

Classification Binaire & Perceptron

Jour 2 — Matin

Julien Rolland

Formation M2 Développement Fullstack

Jour 2

- 1 Classification Binaire
- 2 L'Impasse du Gradient
- 3 ReLU & Loss du Perceptron
- 4 Algorithme de Rosenblatt
- 5 Régression Logistique
- 6 Architecture d'un Neurone
- 7 Classification Multi-Classes
- 8 Vers le Deep Learning

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

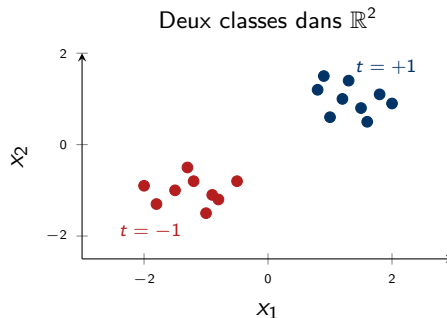
Classification (J2) : classe **discrète** $\hat{t}_n \in \{+1, -1\}$

Exemples :

- Email \rightarrow spam / ham
- Image \rightarrow chat / chien
- Tumeur \rightarrow maligne / bénigne

Enjeu

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

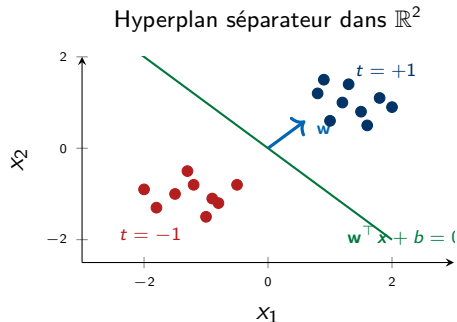


Modèle & Prédiction

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

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

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



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.

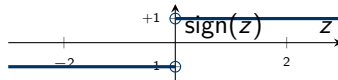
Conséquences

- 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)



Définition

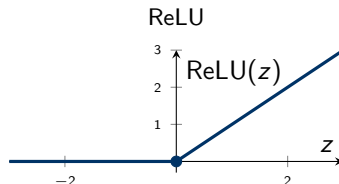
$$\text{ReLU}(z) = \max(0, z)$$

Dérivée

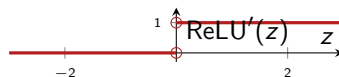
$$\text{ReLU}'(z) = \mathbf{1}[z > 0] = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

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).



Dérivée — toujours calculable



Loss de Rosenblatt

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

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

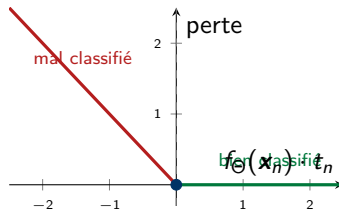
Gradient pour un exemple **mal classifié** ($f \cdot t < 0$) :

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

Règle de mise à jour

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

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

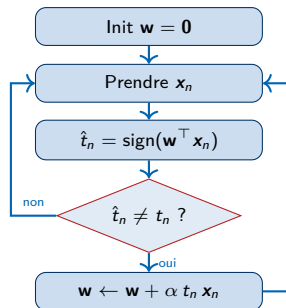


Encodage ± 1 crucial

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

```
1 def rosenblatt(X, t, lr=0.1, n_epochs=50):
2     N, D = X.shape
3     ones = np.ones((N, 1))
4     X_aug = np.hstack([ones, X]) # (N, D+1)
5     w = np.zeros(D + 1)
6
7     for epoch in range(n_epochs):
8         for n in range(N):
9             score = X_aug[n] @ w
10            t_hat = np.sign(score)
11            if t_hat != t[n]:
12                w += lr * t[n] * X_aug[n]
13
14     return w
```



« Online » = un exemple à la fois

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

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

Avantage clé : vectorisation

Un batch (X_b, t_b) de taille B permet de calculer le gradient en un seul produit matriciel.

Exploitable par **NumPy** et les **GPU**.

Mini-Batch GD — Pratique moderne

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

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)

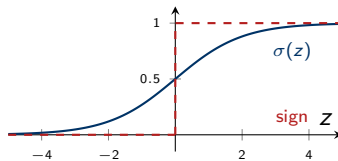
$$\sigma(z) = \frac{1}{1+e^{-z}}, \quad f_{\Theta}(\mathbf{x}_n) = \sigma(\mathbf{w}^{\top} \mathbf{x}_n) \approx P(t_n=1 \mid \mathbf{x}_n)$$

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

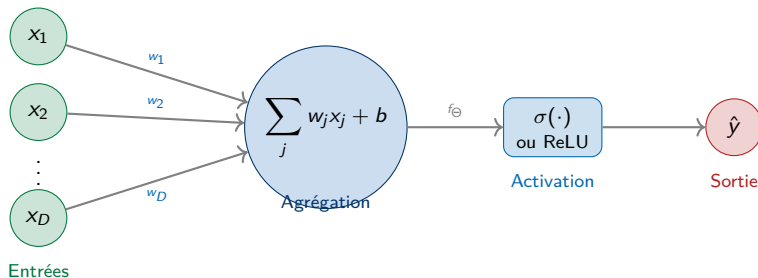
Avantage

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

Sigmoïde vs sign



Un Neurone Artificiel



Perceptron = réseau à 0 couche cachée

Entrées \rightarrow agrégation \rightarrow 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) : $\hat{t}_n \in \{+1, -1\}$

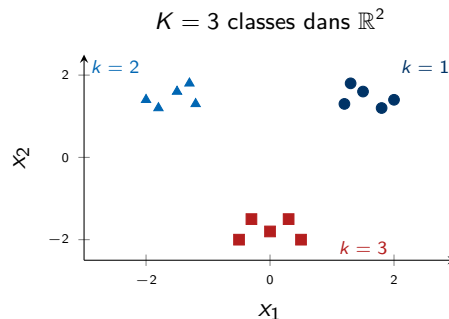
Multi-classes : K catégories discrètes,
 $\hat{k} \in \{1, \dots, K\}$

Exemples :

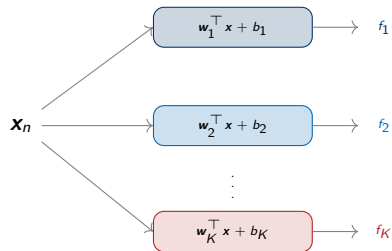
- Chiffres MNIST ($K = 10$)
- Objets CIFAR-10 ($K = 10$)
- Fleurs Iris ($K = 3$)

Enjeu

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



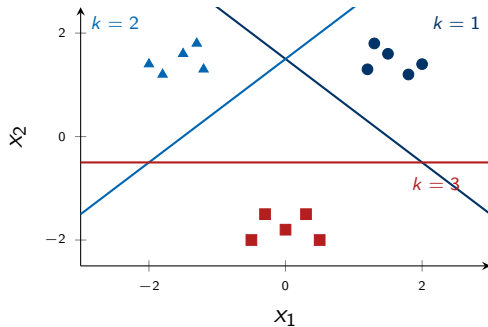
Un Perceptron par Classe



Score de la classe k :

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

Un hyperplan séparateur par classe



Softmax → probabilités

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

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

$\mathbf{t}_n \in \{0, 1\}^K$ (**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 : $\hat{k} = \arg \max_k p_k(\mathbf{x}_n)$

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 :

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

Empilement \Rightarrow 1 équation matricielle

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

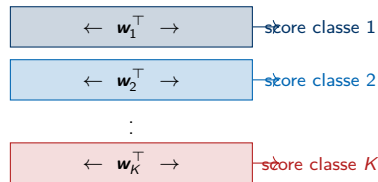
Ligne k de $\mathbf{W} = \mathbf{w}_k^\top$ (perceptron de classe k).

Forme batch (tout le dataset)

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

1 seul produit matriciel pour les N exemples.

Matrice $\mathbf{W} \in \mathbb{R}^{K \times D}$



1 ligne = 1 perceptron

$\mathbf{P} \in \mathbb{R}^{N \times K}$: ligne $n = \mathbf{p}_n$ (probas softmax de \mathbf{x}_n)

$\mathbf{T} \in \mathbb{R}^{N \times K}$: ligne $n = \mathbf{t}_n$ (label one-hot de \mathbf{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 \mathbf{W} et \mathbf{b}

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

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

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

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

Mise à jour & PyTorch

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

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

Limite fondamentale

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

Exemple : XOR

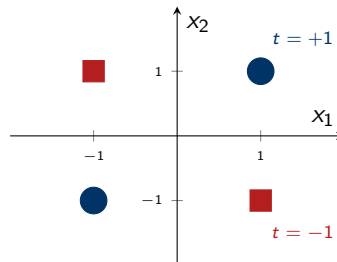
x_1	x_2	t
-1	-1	+1
+1	+1	+1
-1	+1	-1
+1	-1	-1

Aucun hyperplan ne peut séparer ces 4 points.

⇒ Le perceptron **échoue** sur XOR.

⇒ Il faut des couches **non-linéaires**.

XOR — non linéairement séparable



Vers le Multi-Layer Perceptron (MLP)

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

Architecture MLP (L couches)

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

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

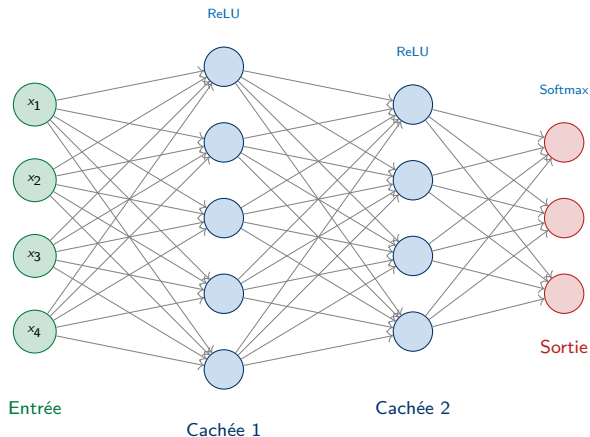
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}^{(L)}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)})$$

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

J3 — Autograd & PyTorch

Comment différencier automatiquement toutes ces couches ?

⇒ **Rétropropagation** & Autograd.



Classification Binaire

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

Régression Logistique

- $\sigma(z) = 1/(1 + e^{-z})$: lisse, $\in (0, 1)$
- $f_\Theta(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) \approx P(t = 1|\mathbf{x})$
- 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 : $\mathbf{w} \leftarrow \mathbf{w} + \alpha t_n \mathbf{x}_n$ 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)