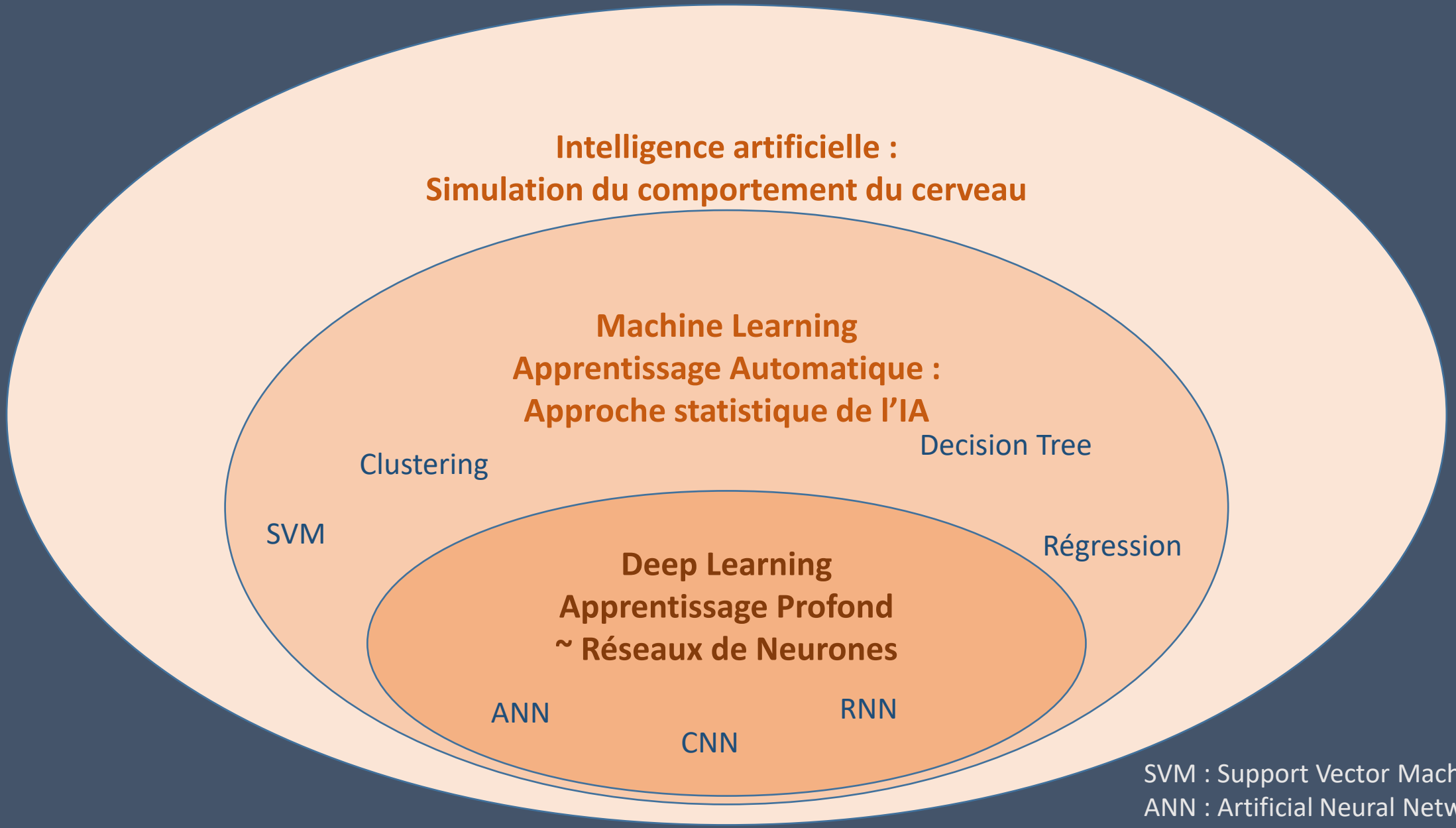
Réseau de neurone : le perceptron multicouche

- ✓ Le neurone formel
- ✓ Perceptron multicouche
- ✓ Propagation /rétropropagation
- ✓ Configuration du réseau
- ✓ Evolution de l'apprentissage

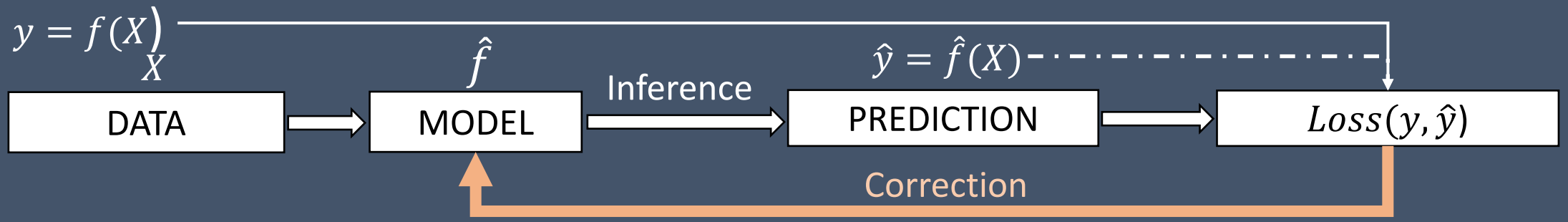
Françoise Bouvet

francoise.bouvet@ijclab.in2p3.fr



SVM : Support Vector Machine
ANN : Artificial Neural Network
CNN : Convolution NN
RNN : Recurrent NN

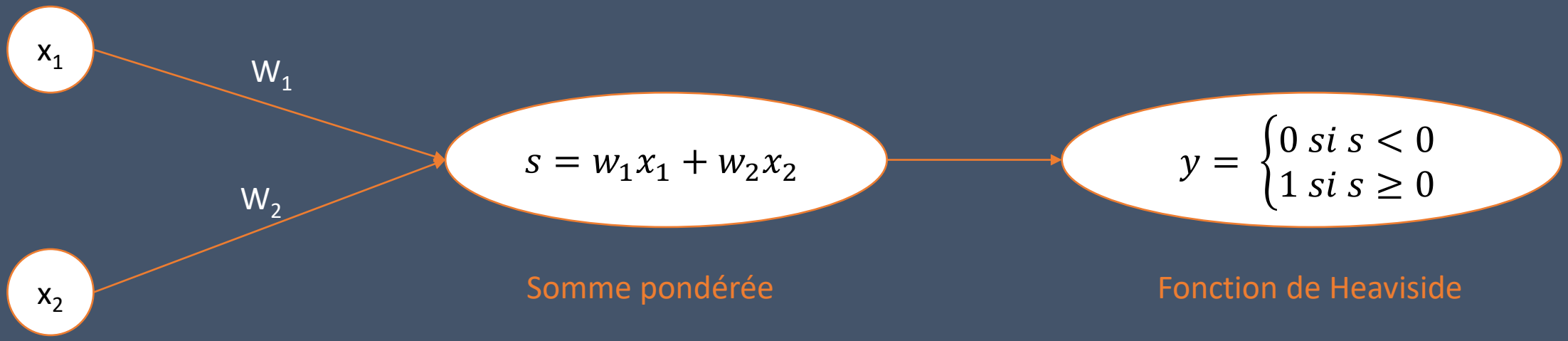
Apprentissage supervisé



- $\hat{f}$ est une approximation de f
- $\hat{f}$ est implémentée par un réseau de neurones (profond)
- $\hat{f}$ peut approximer une fonction f non linéaire

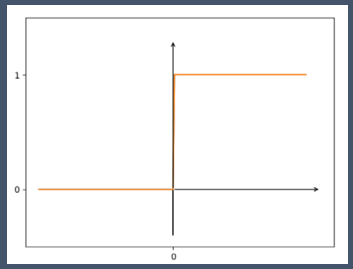
Le neurone formel

Le neurone formel - McCulloch & Pitts, 1943

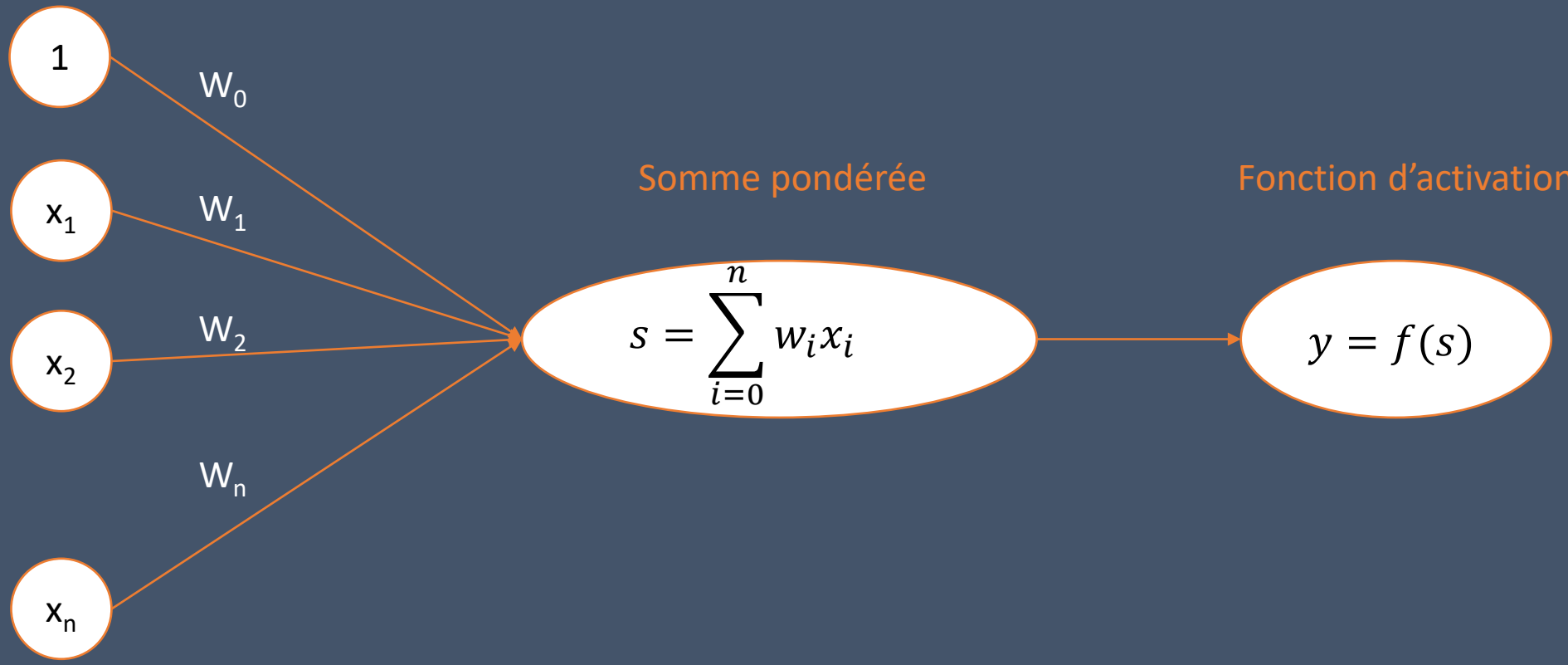


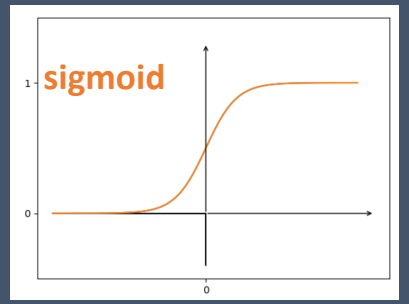
Somme pondérée

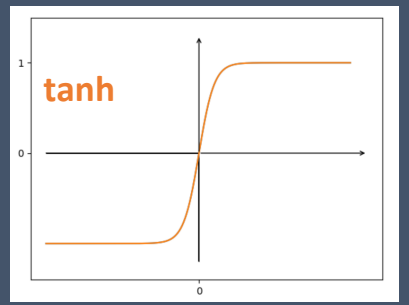
Fonction de Heaviside



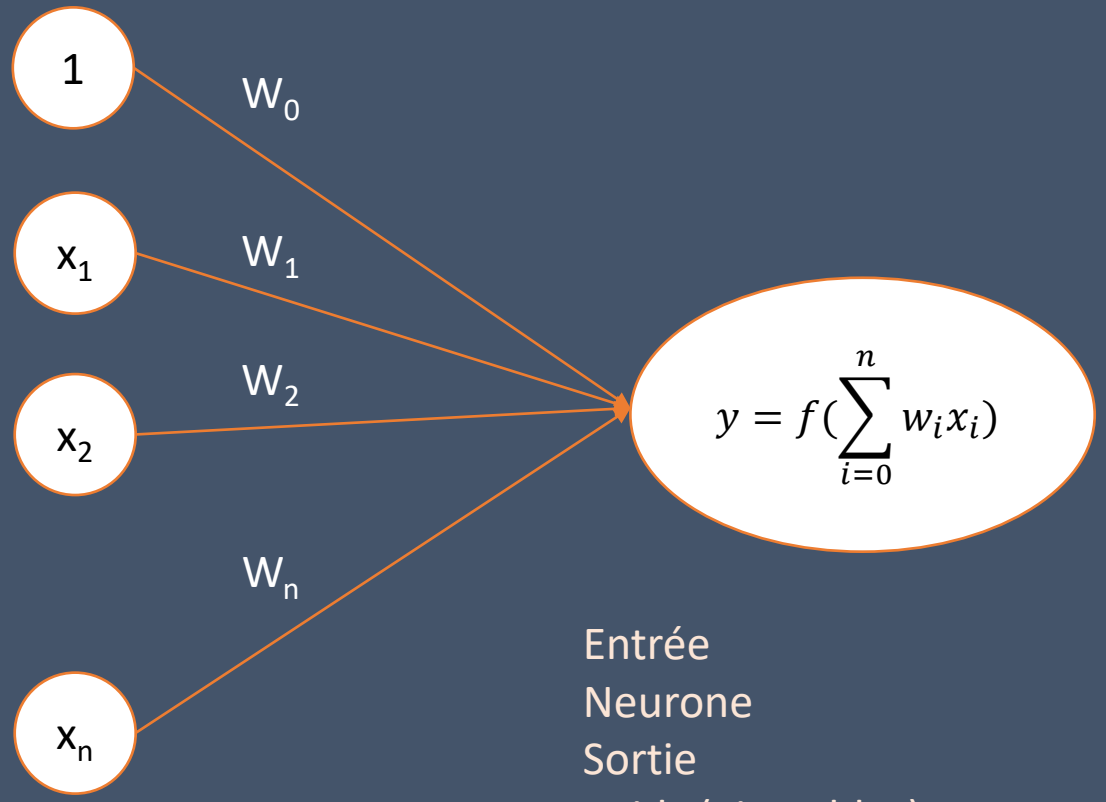
Le neurone formel - généralisation

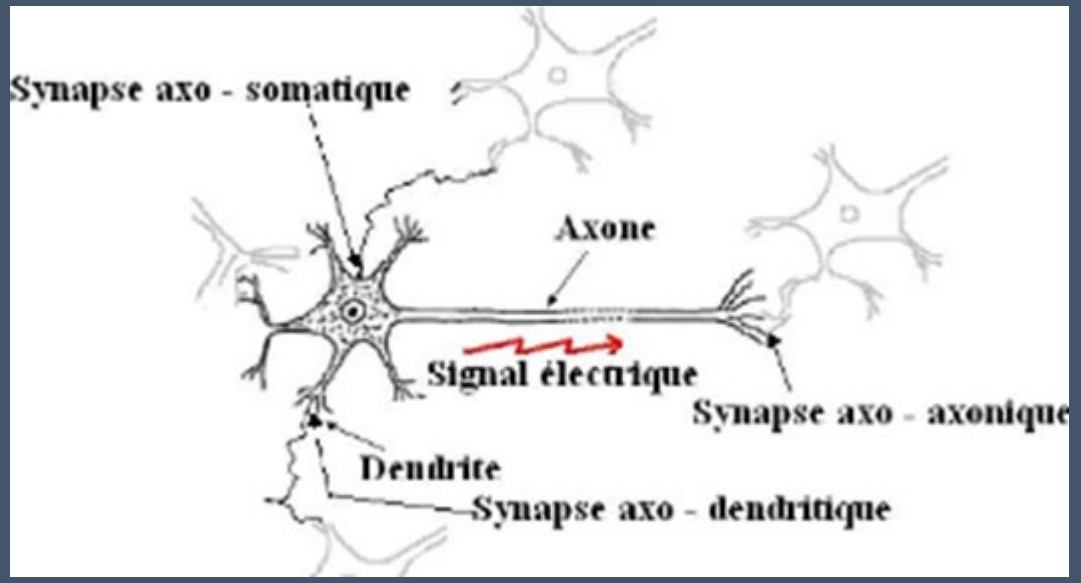






Neurone formel – Neurone biologique





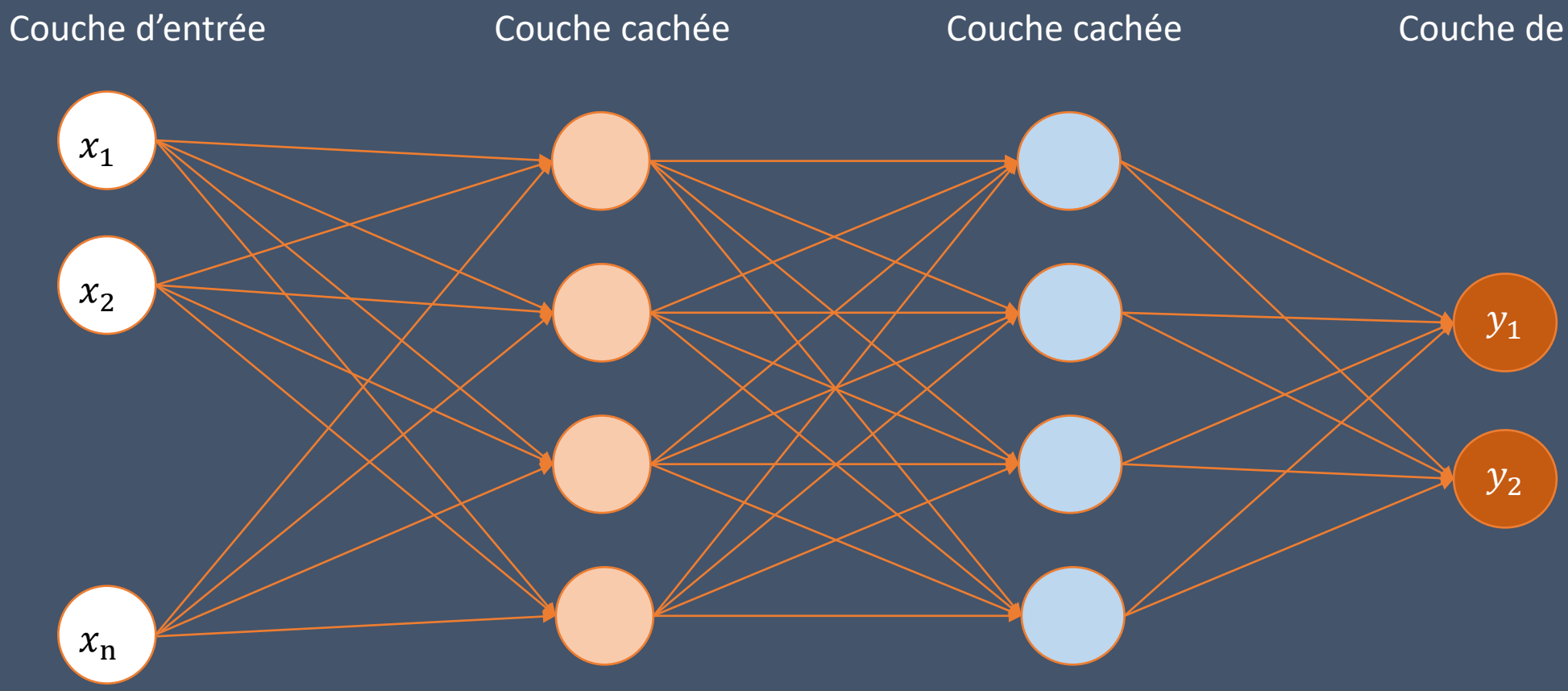
Entrée
Neurone
Sortie
Poids (ajustables)
Fonction d'activation

→ Dendrites
→ Corps cellulaire
→ Axone
→ Synapse (plasticité synaptique)
→ Potentiel postsynaptique

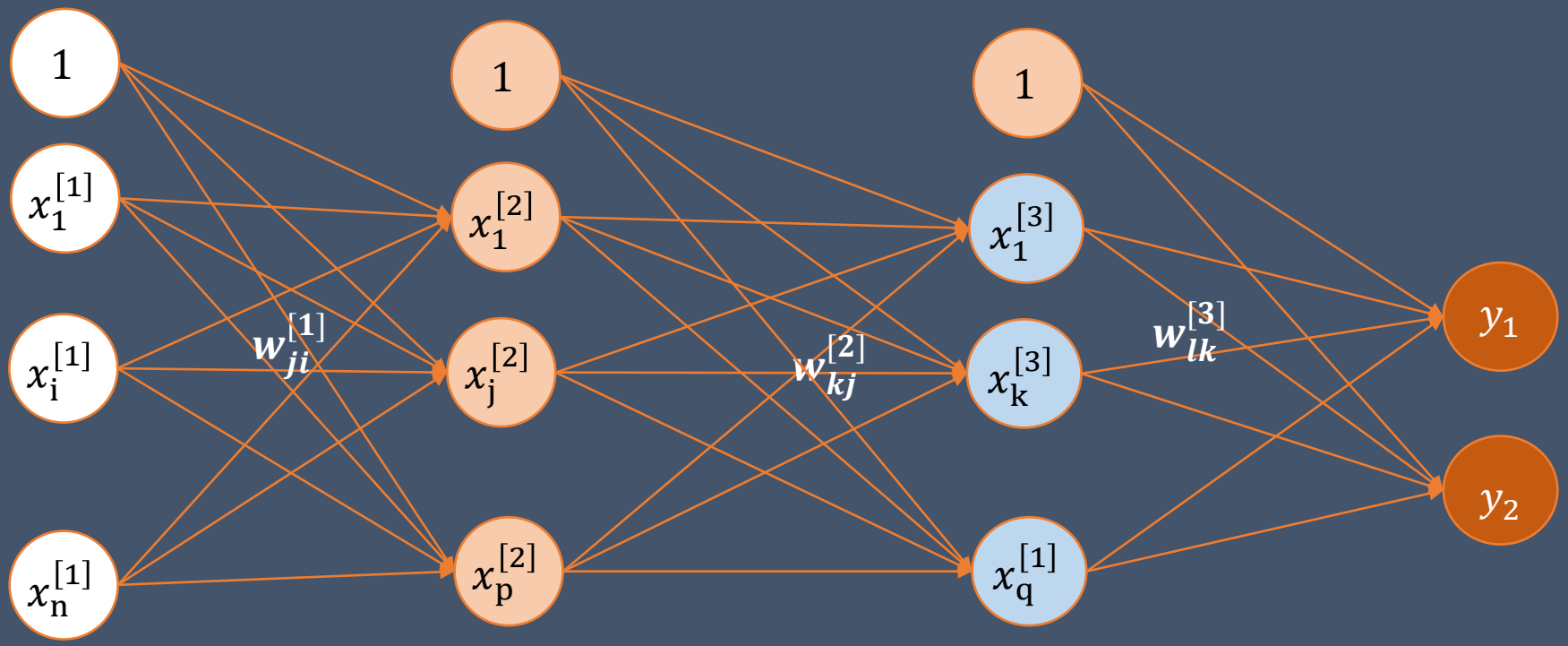
➢ Gardez à l'esprit qu'il s'agit d'une simplification ENORME

Le perceptron multicouche (réseau de neurones dense)

Perceptron multicouche (MultiLayer Perceptron, MLP)



Propagation (feed forward)



$$s_j^{[2]} = \sum_{i=0}^{n} w_{ji}^{[1]}\, x_i^{[1]} \qquad x_j^{[2]} = f\left(s_j^{[2]}\right)$$

$$s_k^{[3]} = \sum_{j=0}^{p} w_{kj}^{[2]}\, x_j^{[2]} \qquad x_k^{[3]} = f\left(s_k^{[3]}\right)$$

$$y_l = \sum_{k=0}^{q} w_{lk}^{[3]}\, x_k^{[3]}$$

Notation matricielle

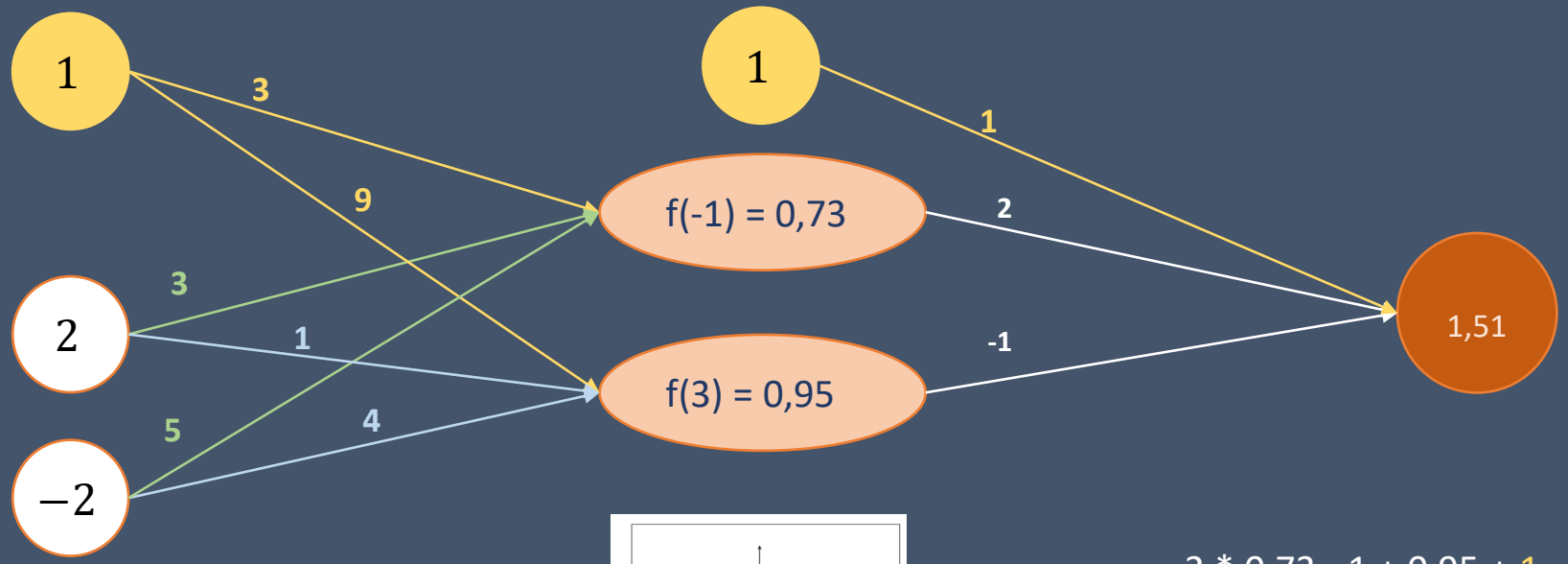
Couches intermédiaires :

$$X^c = f\left(W^{c-1} X^{c-1}\right)$$

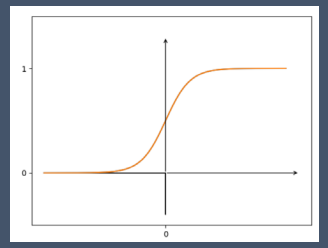
Couche de sortie :

$$Y = W^{C-1} X^{C-1}$$

Propagation : exemple



2 * 3 - 2 * 5 + 3 = -1
2 * 1 - 2 * 4 + 9 = 3



2 * 0,73 - 1 + 0,95 + 1 = 1,51

Notation matricielle : $X^2 = f(W^1 X^1)$ $\quad Y = W^2 X^2$

$$\begin{pmatrix} 3 & 3 & 5 \\ 9 & 1 & 4 \end{pmatrix}\begin{pmatrix} 1 \\ 2 \\ -2 \end{pmatrix}=\begin{pmatrix} -1 \\ 3 \end{pmatrix} \qquad f\begin{pmatrix} -1 \\ 3 \end{pmatrix}=\begin{pmatrix} 0{,}73 \\ 0{,}95 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0{,}73 \\ 0{,}95 \end{pmatrix} = 1{,}51$$

Rétropropagation du gradient

Fonction de coût : $C(y, \hat{y}) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$

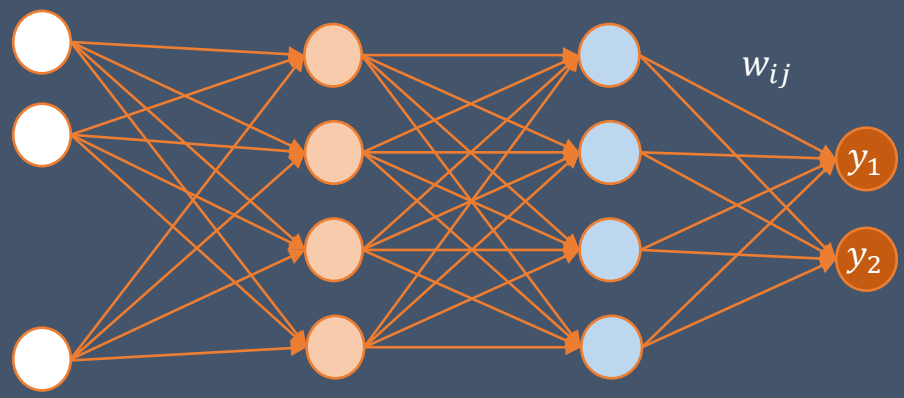
Algorithme de rétropropagation du gradient :

1. Calculer $\frac{\partial C}{\partial \omega_{ij}}$

2. Modifier rétroactivement les poids

$$w_{ij} \coloneqq w_{ij} - \alpha \frac{\partial C}{\partial \omega_{ij}}$$

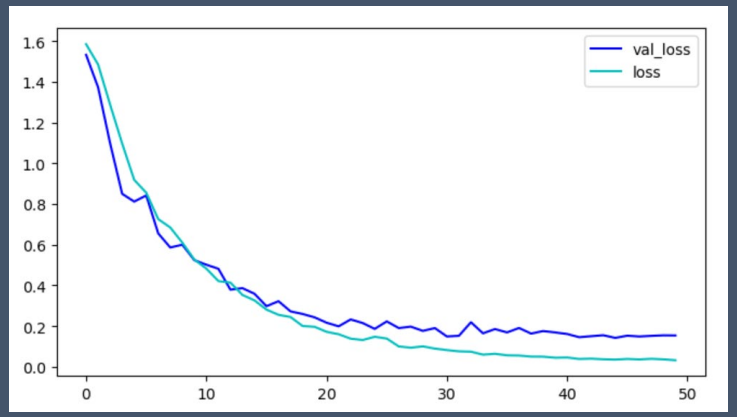
α : taux d'apprentissage (learning rate)



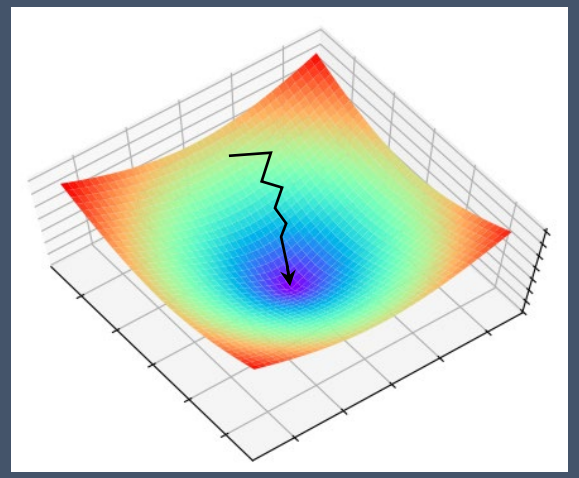
Algorithme

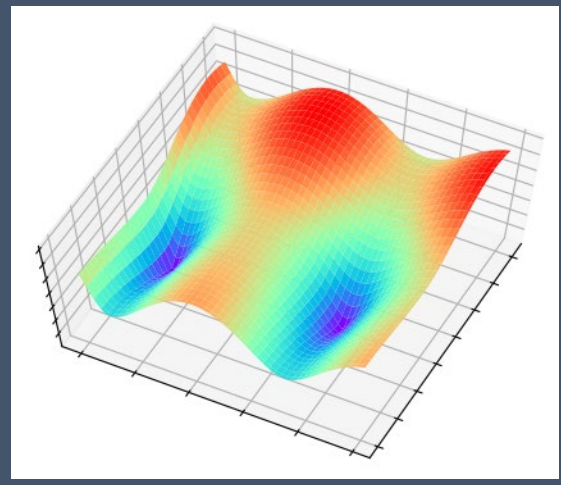
Répéter n fois (n = nombre d'époques)

1. Propager les données d'apprentissage
2. Calculer l'erreur (fonction de coût)
3. Pour chaque couche en partant de la dernière
   - calculer le gradient et transmettre à la couche précédente
   - ajuster les poids



Evolution de la fonction de coût en fonction du nombre d'époques
loss : coût calculé et utilisé pour la rétropropagation
val_loss : coût calculé sur l'échantillon de validation





Fonction de coût dans l'espace des poids

Configuration du réseau

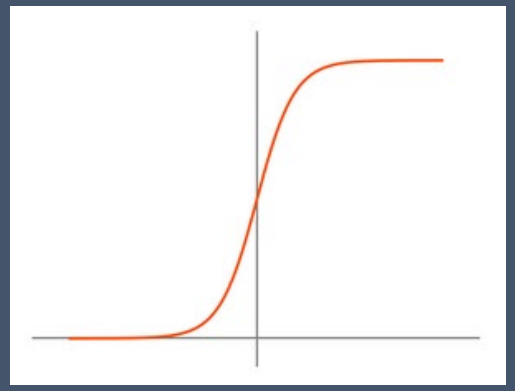
Prétraitement des données

cf présentation précédente

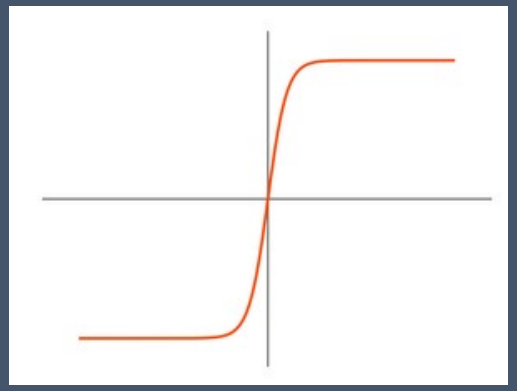
Initialisation des poids

- Surtout pas nul et/ou uniforme
- Initialisation aléatoire, petites valeurs
- Distribution uniforme ou gaussienne

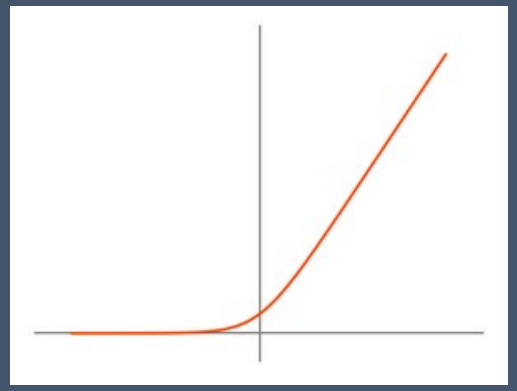
Fonction d’activation



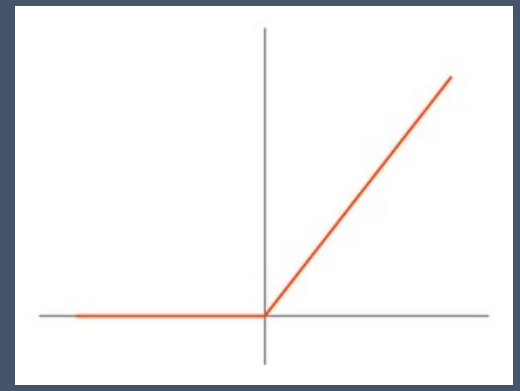
Sigmoid



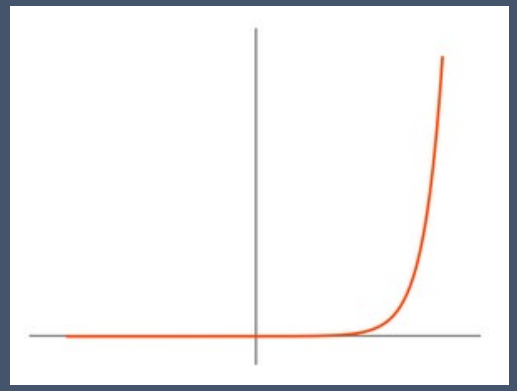
Tanh



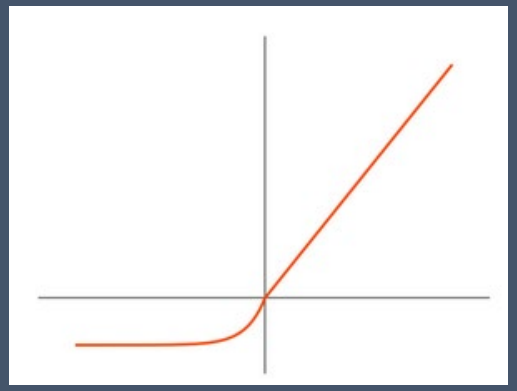
Softplus



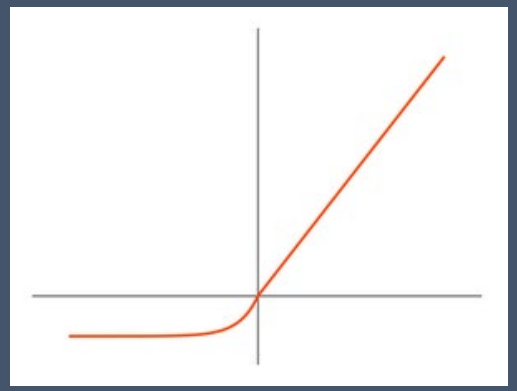
Relu



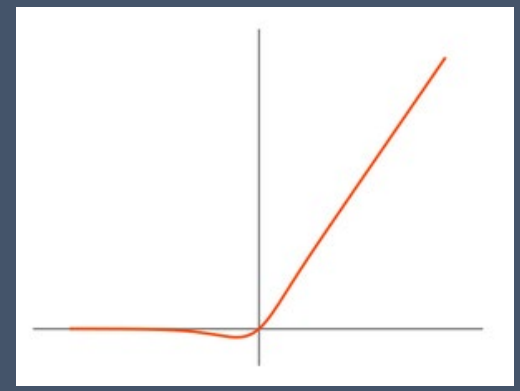
Softmax
(probabilité)



Elu (α=2)



Selu
(α=1.67326324, scale=1.05070098)



Mish

Fonction de coût

$$C(Y, \hat{Y}) = \sum_{m=1}^{M} c(\hat{y}_m - y_m)$$

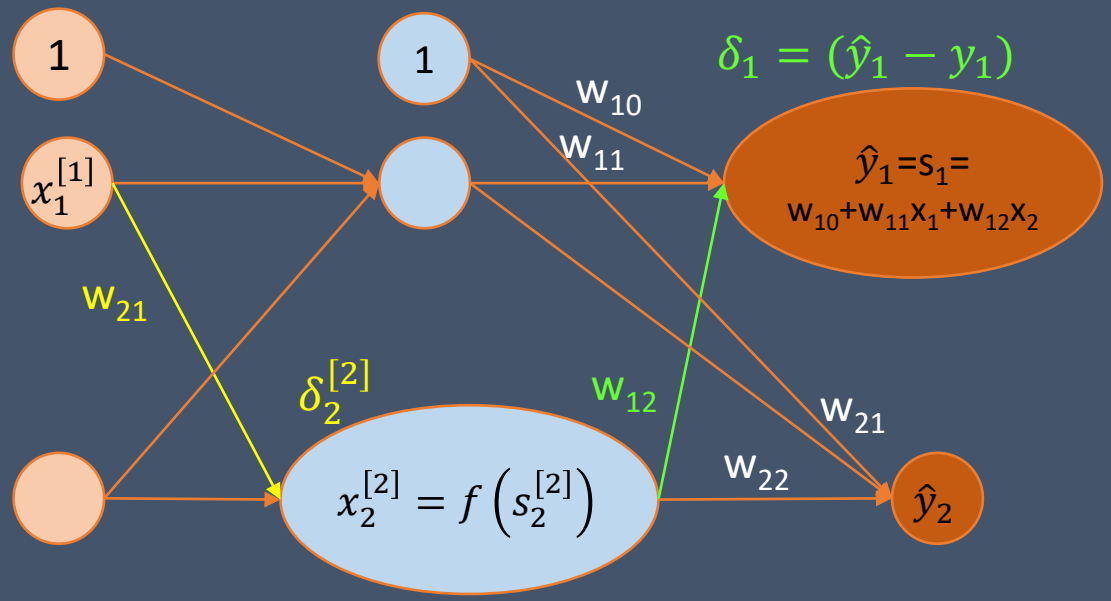
	Fonction d'activation (dernière couche)	Fonction de coût
Régression (quantitatif)	Linéaire	Mean squared error $c(y, \hat{y}) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$
Classification à 2 classes (régresssion logistique)	Sigmoïde	Cross-entropy ce(y, $\hat{y}$) = ((y == 1)? -log($\hat{y}$) : -log(1 - $\hat{y}$))
Classification multi-classes	Soft-max (probabilité)	Cross-entropy (categorical_crossentropy)

Rétropropagation du gradient

Fonction de coût : $C(y,\hat{y}) = \frac{1}{2}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$

Principe :

1. Calculer $\frac{\partial C}{\partial \omega_{ij}}$
2. Modifier les poids $w_{ij} \coloneqq w_{ij} - \alpha \frac{\partial C}{\partial \omega_{ij}}$



$$\frac{\partial C}{\partial w_{12}} = \frac{\partial C}{\partial s_1}\frac{\partial s_1}{\partial w_{12}}$$

$$\frac{\partial s_1}{\partial w_{12}} = x_2$$

$$\frac{\partial C}{\partial s_1} = \delta_1 = \hat{y}_1 - y_1$$

$$w_{12} \coloneqq w_{12} - \alpha\delta_1 x_2$$

$$\frac{\partial C}{\partial w_{21}} = \frac{\partial C}{\partial s_2^{[2]}}\frac{\partial s_2^{[2]}}{\partial w_{21}}$$

$$\frac{\partial s_2^{[2]}}{\partial w_{21}} = x_1^{[1]}$$

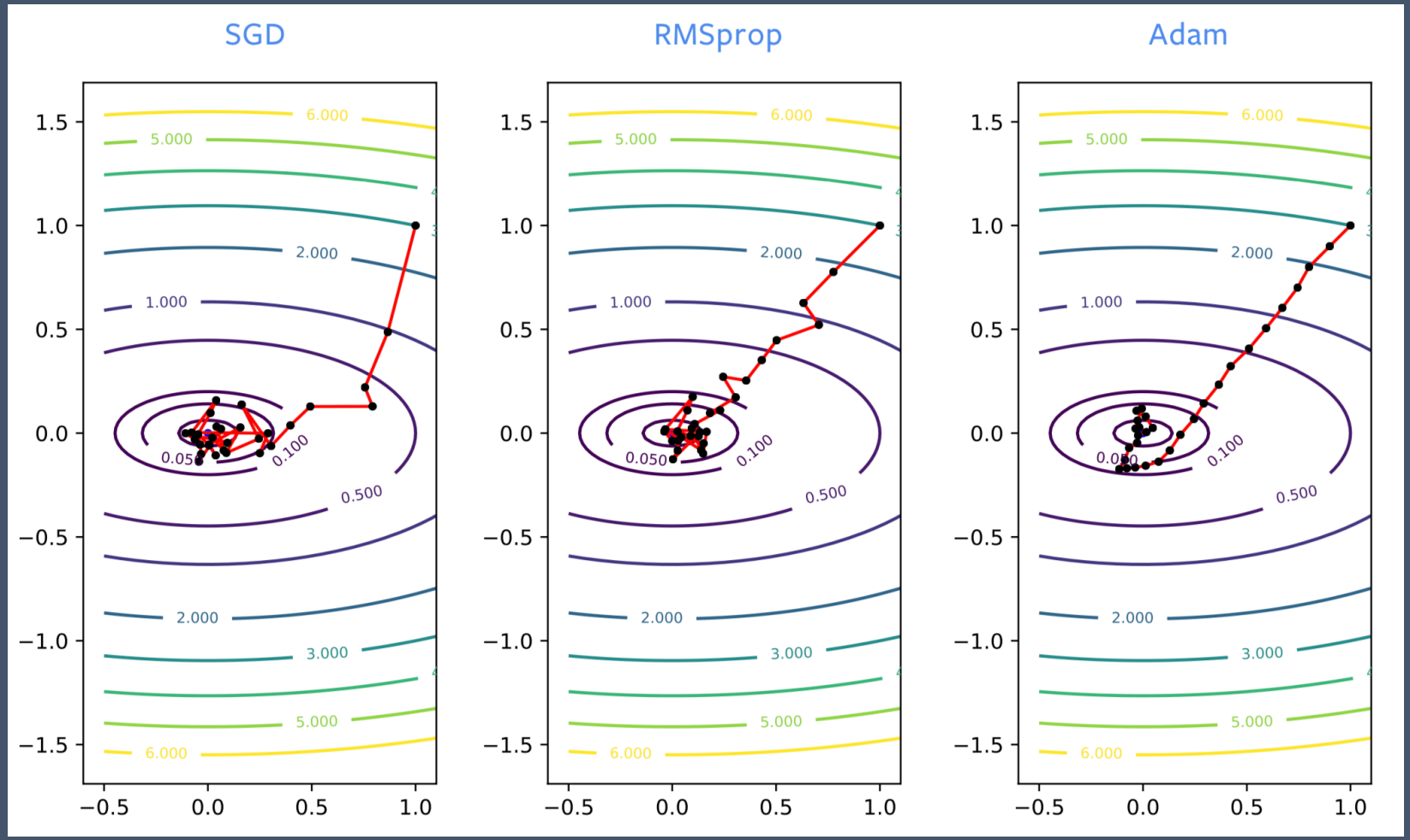
$$\frac{\partial C}{\partial s_2^{[2]}} = \sum_k \delta_k \frac{\partial \delta_k}{\partial s_2^{[2]}}$$

$$\frac{\partial \delta_k}{\partial s_2^{[2]}} = \frac{\sum_j w_{kj} f(s_j^{[2]}) - y_k}{\partial s_2^{[2]}} = w_{k2} f'(s_2^{[2]})$$

$$\frac{\partial C}{\partial s_2^{[2]}} = f'(s_2^{[2]}) \sum_k \delta_k \, w_{k2} = \delta_2^{[2]}$$

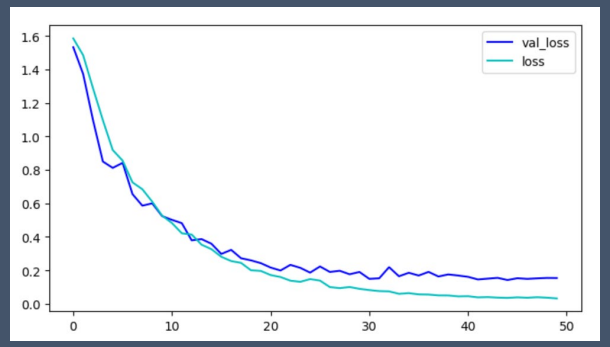
Optimiser

- SGD (Stochastic Gradient Descent)
- SGD + Momentum
- AdaGrad (Adaptive Gradient)
- RMSprop
- Adam
- AdaDelta
- …

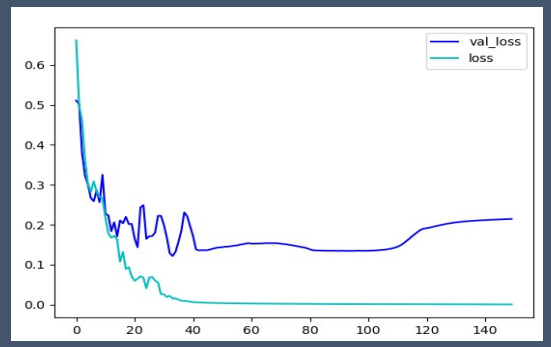


Evolution de l’apprentissage

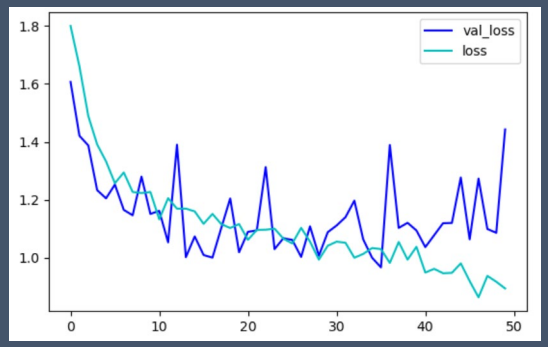
Détecter les problèmes de biais et de de variance



Fonction de coût = f(nb époque)



Problème de variance



Problème de biais

Biais élevé	Sous-apprentissage Underfitting	↗ taille du réseau ↗ nombre de paramètres en entrée (données) ↘ α Plus réguler l'apprentissage
Variance élevée	Sur-apprentissage Overfitting	↘ taille du réseau ↘ nombre de paramètres en entrée ↗ taille de l'échantillon ↗ α Moins réguler l'apprentissage

Régularisation de l'apprentissage (1)

Taux d'apprentissage α (learning rate)

- Diminuer α au cours de l'apprentissage

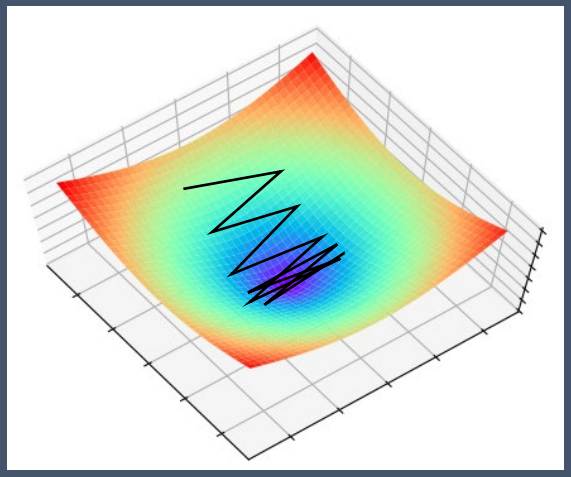
$$w_{ij} \coloneqq w_{ij} - \alpha \frac{\partial C}{\partial \omega_{ij}}$$

Batch

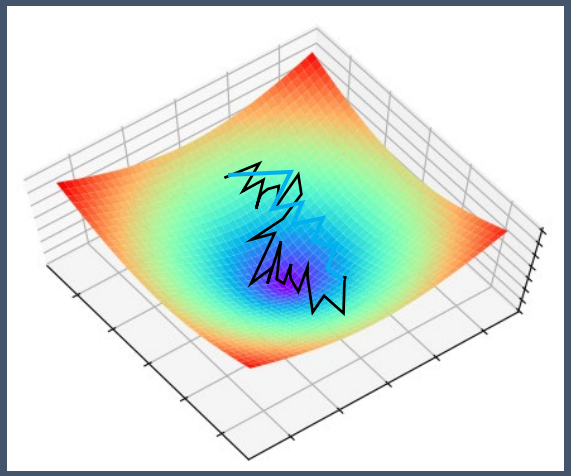
- Apprentissage par paquets

Dropout

- X% des poids sont mis à 0 à chaque époque
- Appliquer couche/couche



α trop grand



Apprentissage par batch

Régularisation de l'apprentissage (2)

Pénalité de la fonction de coût

- Ajout d'un terme dépendant des poids à la fonction de coût
- Evite l'explosion des gradients

  Régularisation **L1 :** $\lambda|w|$ (peu de données)

  Régularisation **L2 :** $\lambda\|w\|^2$

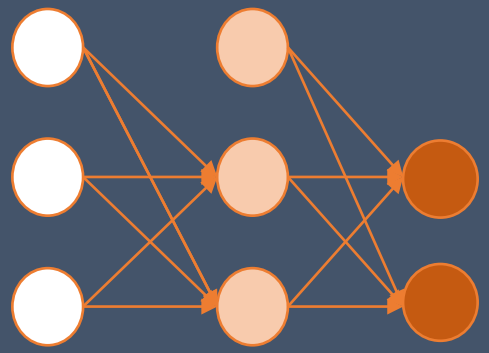
Gradient clipping

- Evite l'explosion des gradients

Evolution de l’apprentissage : règles de base

- ✓ Avoir une évaluation **chiffrée** de la qualité du résultat
- ✓ Commencer par un algorithme **très simple**
- ✓ Tracer les **courbes d’apprentissage** pour détecter un problème de biais, variance, etc.
- ✓ Analyser les échantillons mal classés
- ✓ Ne pas passer trop vite à l’optimisation des paramètres
- ✓ L’optimisation ne doit être réalisée sur l’ensemble de test

Paramètres : résumé



Paramètres calculés

- ✓ Poids

Données du problème

- ✓ Nombre de neurones / couche : entrée/sortie

Hyper-paramètres

Structure du réseau

- ✓ Nombre de couches cachées
- ✓ Nombre de neurones / couche cachée
- ✓ Fonction d'activation / couche cachée

Apprentissage

- ✓ Fonction de coût
- ✓ Initialisation des poids

- ✓ Nombre d'époques
- ✓ Régularisation :
  - α
  - λ ($\lambda \|w\|^2$)
  - Batch
  - Dropout

De la théorie à la pratique …

Keras

API de haut niveau pour créer et entraîner des modéles de « Deep Learning » construite sur TensorFlow

TensorFlow

Code open-source, développé par Google, sous licence Apache depuis 2015 ; fonctionne sur CPU et GPU

https://keras.io/