Classification Binaire & Perceptron

Jour 2 — Matin

Julien Rolland

Formation M2 Développement Fullstack

Jour 2

Plan du module

Régression (J1) : sortie **continue** $\hat{y}_i \in \mathbb{R}$

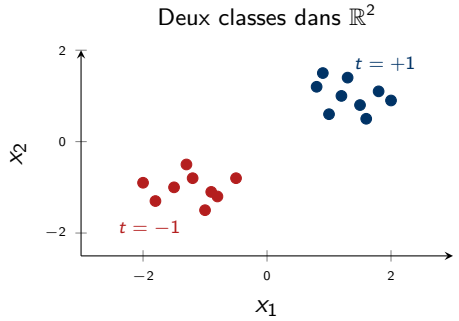
Classification (J2) : classe **discrète** $\hat{t}_n \in \{+1, -1\}$

Exemples :

- Email → spam / ham
- Image → chat / chien
- Tumeur → maligne / bénigne

Enjeu

Trouver une **frontière de décision** qui sépare les classes dans l'espace des features.



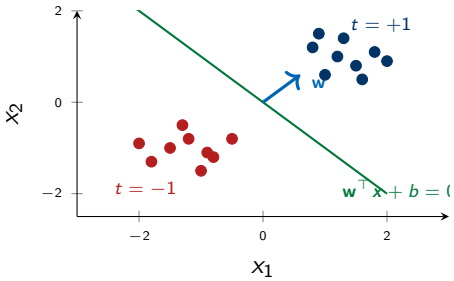
Modèle & Prédiction

$$f_\Theta(\boldsymbol{x}_n) = \mathbf{w}^\top \boldsymbol{x}_n + b \in \mathbb{R}$$

$$\hat{t}_n = \text{sign}(f_\Theta(\boldsymbol{x}_n)) \in \{+1, -1\}$$

Le signe de $f_\Theta(\boldsymbol{x}_n)$ indique de quel côté de l'hyperplan se trouve $\boldsymbol{x}_n$.

Hyperplan séparateur dans $\mathbb{R}^2$



Pourquoi le Gradient Échoue

Pour optimiser Θ par descente de gradient :

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}$$

Problème : le readout $\text{sign}(z)$ est **non-différentiable** en 0, dérivée **nulle** ailleurs.

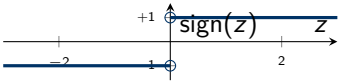
Conséquences

- sign dans la loss $\Rightarrow \nabla_{\mathbf{w}} \mathcal{L} = 0$ presque partout.
- Le gradient ne transporte **aucune information**.
- GD est **bloqué mathématiquement**.

Solution

Ne **pas** mettre le readout dans la loss.
Utiliser une fonction **différentiable** à la place.

Readout : sign



Dérivée : nulle partout (infinie en 0)

ReLU — La Fonction Clé

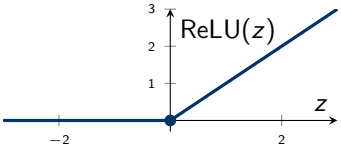
Définition

$$\text{ReLU}(z) = \max(0, z)$$

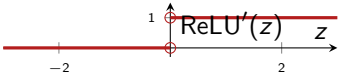
Dérivée

$$\text{ReLU}'(z) = \mathbf{1}[z > 0] = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

ReLU



Dérivée — toujours calculable



Pourquoi c'est le standard :

- Gradient **constant** ($= 1$) pour $z > 0$ — l'information circule.
- Ultra-rapide : pas d'exponentielle.
- Atténue le *Vanishing Gradient* (Deep Learning, J3).

Idée : mesurer l'erreur par le produit $f \cdot t_n$.

- $f \cdot t > 0$: bonne classe $\Rightarrow$ perte $= 0$.
- $f \cdot t < 0$: mauvaise classe $\Rightarrow$ pénalité.

Loss de Rosenblatt

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \mathrm{ReLU}\big(-f_\Theta(\boldsymbol{x}_n) \cdot t_n\big)$$

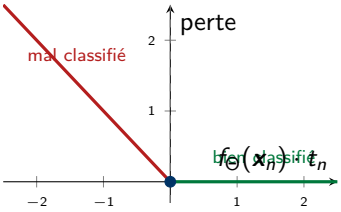
Gradient pour un exemple **mal classifié** ($f \cdot t < 0$) :

$$\nabla_{\mathbf{w}} \,\mathrm{ReLU}(-f \cdot t) = -t_n \, \boldsymbol{x}_n$$

Règle de mise à jour

Si $\hat{t}_n \neq t_n$: $\quad \mathbf{w} \leftarrow \mathbf{w} + \alpha \, t_n \, \boldsymbol{x}_n$

$\mathrm{Loss} = \mathrm{ReLU}(-f \cdot t)$



Encodage ± 1 crucial

Avec $t \in \{0, 1\}$, le produit $f \cdot t$ serait nul pour la classe 0 — la loss ne fonctionnerait pas.

Algorithme Online de Rosenblatt

Algorithme

Entrée : $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$, $t_1, \ldots, t_N \in \{+1, -1\}$, α

$\tilde{\boldsymbol{x}}_n \leftarrow [1,\ \boldsymbol{x}_n^\top]^\top \in \mathbb{R}^{D+1}$ *(biais absorbé)*

$\boldsymbol{w} \leftarrow \vec{0} \in \mathbb{R}^{D+1}$

pour chaque époque **faire**

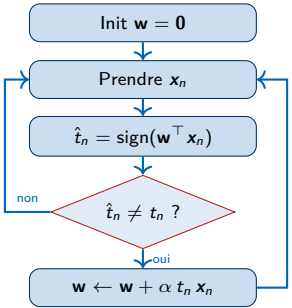
  pour $n = 1$ **à** N **faire**

    $\hat{t}_n \leftarrow \text{sign}(\boldsymbol{w}^\top \tilde{\boldsymbol{x}}_n)$

    si $\hat{t}_n \neq t_n$ **alors**

      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\, t_n\, \tilde{\boldsymbol{x}}_n$

retourner $\boldsymbol{w}$



« Online » = un exemple à la fois

Traite les données comme un **flux**, pas comme un lot statique. Instable sur données **bruitées**.

Online vs Batch Gradient Descent

Stratégie	Gradient sur	Stabilité
Online (SGD pur)	1 exemple	Très instable
Mini-Batch	B exemples (32–256)	Bon équilibre
Batch (full)	Tout le dataset (N)	Très stable

Mini-Batch GD — Pratique moderne

1. Mélanger le dataset.
2. Découper en batches de taille B.
3. Pour chaque batch : gradient $\rightarrow$ update $\mathbf{w}$.
4. Répéter sur toutes les **époques**.

Avantage clé : vectorisation

Un batch $(X_b, \boldsymbol{t}_b)$ de taille B permet de calculer le gradient en un seul produit matriciel.
Exploitable par **NumPy** et les **GPU**.

Convergence

Si les données sont **linéairement séparables**, le perceptron converge.
Sinon : il oscille $\Rightarrow$ utiliser une loss différentiable.

Remplacer sign par une fonction **lisse** qui retourne une **probabilité**.

Modèle & Loss (BCE)

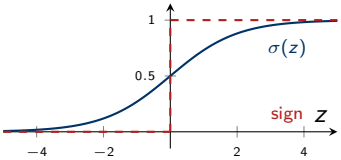
$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad f_\Theta(\boldsymbol{x}_n) = \sigma(\mathbf{w}^\top \boldsymbol{x}_n) \approx P(t_n{=}1 \mid \boldsymbol{x}_n)$$

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N} \sum_n \big[t_n \log f_n + (1 - t_n) \log(1 - f_n) \big]$$

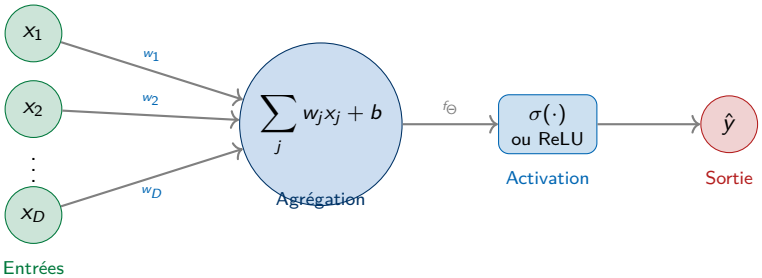
Avantage

σ diff. partout — pénalise lourdement les erreurs confiantes ($\mathcal{L} \to +\infty$).

Sigmoïde vs sign



Un Neurone Artificiel



Perceptron = réseau à 0 couche cachée

Entrées → agrégation → readout.
Premier réseau de neurones (Rosenblatt, 1958).

Readout vs Activation

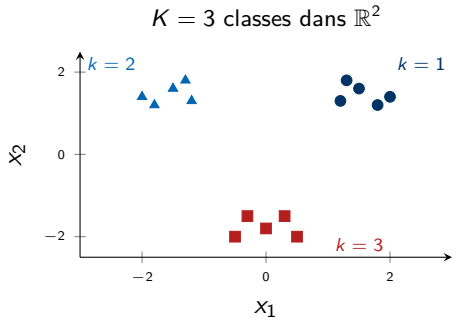
Activation : dans le réseau (ReLU, tanh, σ) — différentiable.
Readout : décision finale (sign, arg max).

Classification binaire (jusqu'ici) : $\hat{t}_n \in \{+1, -1\}$

Multi-classes : K catégories discrètes,
$\hat{k} \in \{1, \ldots, K\}$

Exemples :

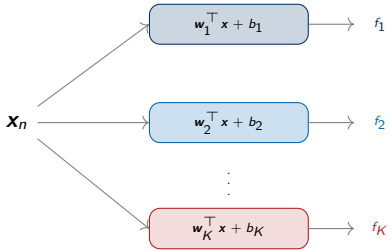
- Chiffres MNIST ($K = 10$)
- Objets CIFAR-10 ($K = 10$)
- Fleurs Iris ($K = 3$)



Enjeu

Séparer K classes — une **frontière de décision** par classe.

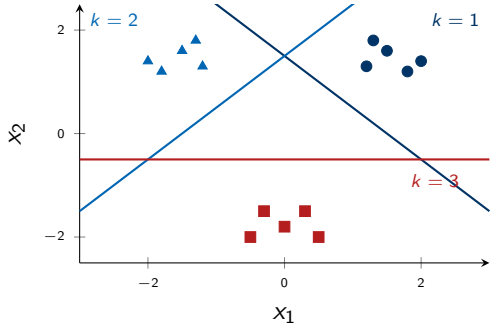
Un Perceptron par Classe



Score de la classe k :

$$f_k(\boldsymbol{x}_n) = \boldsymbol{w}_k^\top \boldsymbol{x}_n + b_k \in \mathbb{R}$$

Un hyperplan séparateur par classe



Softmax → probabilités

$$p_k = \frac{e^{f_k}}{\sum_{j=1}^{K} e^{f_j}} \in (0,1), \qquad \sum_{k=1}^{K} p_k = 1$$

Lisse et différentiable : adaptée à la descente de gradient.

Les scores bruts f_k (**logits**) sont convertis en distribution de probabilité sur les K classes.

Encodage & Prédiction

$\boldsymbol{t}_n \in \{0,1\}^K$ (**one-hot**) : une seule composante à 1.

$$k = 1 \rightarrow \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad k = 2 \rightarrow \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Readout : $\hat{k} = \arg\max_k \; p_k(\boldsymbol{x}_n)$

Exemple $K = 3$

Logits :	[3.0, 1.5, 0.5]
Softmax :	[0.77, 0.17, 0.06]
Prédiction :	$\hat{k} = 1$

K Perceptrons = 1 Matrice

K équations séparées, une par classe :

$$f_k(\boldsymbol{x}_n) = \boldsymbol{w}_k^\top \boldsymbol{x}_n + b_k, \quad k = 1, \ldots, K$$

Empilement $\Rightarrow$ 1 équation matricielle

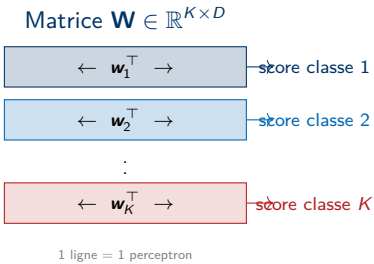
$$\underbrace{\boldsymbol{f}(\boldsymbol{x}_n)}_{\mathbb{R}^K} = \underbrace{\mathbf{W}}_{\mathbb{R}^{K\times D}} \underbrace{\boldsymbol{x}_n}_{\mathbb{R}^D} + \underbrace{\boldsymbol{b}}_{\mathbb{R}^K}$$

Ligne k de $\mathbf{W} = \boldsymbol{w}_k^\top$ (perceptron de classe k).

Forme batch (tout le dataset)

$\boldsymbol{F} = \boldsymbol{X}\mathbf{W}^\top + \mathbf{1}_N \boldsymbol{b}^\top$, $\quad \boldsymbol{X} \in \mathbb{R}^{N\times D}$, $\boldsymbol{F} \in \mathbb{R}^{N\times K}$.

1 seul produit matriciel pour les N exemples.



$\boldsymbol{P} \in \mathbb{R}^{N\times K}$: ligne $n = \boldsymbol{p}_n$ (probas softmax de $\boldsymbol{x}_n$)
$\boldsymbol{T} \in \mathbb{R}^{N\times K}$: ligne $n = \boldsymbol{t}_n$ (label one-hot de $\boldsymbol{x}_n$)

Cross-Entropy Catégorielle (CCE)

$$\mathcal{L} = -\frac{1}{N}\sum_{n}\sum_{k} t_{n,k}\,\log(p_{n,k})$$

Seul le log de la **vraie classe** contribue ($t_{n,k} = 0$ pour les autres).

Gradient w.r.t. les logits

$$\frac{\partial \mathcal{L}}{\partial f_{n,k}} = p_{n,k} - t_{n,k}$$

Erreur = proba prédite − label one-hot.
Résultat **élégant** : Softmax + log + CCE.

Gradients pour W et b

Par règle de chaîne ($f_{n,k} = \boldsymbol{w}_k^\top \boldsymbol{x}_n + b_k$) :

$$\nabla_{\mathbf{W}}\mathcal{L} = \frac{(\boldsymbol{P}-\boldsymbol{T})^\top \boldsymbol{X}}{N} \in \mathbb{R}^{K\times D}$$

$$\nabla_{\boldsymbol{b}}\mathcal{L} = \frac{1}{N}\sum_{n}(\boldsymbol{p}_n - \boldsymbol{t}_n) \in \mathbb{R}^{K}$$

$\boldsymbol{P}, \boldsymbol{T} \in \mathbb{R}^{N\times K}$: probas et labels du batch.

Mise à jour & PyTorch

$\mathbf{W} \leftarrow \mathbf{W} - \alpha\,\nabla_{\mathbf{W}}\mathcal{L} \quad \boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha\,\nabla_{\boldsymbol{b}}\mathcal{L}$

PyTorch : `nn.CrossEntropyLoss()`
Softmax intégré — passer les **logits** $\boldsymbol{f}$ directement.

Limite fondamentale

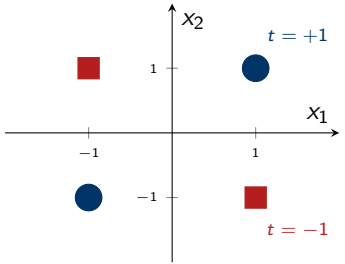
Un perceptron (couche linéaire unique) ne peut séparer que des données **linéairement séparables**.

Exemple : XOR

x_1	x_2	t
-1	-1	$+1$
$+1$	$+1$	$+1$
-1	$+1$	-1
$+1$	-1	-1

Aucun hyperplan ne peut séparer ces 4 points.
$\Rightarrow$ Le perceptron **échoue** sur XOR.
$\Rightarrow$ Il faut des couches **non-linéaires**.

XOR — non linéairement séparable



Solution : empiler des couches avec des activations **non-linéaires**.

Architecture MLP (L couches)

$$\boldsymbol{h}^{(1)} = \text{ReLU}(\mathbf{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)})$$

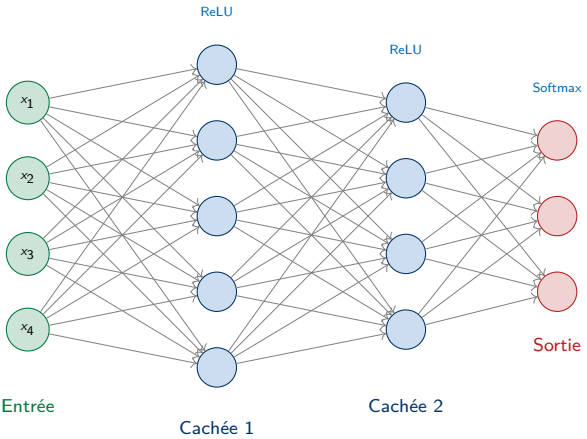
$$\boldsymbol{h}^{(\ell)} = \text{ReLU}(\mathbf{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)})$$

$$\hat{\boldsymbol{y}} = \text{softmax}(\mathbf{W}^{(L)}\boldsymbol{h}^{(L-1)} + \boldsymbol{b}^{(L)})$$

Le réseau apprend des frontières de décision **courbes et complexes**.

J3 — Autograd & PyTorch

Comment différencier automatiquement toutes ces couches ?
⇒ **Rétropropagation** & Autograd.



Récapitulatif — Ce qu'on a vu

Classification Binaire

- Labels $t_n \in \{+1, -1\}$, hyperplan $\mathbf{w}^\top \boldsymbol{x} + b = 0$
- Readout $\text{sign}(f)$: **hors** de la loss
- $\nabla \text{sign} = 0 \Rightarrow$ GD bloqué

Régression Logistique

- $\sigma(z) = 1/(1 + e^{-z})$: lisse, $\in (0, 1)$
- $f_\Theta(\boldsymbol{x}) = \sigma(\mathbf{w}^\top \boldsymbol{x}) \approx P(t = 1|\boldsymbol{x})$
- Loss : Binary Cross-Entropy, encodage $t \in \{0, 1\}$

ReLU & Perceptron de Rosenblatt

- $\text{ReLU}(z) = \max(0, z)$: dérivée 0/1
- $\mathcal{L} = \frac{1}{N} \sum \text{ReLU}(-f \cdot t)$
- Update : $\mathbf{w} \leftarrow \mathbf{w} + \alpha t_n \boldsymbol{x}_n$ si mal classifié
- Converge $\Leftrightarrow$ données lin. séparables

Multi-Classes & Vers le MLP

- K sorties, encodage one-hot
- Softmax $\rightarrow$ distribution de probabilité
- CCE : $\nabla_{f_k} = p_k - t_k$
- Couches cachées + activations $\rightarrow$ **MLP** (J3)