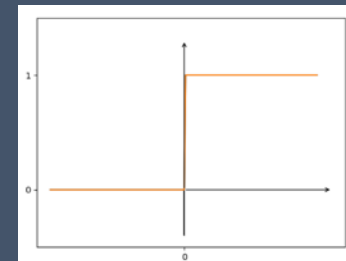
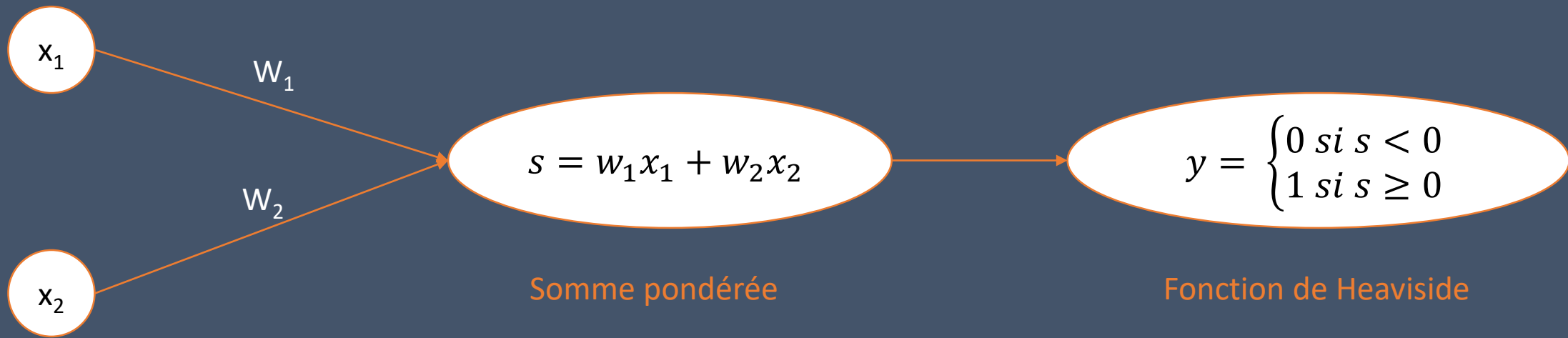


Perceptron multicouche

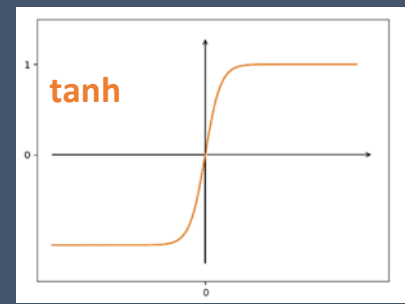
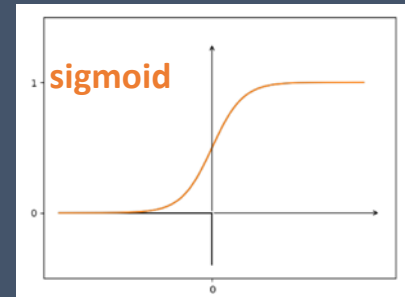
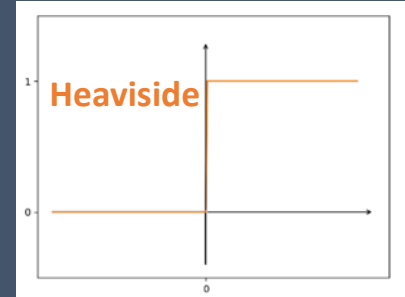
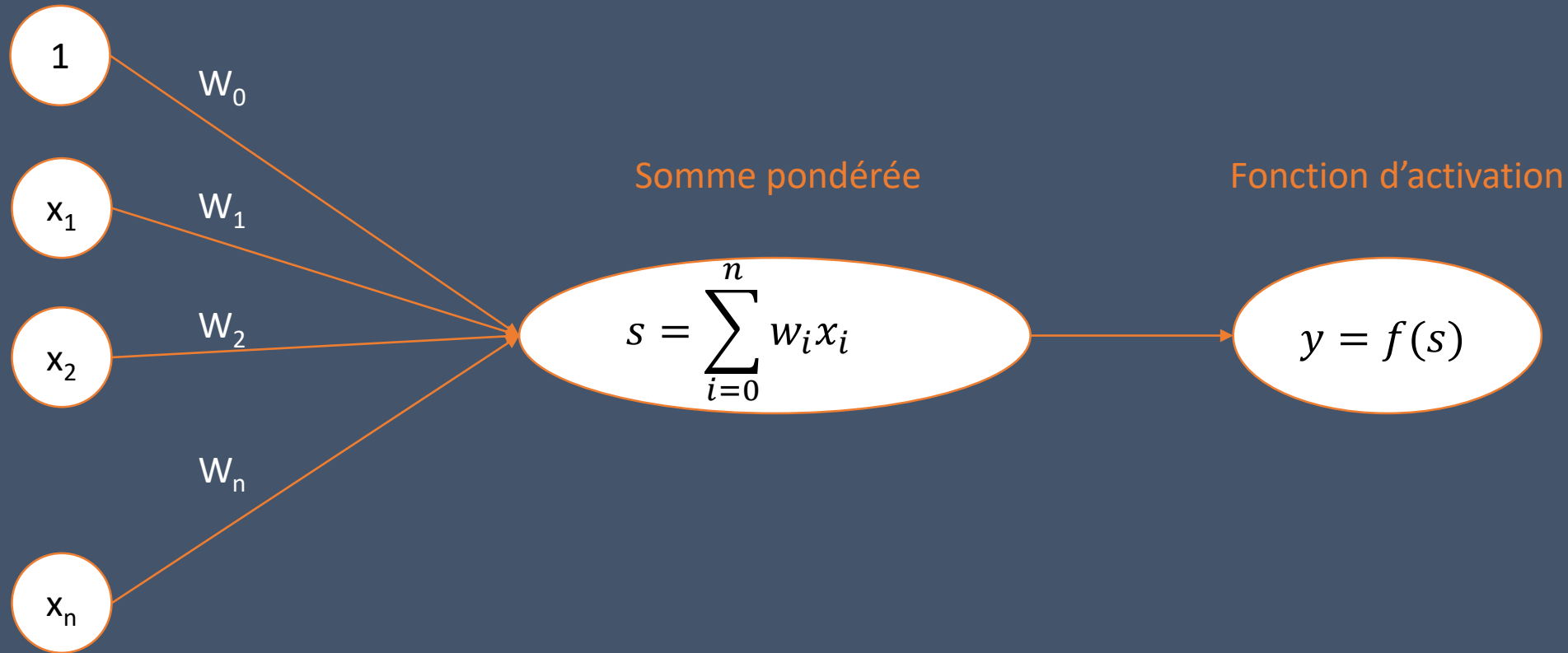
- ✓ Le neurone formel
- ✓ Perceptron multicouche
- ✓ Propagation /rétropropagation
- ✓ Configuration du réseau
- ✓ Evolution de l'apprentissage
- ✓ Autres réseaux de neurones

Françoise Bouvet
francoise.bouvet@ijclab.in2p3.fr

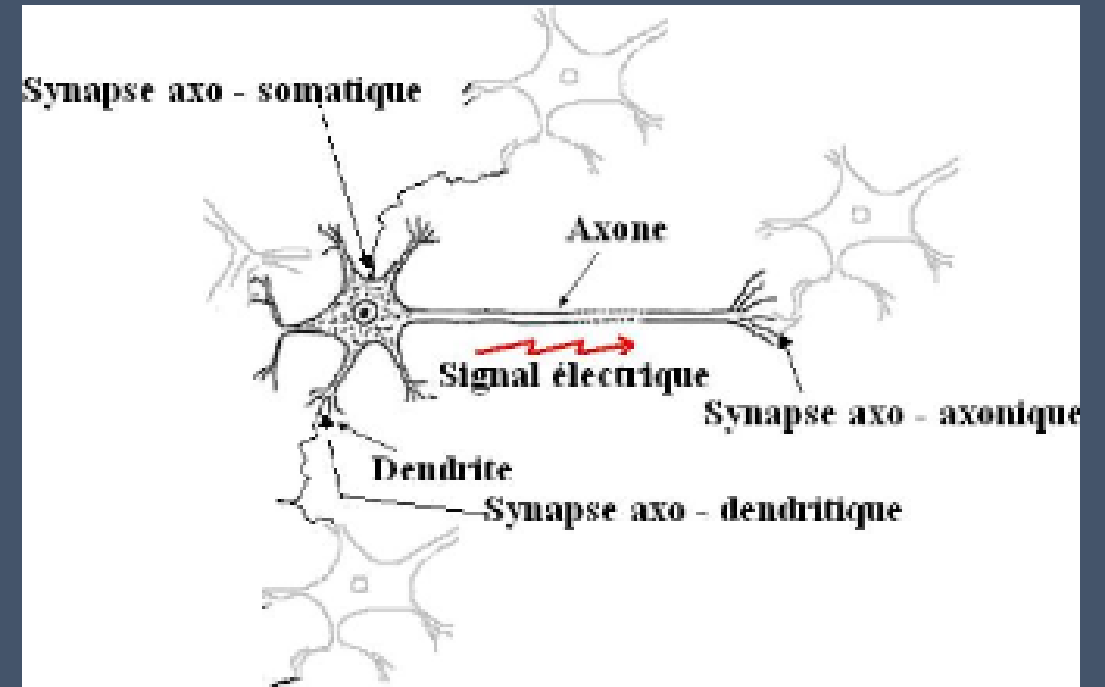
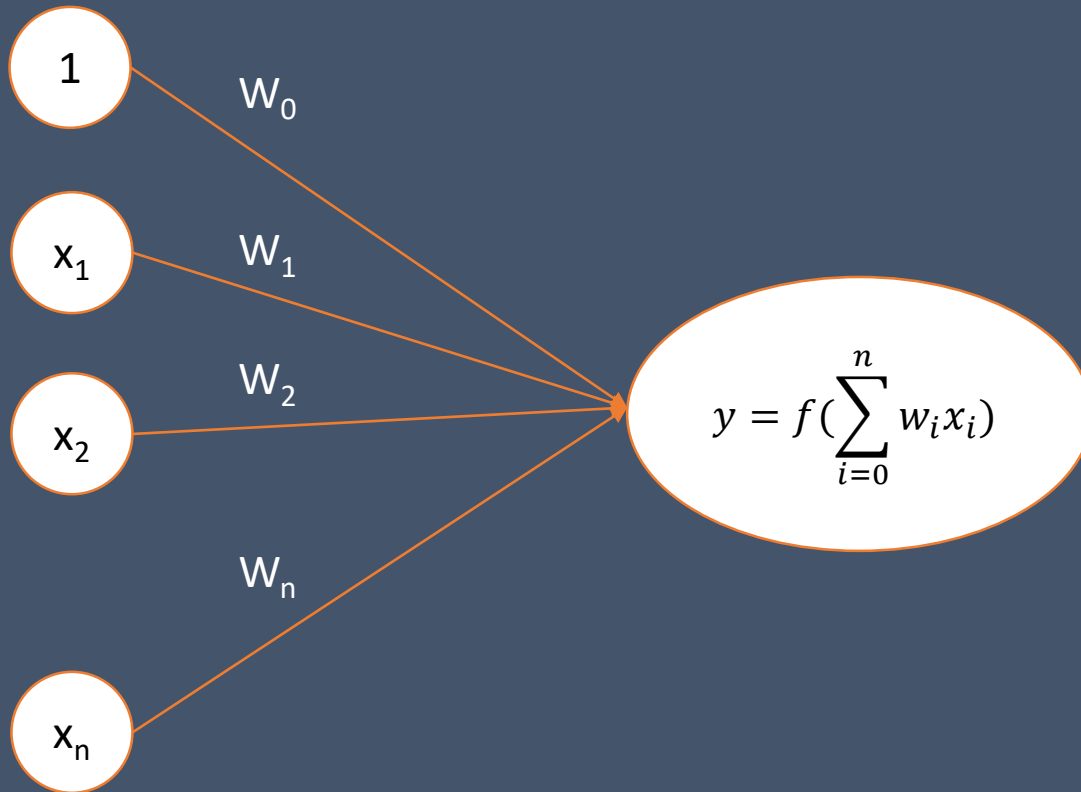
Le neurone formel - McCulloch & Pitts, 1943



Le neurone formel - généralisation



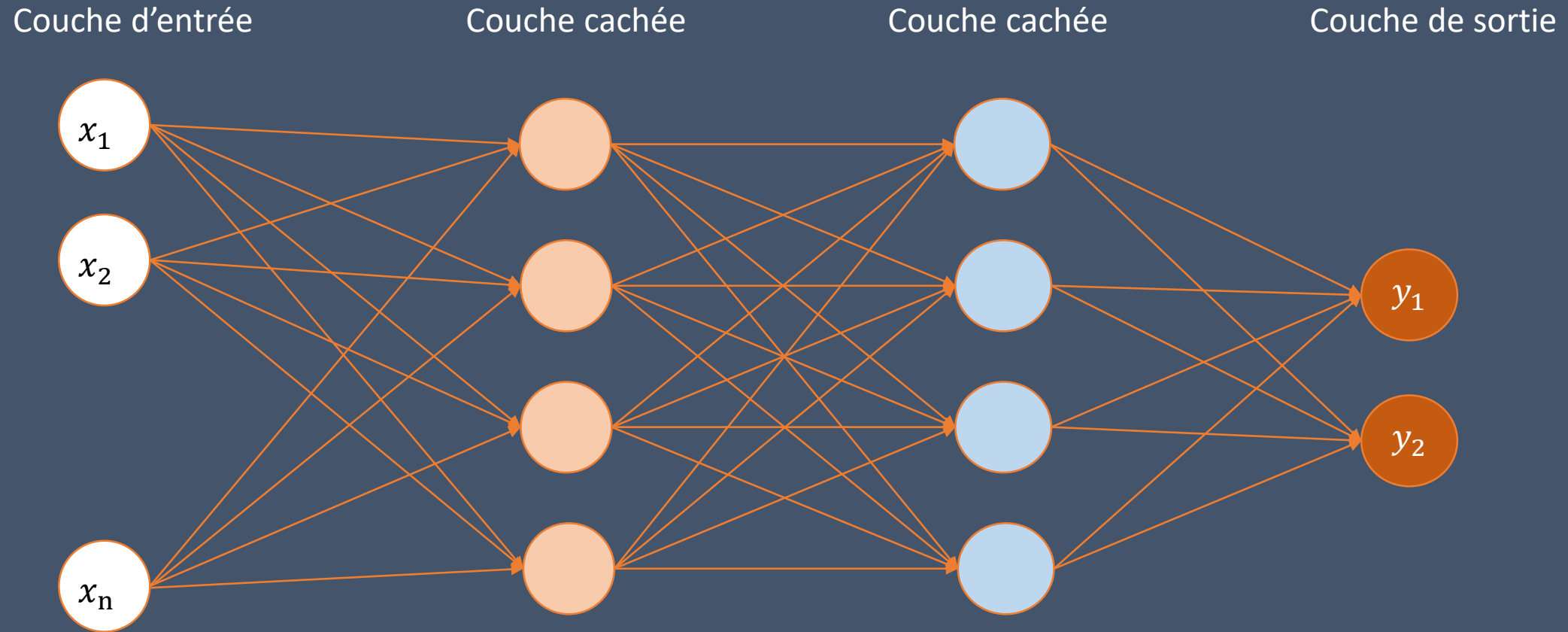
Neurone formel – Neurone biologique



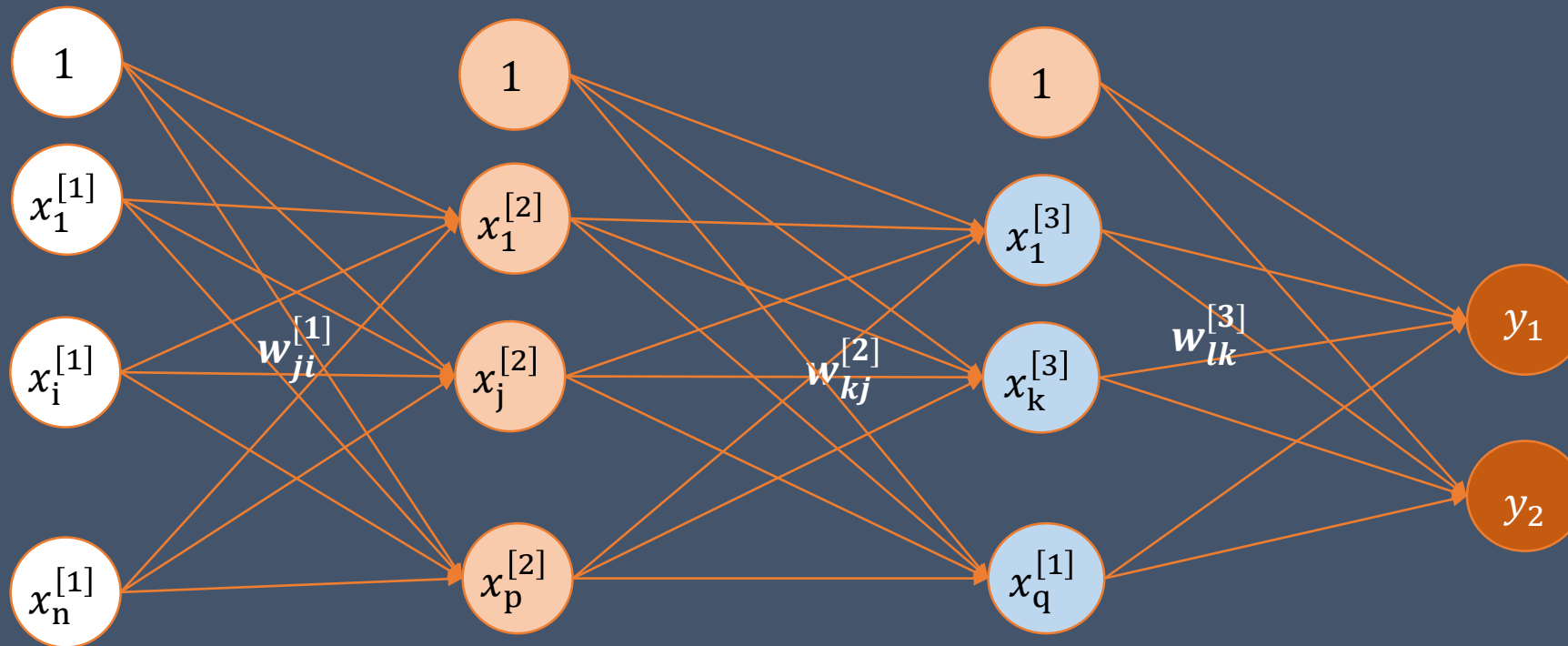
1982 : modèle de Hopfield : réseau récurrent entièrement connecté

1986 : perceptron multicouche + algorithme de rétropropagation du gradient (Rumelhart, Yann le Cun)

Perceptron multicouche (MultiLayer Perceptron, MLP)



Propagation (feed forward)



$$s_j^{[2]} = \sum_{i=0}^n w_{ji}^{[1]} x_i^{[1]}$$

$$x_j^{[2]} = f(s_j^{[2]})$$

$$s_k^{[3]} = \sum_{j=0}^p w_{kj}^{[2]} x_j^{[2]}$$

$$x_k^{[3]} = f(s_k^{[3]})$$

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

Notation matricielle

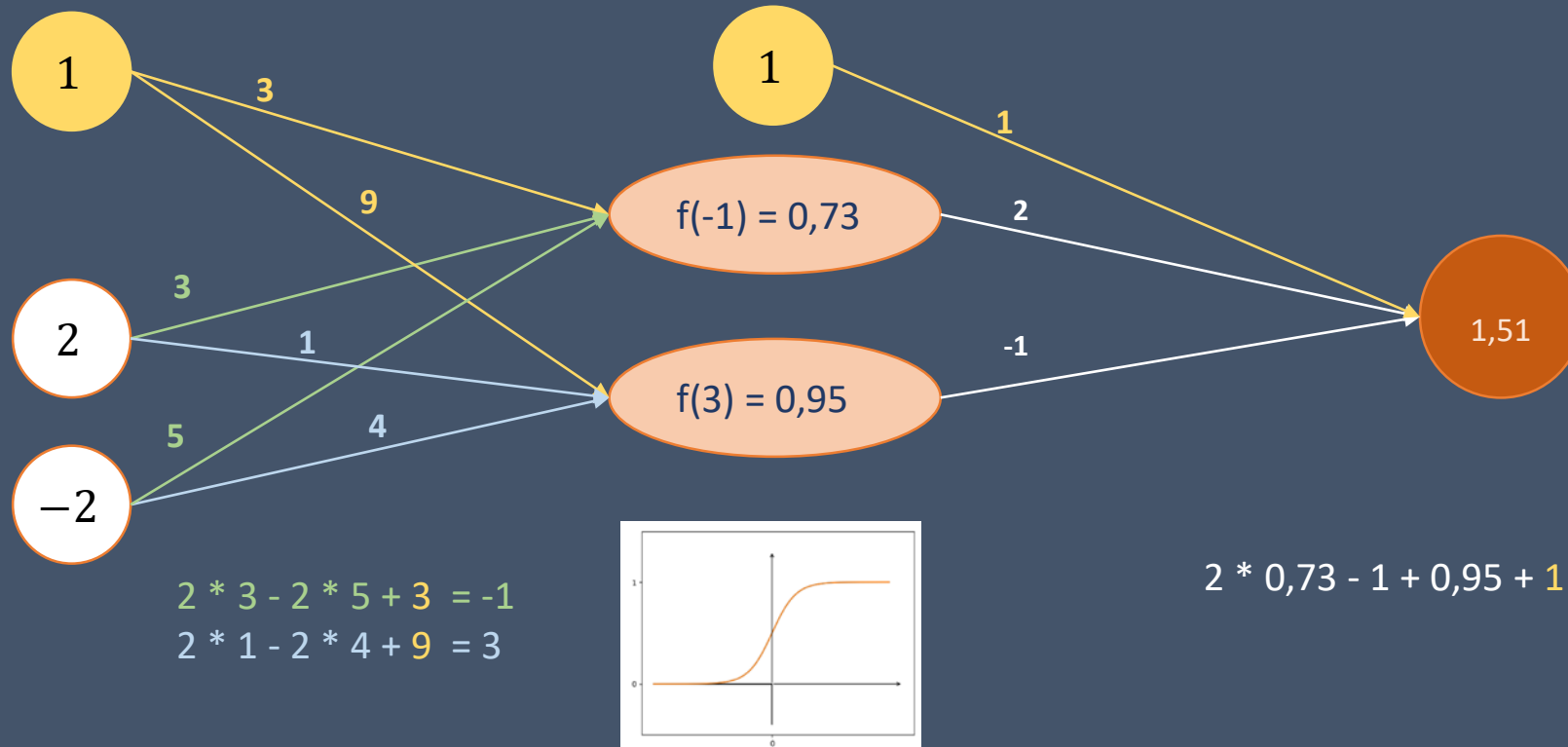
Couches intermédiaires :

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

Couche de sortie :

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

Propagation : exemple



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

$Y = W^2 X^2$

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

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

Rétropropagation du gradient

Fonction de coût :

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

Principe :

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

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

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

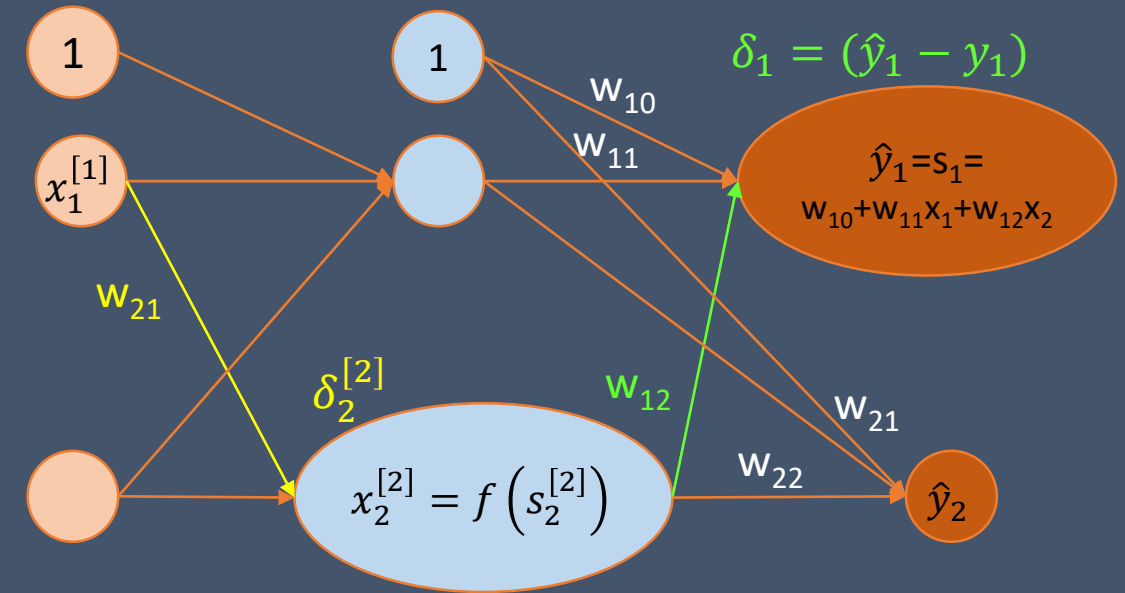
$$\frac{\partial C}{\partial s_1} = \delta_1 = \hat{y}_1 - y_1$$

$$w_{12} := w_{12} - \alpha (\hat{y}_1 - y_1) x_2$$

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

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

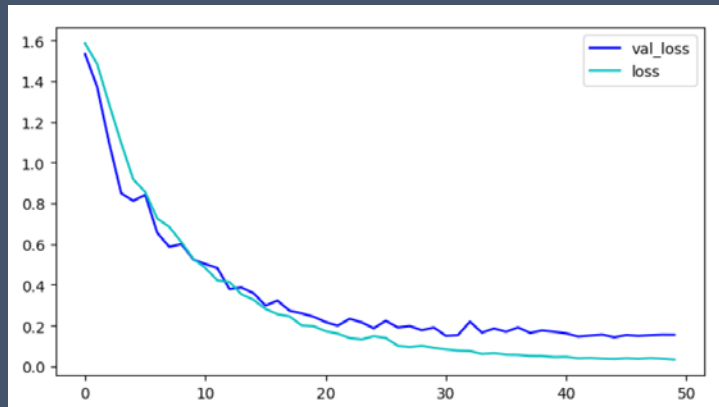
$$\frac{\partial C}{\partial s_2^{[2]}} = \delta_2^{[2]} = f'(s_2^{[2]}) \sum_k \delta_k w_{k2}$$



Algorithme

Répéter n fois (n = nombre d'époques)

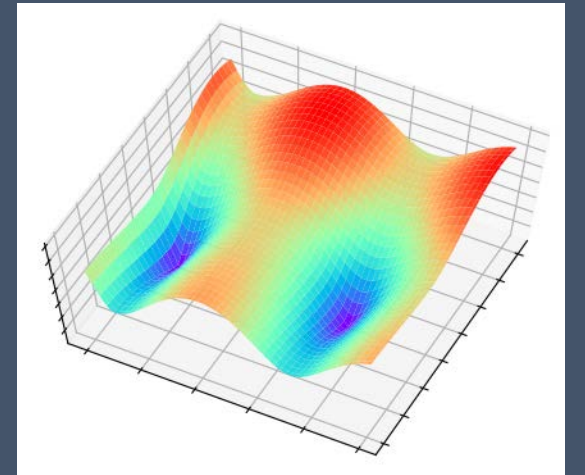
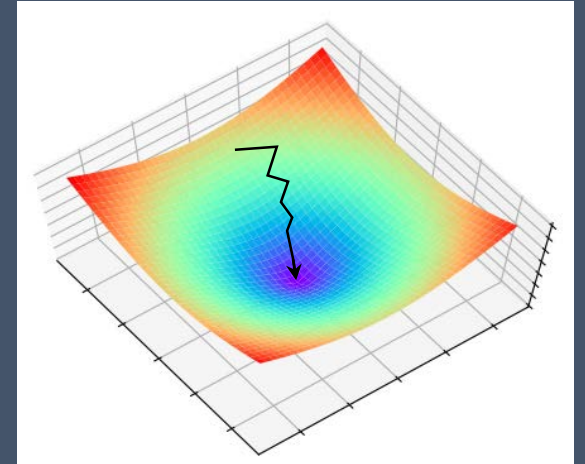
1. Propager les données d'apprentissage
2. Calculer l'erreur (fonction de coût)
3. Pour chaque couche en partant de la dernière
 - calculer le gradient et transmettre à la couche précédente
 - ajuster les poids



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

Preprocessing / configuration

- Centrage et normalisation des données en entrée

- $X = \{x^m\}, m \in \{1, M\}, x^m \in R^N$
- $\mu_i = \frac{1}{M} \sum_{m=1}^M x_i^m, \sigma_i^2 = \frac{1}{M} \sum_{m=1}^M (x_i^m - \mu_i)^2$
- $x_i^m \leftarrow \frac{x_i^m - \mu_i}{\sigma_i}$

- Initialisation des poids

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

- Fonction de coût

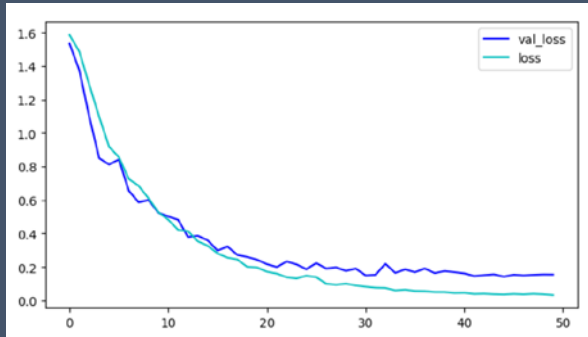
$$C(Y, \hat{Y}) = \sum_{m=1}^M c(\hat{y}_m - y_m)$$

- Régression (quantitatif)
 - Fonction linéaire
 - Mean squared error : $c(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^N (\hat{y}_i - y_i)^2$
- Régression logistique (2 classes)
 - Fonction sigmoïde
 - Cross-entropy : $ce(y, \hat{y}) = ((y == 1)? -\log(\hat{y}) : -\log(1 - \hat{y}))$
- Classification multi-classe
 - Fonction soft max (probabilité)
 - Cross-entropy (categorical_crossentropy)

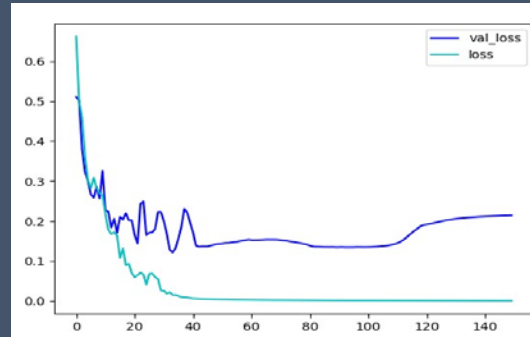
Evolution de l'apprentissage : quelques règles de base

- ✓ 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 doit être réalisée en cross validation, pas sur le test

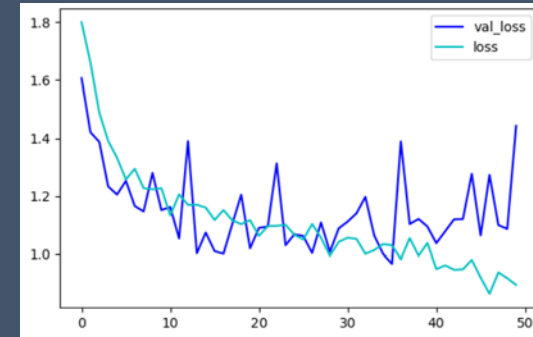
Détecter les problèmes de biais et de de variance



Fonction de coût = $f(\text{nb époque})$



Problème de variance



Problème de biais

Biais élevé	Sous-apprentissage Underfitting	\nearrow taille du réseau \nearrow nombre de paramètres en entrée (données) $\searrow \alpha$ Plus réguler l'apprentissage
Variance élevée	Sur-apprentissage Overfitting	\searrow taille du réseau \searrow nombre de paramètres en entrée \nearrow taille de l'échantillon $\nearrow \alpha$ Moins réguler l'apprentissage

Régularisation de l'apprentissage

Paramètre α

$$w_{ij} := w_{ij} - \alpha \frac{\partial C}{\partial w_{ij}}$$

- Diminuer α au cours de l'apprentissage

Batch

- Apprentissage par paquets

Dropout

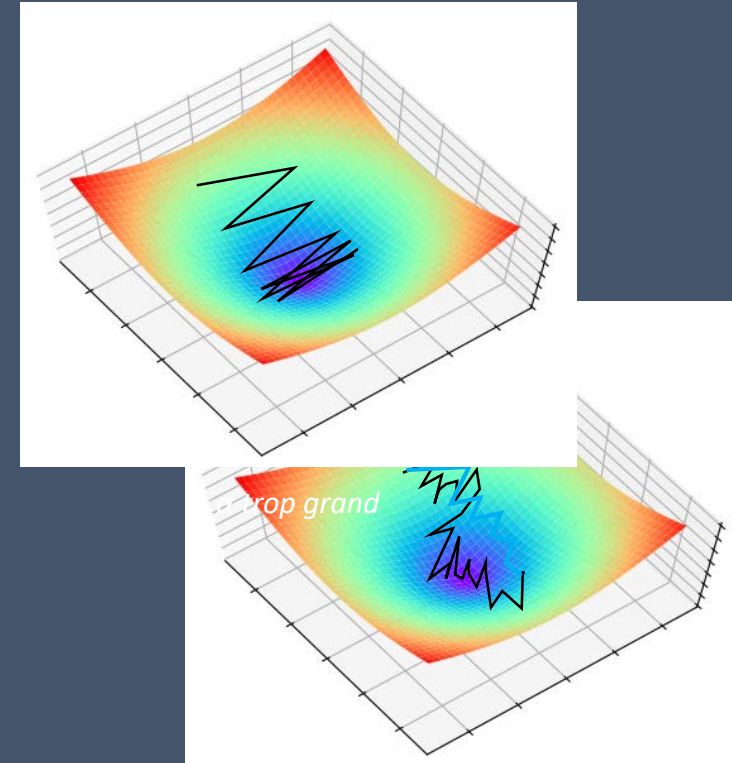
- X% des poids sont mis à 0 à chaque époque
- Appliquer couche/couche

Régularisation L2

- Ajout de $\lambda ||w||^2$ à la fonction de coût
- Evite l'explosion des gradients

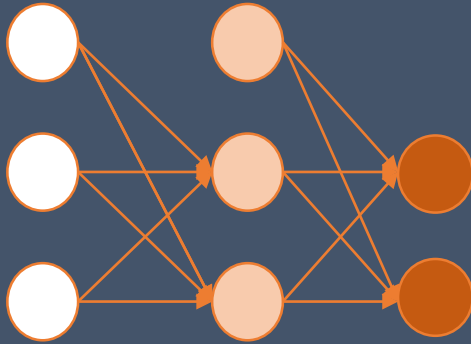
Gradient clipping

- Evite l'explosion des gradients



Apprentissage par batch

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
 - $\lambda (\lambda ||w||^2)$
 - Batch
 - Dropout

A mostly complete chart of Neural Networks

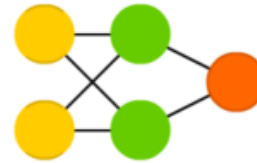
©2019 Fjodor van Veen & Stefan Leijnen asimovinstitute.org

-  Input Cell
-  Backfed Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Capsule Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Gated Memory Cell
-  Kernel
-  Convolution or Pool

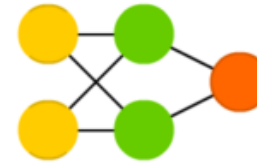
Perceptron (P)



Feed Forward (FF)



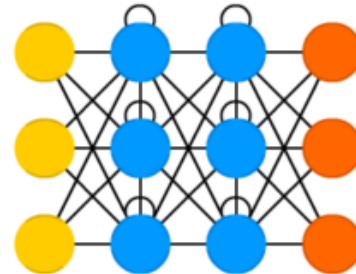
Radial Basis Network (RBF)



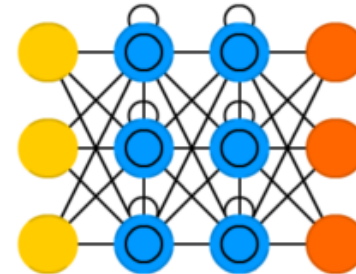
Deep Feed Forward (DFF)



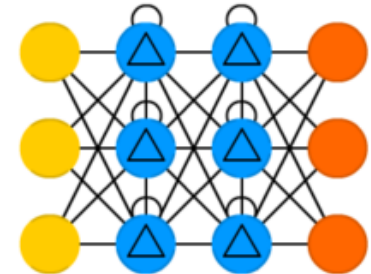
Recurrent Neural Network (RNN)



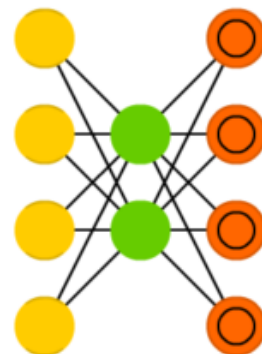
Long / Short Term Memory (LSTM)



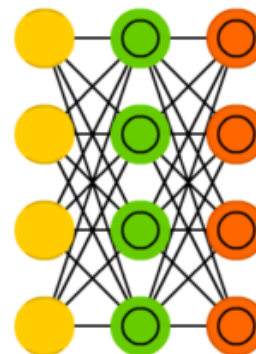
Gated Recurrent Unit (GRU)



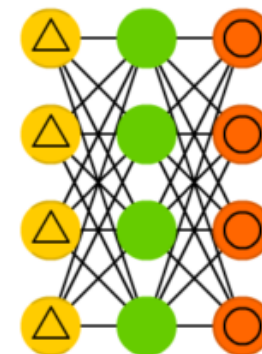
Auto Encoder (AE)



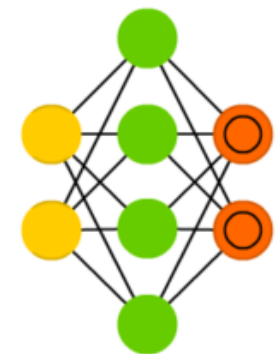
Variational AE (VAE)



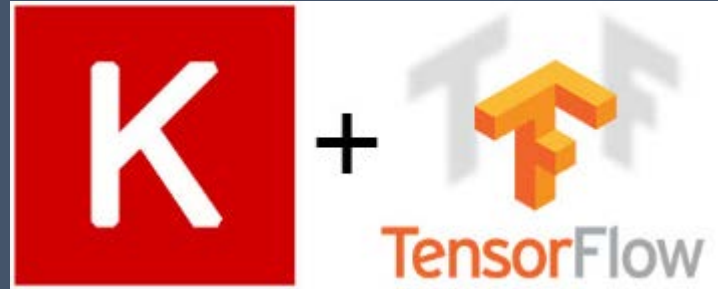
Denoising AE (DAE)



Sparse AE (SAE)



De la théorie à la pratique ...



Keras

API de haut niveau pour créer et entraîner des modèles de « deep learning »
construite sur TensorFlow ou Theano

TensorFlow

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