# Réseaux de neurones artificiels

## ANN

A. Carlier

2026

# Aperçu du cours

| Schedule | | |
|---|---|---|
| 13/10 AM | Artificial Neural Networks | ANNs, backpropagation, Stochastic Gradient Descent |
| 13/10 PM | Deep Learning | layers, convolution, architectures, training |
| 03/11 AM | Deep Learning for Computer Vision, pt 1 | Convolutional Neural Networks |
| 03/11 PM | Deep Learning for Computer Vision, pt 2 | CNNs and satellite imagery |
| 10/11 AM | Autoencoders and Self-Supervised Learning | autoencoders, SSL |
| 10/11 PM | Dimensionality Reduction | t-SNE, SAEs |
| 17/11 AM | Deep Learning in Practice | training, debugging, and deploying deep models |
| 17/11 PM | RNNs | Recurrent Neural Networks, LSTM, GRU |
| 24/11 AM | Image generation, pt 1 | VAEs and GANs |
| 24/11 PM | Image generation, pt 2 | Diffusion Models |
| 30/11 PM | Transformers | the Transformer architecture |
| 01/12 AM | LLMs | Large Language Models |
| 01/12 PM | Retrieval-Augmented Generation | RAG |

# Évaluation du cours

Projet Deep Learning (Décembre-Janvier) :

- Choix d'un article de recherche parmi environ soixante possibles
- Lecture et compréhension de l'article
- Ré-implémentation de l'article et mise en évidence d'au moins un résultat
- Proposition, implémentation et évaluation d'une petite extension à l'article
- Production d'une vidéo de présentation de votre travail

Evaluation par les pairs : chacun évaluera 5 autres étudiants

# Plan du cours

1. Régression linéaire et régression logistique
2. Neurones artificiels

# Apprentissage supervisé

Dans le cadre de l'**apprentissage supervisé**, on dispose d'observations et de leurs étiquettes (appelées encore cibles (*targets*), catégories ou *labels*) qui constituent un ensemble d'apprentissage. On le note :

$$\mathcal{D} = \{(\mathrm{x}^{(1)}, y^{(1)}), .., (\mathrm{x}^{(m)}, y^{(m)})\}.$$

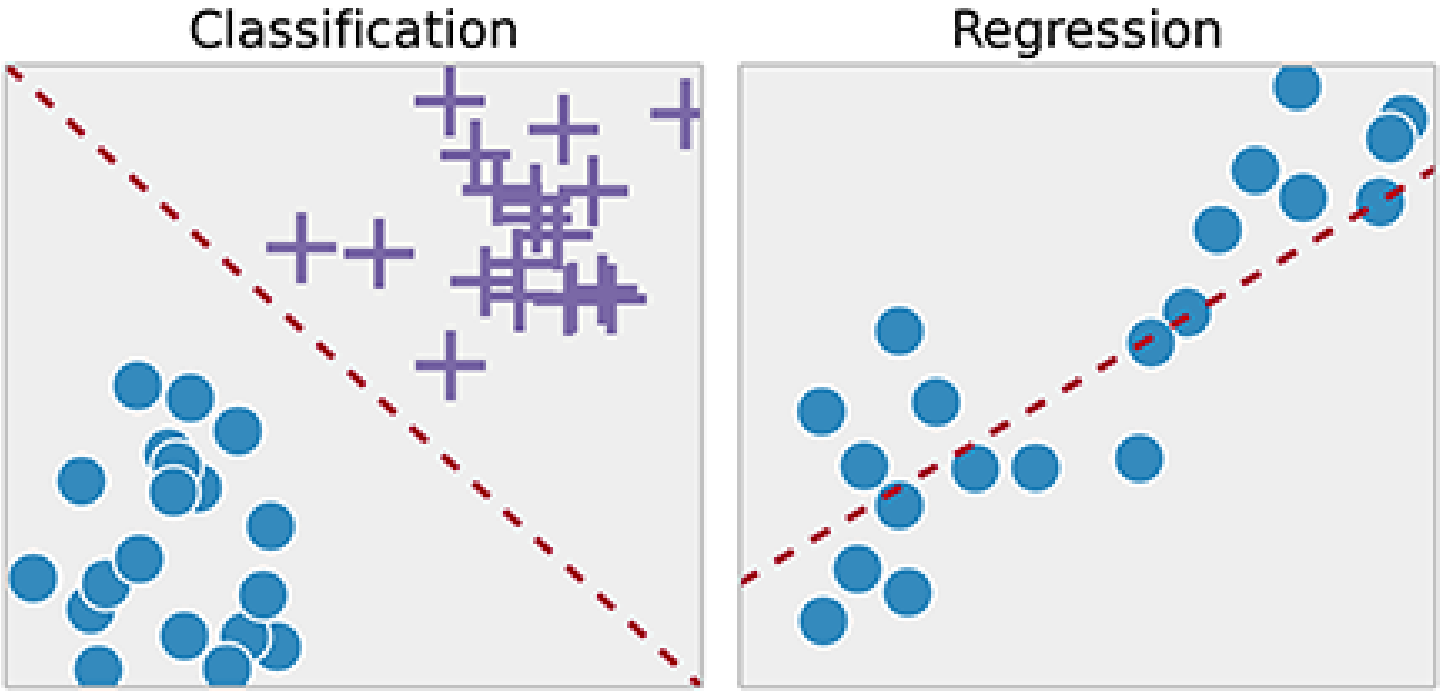
Les labels permettent d'enseigner à l'algorithme à établir des correspondances entre les observations et les labels.

# Apprentissage supervisé

Il existe deux principaux types d'apprentissage supervisé :

- **Classification :** Assigner une catégorie à chaque observation :
  - Les catégories sont discrètes
  - La cible est un indice de classe : $y \in \{0, .., K-1\}$
  - Exemple : reconnaissance de chiffres manuscrits :
    - ★ $\mathtt{x}$ : vecteur ou matrice des intensités des pixels de l'image
    - ★ $y$ : identité du chiffre
- **Régression :** Prédire une valeur réelle à chaque observation :
  - les catégories sont continues
  - la cible est un nombre réel $y \in \mathbb{R}$
  - Exemple : prédire le cours d'une action
    - ★ $\mathtt{x}$ : vecteur contenant l'information sur l'activité économique
    - ★ $y$ : valeur de l'action le lendemain

# Classification et Régression




$\Rightarrow$ Régression

# Régression linéaire

Soit l'ensemble $\mathcal{D}$ contenant $m$ exemples d'apprentissage.

Pour l'exemple, $\mathbf{x}^{(i)}$, le modèle linéaire s'écrit :

$$\hat{y}^{(i)} = \theta^T \mathbf{x}^{(i)}$$

et la fonction de coût quadratique s'écrit :

$$(\hat{y}^{(i)} - y^{(i)})^2$$

$\rightarrow$ Formulation aux **Moindres Carrés** !

Trouver $\theta$ qui minimise la perte sur tous les exemples d'apprentissage (fonction objectif $J(\theta)$) :

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{m} (\theta^T x^{(i)} - y^{(i)})^2 = J(\theta) \tag{1}$$

# Régression linéaire

Soit l'ensemble $\mathcal{D}$ contenant $m$ exemples d'apprentissage en dimension $d$ ($d$ variables), on définit :

- $\mathtt{X} \in \mathbb{R}^{m \times d}$ matrice des données
- $\mathtt{y} \in \mathbb{R}^m$ : vecteur de cibles
- $\hat{\mathtt{y}} \in \mathbb{R}^m$ : vecteur de prédictions avec $\hat{y} = \mathtt{X}\theta$
- $\theta \in \mathbb{R}^d$ vecteur des paramètres du modèle à estimer

## Régression aux moindres carrés

Estimer le modèle linéaire $\theta$ entre les données et les cibles en minimisant la somme des résidus quadratiques :

$$\min_{\theta} \|\mathtt{X}\theta - \mathtt{y}\|^2 = J(\theta)$$

# Régression linéaire

Résolution des problèmes aux moindres carrés :

*Condition Nécessaire du premier ordre :*

$$\frac{dJ(\theta)}{d\theta} = 0 = 2\mathrm{X}^T(\mathrm{X}\theta - \mathrm{y}).$$

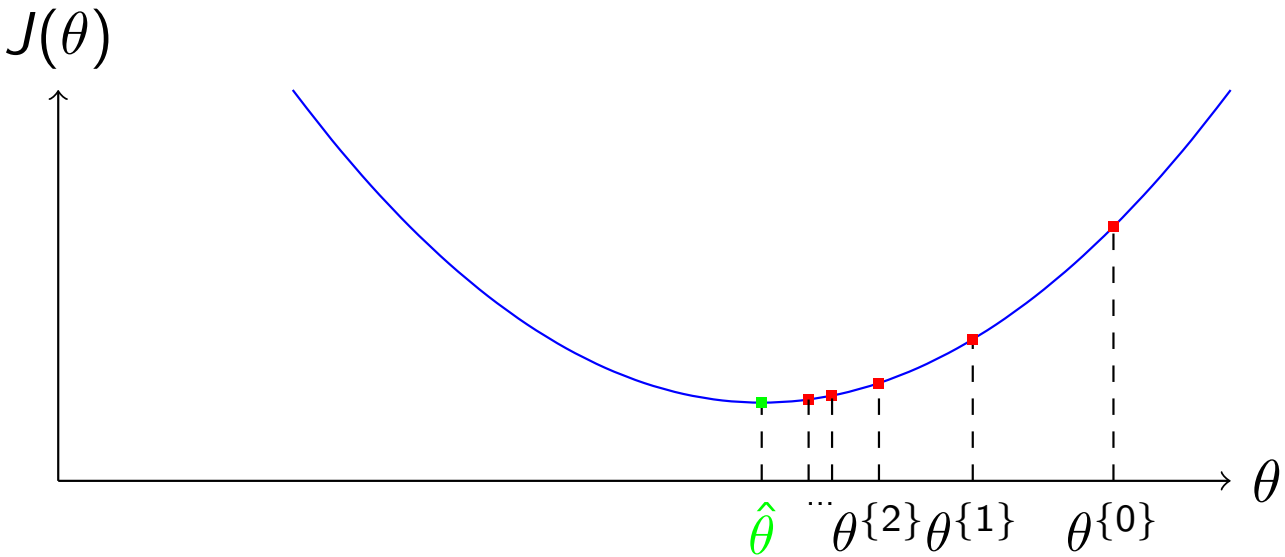
Si $det(\mathrm{X}^T\mathrm{X}) \neq 0$, la solution analytique est :

$$\hat{\theta} = (\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{y}$$

En pratique, calculs coûteux (inversion de matrice) donc solution itérative : **descente de gradient** !

**Remarque :** les problèmes aux moindres carrés sont **convexes** $\rightarrow$ minimum local est global !

# Régression linéaire




**Algorithme** : Descente de gradient $(\mathcal{D}, \alpha)$

Initialiser $\theta^{\{0\}} \leftarrow 0$, $k \leftarrow 0$

**TANT QUE** pas convergence **FAIRE**

  **POUR** $j$ de 1 à $d$ **FAIRE**

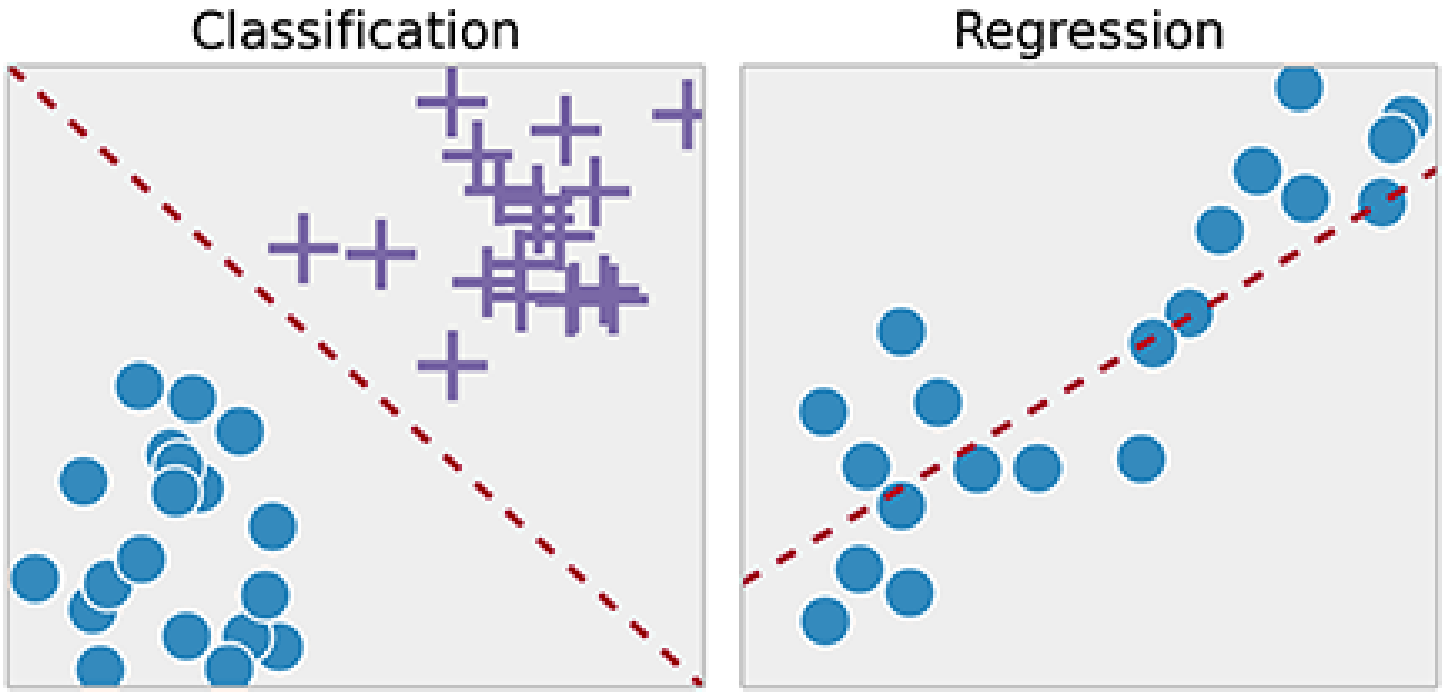
$$\theta_j^{\{k+1\}} \leftarrow \theta_j^{\{k\}} - \alpha \frac{\partial J(\theta^{\{k\}})}{\partial \theta_j}$$

  **FIN POUR**

  $k \leftarrow k + 1$

**FIN TANT QUE**

# Classification et Régression




$\Rightarrow$ Classification

# Régression logistique

En sortie, la sortie du modèle peut être :

- **binaire** : échec/succès, 0/1, -/+
  $\Rightarrow$ fonction **sigmoïde ou logistique**
- **multinomiale** (multi-classes) : chiffres
  $\Rightarrow$ fonction **softmax**

# Régression logistique : cas binaire

Comme avec la régression linéaire, on prend un modèle linéaire type :

$$z = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n = \theta^T \mathbf{x}$$

Ce modèle linéaire agit comme **séparateur** des 2 classes.
Puis on veut une probabilité : $0 \leq h_\theta(x) = g(z) \leq 1$ telle que :

- si $h_\theta(x) \leq 0.5$, alors la classe est 0 ($y = 0$)
- si $h_\theta(x) > 0.5$, alors la classe est 1 ($y = 1$)

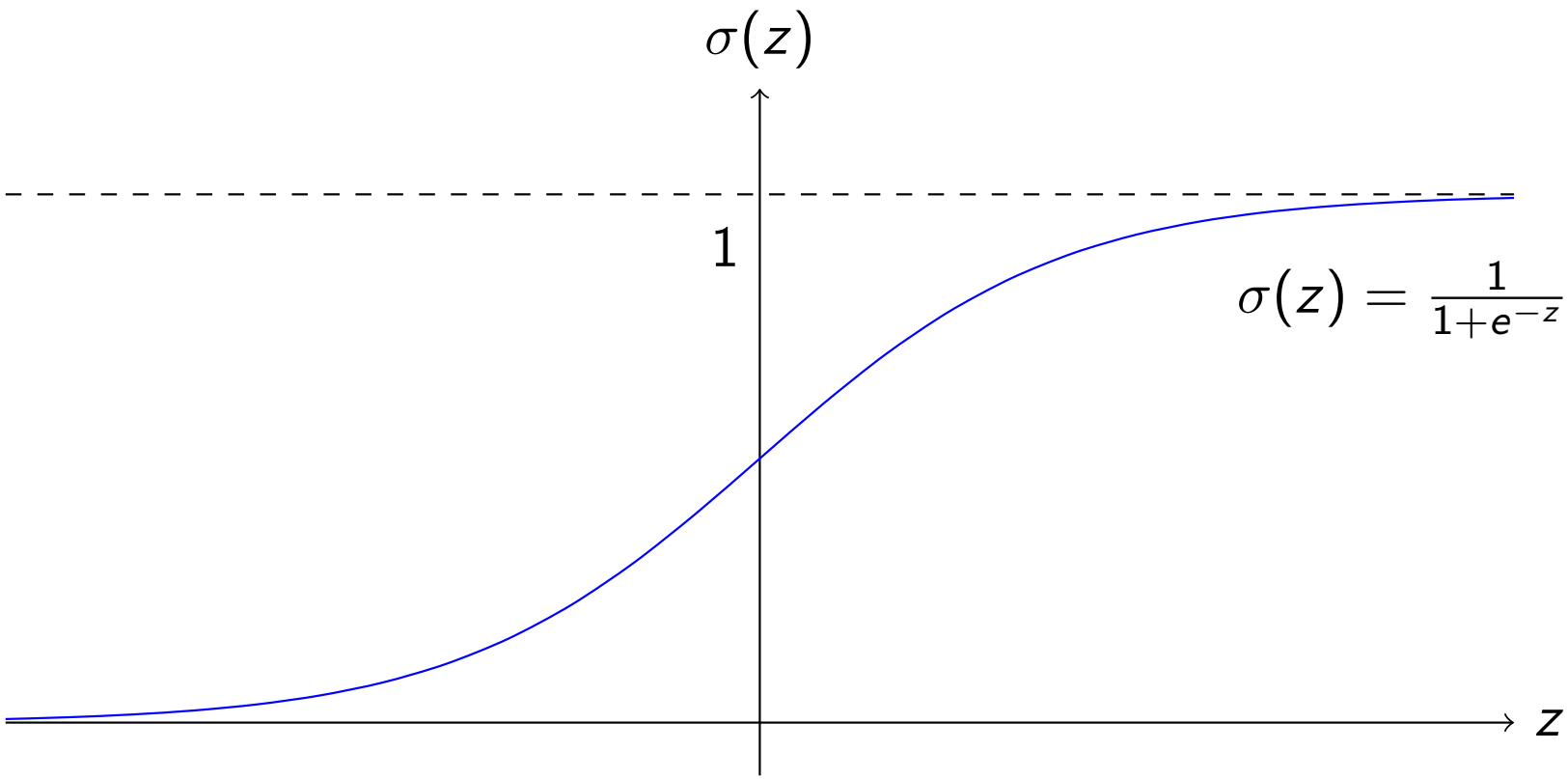
Quelle est cette fonction $g$ telle que :

$$h_\theta(x) = g(z) = g(\theta^T \mathbf{x}) = P(y = 1|x; \theta) \ ?$$

# Régression logistique : cas binaire

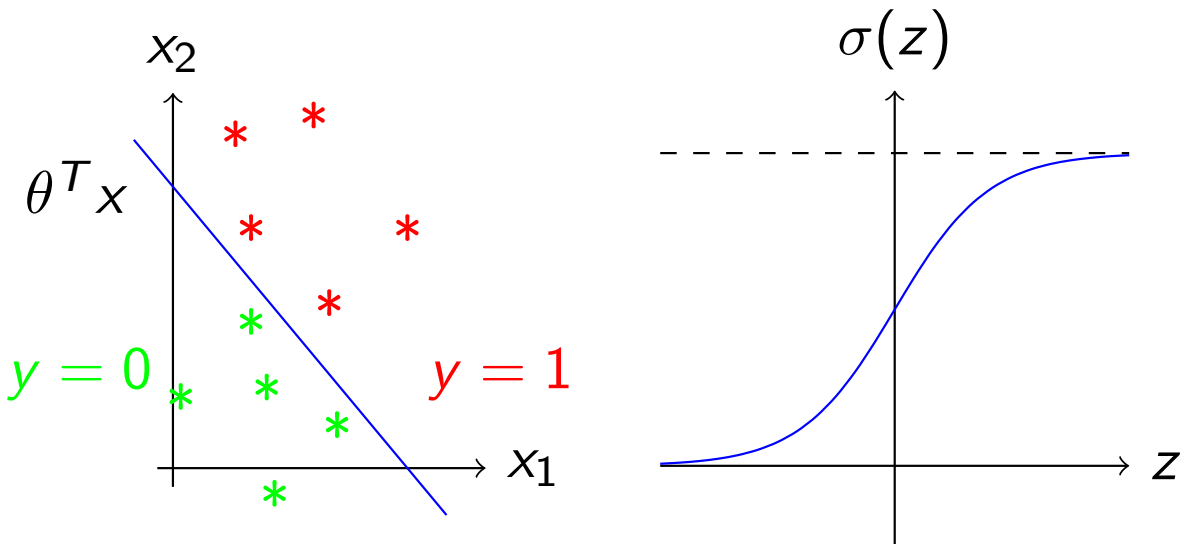
La **sigmoïde** ou **fonction logistique** définie par :

$$\sigma(z) = \frac{1}{1 + exp(-z)}$$



# Régression logistique : cas binaire

**Cas 2D où les données sont linéairement séparables**




$$h_\theta(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

- on prédit $y = 1$ si $-3 + x_1 + x_2 > 0$
- on prédit $y = 0$ si $x_1 + x_2 \leq 3$

# Régression logistique : cas binaire

Comment estimer automatiquement $\theta$ ?

On a un **corpus d'apprentissage** $\mathcal{D}$, contenant $m$ exemples avec :

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_d^{(i)} \end{bmatrix} \in \mathbb{R}^d \text{ et } y^{(i)} \in \{0,1\}, \forall i \in \{1,..,m\}$$

et le **modèle** est défini par la fonction sigmoïde appliquée à l'équation de l'hyperplan séparateur :

$$P(y=1|x;\theta) = \frac{1}{1+e^{-\theta^T x}}$$

# Régression logistique : cas binaire

Il faut minimiser une **fonction objectif** à l'aide d'une technique d'optimisation (descente de gradient).

Peut on utiliser la même fonction de perte que pour la régression linéaire ?

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} perte\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

avec

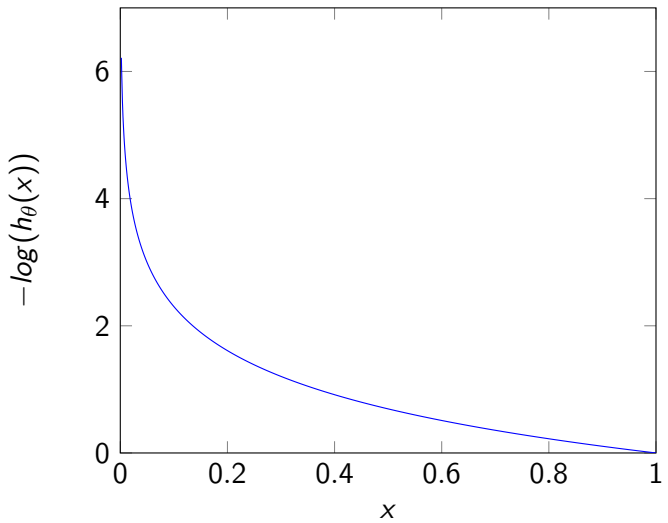
$$perte(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 = \frac{1}{2}\left(\frac{1}{1+e^{-\theta^T x^{(i)}}} - y^{(i)}\right)^2$$

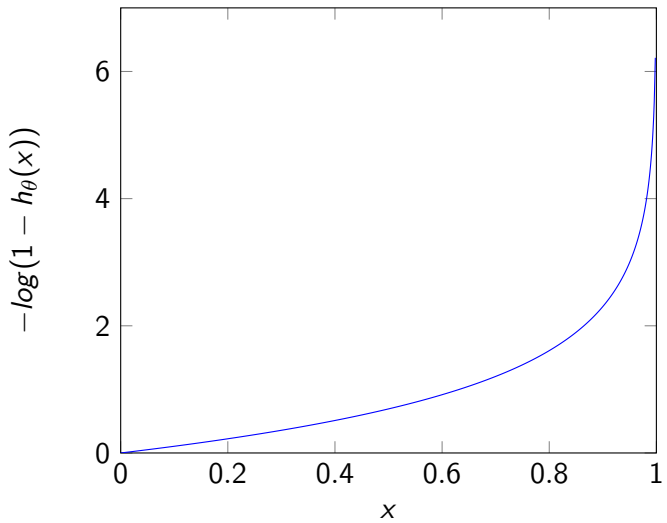
Cette fonction n'est **pas convexe** donc la descente de gradient ne garantit pas un minimum global !

# Régression logistique : cas binaire

On introduit donc la **fonction de perte logistique ou entropie croisée (cross-entropy)** définie par :

$$\begin{aligned} perte(h_{\theta(x)}, y) &= \begin{cases} -log(h_\theta(x)) \text{ si } y = 1 \\ -log(1 - h_\theta(x)) \text{ si } y = 0 \end{cases} \\ &= -ylog(h_\theta(x)) - (1 - y)log(1 - h_\theta(x)) \end{aligned}$$



# Régression logistique : cas binaire

Sur les $m$ exemples, la fonction de perte devient :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} log(h_\theta(x^{(i)})) - (1-y^{(i)}) log(1-h_\theta(x^{(i)}))$$

Pour minimiser cette fonction, on applique la descente de gradient.

$$\begin{aligned}
\frac{\partial J}{\partial \theta_j} &= \frac{\partial}{\partial \theta_j}\frac{1}{m}\sum_{i=1}^{m} -y^{(i)} log(h_\theta(x^{(i)})) - (1-y^{(i)}) log(1-h_\theta(x^{(i)})) \\
&= \frac{1}{m}\sum_{i=1}^{m} -y^{(i)} \frac{\partial}{\partial \theta_j}(log(h_\theta(x^{(i)}))) - (1-y^{(i)}) \frac{\partial}{\partial \theta_j} log(1-h_\theta(x^{(i)})) \\
&= \frac{1}{m}\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} \qquad \text{(tous calculs faits)}
\end{aligned} \tag{2}$$

# Précision sur le calcul précédent

Pour le calcul de $\frac{\partial}{\partial \theta_j}(log(h_\theta(x)))$, on pose $z = \theta^T x$, $u = \sigma(z)$ and $v = log(u)$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j}(log(h_\theta(x))) &= \frac{\partial}{\partial \theta_j}(log(\sigma(\theta^T x))) \\
&= \frac{\partial v}{\partial u}\frac{\partial u}{\partial z}\frac{\partial z}{\partial \theta_j} && \text{chain-rule} \\
&= \frac{1}{\sigma(z)}\sigma(z)(1-\sigma(z))x_j && \text{car } \sigma'(z) = \sigma(z)(1-\sigma(z)) \\
&= (1-h_\theta(x))x_j
\end{aligned}$$

On obtient par le même procédé $\frac{\partial}{\partial \theta_j}(log(1-h_\theta(x))) = -h_\theta(x)x_j$

# Régression logistique : cas multiclasse

Comment faire quand on a $k$ **classes avec** $k > 2$ ?

On utilise la **fonction softmax** :

$$P(y = i|x, \theta) = \frac{e^{\theta_i^T x}}{\sum_{j=1}^{k} e^{\theta_j^T x}}$$

avec $y^{(i)} = [0 \ \ldots \ 0 \ 1 \ 0 \ldots \ 0]^T$ avec 1 à la $i$ème coordonnée.

# Régression logistique : cas multiclasse

Pour chaque vecteur de données de test $x^{(i)}$, on calcule un vecteur de probabilités d'obtenir l'une des $k$ classes.

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1|x^{(i)};\theta) \\ p(y^{(i)} = 2|x^{(i)};\theta) \\ \vdots \\ p(y^{(i)} = k|x^{(i)};\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

# Régression logistique : cas multiclasse

La fonction objectif devient :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k}\mathbb{I}(y^{(i)}=j)log\left(\frac{e^{\theta_j^T x^{(i)}}}{\sum_{p=1}^{k} e^{\theta_p^T x^{(i)}}}\right) \quad (3)$$

Le gradient s'écrit :

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\left(\mathbb{I}(y^{(i)}=j) - P(y^{(i)}=j|x^{(i)};\theta)\right)$$

# Plan du cours

1 Régression linéaire et régression logistique

2 Neurones artificiels

# Perceptron monocouche

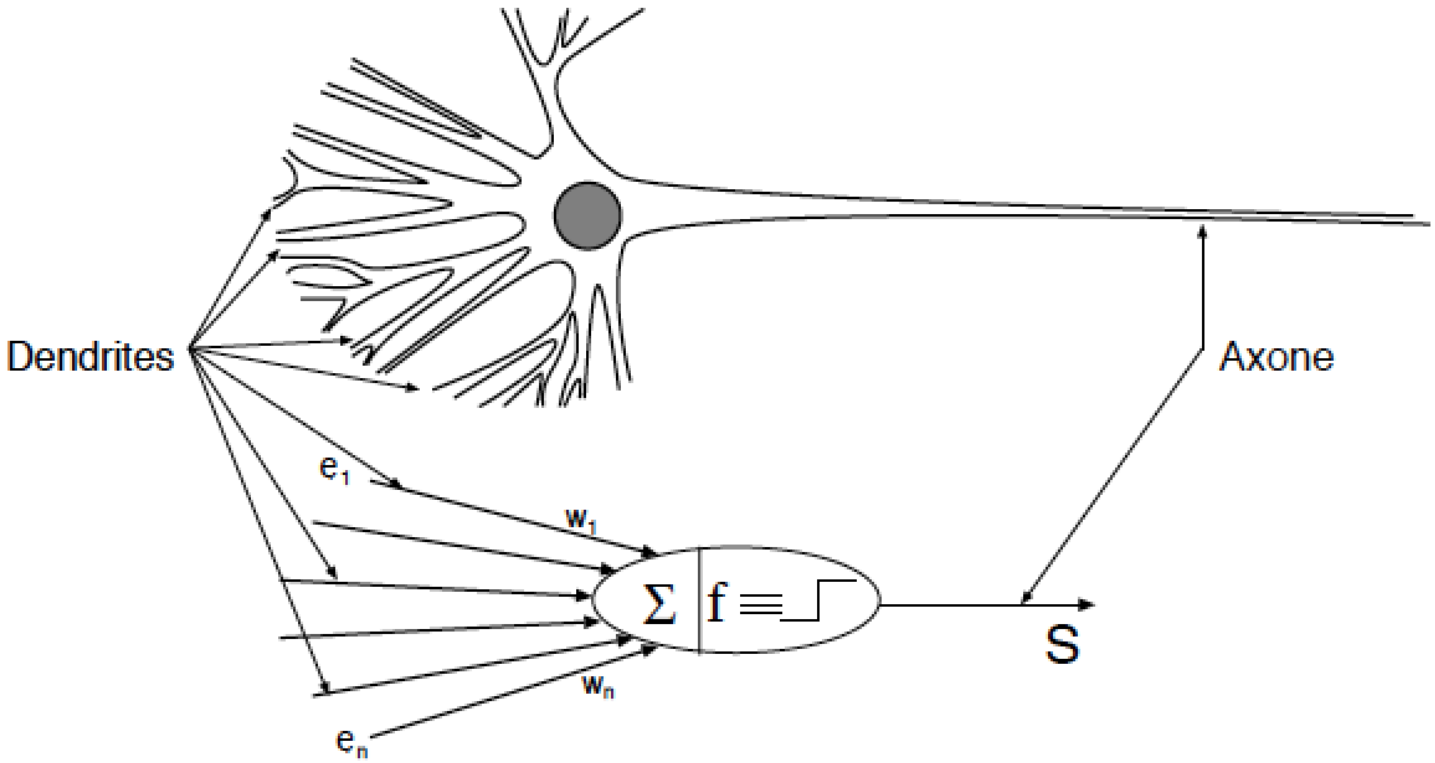
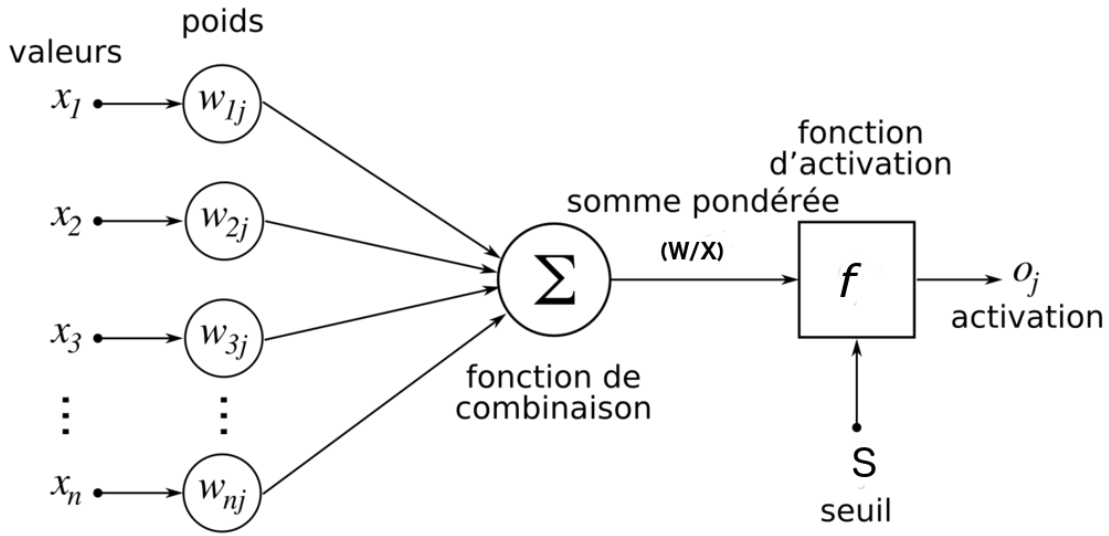



Figure – Structure d'un neurone biologique/artificiel

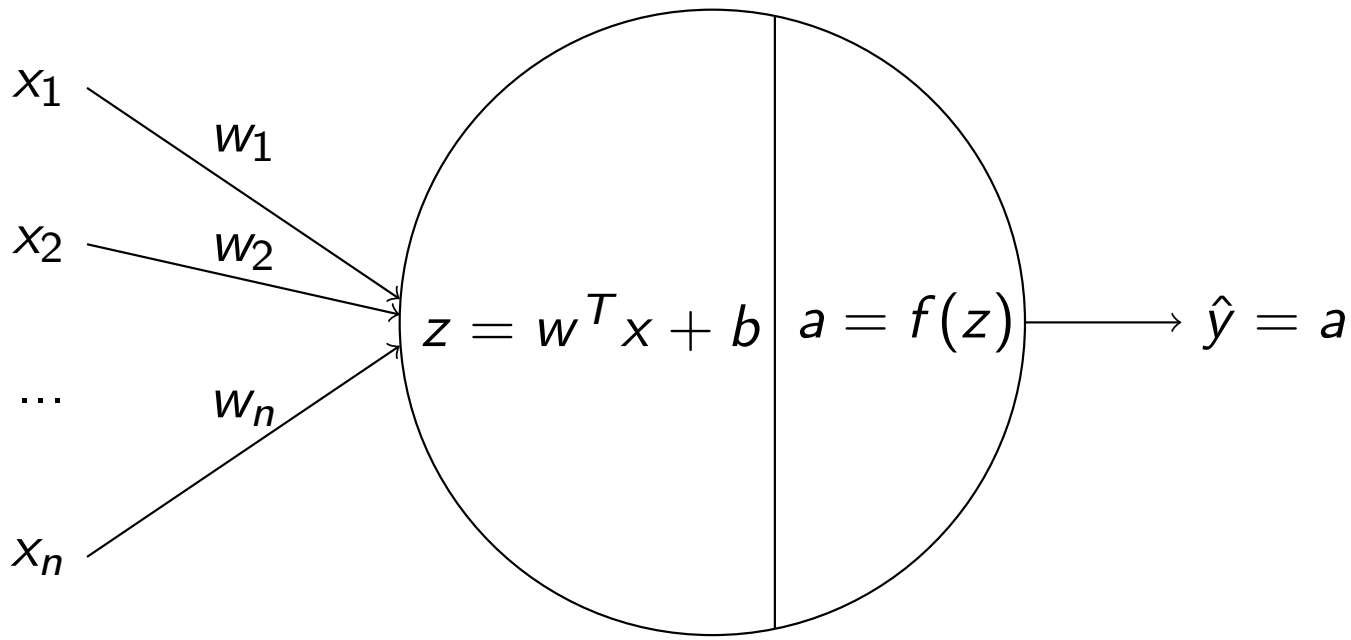
# Perceptron monocouche : Fonctionnement




On assimile un neurone à un triplet : poids synaptique $w$, seuil $S$ (ou biais $b$), fonction d'activation $f$.

1. produit scalaire entre les entrées $x$ et les poids synaptiques $w$ : $w^T x$ ;
2. ajout d'une valeur de référence (biais $b$) : $z = w^T x + b$
3. application de la fonction d'activation à la valeur obtenue $z$ : $a = f(z)$

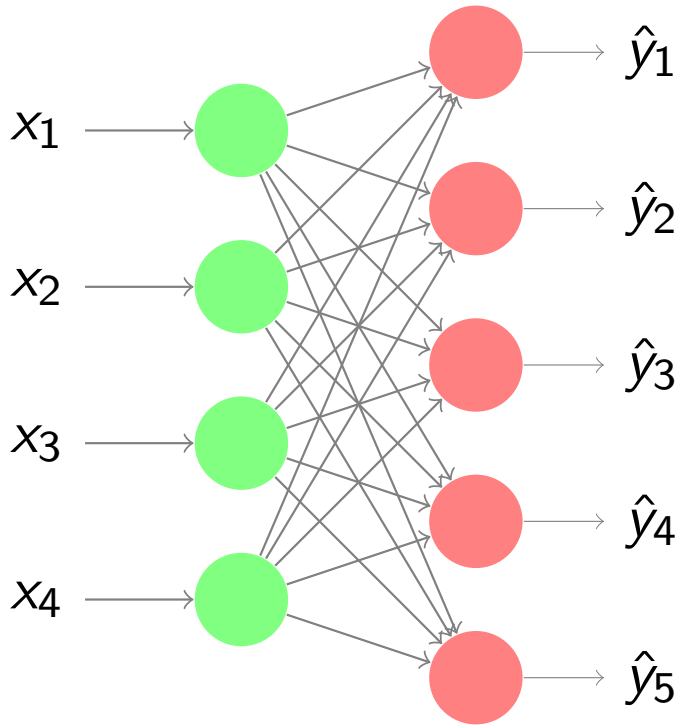
# Perceptron monocouche : Fonctionnement




1. produit scalaire entre les entrées $x$ et les poids synaptiques $w$ : $w^T x$ ;
2. ajout d'une valeur de référence (biais $b$) : $z = w^T x + b$
3. application de la fonction d'activation à la valeur obtenue $z$ : $a = f(z)$

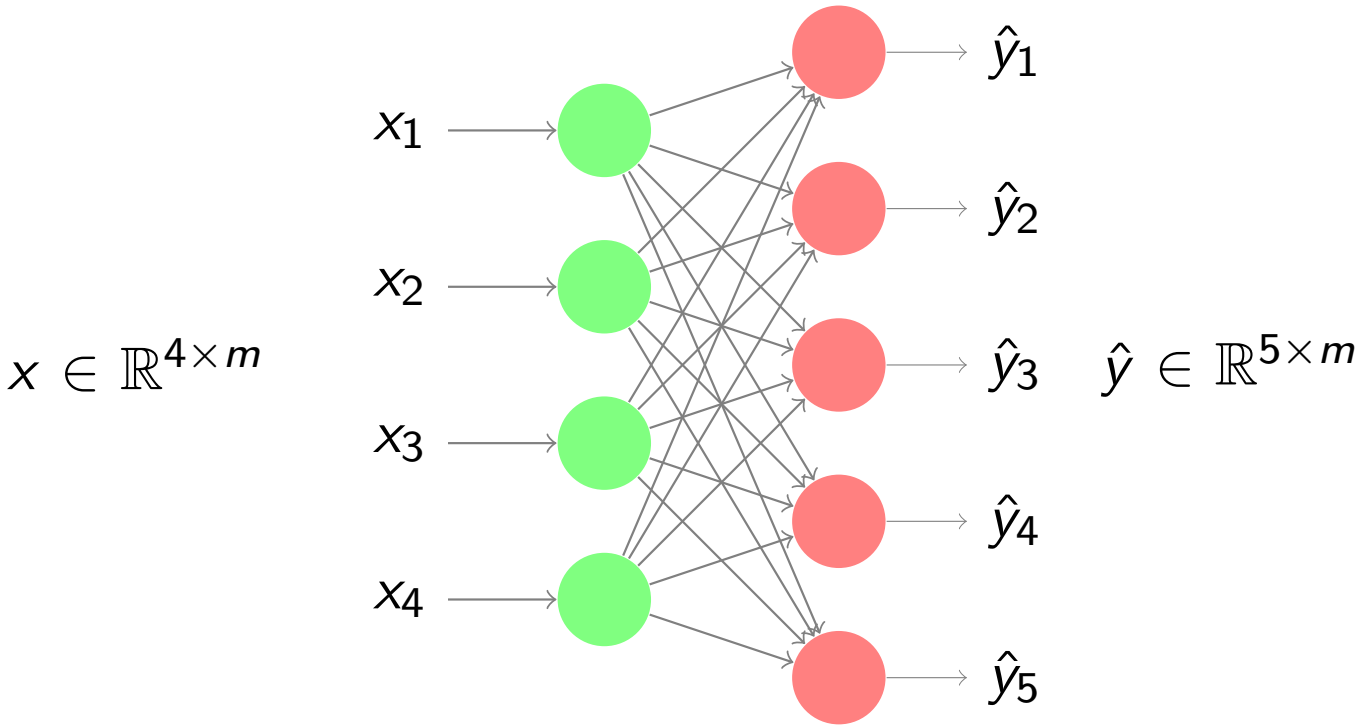
Représentation "neurone"

# Réseau de neurones : Perceptron monocouche




Le vocable de perceptron monocouche est en fait un peu plus général et désigne également les cas où la sortie est définie par plusieurs neurones tels que décrits dans les diapositives précédentes.

# Réseau de neurones : Perceptron monocouche

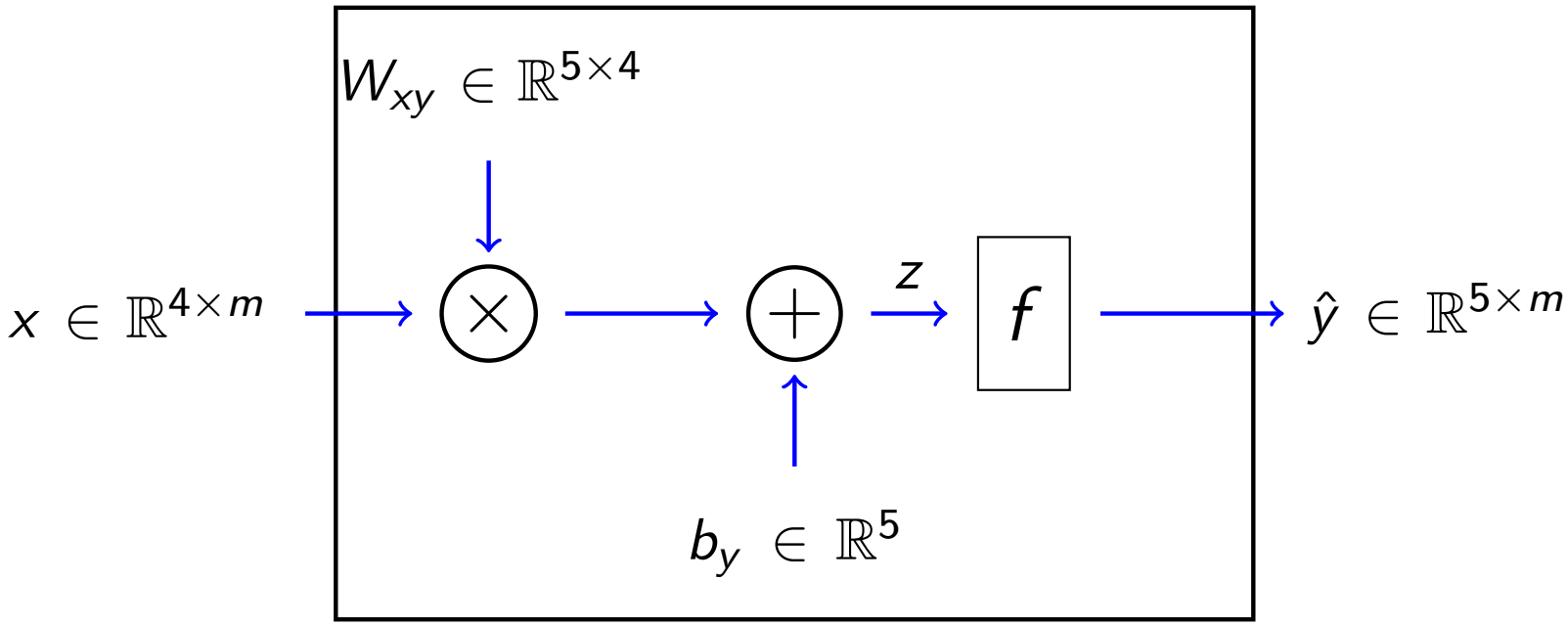



La prédiction du perceptron est implémentée par un calcul matriciel et permet la prédiction de plusieurs entrées simultanément :

$$\hat{y} = f(W_{xy}x + b_y)$$

où $f$ est la fonction d'activation.

# Réseau de neurones : Perceptron monocouche

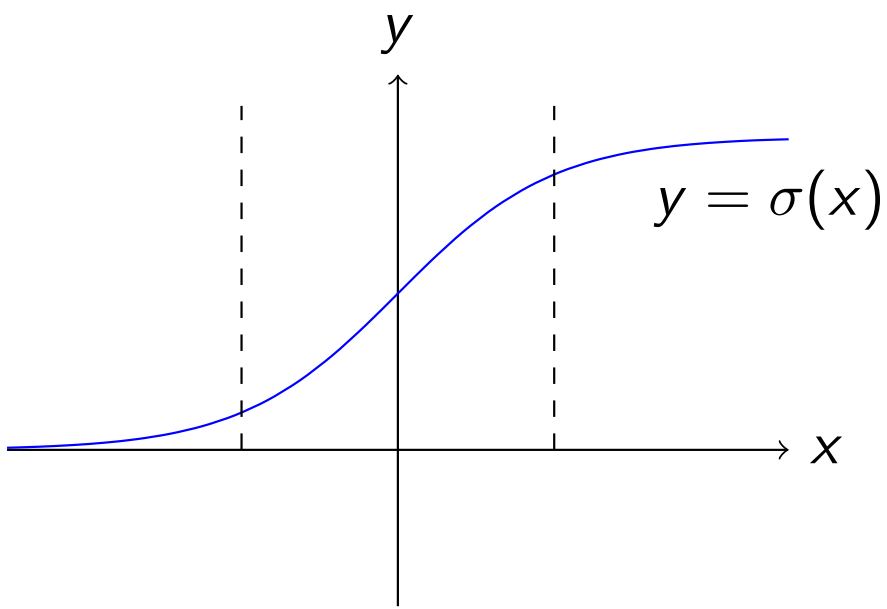



Une autre représentation, utile pour la suite.

# Perceptron monocouche

Les **fonctions d'activation**, notées $f$, sont des fonctions de seuillage qui peuvent souvent se décomposer en trois parties, comme c'est le cas pour la fonction sigmoïde (ci-dessous) :

- une partie non-activée, en dessous du seuil ;
- une phase de transition, aux alentours du seuil ;
- une partie activée, au dessus du seuil.



# Perceptron monocouche

**Algorithm 1** Algorithme du perceptron monocouche

1. Initialisation des poids $W_{xy}^{\{0\}}$ et biais $b_y^{\{0\}}$
2. Présentation d'un vecteur d'entrées $x^{(1)}, ..., x^{(m)}$ et du vecteur de sortie correspondant $y^{(1)}, .., y^{(m)}$
3. Calcul de la sortie prédite et de la fonction objectif :

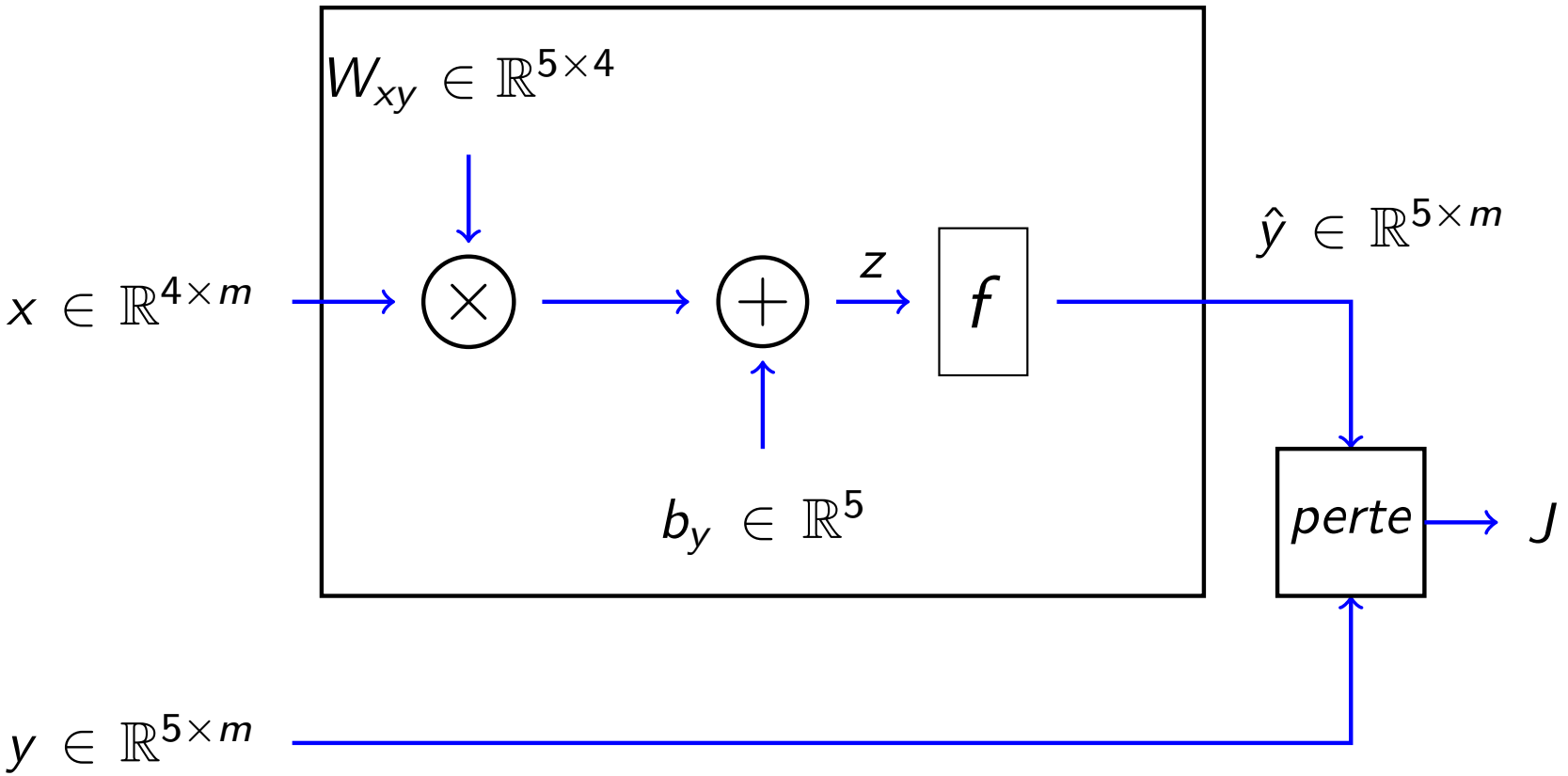
$$\hat{y}^{(i)\{k\}} = f(W_{xy}^{\{k\}} x^{(i)} + b_y^{\{k\}}) \text{ et } J = \sum_{i=1}^{m} \text{perte}(\hat{y}^{(i)\{k\}}, y^{(i)})$$

4. Mise à jour des paramètres

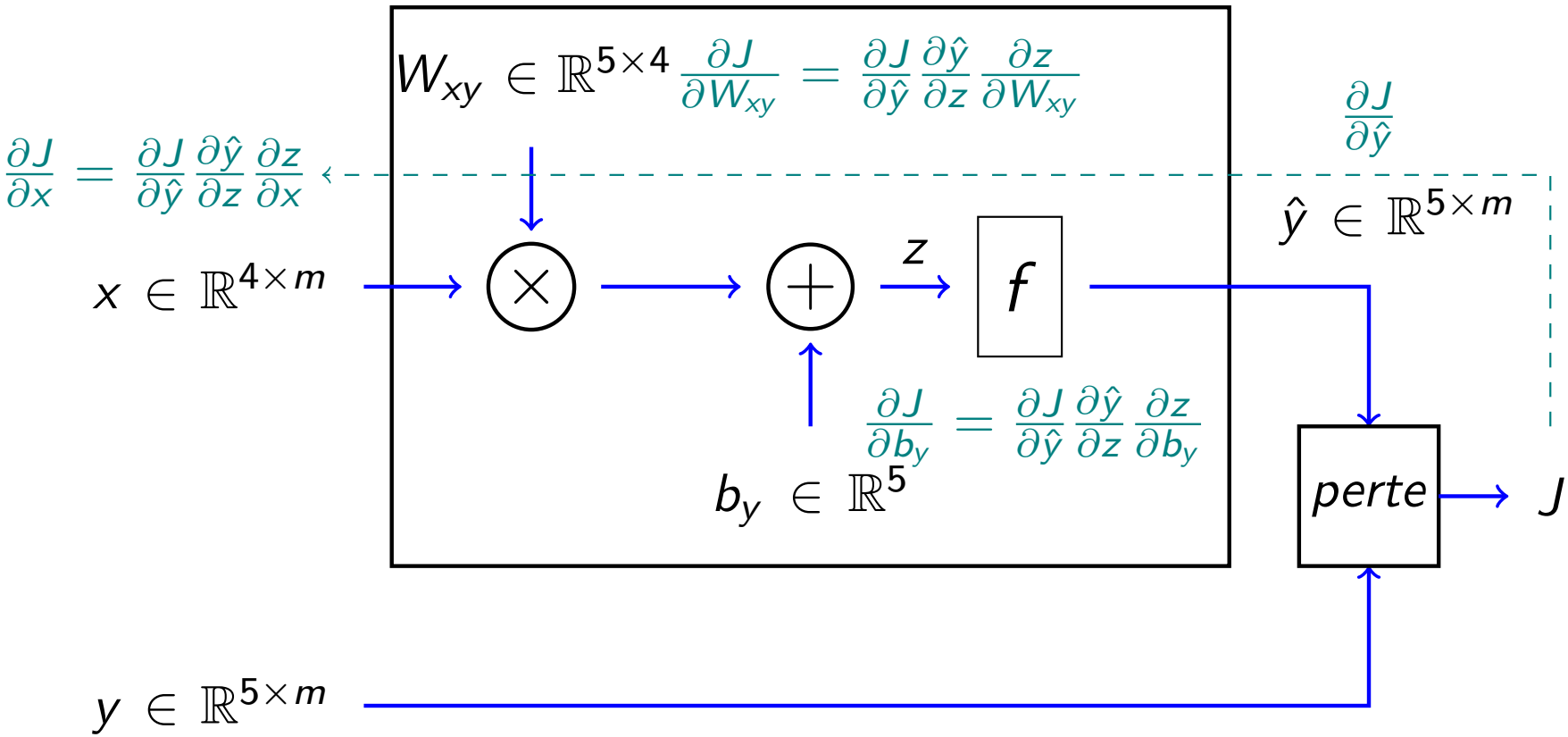
$$W_{xy}^{\{k+1\}} = W_{xy}^{\{k\}} - \alpha \frac{\partial J}{\partial W_{xy}}, \text{ et } b_y^{\{k+1\}} = b_y^{\{k\}} - \alpha \frac{\partial J}{\partial b_y}$$

   où $\alpha$ est le taux d'apprentissage ($0 \leq \alpha \leq 1$).
5. Retourner en 2 jusqu'à la convergence (c'est-à-dire $\hat{y}^{(i)\{k\}} \approx y^{(i)}$).

# Perceptron monocouche : fonction objectif



# Perceptron monocouche : calcul des gradients




Plus de détails en TP !

# IMPORTANT

Pour résoudre un problème de **régression**, choisir :

- Fonction d'activation **linéaire** : $f(x) = x$
- Fonction de coût **quadratique** :

$$J = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2$$

$\Rightarrow$ Perceptron monocouche **équivalent à la régression linéaire** !

# Classification binaire

# IMPORTANT

Pour résoudre un problème de **classification binaire**, choisir :

- Fonction d'activation **sigmoïde** : $\sigma(x) = \frac{1}{1+e^{-x}}$
- Fonction de coût **entropie croisée binaire** :

$$J = \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} log(\hat{y}^{(i)}) - (1 - y^{(i)}) log(1 - \hat{y}^{(i)})$$

$\Rightarrow$ Perceptron monocouche **équivalent à la régression logistique** !

# Classification n-aire

# IMPORTANT

Pour résoudre un problème de **classification n-aire** (plus de deux classes), choisir :

- Fonction d'activation **softmax** : $\text{softmax}(x) = \frac{1}{\sum_{j=1}^{n} e^{x_j}} \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$
- Fonction de coût **entropie croisée multinomiale** :

$$J = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} -y_j^{(i)} log(\hat{y}_j^{(i)})$$

$\Rightarrow$ Perceptron monocouche **équivalent à la régression logistique** !