Mathématiques Numériques 1A
MN2

Chapitre 3 - Une brève introduction aux réseaux de neurones

J.-F. Scheid

Télécom Nancy

2025-2026

version du 15/6/2026

I. Perceptron

1. Introduction et définition

Classification de données

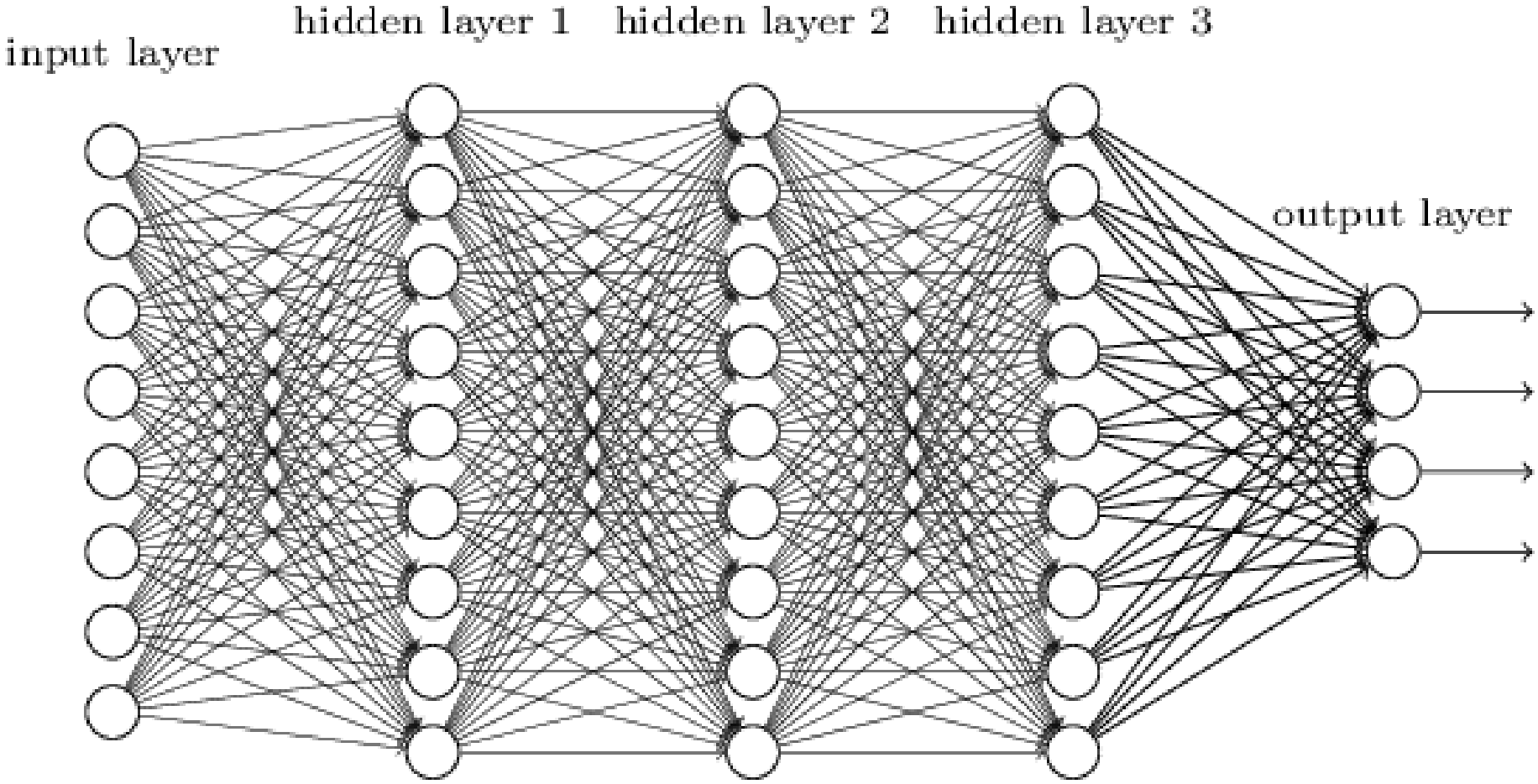
- *supervisée* : les classes sont prédéfinies et les prédictions sont réalisées avec des données historiques (apprentissage)
- *binaire* : 2 classes
- *linéaire* (avec biais = affine)

Un perceptron correspond à un (seul) *neurone formel*.

Inventé en 1943 par McCulloch et Pitts mais première mise en œuvre avec une machine construite par RosenBlatt en 1958. Machine conçue pour la reconnaissance d'images [1].

1. Cette machine était constituée d'un réseau de 400 photocellules, reliées aléatoirement aux "neurones". Les poids étaient encodés dans des potentiomètres et les mises à jour des poids pendant l'apprentissage étaient effectuées par des moteurs électriques.

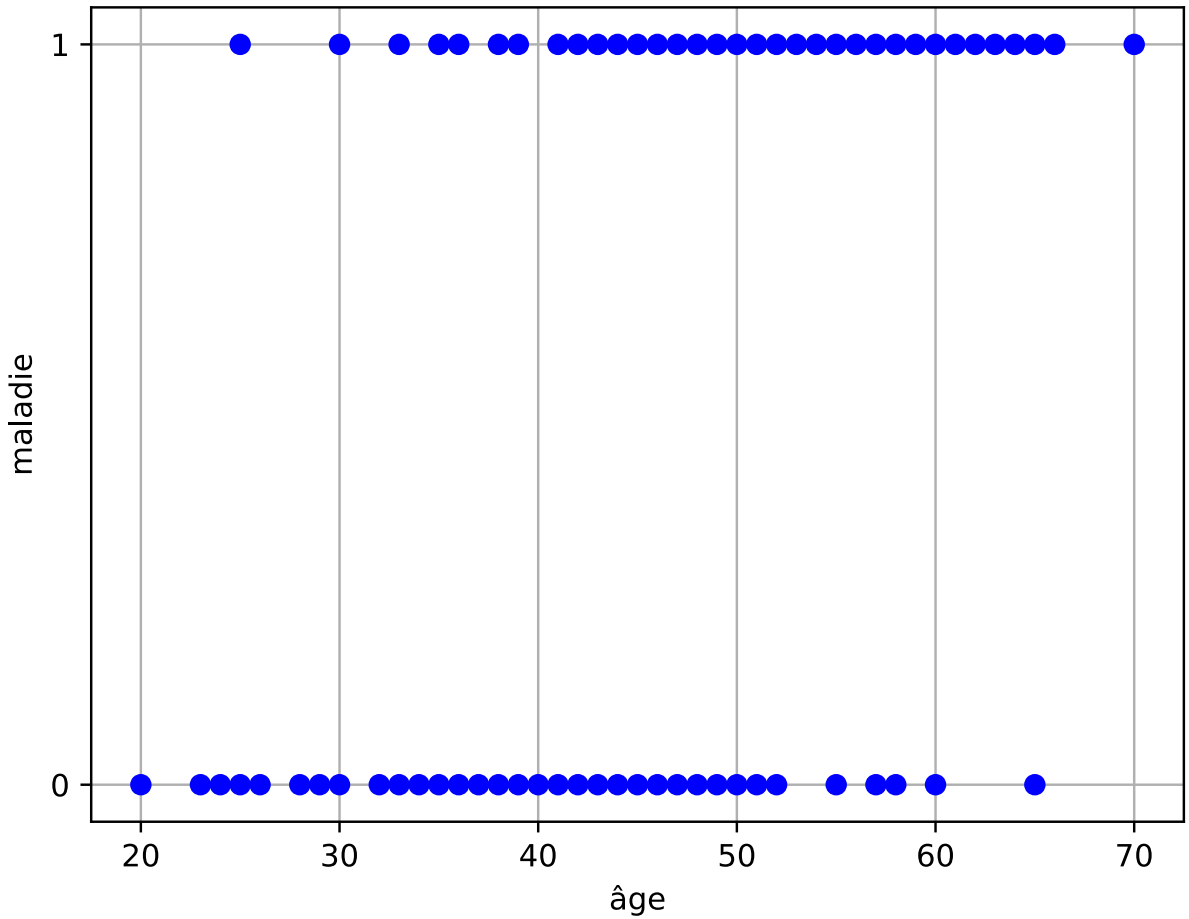
Des perceptrons disposés en couches successives et reliés entre eux couche par couche forment un *réseau de neurones*.



Crédit S. Ravindra

Un exemple avec un perceptron. Etude du lien entre une maladie et l'âge. Deux variables :

- x âge du patient
- $z \in \{0, 1\}$: absence (0) ou présence (1) de la maladie



☛ Prédire si un patient d'âge x est malade ($z = 1$) ou non ($z = 0$).

Dans cet exemple, on cherche une fonction (un *classificateur*) $F : \mathbb{R} \to \{-1, 1\}$ telle que :

x est dans la classe "$+1$" si le patient est malade
x est dans la classe "-1" sinon.

Perceptron

Soit $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ et les paramètres $w = (w_0, a) \in \mathbb{R} \times \mathbb{R}^n$ avec

$$w_0 \in \mathbb{R}, \quad a = (w_1, \ldots, w_n) \in \mathbb{R}^n.$$

Un perceptron est caractérisé par la fonction $F : \mathbb{R}^n \to \{-1, 1\}$ de la forme $F(x) = \phi(f_w(x))$ avec

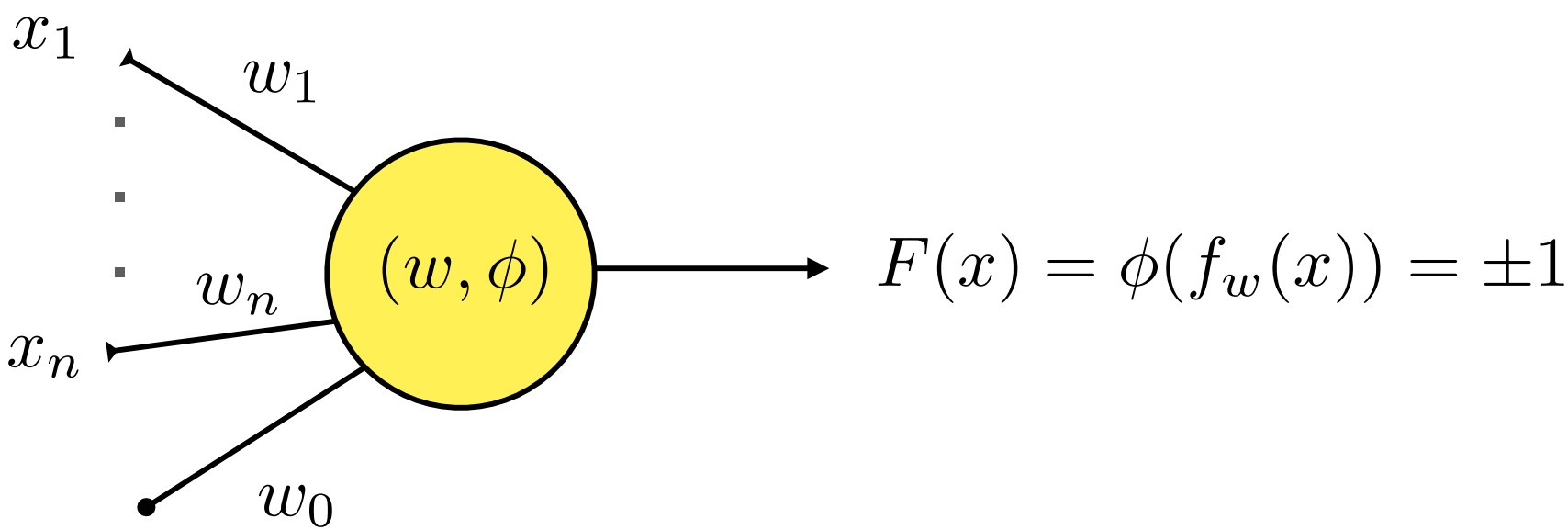
- $f_w(x) = (a \,|\, x) + w_0 = \sum_{i=1}^{n} w_i x_i + w_0$
- ϕ est la fonction d'activation $\phi(s) = \text{sign}(s) = \begin{cases} 1 & \text{si } s \geq 0 \\ -1 & \text{sinon} \end{cases}$

Remarque. $n = 1$ dans l'exemple précédent.

Les paramètres $a = (w_1, \ldots, w_n)$ sont les *poids* du perceptron et w_0 le *biais*.

Le classificateur F est donc donné par

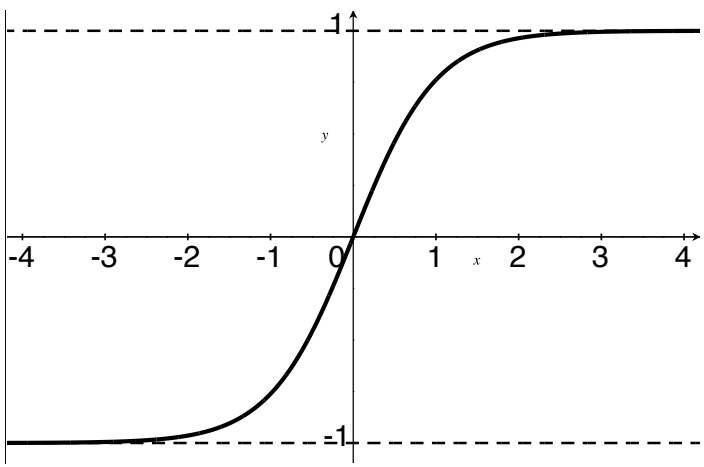
$$F(x) = \begin{cases} +1 & \text{si } (a \,|\, x) + w_0 \geq 0 \\ -1 & \text{sinon} \end{cases} \tag{1}$$



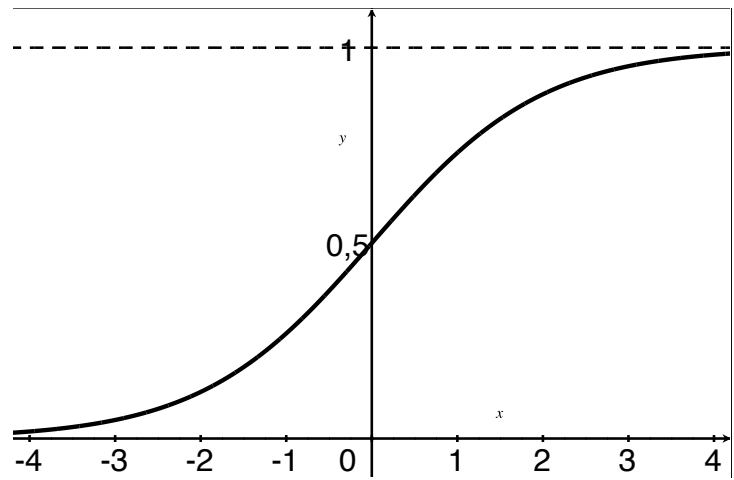
Fonctions d'activation.

D'autres fonctions d'activations ϕ sont possibles :

- linéaire $\phi(x) = x$ (on ne fait rien de plus donc...)
- ReLU (Rectified Linear Unit) $\phi(x) = \max(0, x)$
- tangente hyperbolique $\phi(x) = \tanh(x)$
- sigmoïde $\phi(x) = \sigma(x) := 1/(1 + \exp(-\alpha x))$, $\alpha > 0$



$\phi(x) = tanh(x)$



$\phi = \sigma$ sigmoïde

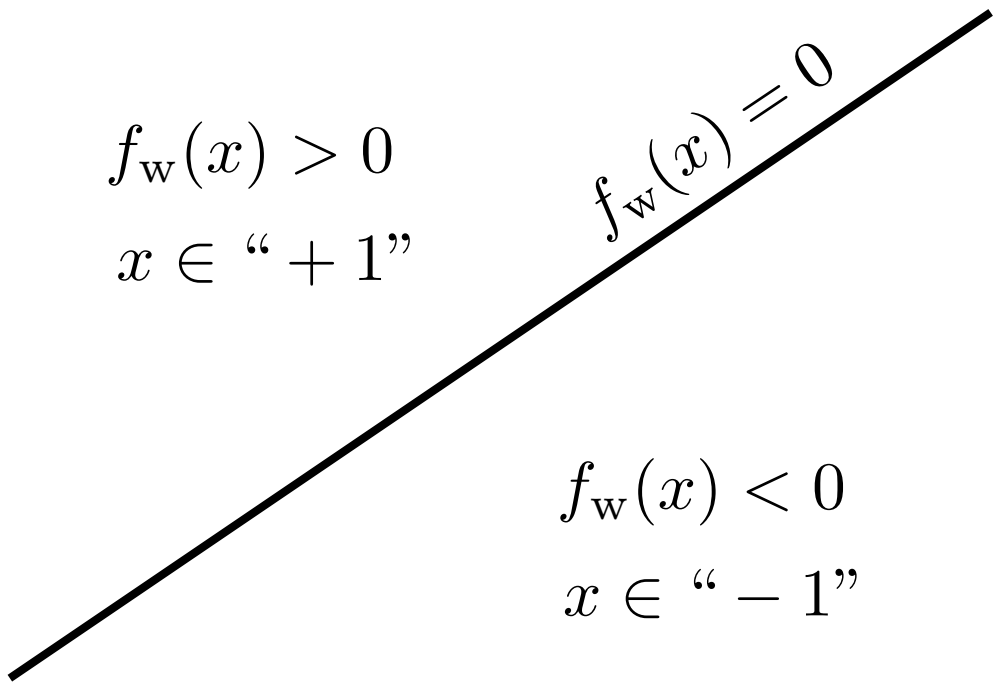
2. Apprentissage

a) Séparabilité. Dans $\mathbb{R}^n$, un hyperplan est défini par

$$H = \{x \in \mathbb{R}^n,\ f_w(x) := \sum_{i=1}^{n} w_i x_i + w_0 = 0\}$$

où $w_0, w_1, \ldots, w_n \in \mathbb{R}$. Un perceptron sépare $\mathbb{R}^n$ en deux parties telles que

x est dans la classe "$+1$" si $f_w(x) \geq 0$
x est dans la classe "-1" sinon.



b) Apprentissage des paramètres.

- On dispose d'un jeu de données $(x^{(j)}, z^{(j)}) \in \mathbb{R}^n \times \{-1, 1\}$ pour $j = 1, \ldots, m$.
- On veut déterminer par apprentissage les paramètres (poids et biais) sur ce jeu de données.
- Une donnée $(x^{(j)}, z^{(j)})$ est **mal classée** si

  $z^{(j)} = +1$ et $f_w(x^{(j)}) < 0$ (i.e. $x^{(j)}$ dans la classe "-1")

  ou

  $z^{(j)} = -1$ et $f_w(x^{(j)}) > 0$ (i.e. $x^{(j)}$ dans la classe "$+1$")

☛ La donnée $(x^{(j)}, z^{(j)})$ est **mal classée** si

$$\boxed{z^{(j)} f_w(x^{(j)}) < 0} \tag{2}$$

On cherche les paramètres optimaux $w^* = (w_0^*, a^*)$ minimisant la distance entre les données *mal classées* par F et la frontière de décision du modèle $f_w(x) = 0$.

☛ On cherche donc à minimiser la fonction (*risque empirique*)

$$\widehat{L}(w) = -\sum_{j \in I} z^{(j)} f_w(x^{(j)}) \tag{3}$$

où I est l'ensemble des indices des données *mal classées*.

3. Minimisation de fonction : méthode de descente de gradient

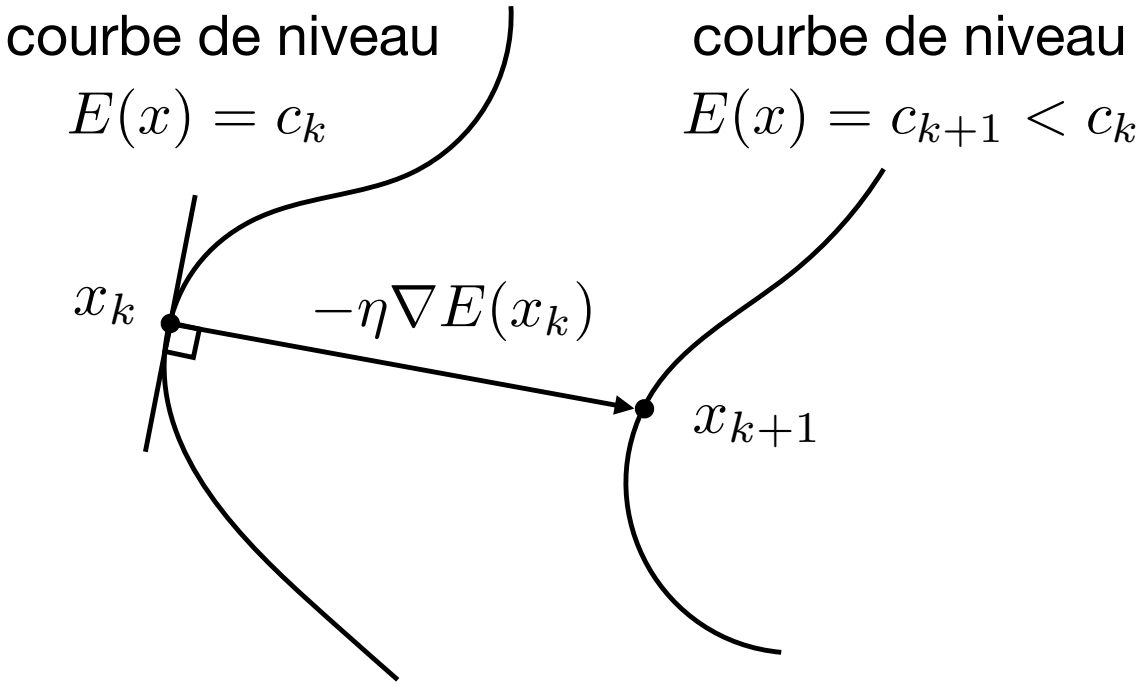
On cherche à minimiser une fonction $E : \mathbb{R}^n \to \mathbb{R}$ qui admet un minimum (local) en x^*. On a vu (cf. MN2, Chap.2) que le gradient ∇E en un point x est dirigé dans la direction de plus forte pente partant de x.

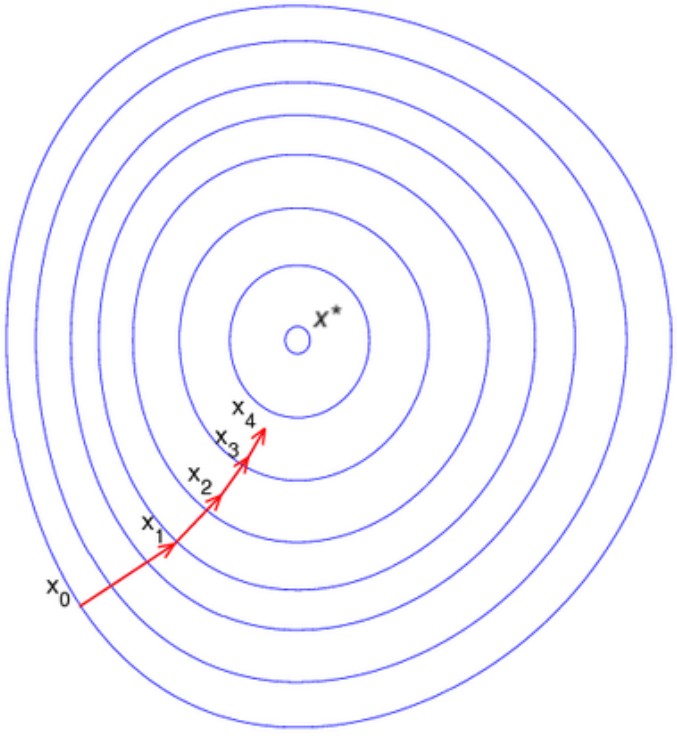
On peut construire une suite de vecteurs x_k qui converge vers x^* en utilisant le gradient :

$$\boxed{x_{k+1} = x_k - \eta \nabla E(x_k), \quad \eta > 0} \tag{4}$$

Remarques. La relation (4) définit une méthode *itérative* (on boucle sur k).

- Il faut préciser le point de départ x_0, la convergence dépendant évidemment de ce choix.
- Un test d'arrêt pour la méthode de descente de gradient porte généralement sur le gradient : on choisit une tolérance $\epsilon > 0$ et on arrête les itérations quand $\|\nabla E(x_k)\| \leq \epsilon$.





Descente de gradient : $x_{k+1} = x_k - \eta \nabla E(x_k)$

4. Algorithme du perceptron

a) Minimisation du risque empirique $\widehat{L}(w)$.

On a $\widehat{L}(w) = -\sum_{j\in I} z^{(j)} \left(w_0 + w_1 x_1^{(j)} + \cdots + w_n x_n^{(j)}\right)$

On calcule le gradient $\nabla\widehat{L} = \nabla_w \widehat{L}$ avec $w = (w_0, w_1, \ldots, w_n)$ et $a = (w_1, \ldots, w_n)$:

$$\nabla\widehat{L} = (\partial\widehat{L}/\partial w_0, \nabla_a \widehat{L}) = (\partial\widehat{L}/\partial w_0, \partial\widehat{L}/\partial w_1, \ldots, \partial\widehat{L}/\partial w_n),$$

$$\boxed{\frac{\partial\widehat{L}}{\partial w_0}(w) = -\sum_{j\in I} z^{(j)}, \quad \nabla_a \widehat{L}(w) = -\sum_{j\in I} z^{(j)} x^{(j)}}$$

Algorithme de descente du gradient.

- Initialisation $w^{(0)} = (w_0^{(0)}, a^{(0)})$.
- Pour $k = 0, 1, \ldots$:

$$w^{(k+1)} = w^{(k)} - \eta\nabla\widehat{L}(w^{(k)}) = \begin{pmatrix} w_0^{(k)} \\ a^{(k)} \end{pmatrix} + \eta \begin{pmatrix} \sum_{j\in I} z^{(j)} \\ \sum_{j\in I} z^{(j)} x^{(j)} \end{pmatrix}$$

 L'ensemble I peut être très grand $\Rightarrow$ algo. pas utilisé en pratique.

Une alternative : algorithme de gradient **stochastique**.

Au lieu de considérer les sommes sur $j \in I$ dans l'algo. du gradient, on choisit *au hasard un seul terme*.

Algorithme du perceptron (gradient stochastique).

- Initialisation à 0 : $k = 0$, $w^{(0)} = (w_0^{(0)}, a^{(0)}) = 0$.
- Tant qu'il existe j tel que $z^{(j)} \left((a^{(k)} \mid x^{(j)}) + w_0^{(k)} \right) < 0$
  - choisir j **aléatoirement** tel que $z^{(j)} \left((a^{(k)} \mid x^{(j)}) + w_0^{(k)} \right) < 0$.
  - $\begin{pmatrix} w_0^{(k+1)} \\ a^{(k+1)} \end{pmatrix} = \begin{pmatrix} w_0^{(k)} \\ a^{(k)} \end{pmatrix} + \eta \begin{pmatrix} z^{(j)} \\ \\ z^{(j)} x^{(j)} \end{pmatrix}$
  - $k = k + 1$

b) Convergence du perceptron.

Théorème [Novikoff, 1962]

On suppose que

- il existe $\gamma > 0$ et un vecteur $\bar{w} \in \mathbb{R} \times \mathbb{R}^n$ avec $\|\bar{w}\| = 1$ tel que

$$\forall j, \quad z^{(j)} f_{\bar{w}}(x^{(j)}) \geq \gamma \quad \text{(séparabilité linéaire des données)}$$

- $\forall j, \quad \|x^{(j)}\| \leq R$ (données bornées)

Alors l'algorithme du perceptron converge (il n'y a plus de données mal classées) en au plus $(1 + R^2)/\gamma^2$ itérations.

Convergence : il n'y a plus de donnée mal classée (démo. en TD).

c) Choix de η paramètre d'apprentissage (*learning rate*).

Dans l'algorithme du perceptron présenté ici, le paramètre η est constant à chaque itération. Quand on s'approche d'un minimum, le gradient tend vers 0. Le vecteur $\eta \nabla E(x_k)$ tend donc vers 0, même si η reste constant.

Cependant, il vaut mieux choisir η ni trop grand, ni trop petit : si η est trop grand alors les points x_k vont osciller autour du minimum mais si η est trop petit alors il faudra beaucoup d'itérations pour que les points x_k s'approchent du minimum.

☛ *Solution :* faire varier η avec les itérations k. Pour les premières itérations, on choisit un η_k assez grand, puis de plus en plus petit avec les itérations.

☛ *Choix possible* (décroissance utilisée par `keras`) :

$$\eta_k = \frac{\eta_0}{\alpha k + 1} \tag{5}$$

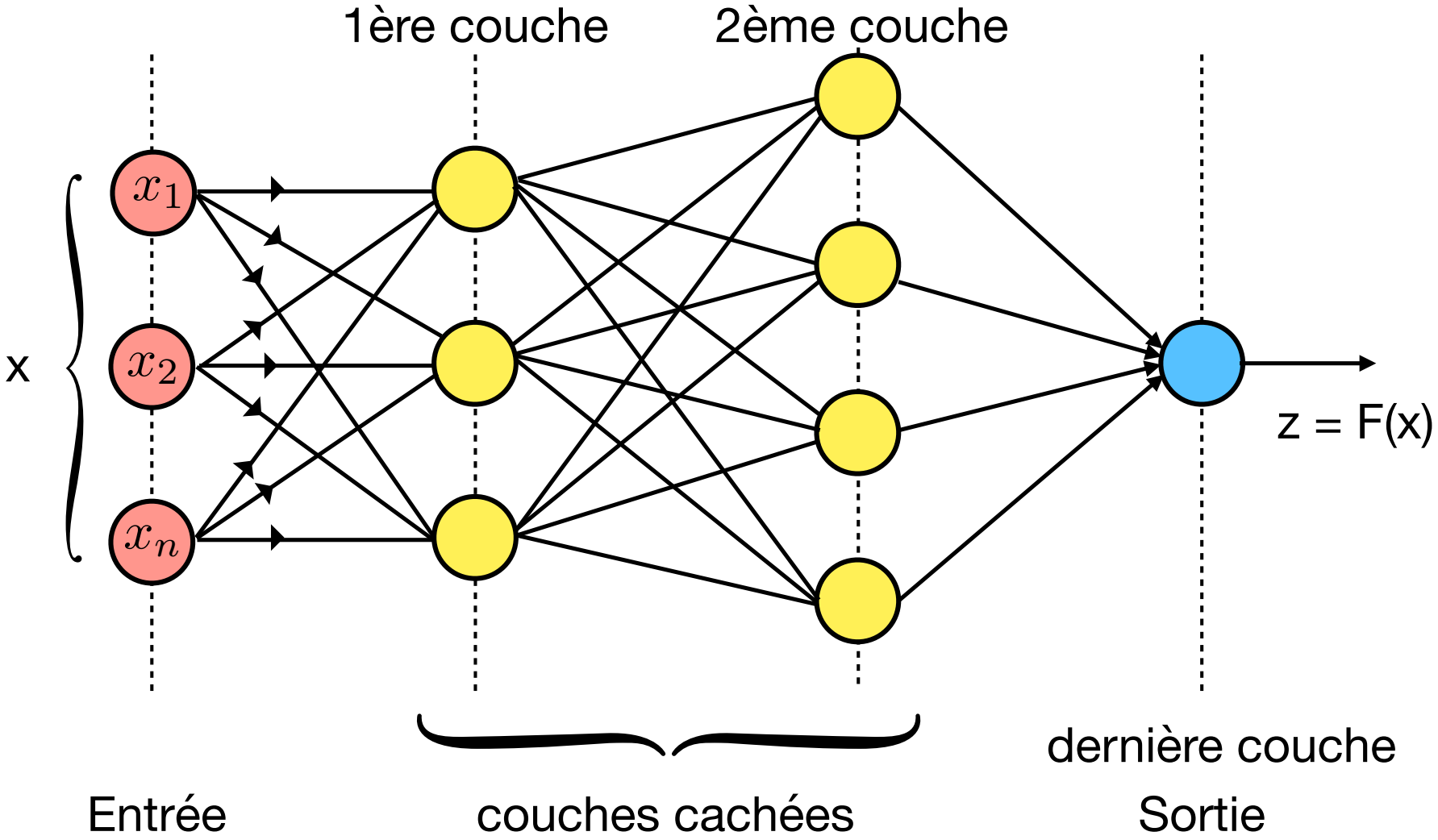
avec $\eta_0 \in [0.01,\ 0.1]$ et $\alpha \in [1e-6,\ 1e-4]$.

II. Réseaux de neurones (perceptron multicouche)

Motivation : un perceptron est trop élémentaire pour résoudre des problèmes complexes. Il permet seulement de séparer l'espace en deux parties.

Amélioration : des neurones connectés entre eux formant un réseau.

- Neurones disposés en plusieurs couches successives.
- Les connexions entre neurones se font uniquement des neurones d'une couche vers ceux de la couche directement de niveau supérieur : chaque neurone d'une couche $(k-1)$ est relié *à tous les neurones* de la couche supérieure k.
- Il n'y a pas de connexion entre neurones d'une même couche.
- Les neurones d'entrée représentent les données $x \in \mathbb{R}^n$. Ils n'ont pas de fonction d'activation (ce sont les données).
- Il peut y avoir plusieurs sorties mais pour simplifier on considère ici *une seule sortie*. La valeur de sortie est notée $z = F(x) \in \mathbb{R}$.

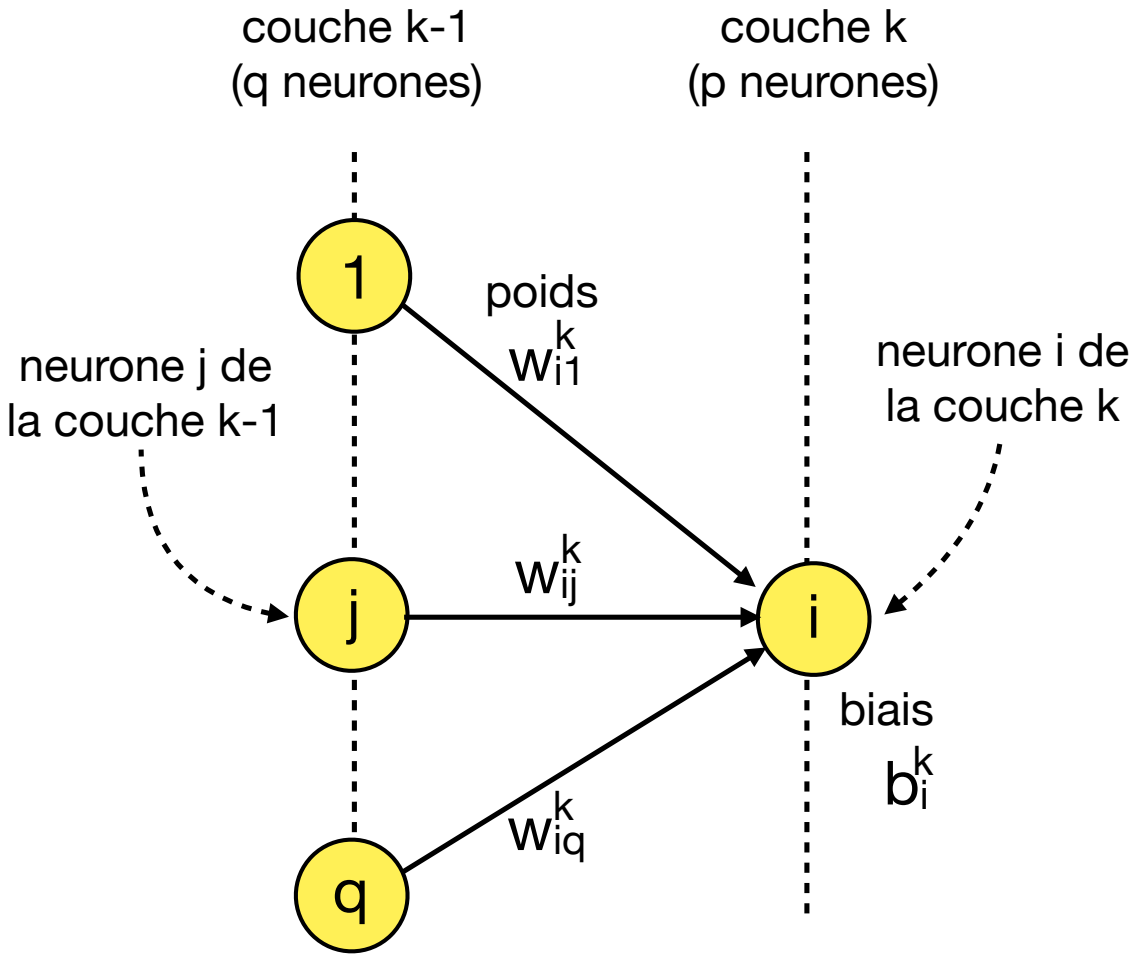


- La sortie n'est pas binaire mais réelle.
- Les neurones des couches cachées ont des fonctions d'activation sigmoïde $\phi(t) = \sigma(t) := 1/(1 + e^{-\alpha t})$.

1. Paramètres du réseau.

a) Poids et biais.

Le réseau est défini couche par couche et il y a M couches ($k = M$ étant la couche de sortie). On considère la couche k du réseau comportant p neurones et la couche $(k-1)$ en comportant q.



Sur la couche k, le réseau est défini par :

- W^k la matrice des poids des neurones de la couche $(k-1)$ vers la couche k :

  ☛ w_{ij}^k est le poids du neurone j de la couche $(k-1)$ vers le neurone i de la couche k.

  La matrice W^k est de taille (p, q).
- b^k le vecteur des biais des neurones de la couche k (taille p).

b) Fonctions d'activation.

On prend la même fonction d'activation ϕ^k pour tous les neurones de la couche k (sauf ceux de la couche d'entrée qui n'en ont pas). Pour les couches cachées, on prend généralement $\phi = \sigma$ la sigmoïde.

2. Fonctions d'évaluation

a) Fonction d'évaluation associée à un neurone.

- A chaque neurone $i \in [\![1, p]\!]$ de la couche $k \in [\![1, M]\!]$ du réseau, on définit la fonction d'évaluation du neurone :

$$f_i^k(y) = \phi^k\left(b_i^k + w_{i1}^k y_1 + \cdots + w_{iq}^k y_q\right) \tag{6}$$

  pour $y = (y_1, \ldots, y_q)^\top \in \mathbb{R}^q$.
- Pour les neurones i de la couche d'entrée $(k = 0)$ i.e. avec les données $x \in \mathbb{R}^n$, on a

$$f_i^0(x) = x_i \tag{7}$$

b) Fonctions associées au réseau.

A chaque neurone i de la couche k, on associe une fonction de réseau $F_i^k : \mathbb{R}^n \to \mathbb{R}$ traduisant l'état du réseau au niveau du neurone i à partir des données d'entrée $x \in \mathbb{R}^n$.

Elle est définie *par récurrence* avec les fonctions de réseau des neurones de la couche inférieure ($k \in [\![1, M]\!]$) :

$$F_i^k(x) = f_i^k\left(F_1^{k-1}(x), \cdots, F_q^{k-1}(x)\right) \tag{8}$$

et avec

$$F_i^0(x) = f_i^0(x) = x_i. \tag{9}$$

Ceci s'écrit encore (cf. (6)) :

Pour $k \in [\![1, M]\!]$,

$$\boxed{\begin{array}{l} F_i^k(x) = \phi^k\left(\alpha_i^k\right) \\ \alpha_i^k := b_i^k + w_{i1}^k F_1^{k-1}(x) + \cdots + w_{iq}^k F_q^{k-1}(x) \end{array}} \tag{10}$$

On note

$$\boxed{a_i^k := F_i^k(x)} \tag{11}$$

de sorte que

$$\boxed{\alpha_i^k = b_i^k + w_{i1}^k a_1^{k-1} + \cdots + w_{iq}^k a_q^{k-1}} \tag{12}$$

☛ α_i^k est le « signal » en entrée du neurone i de la couche k qui est une combinaison linéaire des « signaux » émis par les neurones de la couche précédente $k-1$.

Notation vectorielle et fonction d'évaluation F du réseau.

On note

$$\begin{aligned} a^k &:= F^k(x) = \left(F_1^k(x), \ldots, F_p^k(x)\right)^\top \in \mathbb{R}^p \\ \alpha^k &= \left(\alpha_1^k, \ldots, \alpha_p^k\right)^\top \in \mathbb{R}^p \end{aligned} \tag{13}$$

La relation (10) s'écrit alors vectoriellement :

$$\boxed{\begin{array}{rcl} \alpha^k & = & b^k + W^k a^{k-1} \\ a^k & = & \varphi^k(\alpha^k) \end{array}} \tag{14}$$

avec $\varphi^k(\alpha^k) = \left(\phi^k(\alpha_1^k), \ldots, \phi^k(\alpha_p^k)\right)^\top \in \mathbb{R}^p$.

La formule de récurrence (14) permet de calculer les valeurs a^k de la couche k en fonction des valeurs a^{k-1} de la couche précédente $(k-1)$.

Partant de la donnée $a^0 = x$, on peut ainsi calculer la valeur $F(x)$ du réseau à la sortie en propageant les valeurs des couches de l'entrée vers la sortie. Si $k = M$ est la couche de sortie, on a

$$F(x) := F^M(x) = a^M. \tag{15}$$

☛ *Algorithme de propagation*

$a^0 = x$ `# les données d'entrée`

Pour k de 1 à M

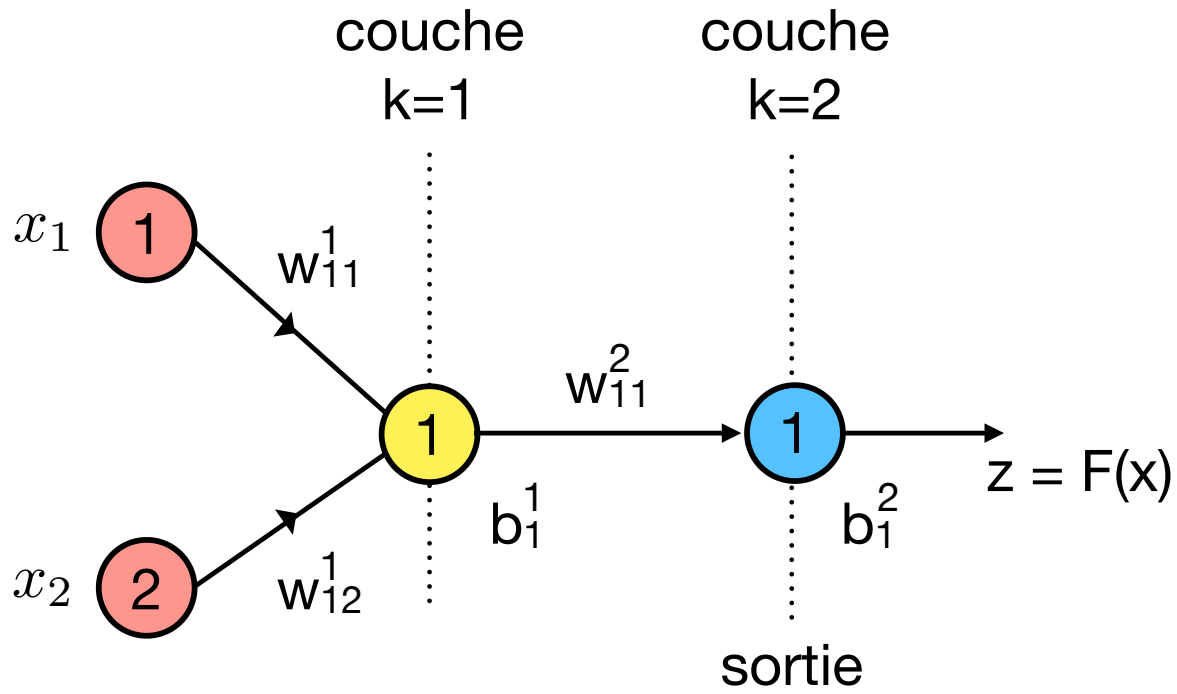
$\alpha^k = b^k + W^k a^{k-1}$

$a^k = \varphi^k(\alpha^k)$

fin pour

En sortie, on a $a^M = F(x)$.

Exemple. On considère un réseau avec une seule couche cachée composée d'un seul neurone.



$$
\begin{aligned}
F_1^1(x) &= f_1^1(x) = \sigma(b_1^1 + w_{11}^1 x_1 + w_{12}^1 x_2) \\
F(x) &= F_1^2(x) = f_1^2(F_1^1(x)) \\
&= \phi\left(b_1^2 + w_{11}^2 F_1^1(x)\right) \\
&= \phi\left(b_1^2 + w_{11}^2\, \sigma(b_1^1 + w_{11}^1 x_1 + w_{12}^1 x_2)\right)
\end{aligned}
$$

ϕ est la fonction d'activation du neurone de sortie.

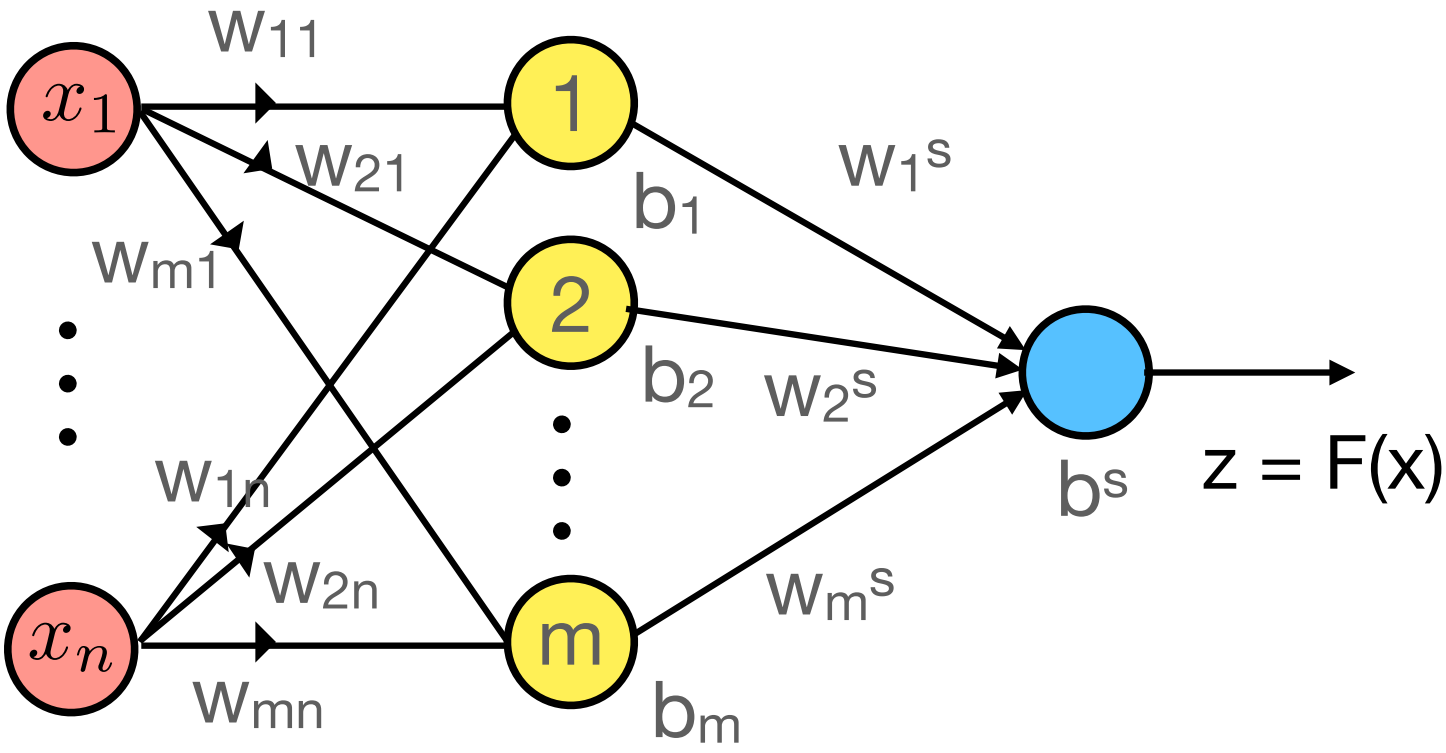
3. Résultat d'approximation.

Quelles fonctions peuvent être calculées correctement par un réseau de neurones (NN, neural network) fournissant la sortie $z = F(x) \in \mathbb{R}$ avec les données d'entrée $x \in \mathbb{R}^n$?

Autrement dit étant donnée une fonction f, peut-on construire un NN avec F telle que $F \simeq f$?

☛ Pour répondre à cette question, on considère un RN simple :

- n données $x \in \mathbb{R}^n$ en entrée
- une seule couche cachée avec m neurones (et avec une fonction d'activation commune ϕ)
- un neurone sans fonction d'activation ($\phi = I_d$) en sortie fournissant $F(x) \in \mathbb{R}$.


x_1
x_n
w11
w21
wm1
w1n
w2n
wmn
1
2
m
b1
b2
bm
w1s
w2s
wms
bs
z = F(x)

On note :

- w_{ij} et b_i le poids et le biais du neurone i avec l'entrée x_j
- w_i^s et b^s le poids et le biais du neurone de sortie avec le neurone i de la couche cachée.

On a donc

$$\boxed{z = F(x) = \sum_{i=1}^{m} \left(w_i^s \phi \left((w_i | x) + b_i \right) \right) + b^s} \tag{16}$$

où $w_i = (w_{i1}, \cdots, w_{in})^\top$ sont les poids du neurone i avec les entrées $x = (x_1, \cdots, x_n)^\top$.

On a alors le résultat suivant d'approximation.

Théorème d'approximation universelle (Cybenko, 1989)

Soient $\phi : \mathbb{R} \to \mathbb{R}$ une fonction (la fonction d'activation) non constante, bornée, continue et croissante et K un sous-ensemble fermé, borné de $\mathbb{R}^n$. Étant donnés $\varepsilon > 0$ et une fonction f continue sur K, il existe un entier m, m scalaires $\{w_i^s\}_{i=1,\ldots,m}$, m scalaires $\{b_i\}_{i=1,\ldots,m}$, et m vecteurs $\{w_i\}_{i=1,\ldots,m}$ de $\mathbb{R}^n$ tels que, pour tout $x \in K$:

$$|f(x) - F(x)| < \varepsilon.$$

où $F(x)$ est la sortie du réseau de neurones donnée par (16)

☛ En d'autres termes, toute fonction continue sur un sous-ensemble compact de $\mathbb{R}^n$ peut être approchée avec un degré de précision arbitraire par un NN à une couche cachée contenant un nombre fini de neurones.

Remarques.

- Ce résultat dit qu'un réseau de neurones peut apprendre toute sorte de fonction, mais attention : on ne connait ni le nombre de neurones devant composer la couche cachée, ni les poids de connexion à utiliser.
- Les réseaux de neurones à une seule couche cachée sont en général peu efficaces. Les réseaux avec plus de couches sont plus performants.

4. Apprentissage.

La définition du réseau de neurones multi-couches a été faite avec une donnée x. Dans la pratique, on dispose d'un jeu de plusieurs données : $x^{(l)}$ pour les données d'entrée et $z^{(l)}$ pour les données de sorties, pour $l = 1, \ldots, m$.

Apprentissage (détermination de W).

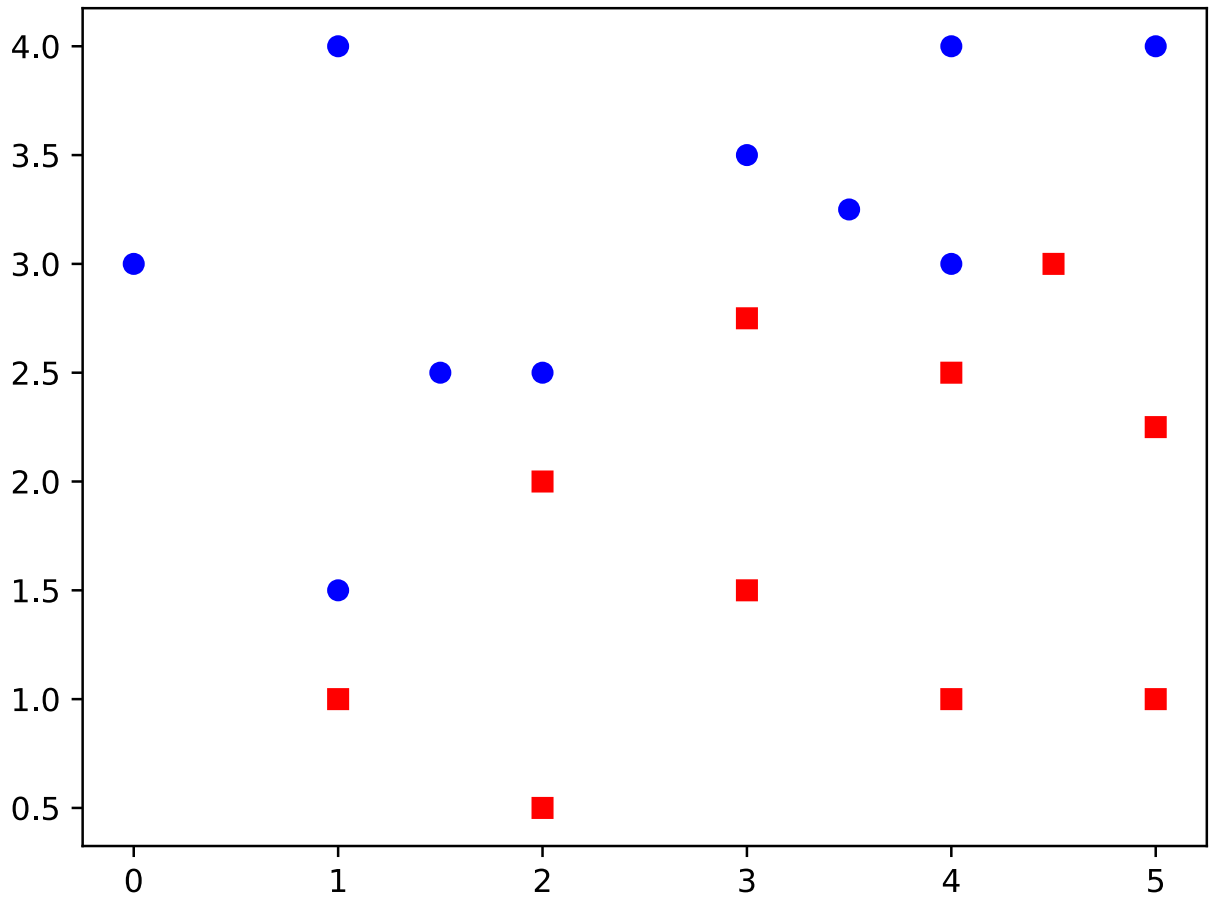
- On dispose de données $(x^{(l)}, z^{(l)}) \in \mathbb{R}^n \times \mathbb{R}$, $l = 1, \ldots, m$.
- On détermine W (poids et biais du réseau) pour minimiser l'erreur

$$E(W) = \sum_{l=1}^{m} E_l(W) = \sum_{l=1}^{m} (F(x^{(l)}) - z^{(l)})^2 \qquad (17)$$

C'est un problème de moindres carrés **non-linéaire**.

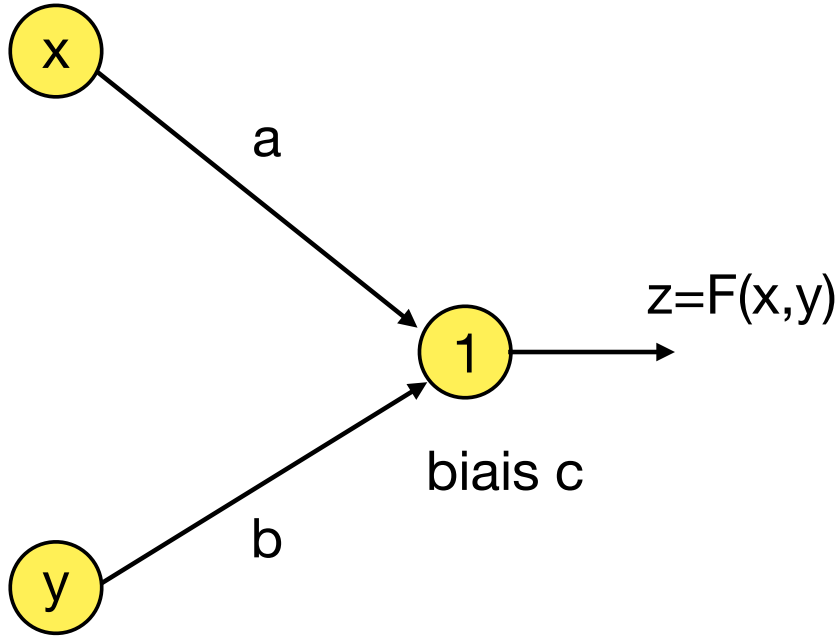
L'apprentissage se fait par une méthode de gradient (stochastique). Il faut donc calculer le gradient ∇E de l'erreur E par rapport aux paramètres poids/biais du réseau regroupés dans W.

Exemple. On dispose de n points du plan, de coordonnées $(x^{(l)}, y^{(l)})$, $l \in [\![1, m]\!]$ répartis en 2 catégories : des ronds bleus $(z^{(l)} = 0)$ et des carrés rouges $(z^{(l)} = 1)$.



On cherche une droite $ax + by + c = 0$ séparant les deux catégories ronds bleus/carrés rouges.

Pour déterminer les paramètres $W = (a, b, c)$, on considère le réseau de neurones suivant :



On a $F(x, y) = \sigma(ax + by + c)$.

- La région des ronds bleus est caractérisée par

$$\{(x, y) \text{ tel que } F(x, y) < 1/2\}$$

- La région des carrés rouges est caractérisée par

$$\{(x, y) \text{ tel que } F(x, y) > 1/2\}.$$

- La valeur $F(x, y) = 1/2$ correspond à la droite $ax + by + c = 0$

La fonction erreur s'écrit

$$E(W) = \sum_{l=1}^{m} \left(F(x^{(l)}, y^{(l)}) - z^{(l)} \right)^2$$

avec $F(x, y) = \sigma\,(ax + by + c)$.

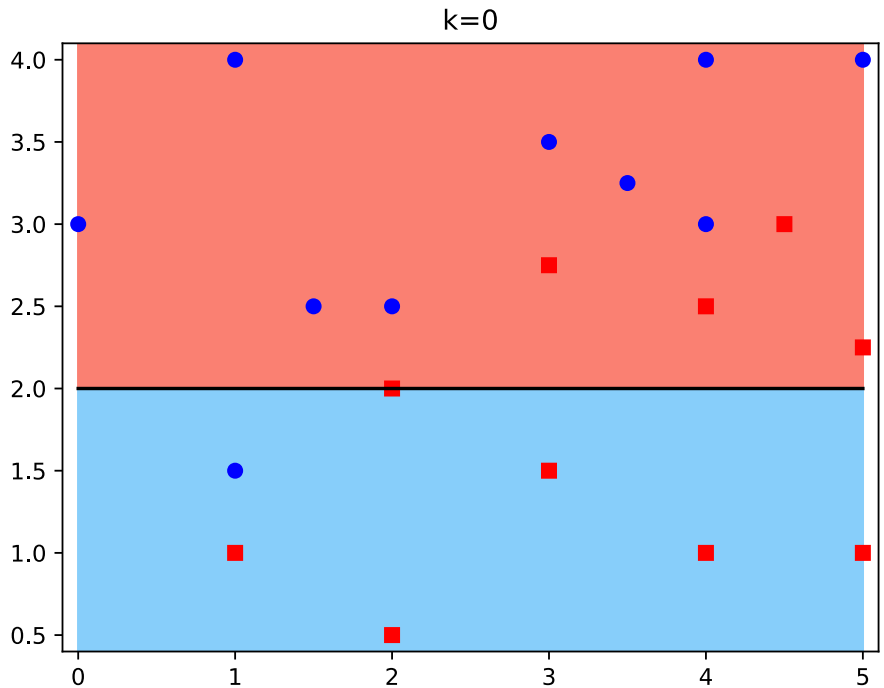
Remarque. La fonction sigmoïde σ vérifie la relation $\boxed{\sigma' = \sigma(1 - \sigma)}$.

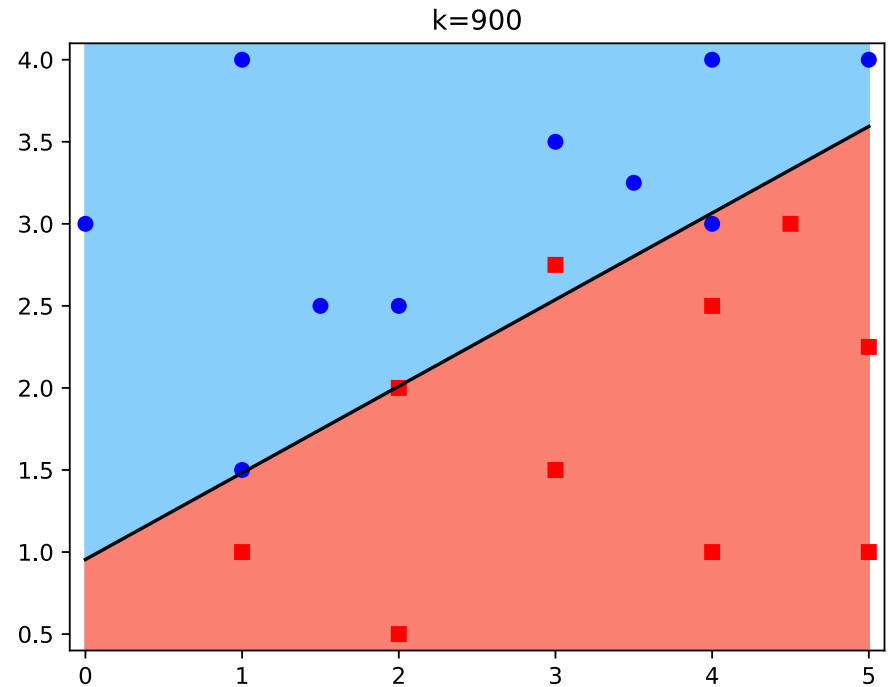
Calcul du gradient de E.

$$\frac{\partial E}{\partial a} = 2 \sum_{l=1}^{m} \sigma^{(l)}(1 - \sigma^{(l)})(\sigma^{(l)} - z^{(l)})x^{(l)}$$
$$\frac{\partial E}{\partial b} = 2 \sum_{l=1}^{m} \sigma^{(l)}(1 - \sigma^{(l)})(\sigma^{(l)} - z^{(l)})y^{(l)}$$
$$\frac{\partial E}{\partial c} = 2 \sum_{l=1}^{m} \sigma^{(l)}(1 - \sigma^{(l)})(\sigma^{(l)} - z^{(l)})$$

avec $\sigma^{(l)} = F(x^{(l)}, y^{(l)}) = \sigma(ax^{(l)} + by^{(l)} + c)$.

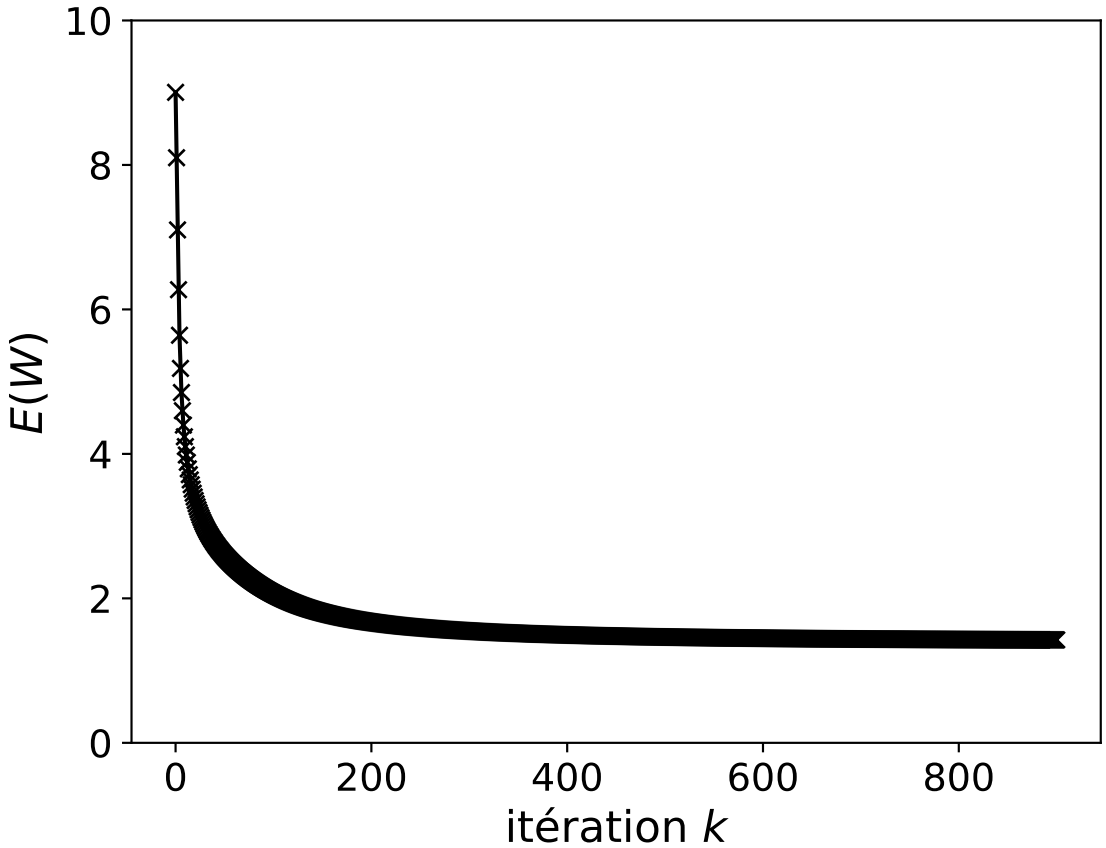
On initialise les paramètres avec la droite $y = 2$ ($a = 0$, $b = 1$, $c = -2$). On remarque qu'initialement la région des carrés rouges se trouve au-dessus de la droite $y = 2$.





On obtient au bout de $k = 900$ itérations de la méthode de gradient : $a = 1.995079$, $b = -3.780077$, $c = 3.608953$.

Evolution de $E(W)$ en fonction des itérations k de la méthode de descente de gradient :



Remarques.

1. Si on dispose de données $(x^{(l)}, z^{(l)})$ avec $z^{(l)} \in \mathbb{R}^q$ c'est-à-dire avec q données de sorties et non plus une seule, on peut généraliser les réseaux de neurones précédents aux réseaux à q neurones de sortie.

   On définit une fonction de réseau $F : \mathbb{R}^n \to \mathbb{R}^q$ avec l'erreur

   $$E(W) = \sum_{l=1}^{m} \|F(x^{(l)}) - z^{(l)}\|^2$$

   où $\|\cdot\|$ représente la norme (euclidienne) dans $\mathbb{R}^q$.

2. La fonction erreur E précédente (ou celle en (17)) est souvent utilisée pour les problèmes de régression ($z^{(l)} \in \mathbb{R}^q$). D'autres fonctions erreurs peuvent être choisies en fonction des problèmes.

Exemples.

1. *Problème de classification en 2 classes.*

   On a $z^{(l)} = 0$ ou 1. On choisit plutôt l'erreur appelée *entropie croisée* :

$$E_l(W) = -\left[z^{(l)} \log(F(x^{(l)})) + (1 - z^{(l)}) \log(1 - F(x^{(l)}))\right]$$

   - Si la sortie $z^{(l)} = 0$, le premier terme dans $E_l(W)$ s'annule. Cela force alors $F(x^{(l)})$ à être le plus proche possible de 0 pour que l'erreur soit petite.
   - Si la sortie $z^{(l)} = 1$, c'est le deuxième terme dans $E_l(W)$ qui s'annule. Cela force $F(x^{(l)})$ à être le plus proche possible de 1.

2. *Problème de classification en $K > 2$ classes.*

On fixe $q = K$ neurones de sorties et $z^{(l)}$ est un vecteur de K composantes, avec des 0 et un seul 1 correspondant à la classe d'appartenance (*one-hot encoding*). Sur la couche de sortie, on choisit la fonction d'activation « *SoftMax* » définie par

$$z_j := \sigma_j(y) = \exp(y_j) / \sum_{k=1}^{K} \exp(y_k) \quad \in [0,1] \tag{18}$$

pour $y = (y_1, \ldots, y_K)$. Cette transformation permet de séparer fortement la plus grande valeur des autres. De plus, les z_j correspondent à des probabilités avec des valeurs ramenées [1] entre 0 et 1 et $\sum_j z_j = 1$.

On choisit alors [2]

$$E_l(w) = -\sum_{j=1}^{K} z_j^{(l)} \log(\hat{z}_j^{(l)}) \quad \text{avec } \hat{z}_j^{(l)} = \sigma_j(\alpha^M) = F_j^M(x^{(l)})$$

1. d'où l'expression *SoftMax* : c'est la version « soft » de la fonction $Max(y) = 0$ si $y < 1$ et $= 1$ ssi $y = 1$.
2. avec la convention $0 \log(0) = 0$

5. Rétro-propagation du gradient.

Dès qu'un réseau comprend de nombreuses couches et neurones, il devient trop compliqué de calculer le gradient de E directement à partir de l'expression de F.
Une façon efficace de procéder est de calculer le gradient par récurrence couche par couche en partant de la dernière couche de sortie vers la couche d'entrée. C'est ce qu'on appelle la *rétropropagation du gradient.*

Pour un ensemble de m mesures $\{(x^{(l)}, z^{(l)})\}_{l\in[\![1,m]\!]}$, on rappelle que l'erreur s'écrit

$$E(W) = \sum_{l=1}^{m} E_l(W)$$

avec

$$E_l(W) = (F(x^{(l)}) - z^{(l)})^2. \tag{19}$$

La dérivée par rapport au poids w_{ij}^k de la couche k vaut

$$\frac{\partial E}{\partial w_{ij}^k} = \sum_{l=1}^{m} \frac{\partial E_l}{\partial w_{ij}^k}. \tag{20}$$

Pour calculer la dérivée de E_l, on considère pour simplifier une seule donnée notée (x, z), de sorte qu'on a (avec (19)) :

$$\frac{\partial E_l}{\partial w_{ij}^k} = 2\,(F(x) - z)\,\frac{\partial F}{\partial w_{ij}^k}(x). \tag{21}$$

Calcul du gradient de F.

Pour calculer la dérivée de F par rapport aux poids des neurones de la couche k, on introduit la fonction G^k telle que

$$F(x) = G^k(\alpha^k) \tag{22}$$

qui représente la sortie du réseau exprimée en fonction de l'état $\alpha^k = (\alpha_1^k, \ldots, \alpha_p^k)$ du réseau à l'entrée de la couche k.On a [2]

$$\frac{\partial F}{\partial w_{ij}^k} = \sum_{l=1}^{p} \frac{\partial G^k}{\partial \alpha_l^k}\frac{\partial \alpha_l^k}{\partial w_{ij}^k} = \frac{\partial G^k}{\partial \alpha_i^k}\frac{\partial \alpha_i^k}{\partial w_{ij}^k} \tag{23}$$

car $\dfrac{\partial \alpha_l^k}{\partial w_{ij}^k} = 0$ pour $l \neq i$.

2. $\frac{d}{dt}(G(x_1(t), \ldots, x_p(t))) = \frac{\partial G}{\partial x_1}\frac{dx_1}{dt} + \cdots + \frac{\partial G}{\partial x_p}\frac{dx_p}{dt} = \sum_{i=1}^{p} \frac{\partial G}{\partial x_i}\frac{dx_i}{dt}$

On rappelle que (cf. (12)) $\alpha_i^k = b_i^k + w_{i1}^k a_1^{k-1} + \cdots + w_{ip}^k a_p^{k-1}$
d'où

$$\frac{\partial \alpha_i^k}{\partial w_{ij}^k} = a_j^{k-1} \tag{24}$$

On note

$$\boxed{s_i^k = \frac{\partial G^k}{\partial \alpha_i^k}(\alpha^k)} \tag{25}$$

On obtient ainsi

$$\boxed{\frac{\partial F}{\partial w_{ij}^k} = s_i^k a_j^{k-1}} \tag{26}$$

Calcul des s^k.

1. *Calcul de s^M.*

   On commence par calculer s^k sur la dernière couche i.e. pour $k = M$:

   $$s^M = \frac{\partial G^M}{\partial \alpha^M}(\alpha^M)$$

   avec $F(x) = G^M(\alpha^M) = \phi^M(\alpha^M)$.

D'où

$$\boxed{s^M = \left(\phi^M\right)'(\alpha^M)} \tag{27}$$

2. *Calcul de s^k en fonction de s^{k+1}.*

   On calcule s^k pour les couches cachées précédentes à partir de la relation

$$\alpha_i^{k+1} = b_i^{k+1} + w_{i1}^{k+1} a_1^k + \cdots + w_{ip}^{k+1} a_p^k$$

   avec $a_i^k = \phi^k(\alpha_i^k)$.

   D'où

$$\alpha_i^{k+1} = b_i^{k+1} + w_{i1}^{k+1} \phi^k(\alpha_1^k) + \cdots + w_{ip}^{k+1} \phi^k(\alpha_p^k) \tag{28}$$

   ce qui s'écrit aussi vectoriellement

$$\alpha^{k+1} = b^{k+1} + W^{k+1} \phi^k(\alpha^k).$$

On écrit alors $F(x)$, non plus en fonction de α^k mais en fonction de α^{k+1} i.e. en fonction de l'état de la couche $k+1$ avec une fonction G^{k+1} :

$$F(x) = G^k(\alpha^k) = G^{k+1}(\alpha^{k+1}) \tag{29}$$

On a alors

$$s_i^k = \frac{\partial G^k}{\partial \alpha_i^k} = \frac{\partial}{\partial \alpha_i^k}\left[G^{k+1}(\alpha^{k+1})\right] = \sum_j \underbrace{\frac{\partial G^{k+1}}{\partial \alpha_j^{k+1}}}_{=s_j^{k+1}} \underbrace{\frac{\partial \alpha_j^{k+1}}{\partial \alpha_i^k}}_{\substack{=w_{ji}^{k+1}\left(\phi^k\right)'(\alpha_i^k) \\ \text{d'après (28)}}}$$

On obtient ainsi

$$\boxed{s_i^k = \left(\phi^k\right)'(\alpha_i^k) \sum_j w_{ji}^{k+1} s_j^{k+1}} \tag{30}$$

ce qui s'écrit encore vectoriellement

$$\boxed{s^k = \left(\varphi^k\right)'(\alpha^k) \cdot \left(W^{k+1}\right)^\top s^{k+1}} \tag{31}$$

(le $\cdot$ désigne le produit scalaire entre 2 vecteurs).

On calcule donc s^k en fonction de s^{k+1}, d'où le terme de *rétropropagation* du gradient.

Remarque. On a calculé les dérivées de l'erreur E_l par rapport aux poids w_{ij}^k. On peut refaire la même chose pour calculer les dérivées par rapport aux biais b^k i.e. $\partial E_l / \partial b_i^k$. Il suffit d'étendre la matrice W^k en lui ajoutant une première colonne supplémentaire, en notant

$$w_{i0}^k := b_i^k \quad \text{et} \quad a_0^k := 1 \tag{32}$$

La matrice W^k devient ainsi de taille $(p, q+1)$.

☛ *Toutes les formules précédentes restent alors inchangées.*

Descente de gradient (couche par couche).

Les poids sont mis à jour *couche par couche* par la méthode de descente de gradient :

Pour $k \in [\![1, M]\!]$,

$$\left(w_{ij}^k\right)_{\text{new}} = w_{ij}^k - \eta \Delta w_{ij}^k \tag{33}$$

avec (voir (21) et (26)) $\Delta w_{ij}^k = \dfrac{\partial E_I}{\partial w_{ij}^k}(w_{ij}^k) = 2(a^M - z) s_i^k a_j^{k-1}$.

Cela s'écrit matriciellement

$$\boxed{W_{\text{new}}^k = W^k - \eta \Delta W^k \quad \text{avec} \quad \Delta W^k = 2(a^M - z)\ s^k \cdot a^{k-1}} \tag{34}$$

pour $k \in [\![1, M]\!]$.

☛ *Algorithme de rétropropagation du gradient*

Les α^k et a^k sont calculés au préalable par l'algorithme de propagation avec les données (x, z).

Les poids et biais sont mis à jour de la façon suivante.

$s^M = \left(\phi^M\right)'(\alpha^M)$ `# initialisation`

$\Delta W^M = 2(a^M - z)\ s^M \cdot a^{M-1}$

$W^M_{\text{new}} = W^M - \eta \Delta W^M$

Pour k de $M-1$ à 1 (par pas de -1)

$s^k = \left(\varphi^k\right)'(\alpha^k_i) \cdot \left(W^{k+1}\right)^\top s^{k+1}$

$\Delta W^k = 2(a^M - z)\ s^k \cdot a^{k-1}$

$W^k_{\text{new}} = W^k - \eta \Delta W^k$

fin pour

Remarques.

1. Avec plusieurs mesures $(x^{(l)}, z^{(l)})_{l \in [\![1,m]\!]}$, on a (cf. (20))

$$\Delta w_{ij}^k = \frac{\partial E}{\partial w_{ij}^k} = \sum_{l=1}^{m} \frac{\partial E_l}{\partial w_{ij}^k} \tag{35}$$

   Le gradient complet (pour l'ensemble des mesures) est donc donné par

$$\Delta W^k = \sum_{l=1}^{m} \Delta W^{(l),k} \tag{36}$$

   où $\Delta W^{(l),k}$ est le gradient calculé par l'algo. de rétropropagation précédent avec la donnée $(x, z) = (x^{(l)}, z^{(l)})$.

2. Quand m est grand ($m \simeq 10^6$), le calcul de $\sum_{l=1}^{m} \nabla E_l$ peut s'avérer très couteux car il faut stocker les gradients ∇E_l avant d'en faire la somme.

☛ *Alternatives possibles.*

a. *Gradient stochastique.*

Mise-à-jour des poids/biais de façon aléatoire pour chacune des données en prenant le gradient associé : on met à jour chaque poids/biais w par

$$w_{new} = w - \eta \frac{\partial E_l}{\partial w}$$

pour toutes les données l choisies aléatoirement [3].

b. *Mini-batch (lots).*

Mise-à-jour des poids/biais après parcours d'un sous-ensemble B_m de données :

$$w_{new} = w - \eta \sum_{l \in B_m} \frac{\partial E_l}{\partial w}$$

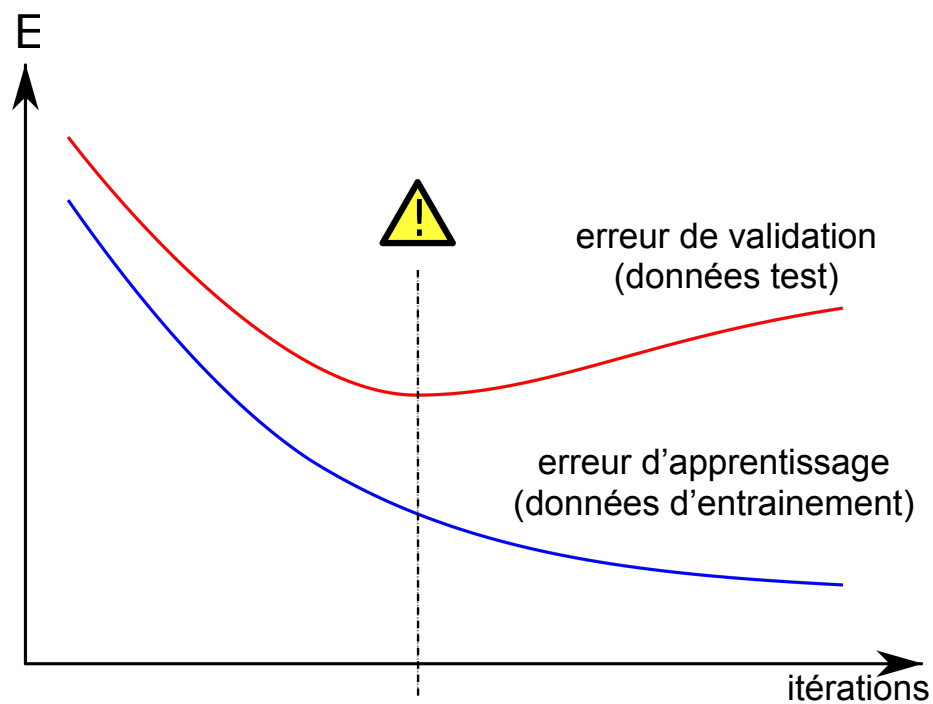
---

3. Plus précisément :
   - On choisit aléatoirement une permutation ϕ de $\{1, \ldots, m\}$
   - Pour $l = 1, \ldots, m$
     $w \leftarrow w - \eta \frac{\partial E_{\phi(l)}}{\partial w}$

6. Surapprentissage et régularisation

Une partie seulement des données est utilisée pour l'apprentissage (*données d'entrainement/training data*). L'autre partie des données sert à tester le modèle et l'apprentissage réalisé (*données test/test data*).

On parle de *surapprentissage* (*overfitting*) lorsque le modèle correspond trop précisément aux données particulières utilisées pour l'entrainement et pas suffisamment aux données test.



Le surapprentissage apparait lorsque l'erreur calculée sur les *données test* se met à augmenter en fonction des itérations[a] de la méthode de descente alors que l'erreur calculée sur les *données d'entrainement* continue de décroitre.

a. on parle d'*époque* (epoch)

Le réseau ainsi entrainé risque de ne pas bien analyser les données utilisées en phase de production et donc de ne pas permettre une utilisation fiable du modèle.

☛ **Remèdes.**

- On arrête l'apprentissage quand l'erreur de validation commence à croitre.
- *Régularisation.* On force les valeurs des poids/biais à ne pas être trop dispersées : on modifie la fonction erreur E en ajoutant un terme qui va forcer les poids/biais à avoir des petites valeurs.

  i) L^1-régularisation.

$$\tilde{E}(W) = \sum_{l=1}^{m} \|F(x^{(l)}) - z^{(l)}\|^2 + \gamma \sum_{k=0}^{M} \sum_{i,j} |w_{ij}^k| \qquad (37)$$

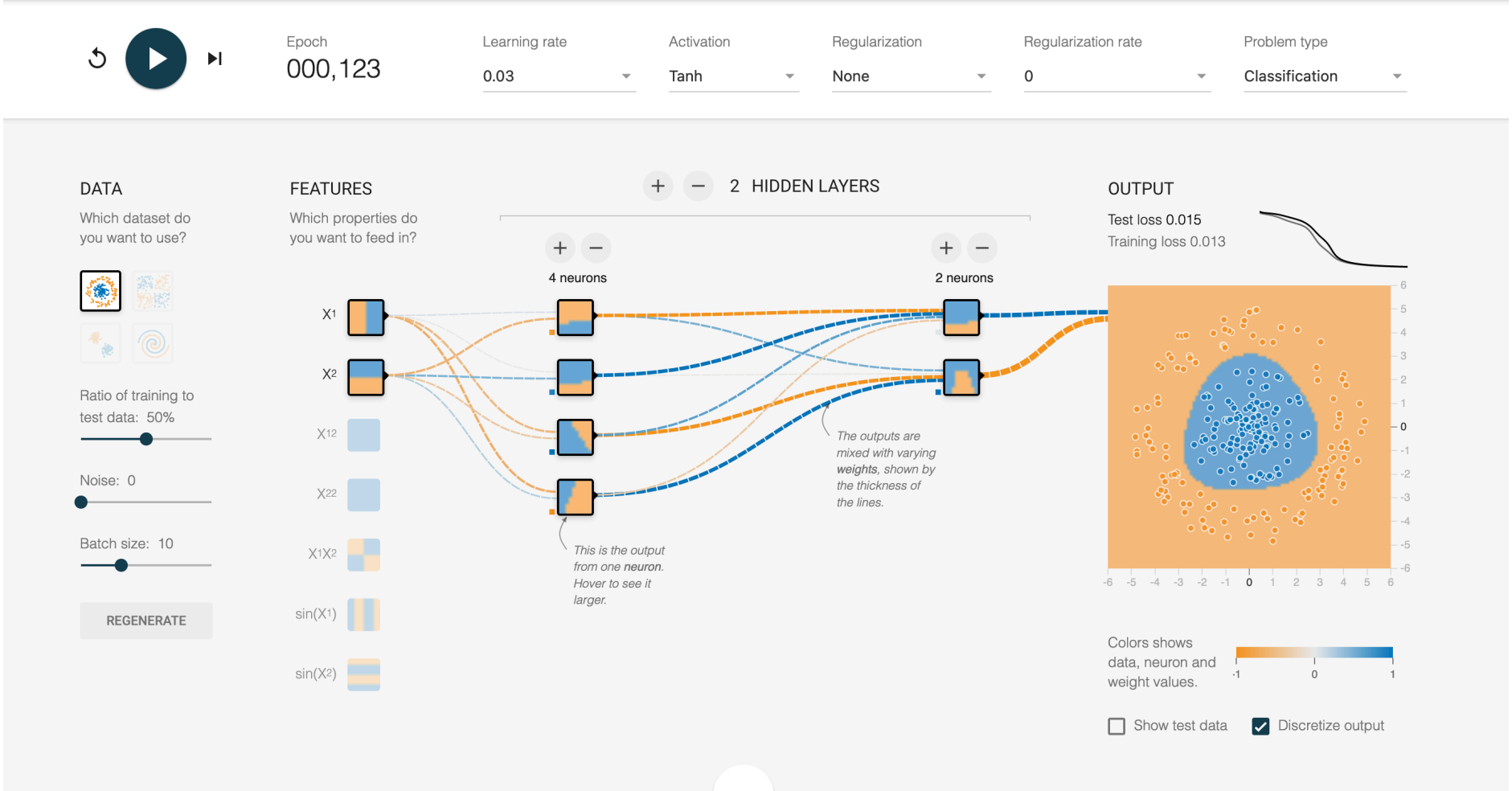
  ii) L^2-régularisation.

$$\tilde{E}(W) = \sum_{l=1}^{m} \|F(x^{(l)}) - z^{(l)}\|^2 + \gamma \sum_{k=0}^{M} \sum_{i,j} (w_{ij}^k)^2 \qquad (38)$$

  où $\eta > 0$ est le *taux de régularisation*.

et pour finir... bac-à-sable `tensorflow` :

`https://playground.tensorflow.org/`



Références.

[1] C.-A. Azencott. *Introduction au Machine Learning*. Dunod, 2018.

[2] Y. Bengio, I. Goodfellow, A. Courville. *Deep Learning*. Volume 1. MIT press, 2017.

[3] B. Després. *Neural networks and numerical analysis*. Volume 2, De Gruyter Series in Applied and Numerical Mathematics, 2022.

[4] K. Spiliopoulos, R. Sowers, J. Sirignano. *Mathematical foundations of Deep Learning models and algorithms*. AMS, 2025.

[5] G. Strang. *Linear Algebra and Learning from data*. Wellesley - Cambridge press,2019.