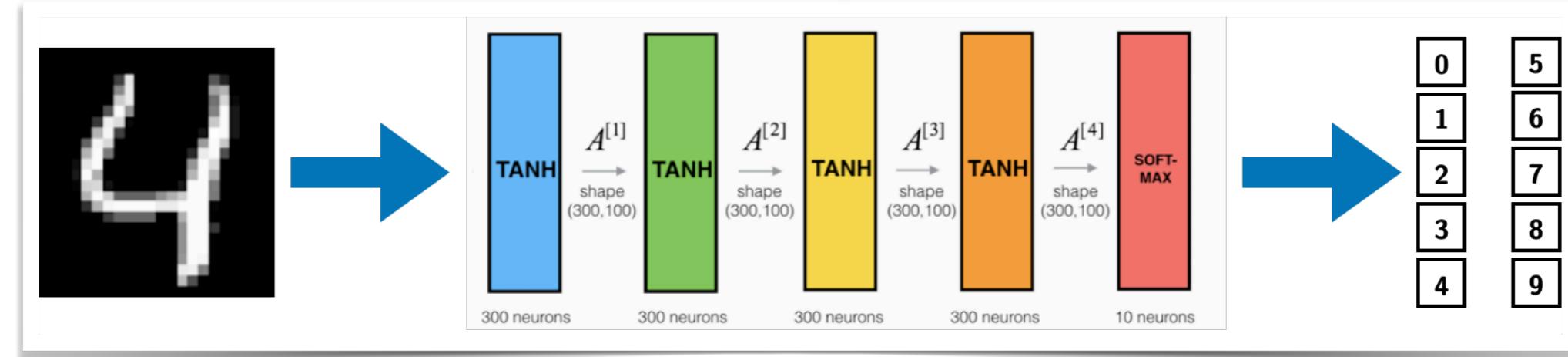
**Idée:** changer la façon d'initialiser en fonction de la couche considérée

**Initialisation de Xavier:**  $W^{[l]} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}\right)$  (pour une activation tanh)

**Initialisation de He:**  $W^{[l]} \sim \mathcal{N}\left(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]}}\right)$  (pour une activation ReLU)

**Objectif:** aider à la stabilisation du gradient dans des réseaux à beaucoup de couches

# Visualisation de l'impact de l'initialisation

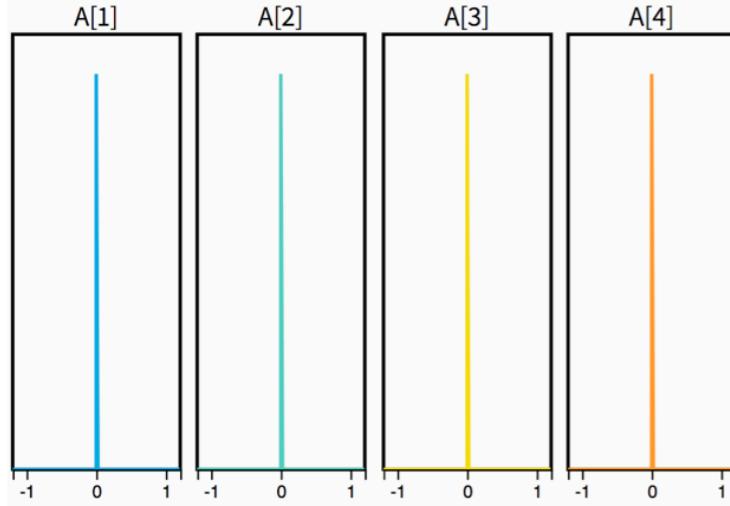


**Tâche:** classification de chiffres écrits à la main (même tâche que pour le devoir)

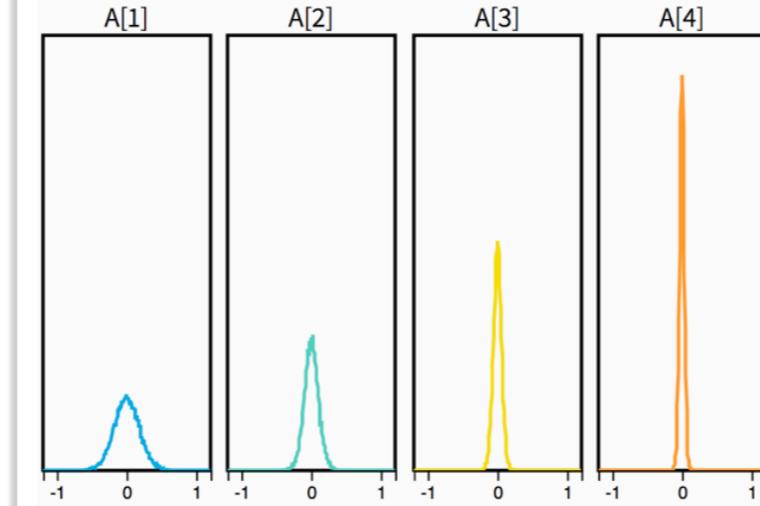
**Architecture:** réseau de neurones à 4 couches avec des activations de type tanh

**Analyse:** impact de l'initialisation des paramètres sur les valeurs des activations à chaque couche

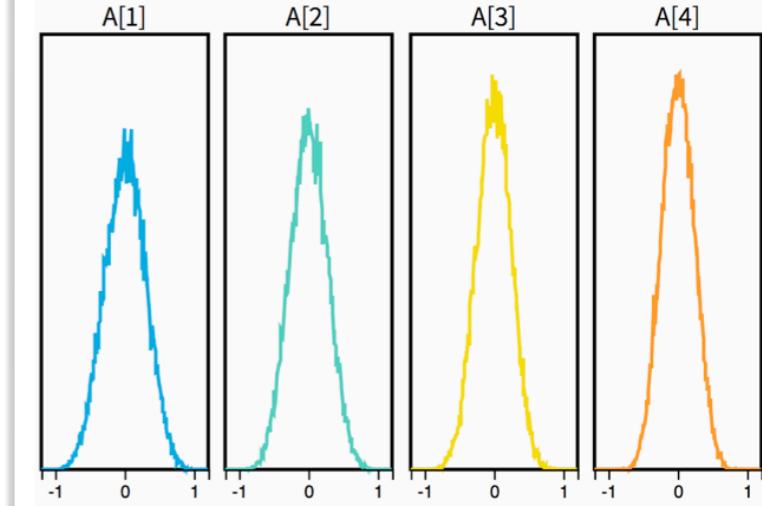
Initialisation à zéro



Initialisation uniforme bornée



Initialisation de Xavier



Le signal globalement nul

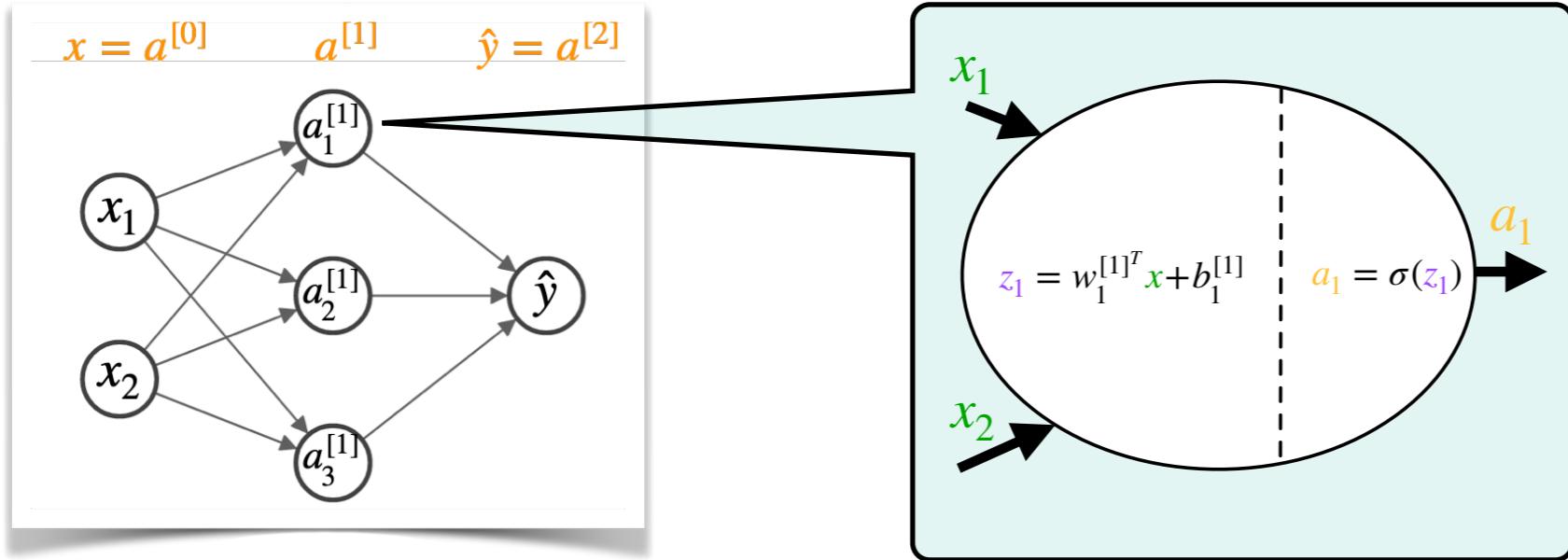
Le signal s'estompe

La signal reste stable

*Understanding the difficulty of training deep feedforward neural networks* [Glorot and Bengio, 2010]

*Delving deep into rectifiers: Surpassing human-level performance on imagenet classification* [He et al., 2015]

# Récapitulatif: réseau de neurones



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

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

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

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

$$\hat{Y} = A^{[2]}$$

## Concepts clefs

**Architecture:** neurones structurés en plusieurs couches

**Passe en avant:** prédiction via les équations fondamentales

**Notation matricielle:** expression compacte et plus efficace

**Entraînement:** descente de gradient et passe en arrière

**Difficulté:** la fonction de coût n'est plus convexe

**Fonction d'activation intermédiaire:** non-linéaire

**Dernière activation:** dépend de la tâche de prédiction

**Initialisation des paramètres:** aléatoire et proche de 0

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

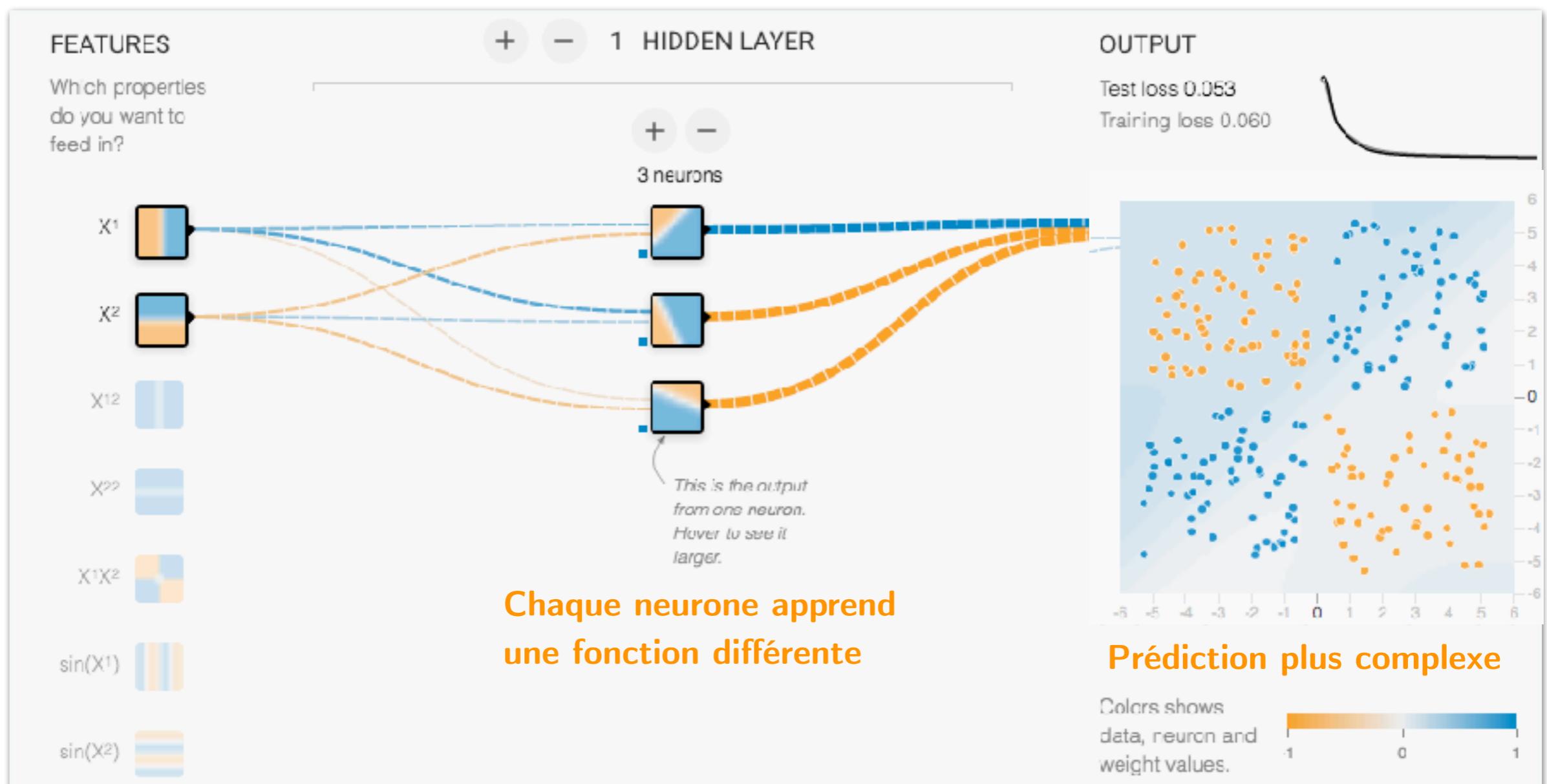
$$\frac{\partial J}{\partial b^{[1]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial b^{[1]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial b^{[1]}} \right)$$

$$\frac{\partial J}{\partial b^{[2]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial b^{[2]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial b^{[2]}} \right)$$

$$\frac{\partial J}{\partial W^{[1]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial W^{[1]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial W^{[1]}} \right)$$

$$\frac{\partial J}{\partial W^{[2]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial W^{[2]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial W^{[2]}} \right)$$

# Visualisation de la prédiction



**Tâche:** classification binaire de points 2D

**Caractéristiques en entrée:** coordonnée en x, coordonnée en y

**Activation:** tangente hyperbolique

**Taux d'apprentissage:** 0.01



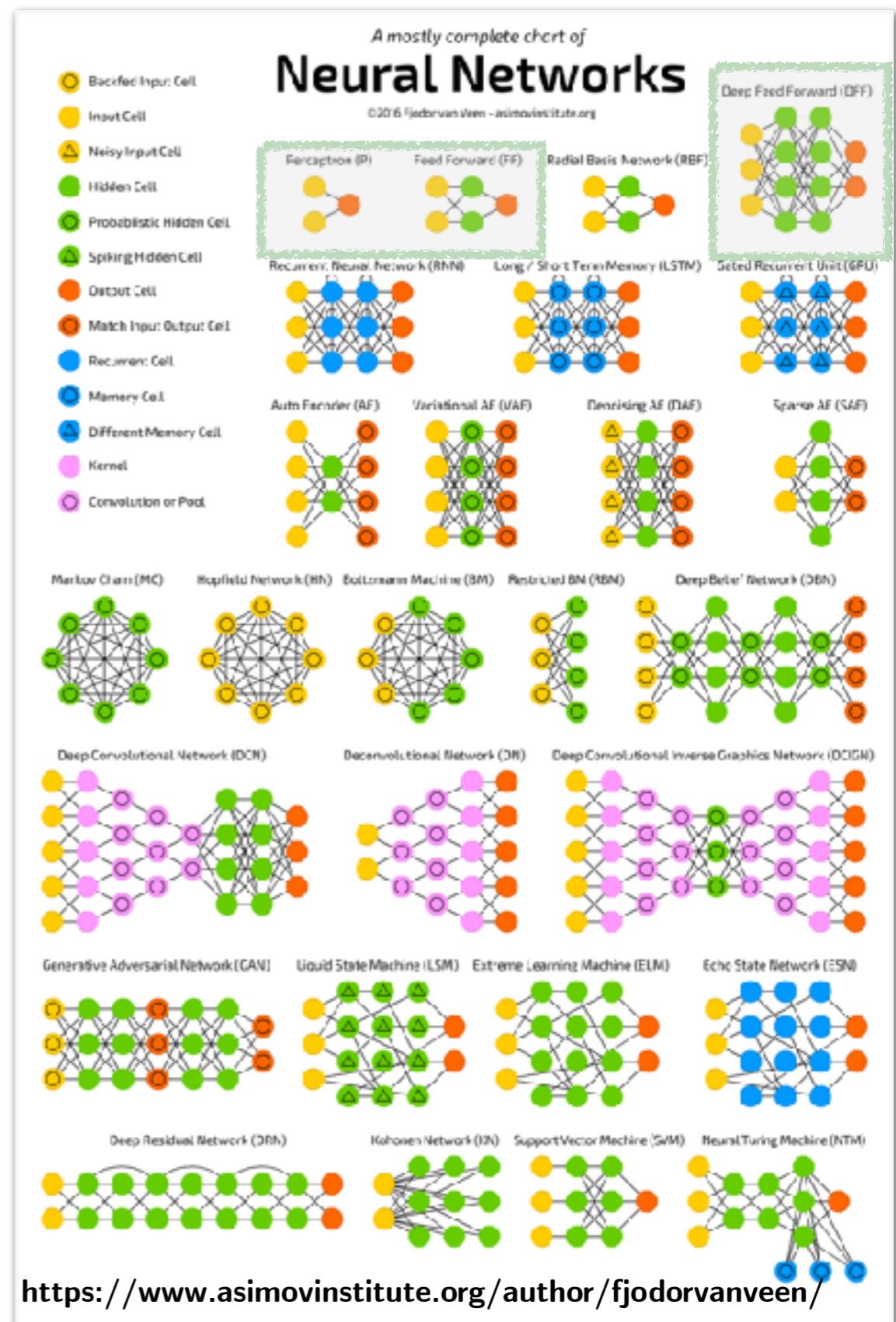
# Table des matières

## Réseaux de neurones

-  1. Concepts principaux des réseaux de neurones
-  2. Extension de la régression logistique
-  3. Notations standards des réseaux de neurones
-  4. Exécution de la *backpropagation* du gradient
-  5. Principes des fonctions d'activation
-  6. Initialisation des paramètres

## Apprentissage profond

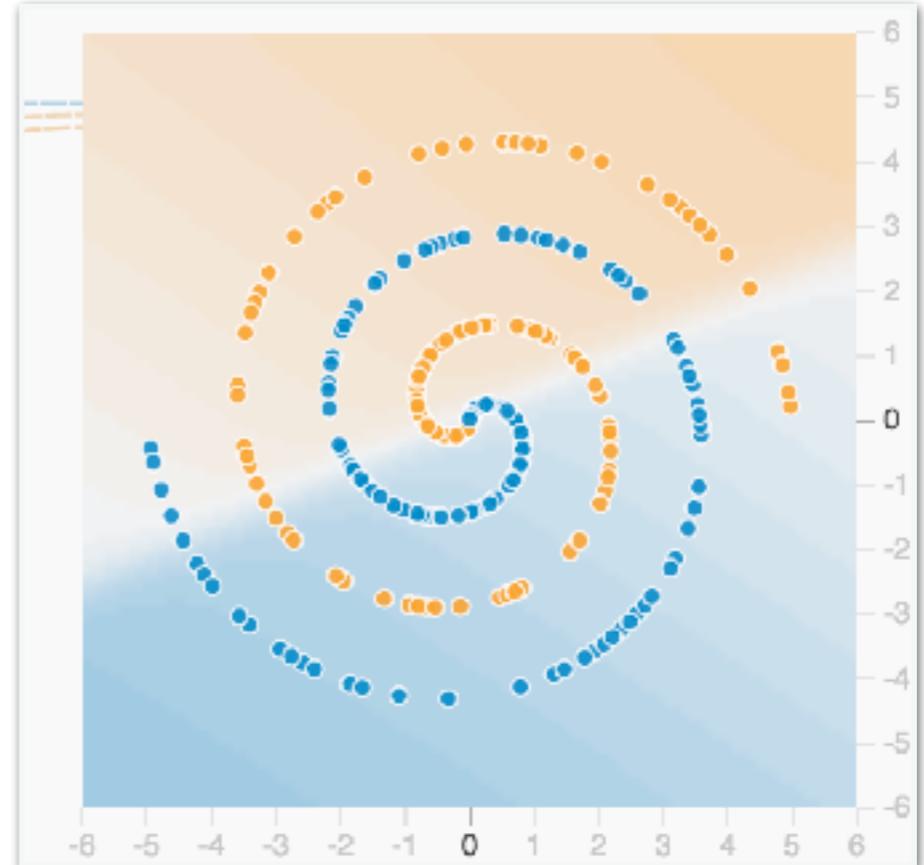
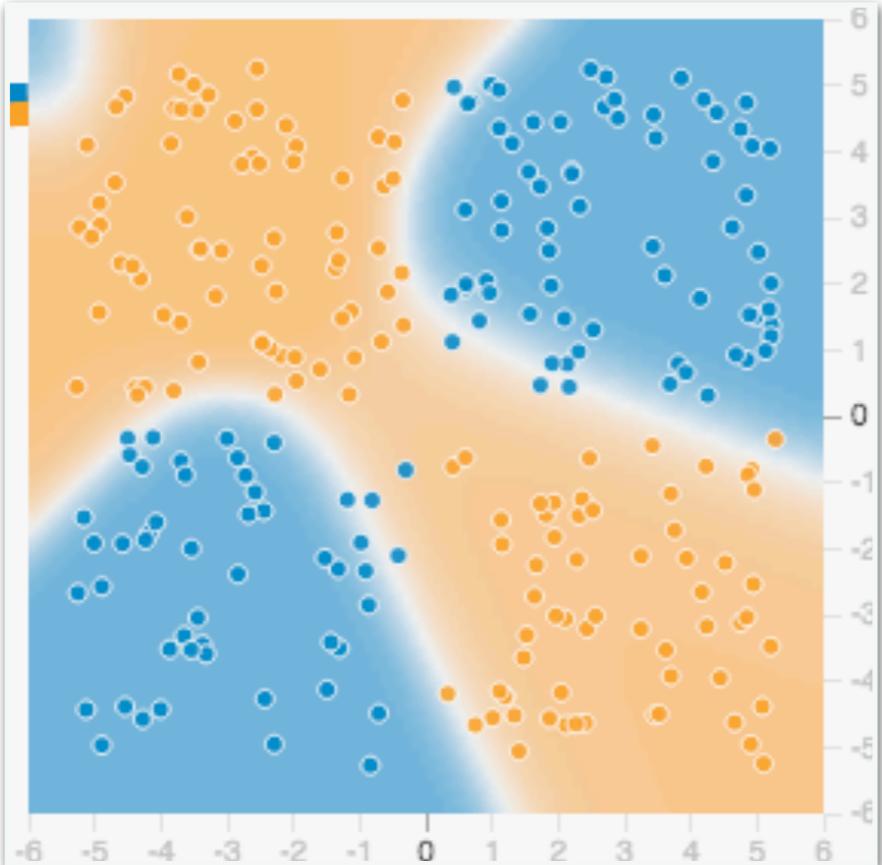
- 1. Limitation des petits réseaux de neurones
- 2. Principes de l'apprentissage profond
- 3. Profondeur d'un réseau de neurones
- 4. Introduction aux hyper-paramètres
- 5. Conseils pratiques



# Limitation de notre réseau de neurones



Quelles sont les limitations de notre réseau de neurones actuel ?



**Limitation:** il peut manquer d'expressivité pour approximer des fonctions complexes

**Figure de gauche:** bonne séparation avec un réseau à deux couches

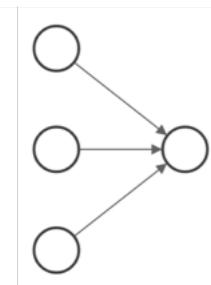
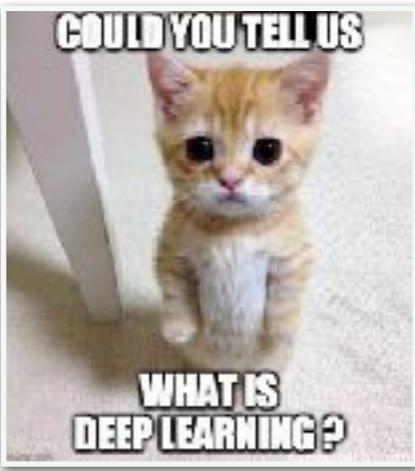
**Figure de droite:** la séparation avec le même réseau devient difficile

**Conclusion:** il faut une fonction de prédiction plus expressive !

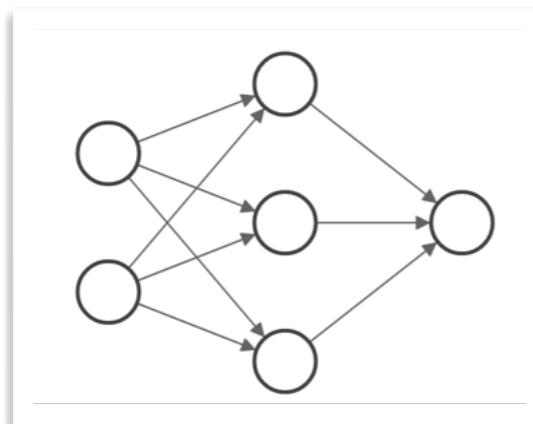


Pour cela, on va simplement augmenter le nombre de couches dans le réseau

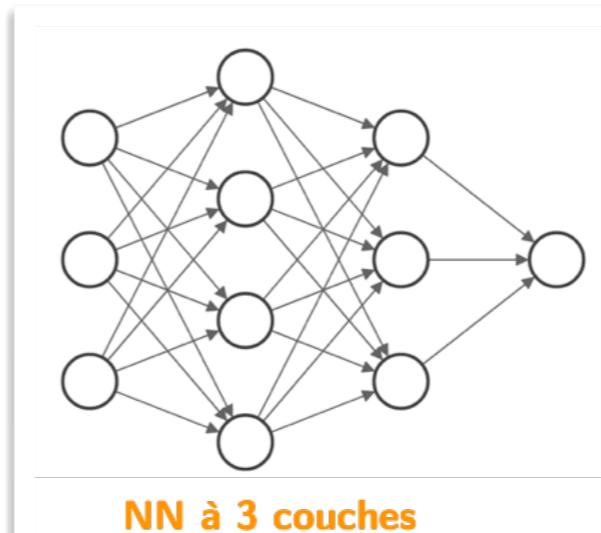
# Apprentissage profond (*deep learning*)



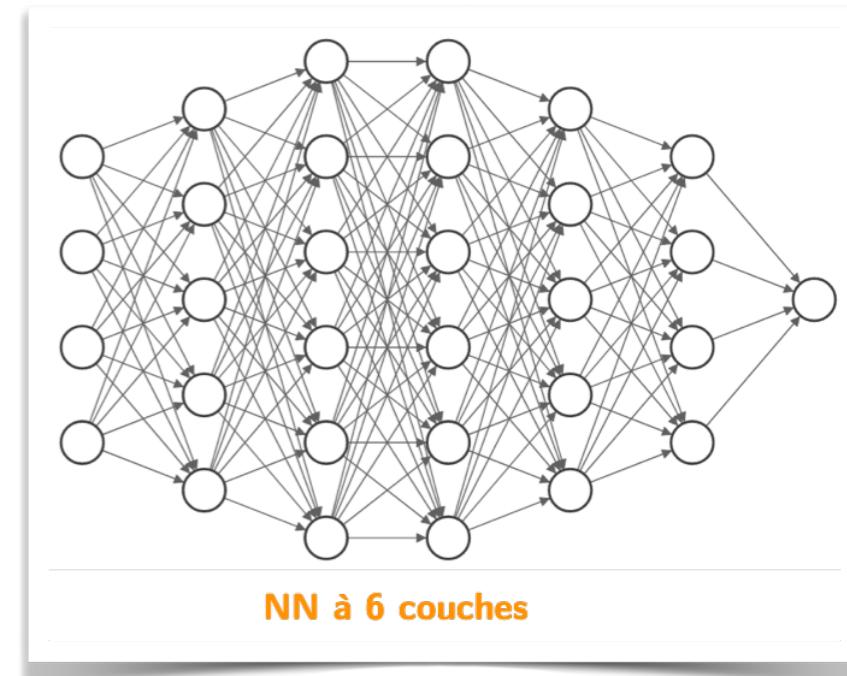
Régression logistique



NN à 2 couches



NN à 3 couches



NN à 6 couches

Conceptuellement, il n'y a pas de profondeur limite (bien que des difficultés apparaissent)

Exemple: le réseau utilisé dans AlphaZero a environ 80 couches, celui de GPT-4 a 120 couches

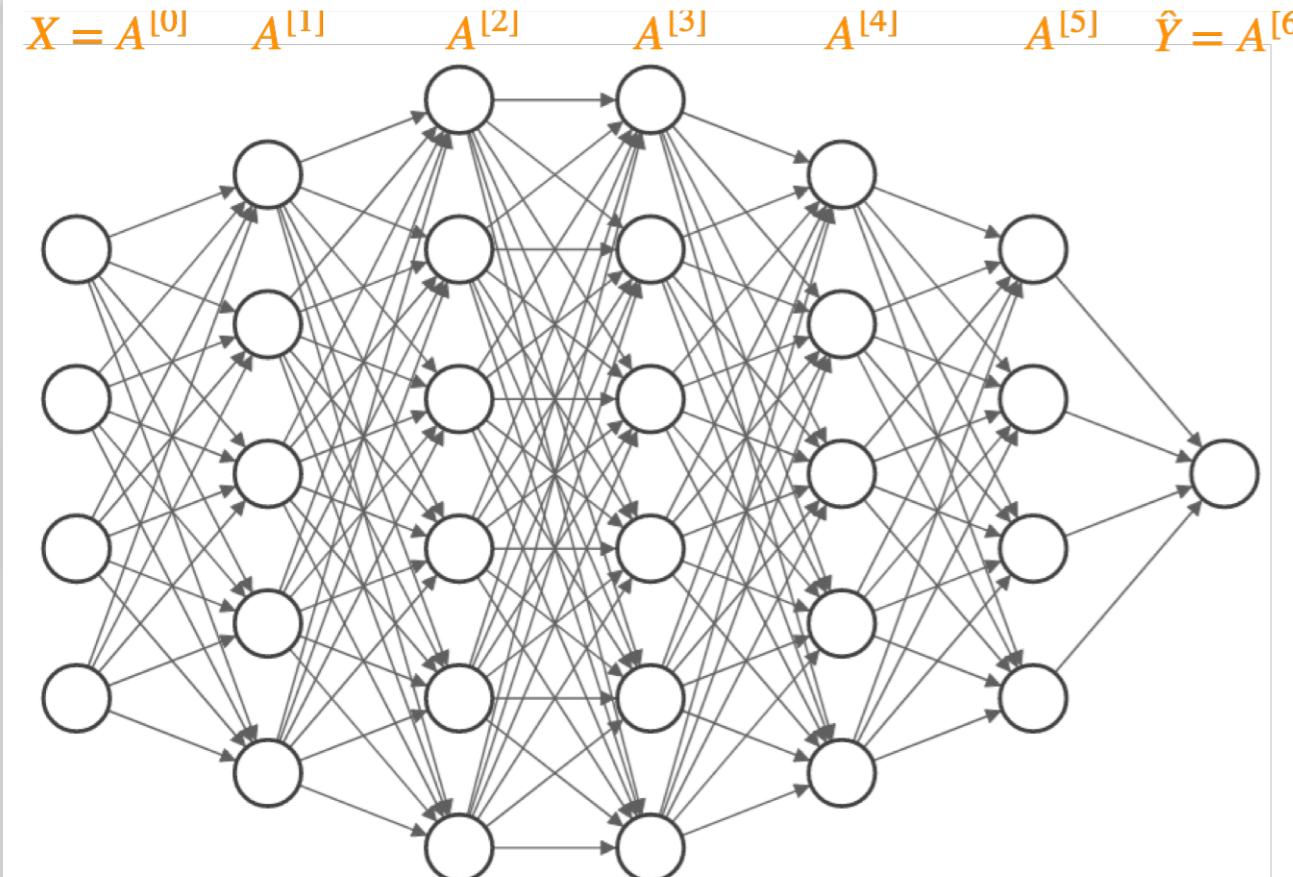


Contenu du module: on a formalisé les réseaux de neurones de façon générique

Avantage: l'extension à l'apprentissage profond va être TRES facile

Pour faire simple, on a déjà fait tout le nécessaire...

# Notations pour le deep learning



## Notations

$L$  : nombre de couches (couche d'entrée non comprise)

$n^{[l]}$  : nombre de neurones pour la couche  $l$

$X$  : caractéristiques en entrée (format matriciel)

$Z^{[l]}$  : valeurs avant l'activation de la couche  $l$

$A^{[l]}$  : sortie de la couche  $l$

$\hat{Y}$  : sortie du réseau (prédiction)

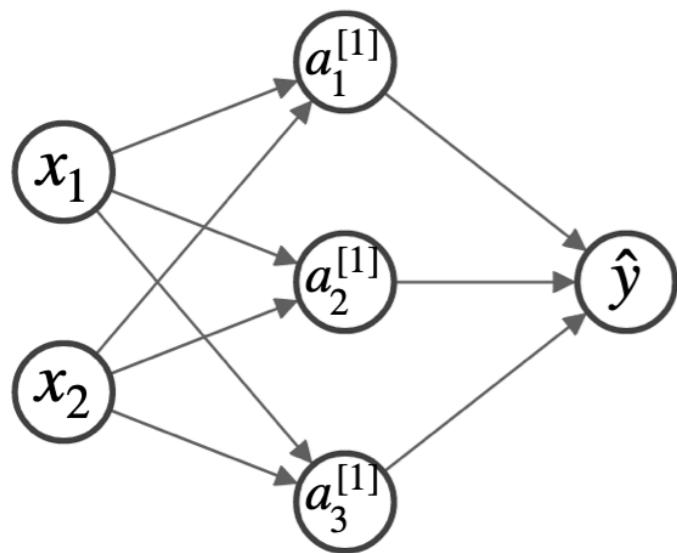
$W^{[l]}, b^{[l]}$  : paramètres de la couche  $l$

**Exercice:** donnez les dimensions de chaque matrice pour 10 données d'entraînement

**Exercice:** donnez le nombre de paramètres à apprendre dans ce réseau

# Equations fondamentales de l'apprentissage profond

$$x = a^{[0]} \quad a^{[1]} \quad \hat{y} = a^{[2]}$$



**Situation précédente:** réseau de neurones à 2 couches

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

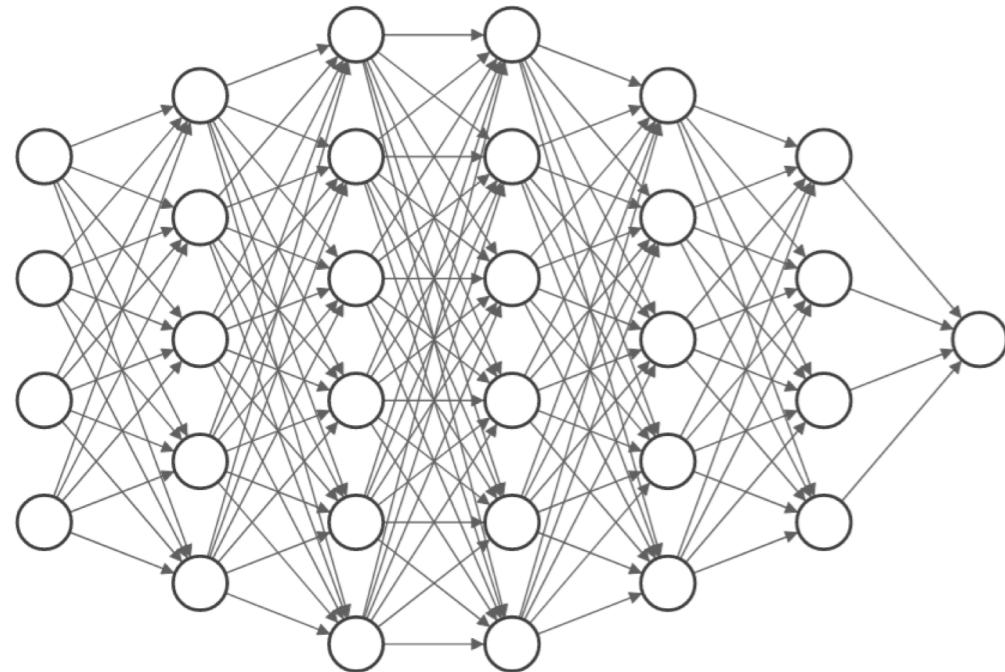
$$A^{[1]} = g^{[1]}(Z^{[1]})$$

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

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

$$\hat{Y} = A^{[2]}$$

$$x = A^{[0]} \quad A^{[1]} \quad A^{[2]} \quad A^{[3]} \quad A^{[4]} \quad A^{[5]} \quad \hat{Y} = A^{[6]}$$



**Situation actuelle:** réseau de neurones à  $L$  couches

$$A^{[0]} = X$$

for  $l \in \{1, \dots, L\}$  :

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

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

$$\hat{Y} = A^{[L]}$$

Il n'y a vraiment rien de nouveau conceptuellement par rapport à précédemment

# Descente de gradient dans un réseau de neurones profond

gradientDescent( $D, \alpha$ ) :

$W^{[l]}, b^{[l]} = \text{initializeRandomly}() \quad \forall l \in \{1, \dots, L\}$

repeat until convergence :

compute  $\hat{Y}$

compute  $J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]})$

compute  $\frac{\partial J}{\partial W^{[L]}}, \frac{\partial J}{\partial b^{[L]}}, \dots, \frac{\partial J}{\partial W^{[1]}}, \frac{\partial J}{\partial b^{[1]}}$

$W^{[l]} = W^{[l]} - \alpha \frac{\partial J}{\partial W^{[l]}} \quad \forall l \in \{1, \dots, L\}$

$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}} \quad \forall l \in \{1, \dots, L\}$

return  $W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}$

Calcul des dérivées par un graphe de dépendance  
et une application de la passe en arrière

## Nombre de paramètres

$$W^{[1]} = n^{[1]} \times n^{[0]} = 5 \times 4 = 20$$

$$W^{[2]} = n^{[2]} \times n^{[1]} = 6 \times 5 = 30$$

$$W^{[3]} = n^{[3]} \times n^{[2]} = 6 \times 6 = 36$$

$$b^{[1]} = n^{[1]} \times 1 = 5 \times 1 = 5$$

$$b^{[2]} = n^{[2]} \times 1 = 6 \times 1 = 6$$

$$b^{[3]} = n^{[3]} \times 1 = 6 \times 1 = 6$$

$$W^{[4]} = n^{[4]} \times n^{[3]} = 5 \times 6 = 30$$

$$W^{[5]} = n^{[5]} \times n^{[4]} = 4 \times 5 = 20$$

$$W^{[6]} = n^{[6]} \times n^{[5]} = 1 \times 4 = 4$$

$$b^{[4]} = n^{[4]} \times 1 = 5 \times 1 = 5$$

$$b^{[5]} = n^{[5]} \times 1 = 4 \times 1 = 4$$

$$b^{[6]} = n^{[6]} \times 1 = 1 \times 1 = 1$$

**Total: 167 paramètres à apprendre**

$$A^{[0]} = X$$

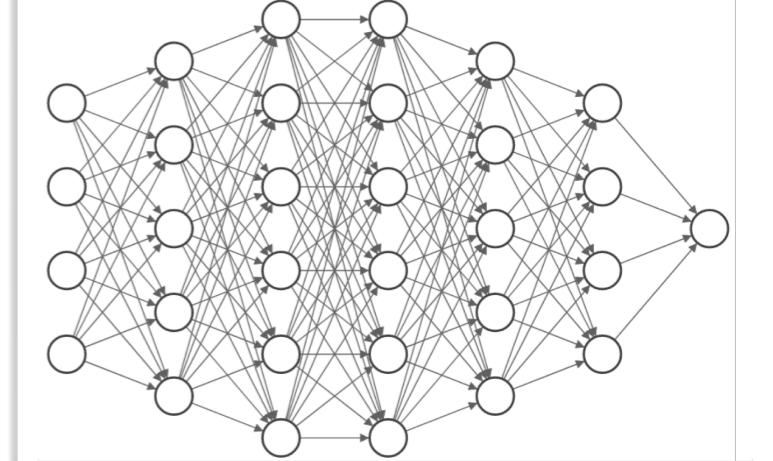
for  $l \in \{1, \dots, L\}$  :

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

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

$$\hat{Y} = A^{[L]}$$

$$X = A^{[0]} \quad A^{[1]} \quad A^{[2]} \quad A^{[3]} \quad A^{[4]} \quad A^{[5]} \quad \hat{Y} = A^{[6]}$$



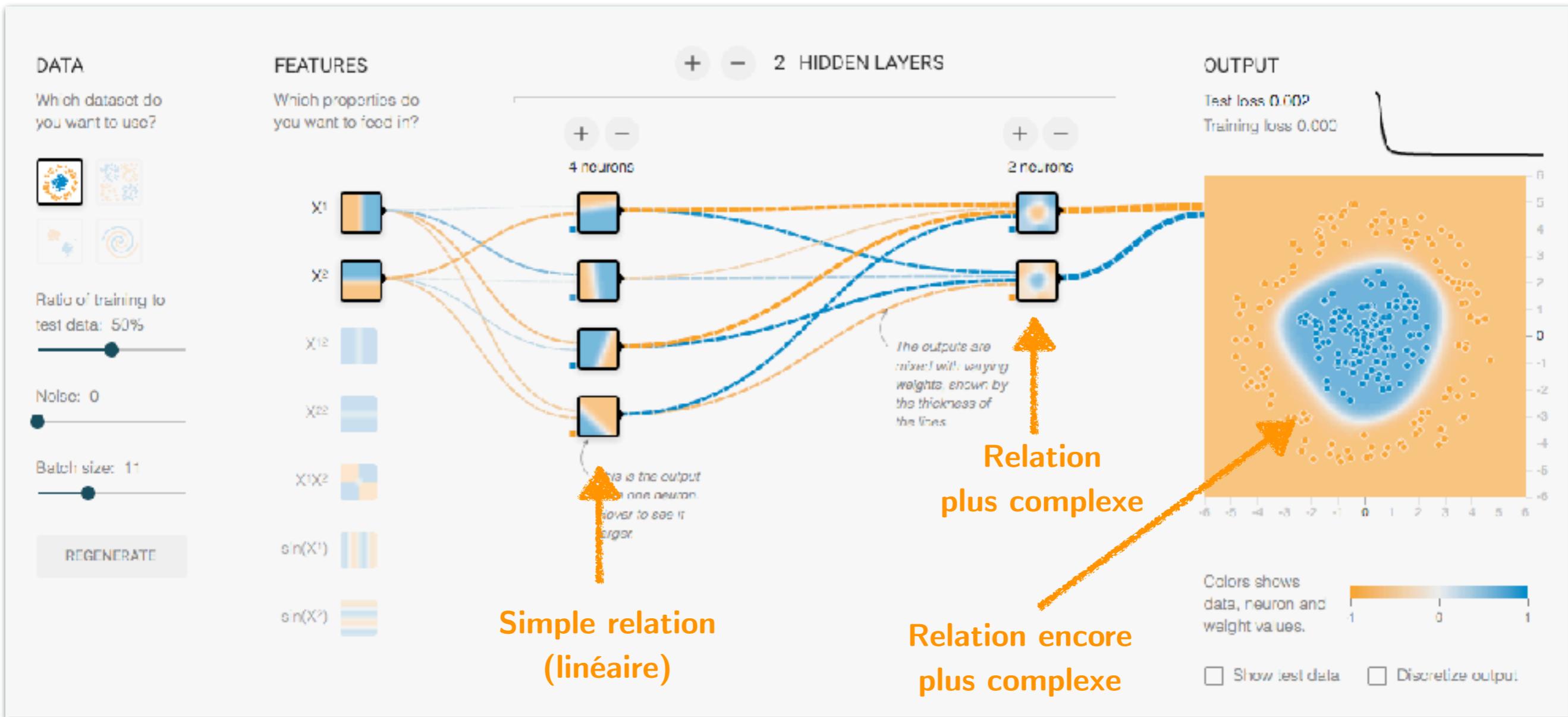
$$J(W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$$

for  $l$  in  $L$  to 1 :

$$\frac{\partial J}{\partial W^{[l]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial W^{[l]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial W^{[l]}} \right)$$

$$\frac{\partial J}{\partial b^{[l]}} = \frac{1}{m} \left( \frac{\partial L(\hat{y}^{(1)}, y^{(1)})}{\partial b^{[l]}} + \dots + \frac{\partial L(\hat{y}^{(m)}, y^{(m)})}{\partial b^{[l]}} \right)$$

# Expressivité de l'apprentissage profond



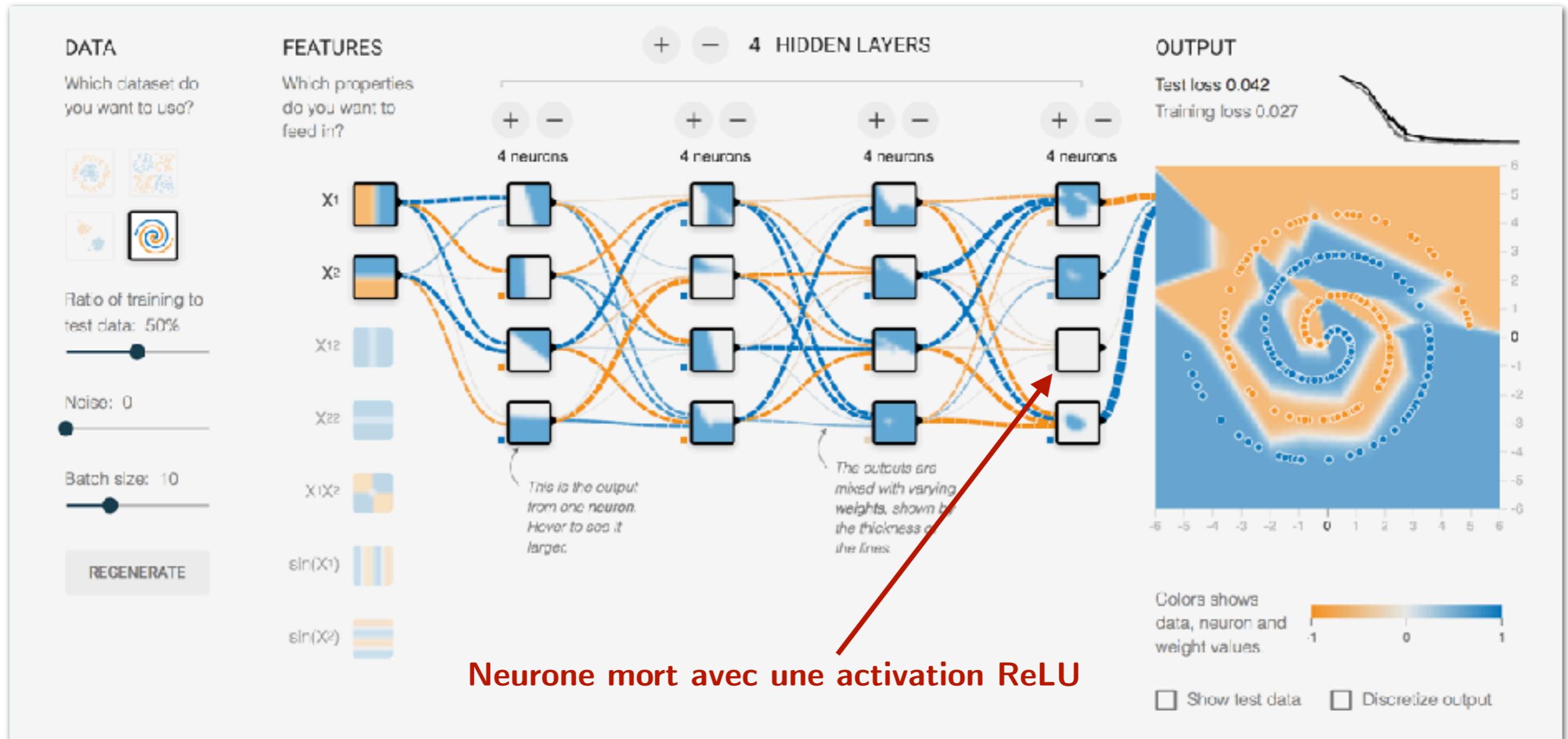
**Intérêt principal:** permet de construire des fonctions plus expressives

**Observation:** les neurones apprennent des relations de plus en plus complexes avec le niveau de profondeur



Qu'en est-il des fonctions encore plus compliquées ?

# Expressivité de l'apprentissage profond



**Situation à résoudre:** assez compliquée, et demande une bonne fonction d'approximation

**Solution:** en augmentant la taille du réseau, on arrive à avoir des meilleures frontières de décision

**Observation:** une activation ReLU a été utilisée, et un neurone est *mort*

**Expéimentez par vous même en essayant d'autres fonctions d'activation et taux d'apprentissage**

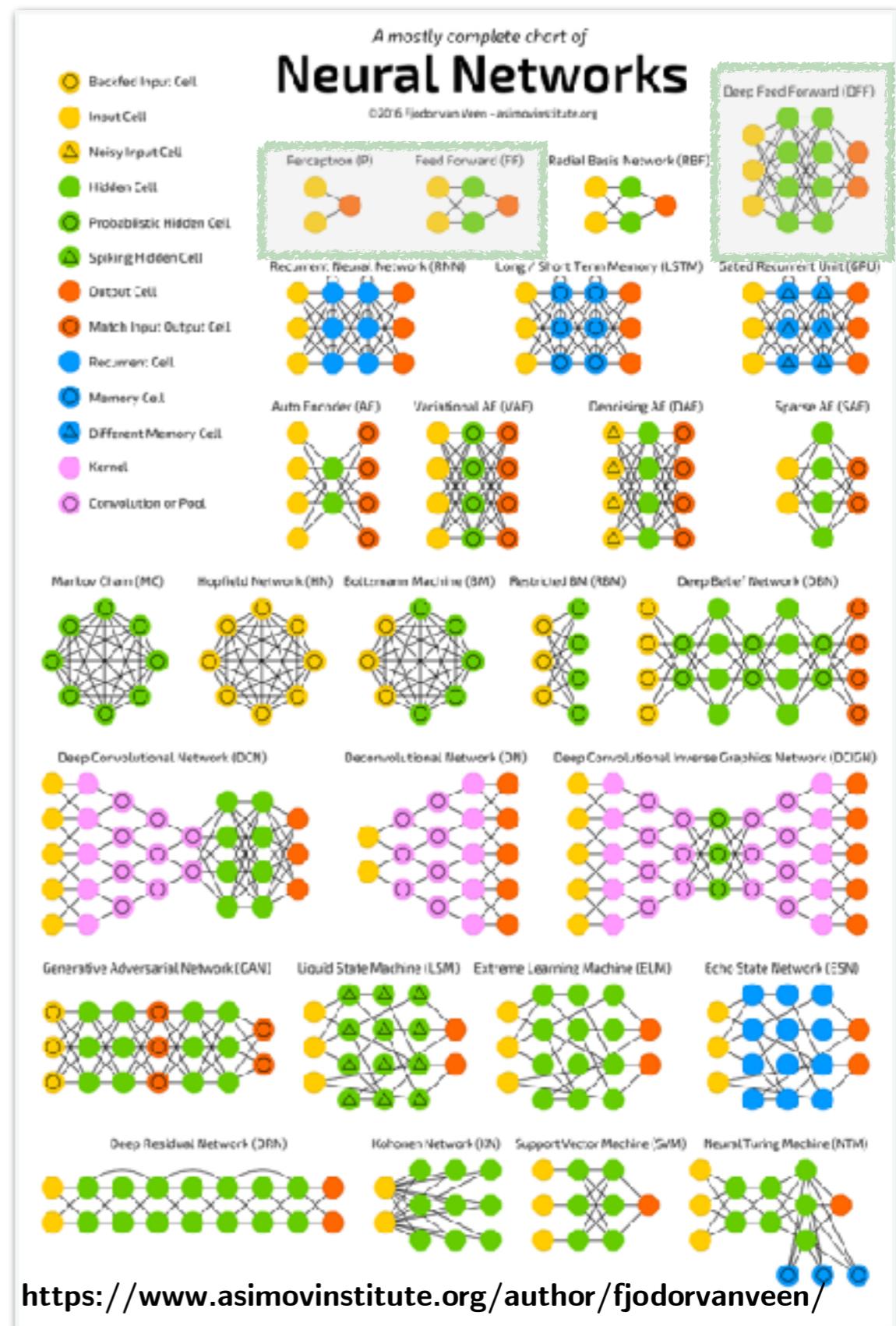
# Table des matières

## Réseaux de neurones

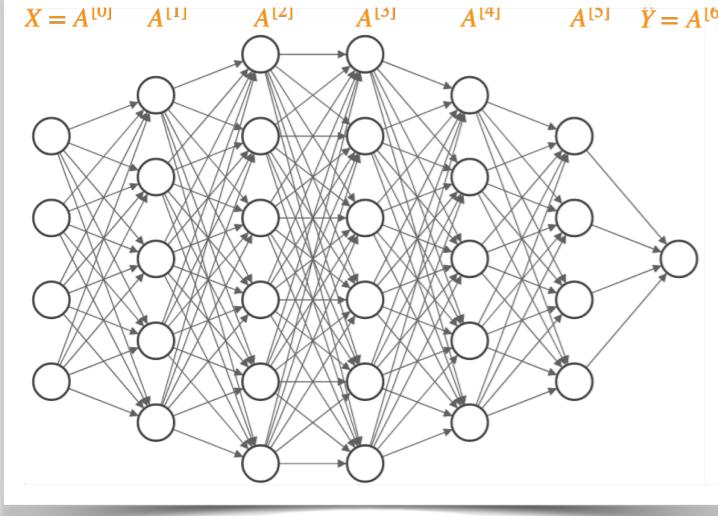
- ✓ 1. Concepts principaux des réseaux de neurones
- ✓ 2. Extension de la régression logistique
- ✓ 3. Notations standards des réseaux de neurones
- ✓ 4. Exécution de la *backpropagation* du gradient
- ✓ 5. Principes des fonctions d'activation
- ✓ 6. Initialisation des paramètres

## Apprentissage profond

- ✓ 1. Limitation des petits réseaux de neurones
- ✓ 2. Principes de l'apprentissage profond
- ✓ 3. Profondeur d'un réseau de neurones
- 4. Introduction aux hyper-paramètres
- 5. Conseils pratiques



# Introduction aux hyper-paramètres



Comment déterminer l'architecture du réseau à utiliser ?

**Exemple:** nombre de couches, fonction d'activation, etc.

**Autre choix à faire:** taux d'apprentissage

Cette question amène à différencier paramètres et hyper-paramètres

**Paramètres:** valeurs qui sont apprises durant la descente de gradient

$W^{[l]}$  : poids de la couche  $l$

$b^{[l]}$  : biais de la couche  $l$

**Intuition:** il s'agit des valeurs qui vont déterminer la fonction que vous construisez

**Point de vue de l'utilisateur:** il n'a aucun contrôle sur la valeur des paramètres



**Hyperparamètre**

Valeurs définies par l'utilisateur utilisées pour contrôler l'apprentissage et le type de modèle

**Importance:** les valeurs choisies ont un impact sur la qualité de l'apprentissage et sur le modèle obtenu

(1) Nombre de couches

(3) Fonction d'activation utilisée

(2) Nombre de neurones par couche

(4) Taux d'apprentissage

Et encore tout plein !

**Objectif idéal:** trouver les hyper-paramètres qui vont résulter au modèle le plus performant

# Sélection de la valeur des hyper-paramètres

LET US FIND THE BEST



HYPERPARAMETERS

**Difficulté de la tâche:** il y a énormément de combinaisons à tester

**Mise en situation:** exemple très petit avec quelques choix à faire

Taux d'apprentissage: {0.0001, 0.001, 0.01, 0.1}

Nombre de couches: {1,2,3,4,5}

Fonction d'activation: {ReLU, tanh, sigmoid}

Total:  $4 \times 5 \times 3 = 60$  configurations à tester



**Lien avec le module 3:** trouver la meilleure configuration est un problème d'optimisation combinatoire

**Difficulté supplémentaire:** évaluer chaque configuration est coûteux car cela demande d'entraîner un réseau

## Conseils pratiques

**Etape 1:** fixez les valeurs standards dans un premier temps (par exemple une activation ReLU)

**Etape 2:** procédez itérativement et analysez les résultats de votre modèle

**Exemple:** une mauvaise précision sur les données d'entraînement peut être dû à un manque d'expressivité

**Action possible:** il serait pertinent d'essayer d'augmenter la profondeur du réseau

## Algorithme de recherche d'hyper-paramètres

Recherche exhaustive (*grid search*)

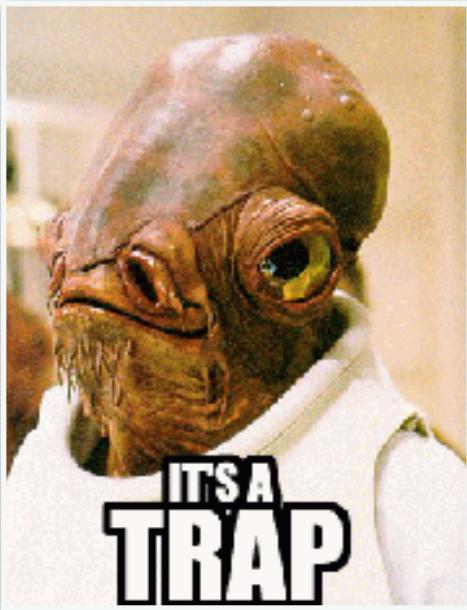
Recherche aléatoire (*random search*)

Recherche locale et métahéuristiques

Optimisation bayésienne

# Autres conseils pratiques

## Difficultés et pièges habituels



- (1) Confondre les dimensions des matrices
- (2) Oublier d'initialiser les poids aléatoirement
- (3) Avoir un taux d'apprentissage trop grand (surtout avec ReLU)
- (4) Oublier d'ajouter une fonction d'activation non-linéaire
- (5) Avoir une fonction d'activation non adaptée au problème à la dernière couche
- (6) Utiliser une fonction de coût non adaptée au problème

## Evaluation rigoureuse et analyse de vos réseaux

**Bonne pratique:** diviser vos données en plusieurs ensembles (entraînement, validation, et de test)

**Notion de sous-apprentissage (*underfitting*):** difficulté à donner de bons résultats sur des données vues

**Notion de sur-apprentissage (*overfitting*):** difficulté à donner de bons résultats sur des données non-vues

Ces considérations seront à prendre en compte lors du devoir 3 et discutées lors de votre laboratoire

## Librairies pour l'apprentissage profond



TensorFlow



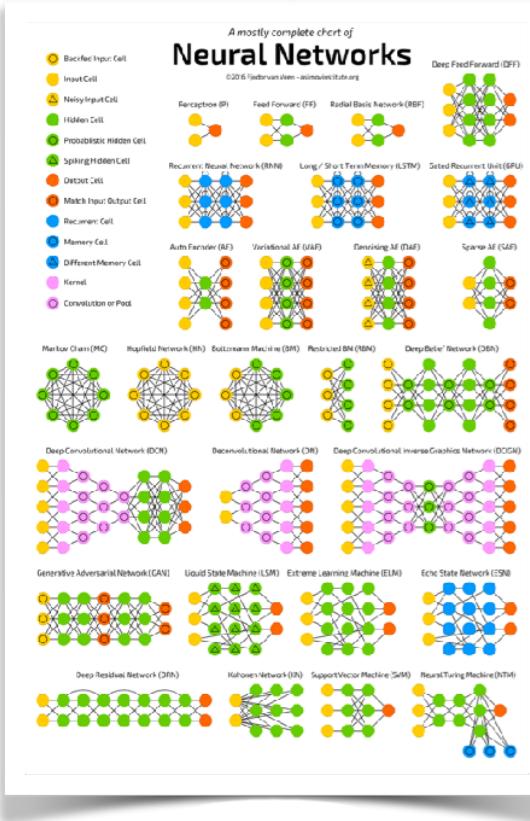
Objectif 1: faciliter la création de réseaux pour un utilisateur



Objectif 2: automatiser l'entraînement d'un modèle

Je vous conseille Keras pour débuter (très facile d'utilisation)

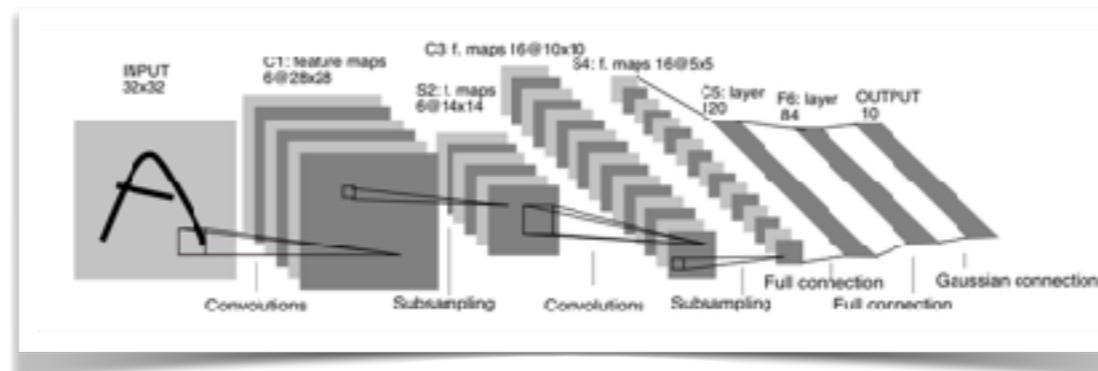
# Autres architectures existantes



Il existe également d'autres architectures de réseau de neurones

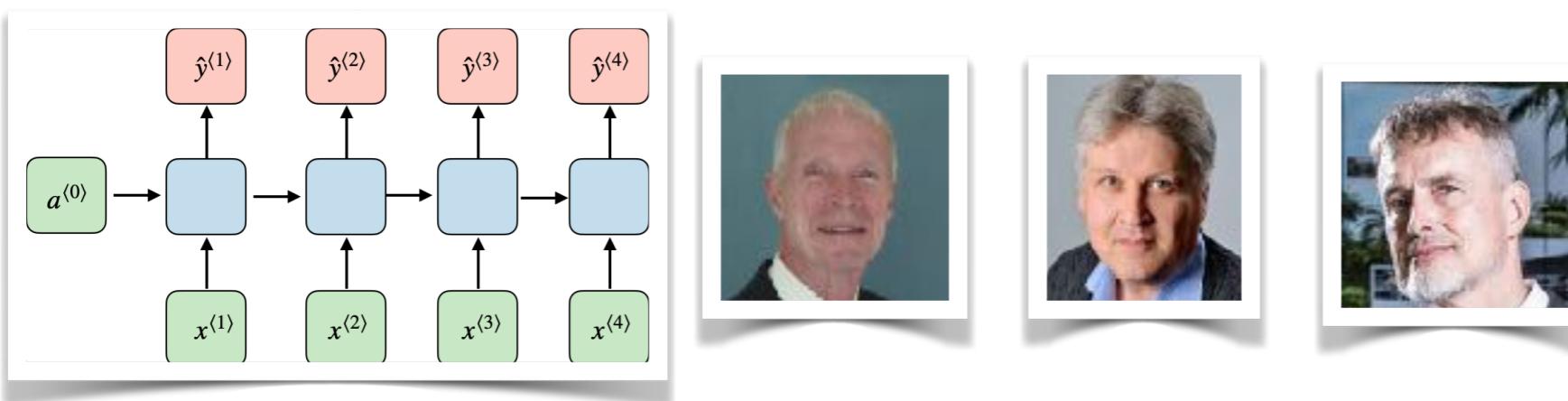
Intérêt: avoir une architecture adaptée aux données en entrées

Réseau à convolution: spécialisé pour l'apprentissage sur des images



Gradient-based learning applied to document recognition [LeCun, Bengio et al., 1998]

Réseau récurrent: spécialisé pour l'apprentissage sur des séquences de données

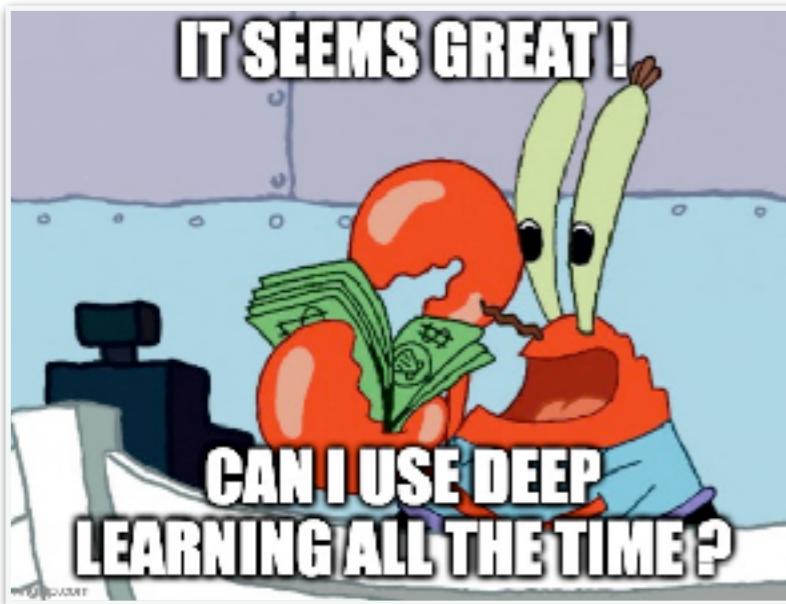


Neural networks and physical systems with emergent collective computational abilities [Hopfield, 1982]

Long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997]

Et d'autres: réseau à convolution de graphe, architecture adversarielle, modèles génératifs, etc.

# Difficultés de l'apprentissage profond et alternatives



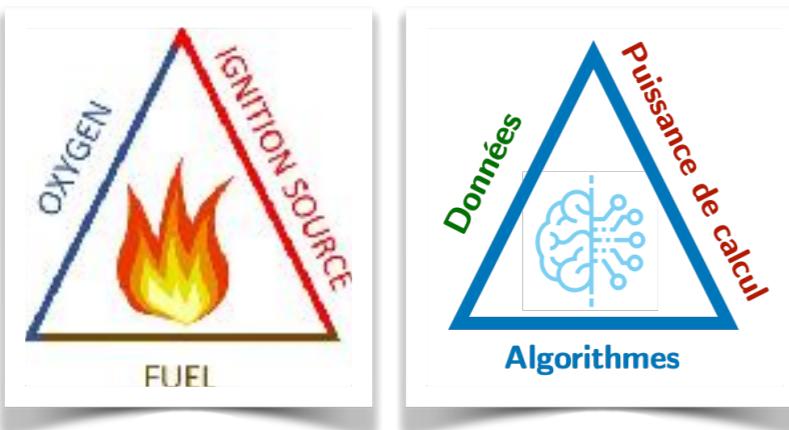
**Besoin de l'apprentissage profond:** trois composants essentiels

- (1) Une méthode d'apprentissage (ce qu'on a vu dans ce module)
- (2) Un nombre conséquent de données labellisées
- (3) De la puissance de calcul

Les besoins (2) et (3) augmentent avec la complexité du réseau

**Analogie:** le triangle du feu donnant les trois besoins pour avoir un feu

Dans les deux cas, si un de ces ingrédients manque, le résultat (ou performances) en sera affecté



?

Quelles sont les alternatives aux réseaux de neurones

Il existe une multitude d'autres méthodes pour l'apprentissage supervisé

Arbres de décision

Modèles probabilistes

Forêts aléatoires

Support vector machine (SVM)

*Gradient boosting (XGBoost)*

Classificateur bayésien

**Conclusion:** les *success stories* de l'apprentissage profond sont souvent liées à des ressources immenses

**Conclusion:** en tout temps, il est pertinent de rester ouvert à d'autres méthodes

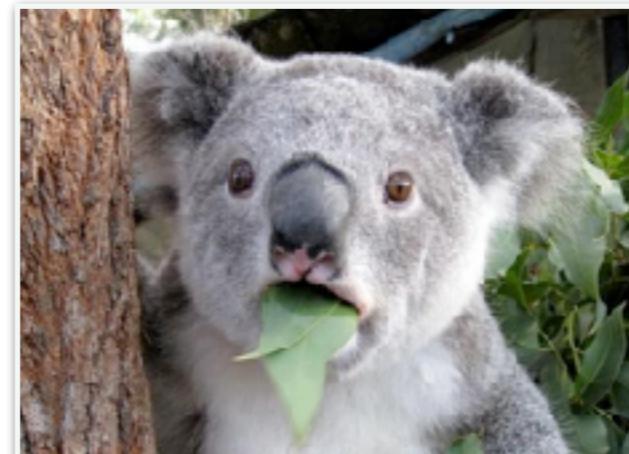
# Exemple récent de réseau profond: Dall-E 2



**Nombre de paramètres à apprendre: plus de 3.5 milliards de paramètres**

**Nombre de données: environ 650 millions images**

**Temps d'entraînement: information malheureusement très difficile à trouver, mais très conséquent**



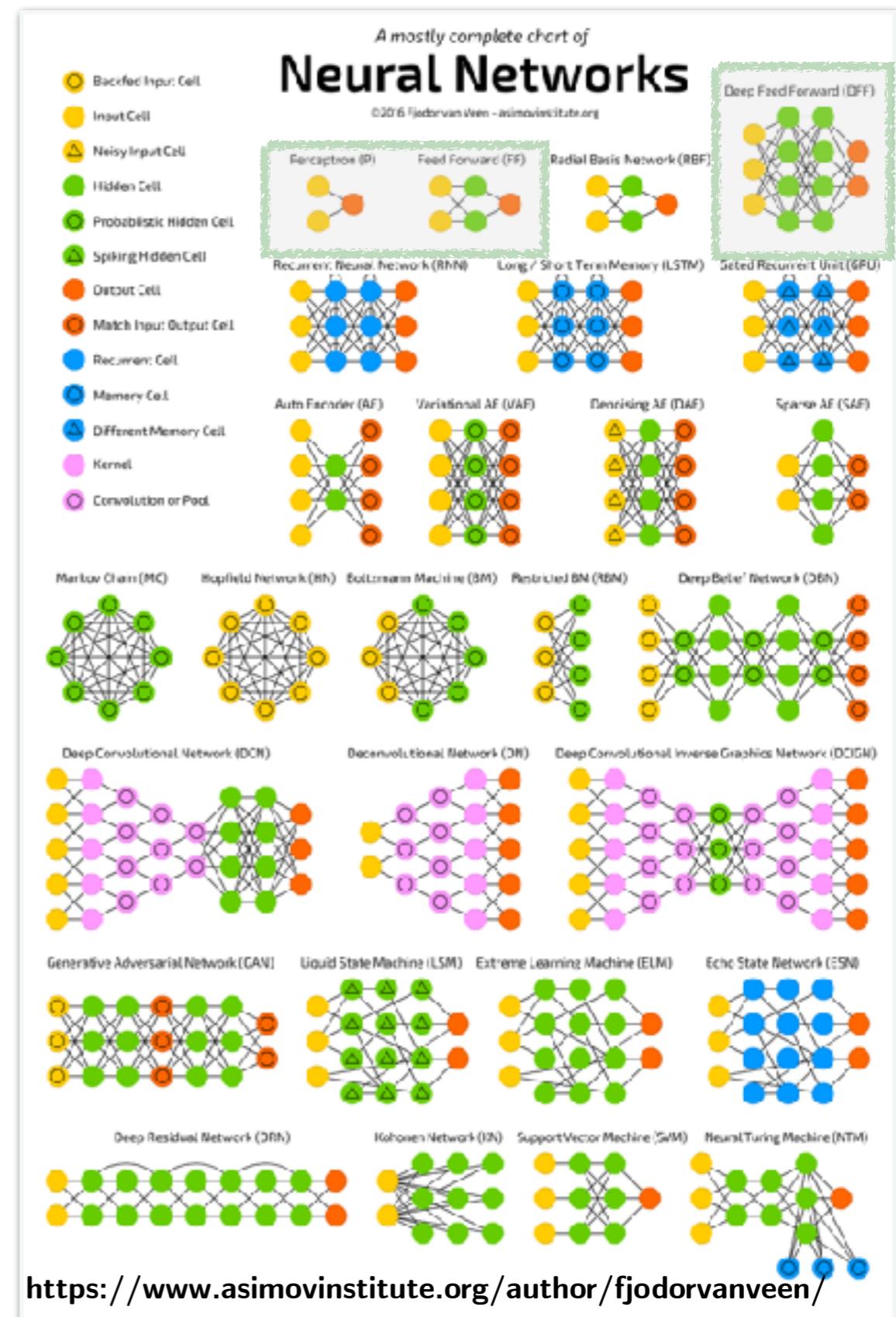
# Table des matières

## Réseaux de neurones

- ✓ 1. Concepts principaux des réseaux de neurones
- ✓ 2. Extension de la régression logistique
- ✓ 3. Notations standards des réseaux de neurones
- ✓ 4. Exécution de la *backpropagation* du gradient
- ✓ 5. Principes des fonctions d'activation
- ✓ 6. Initialisation des paramètres

## Apprentissage profond

- ✓ 1. Limitation des petits réseaux de neurones
- ✓ 2. Principes de l'apprentissage profond
- ✓ 3. Profondeur d'un réseau de neurones
- ✓ 4. Introduction aux hyper-paramètres
- ✓ 5. Conseils pratiques



# Synthèse des notions vues

## Réseaux de neurones

**Définition:** ensemble de neurones organisés en un réseau

**Intérêt:** chaque neurone a ses paramètres et contribue à la prédiction

**Besoin:** activations non-linéaires

**Approximateur universel:** capacité d'approximer n'importe quelle fonction

**Entraînement:** descente de gradient et passe en arrière

**Difficulté:** la fonction de coût n'est plus convexe

**Conséquence:** utilisation de mécanismes pour faciliter l'apprentissage

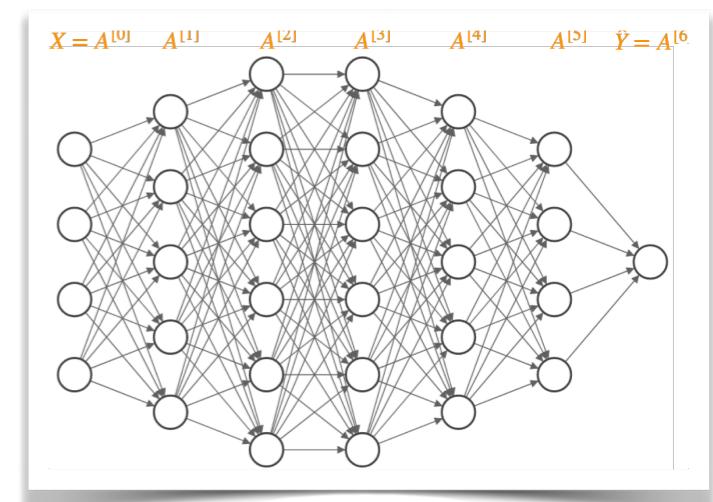
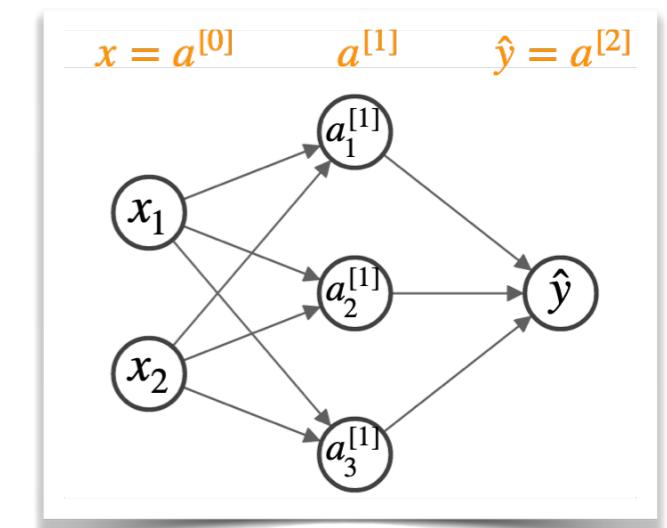
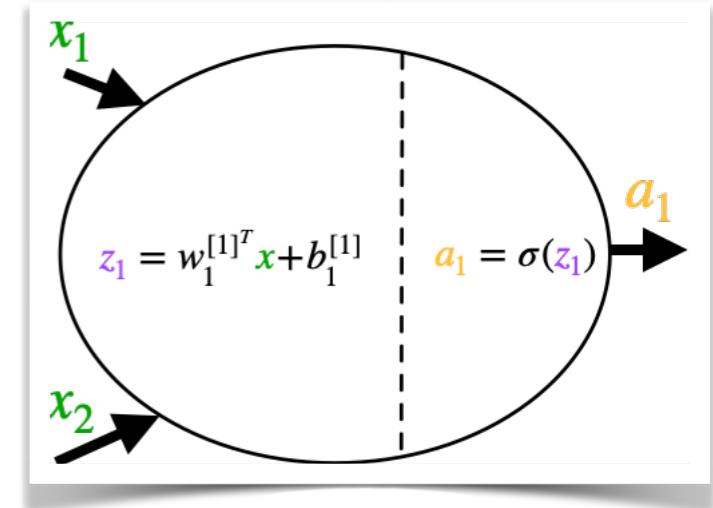
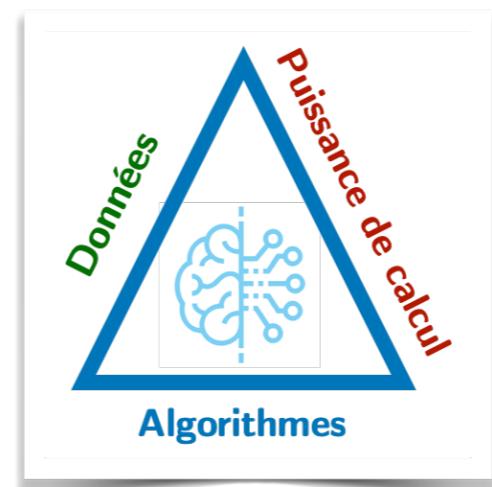
## Apprentissage profond

**Définition:** réseau de neurones ayant plus d'une couche cachée

**Intérêt:** permet de réaliser des approximations plus précises

**Trois ingrédients essentiels:**

- (1) Une méthode d'apprentissage
- (2) Un nombre conséquent de données labellisées
- (3) De la puissance de calcul



# Aller plus loin...

## Cours donnés à Polytechnique

**INF8460:** Traitement automatique de la langue naturelle (Amal Zouaq)

**INF8245E:** Machine learning (Sarah Chandar)

**INF8225:** I.A.: techniques probabilistes et d'apprentissage (Chris pal)



## Cours en ligne

**FastAI:** Practical deep learning for coders



**Coursera:** Machine learning (Andrew Ng)



**Coursera:** Deep learning (Andrew Ng)



<https://course.fast.ai/>

<https://www.coursera.org/learn/machine-learning>

<https://www.coursera.org/specializations/deep-learning>

## Autres ressources

Enormément de ressources sur le web (blog, vidéo youtube, tutoriel, etc.)



**CodeEmporium**

55 k abonnés • 115 vidéos

Everything new and interesting in Machine Learning, Deep Learning, Data Science, & Artificial Intelligence.  
Hoping to build a ...

[https://www.youtube.com/results?search\\_query=code+emporium](https://www.youtube.com/results?search_query=code+emporium)

Machine Learning Research  
Should Be Clear, Dynamic and Vivid.  
**Distill** Is Here to Help.



A DISTILL  
Machine Learning  
research made  
accessible



DISCOVER  
Machine Learning  
research made  
digestible



READ  
Machine Learning  
research made  
digestible

<https://distill.pub/>

# Exemples de questions d'examen

## Théorie

1. Expliquer ce qu'est un réseau de neurones comparé à une régression logistique
2. Expliquer l'importance d'avoir une fonction d'activation non-linéaire
3. Expliquer l'intérêt d'initialiser aléatoirement les paramètres
4. Expliquer la différence entre paramètres et hyper-paramètres

## Pratique

1. Indiquer le nombre de paramètres à entraîner pour un réseau de neurones donné
2. Indiquer la taille des matrices dans une représentation matricielle
3. Donnez les équations de la forward pass pour un réseau de neurones donné





DALLE: *Neural network, digital art*