Fondations Mathématiques & Outils Python

Jour 1 — Matin

Julien Rolland

Formation M2 Développement Fullstack

Jour 1

Plan du module

Scalaire $x \in \mathbb{R}$

Un nombre unique.

Ex : taux d'apprentissage $\alpha = 0.01$

Vecteur $\boldsymbol{x} \in \mathbb{R}^n$

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Ex : features d'un exemple.

Matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$

$$\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}$$

Ex : dataset (m exemples, n features).

Convention de notation

Scalaires : x (italique minuscule) Vecteurs : $\boldsymbol{x}$ (gras minuscule) Matrices : $\mathbf{X}$ (gras majuscule)

Produit scalaire $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$

$$\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{n} u_i v_i = \boldsymbol{u}^\top \boldsymbol{v}$$

Mesure de similarité / projection.

Norme euclidienne

$$\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}} = \sqrt{\sum_i x_i^2}$$

Produit matriciel $\mathbf{X} \in \mathbb{R}^{m \times k}$, $\mathbf{Y} \in \mathbb{R}^{k \times n}$

$$(\mathbf{XY})_{ij} = \sum_{l=1}^{k} X_{il}\, Y_{lj}$$

Attention

$\mathbf{XY} \neq \mathbf{YX}$ en général !
Les dimensions intérieures doivent correspondre.

Transposée $\mathbf{X}^\top$

$$(\mathbf{X}^\top)_{ij} = X_{ji}$$

Propriété utile :

$$(\mathbf{XY})^\top = \mathbf{Y}^\top \mathbf{X}^\top$$

Inverse $\mathbf{X}^{-1}$ (matrice carrée)

$$\mathbf{X}\mathbf{X}^{-1} = \mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$$

Application directe

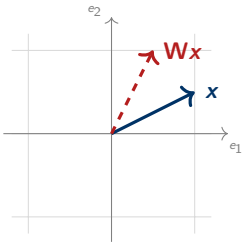
Solution du système $\mathbf{Xw} = \mathbf{y}$:

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

Utilisé dans la régression linéaire.

Multiplier par une matrice **transforme l'espace** :

- **Rotation** — change l'orientation
- **Mise à l'échelle** — étire ou compresse
- **Projection** — réduit la dimension



Pourquoi ça nous importe ?

Un réseau de neurones = **composition** de transformations linéaires entrecoupées de non-linéarités.

$\boldsymbol{h} = \sigma(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$

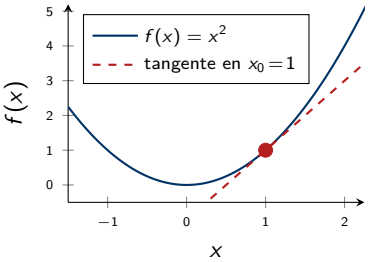
La dérivée mesure comment f **varie localement** en x_0 :

$$\frac{df}{dx}(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}$$

Géométriquement : pente de la **tangente** au point x_0.

Dérivées usuelles :

$f(x)$	$\frac{df}{dx}$
x^n	$n\,x^{n-1}$
e^x	e^x
$\ln x$	$1/x$



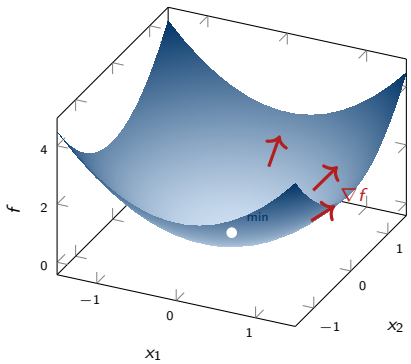
Règle de composition :

$$\frac{d(g \circ f)}{dx} = \frac{dg}{du} \cdot \frac{df}{dx} \quad (u = f(x))$$

Pour $f : \mathbb{R}^n \to \mathbb{R}$, le **gradient** est le vecteur des dérivées partielles :

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \dfrac{\partial f}{\partial x_1} \\ \vdots \\ \dfrac{\partial f}{\partial x_n} \end{pmatrix} \in \mathbb{R}^n$$

Exemple : $f(x_1, x_2) = x_1^2 + x_2^2 \;\Rightarrow\; \nabla f = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix}$



Propriété clé

$\nabla f(\boldsymbol{x})$ pointe dans la direction de **plus forte montée** de f.

$-\nabla f(\boldsymbol{x})$ pointe donc vers la **plus forte descente**.

Variable aléatoire X
Prend des valeurs selon une **distribution** $p(x)$.

Espérance — valeur moyenne

$$\mathbb{E}[X] = \sum_x x\, p(x)$$

Variance — dispersion autour de la moyenne

$$\mathrm{Var}(X) = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right]$$

Exemple concret

X = erreur de prédiction d'un modèle.

$\mathbb{E}[X] \approx 0 \quad \Rightarrow$ modèle non-biaisé.

$\mathrm{Var}(X)$ faible $\quad \Rightarrow$ prédictions stables.

Écart-type $\sigma = \sqrt{\mathrm{Var}(X)}$
Dans la même unité que X — plus interprétable.

Bernoulli $X \sim \mathcal{B}(p)$

$$P(X = 1) = p, \quad P(X = 0) = 1 - p$$

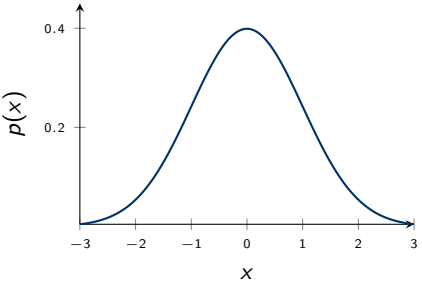
Utilisée pour : classification binaire (sortie d'un neurone sigmoïde).

Gaussienne $X \sim \mathcal{N}(\mu, \sigma^2)$

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Utilisée pour : initialisation des poids, bruit, régression.

Gaussienne $\mathcal{N}(0, 1)$



Idée fondatrice du ML supervisé

On cherche les paramètres θ qui rendent les données **les plus probables** :

$$\theta^* = \arg\max_{\theta} \; p(\mathcal{D} \mid \theta) \quad \Longleftrightarrow \quad \theta^* = \arg\min_{\theta} \; \mathcal{L}(\theta)$$

Log-vraisemblance (plus pratique)

$$\ell(\theta) = \log p(\mathcal{D} \mid \theta) = \sum_i \log p(y_i \mid \boldsymbol{x}_i, \theta)$$

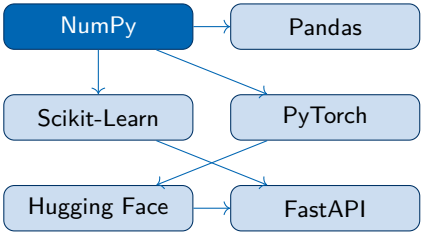
Maximiser ℓ = minimiser une **loss**.

Lien avec la MSE

Si le bruit est gaussien, maximiser la vraisemblance revient à minimiser :

$$\mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

L'Écosystème Python IA



Pourquoi Python ?

- Syntaxe accessible — focus sur les idées
- Bibliothèques C/CUDA sous le capot $\Rightarrow$ performance native
- Standard de facto en recherche et industrie

Ce cours

NumPy $\rightarrow$ Pandas $\rightarrow$ Sklearn $\rightarrow$ PyTorch $\rightarrow$ Hugging Face $\rightarrow$ FastAPI

Jupyter Notebook — L'environnement de travail

Un notebook = **document interactif** mêlant code, résultats et texte.

Deux types de cellules :

- **Code** — exécuté par le *kernel* Python, affiche le résultat juste en dessous
- **Markdown** — texte formaté, formules LaTeX, titres, explications

In

```
import numpy as np
```

In

```
x = np.arange(5)
print(x ** 2)
```

```
[ 0 1 4 9 16 ]
```

Md

```
# cellule Markdown
*Résultat :** vecteur au carré
```

Workflow

Écrire → `Shift+Enter` → voir le résultat
Les variables persistent entre les cellules.

Attention — ordre d'exécution

Les cellules peuvent être exécutées dans n'importe quel ordre. Toujours relancer **Kernel → Restart & Run All** avant de rendre un travail.

Le Problème des Boucles `for`

Approche naïve Python

```python
def dot_loop(u, v):
    result = 0.0
    for u_i, v_i in zip(u, v):
        result += u_i * v_i
    return result
```

Approche vectorisée NumPy

```python
import numpy as np

# Produit scalaire -- vectorise
def dot_numpy(u, v):
    return np.dot(u, v)
```

Problème

Python interprète chaque itération.
Pour $n = 10^6$: ~**200 ms**

Résultat

Exécuté en C/BLAS.
Pour $n = 10^6$: ~**1 ms**

$\approx$ **200$\times$ plus rapide**

Règle d'or

En NumPy/PyTorch : **aucune boucle `for` sur les éléments**. Toujours chercher l'opération matricielle équivalente.

Anatomie d'un `ndarray`

```python
import numpy as np

x = np.array([[1, 2, 3],
              [4, 5, 6]])

print(x.shape)    # (2, 3)
print(x.ndim)     # 2  <- rank
print(x.dtype)    # int64
print(x.size)     # 6
```

Attribut	Signification
`shape`	dimensions $(m, n, \ldots)$
`ndim`	nombre d'axes (rank)
`dtype`	type des éléments
`size`	nombre total d'éléments

Rangs courants

0D scalaire **1D** vecteur
2D matrice **3D** batch d'images

Slicing

```
X = np.arange(12).reshape(3, 4)
# [[ 0  1  2  3]
#  [ 4  5  6  7]
#  [ 8  9 10 11]]

X[0]          # 1e ligne -> [0,1,2,3]
X[:, 2]       # 3e colonne
X[1:, 1:3]    # sous-matrice
```

Reshape / Vue

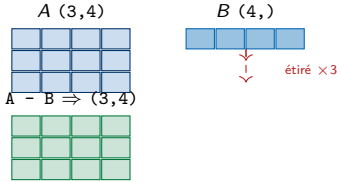
```
v = np.arange(6)          # shape (6,)
M = v.reshape(2, 3)       # shape (2,3)
F = M.flatten()           # retour (6,)

# -1 = "calcule toi-meme"
v.reshape(3, -1)          # (3, 2)
```

Attention

`reshape` retourne une **vue** (pas de copie).
Modifier la vue modifie l'original.

Broadcasting — La Règle des Dimensions

NumPy aligne les shapes **depuis la droite**.
Toute dimension de taille **1** est automatiquement étirée.




```python
X = np.random.randn(100, 5)
# shape (100, 5)

mean = X.mean(axis=0)
# shape (5,) -- une moyenne par feature

X_centered = X - mean
# (100,5) - (5,) => (100,5)
# soustrait la meme moyenne
# sur chaque ligne automatiquement

std = X.std(axis=0)
X_norm = (X - mean) / std
```

NumPy `ndarray`

- Un seul type (`dtype`)
- Accès par indices entiers
- Ultra-rapide pour le calcul
- Pas de métadonnées

```
X[0, 2]      # ligne 0, col 2
```

Pandas `DataFrame`

- Colonnes hétérogènes (int, float, str. . .)
- Accès par **noms de colonnes**
- Index temporel possible
- Idéal pour l'exploration / nettoyage

```
df['age']           # colonne 'age'
df.loc[0, 'age']    # ligne 0, col age
```

Flux de travail typique

Charger avec **Pandas** → Nettoyer → Convertir en **NumPy** / Tensor pour le modèle.

Chargement & Exploration

```python
import pandas as pd

df = pd.read_csv('data.csv')

df.shape          # (nb_lignes, nb_colonnes)
df.dtypes         # types de chaque colonne
df.head(5)        # 5 premieres lignes
df.describe()     # stats : mean, std, min, max...
df.info()         # types + valeurs manquantes
```

Sélection

```python
df[['col1', 'col2']]        # sous-df
df[df['age'] > 18]          # filtre
df.groupby('ville').mean() # agregation
```

Passage à NumPy

```python
X = df[features].values     # ndarray
y = df['target'].values     # ndarray 1D
```

Gérer les valeurs manquantes (NaN)

```
df.isna().sum()  # nb NaN par colonne
df.dropna()  # supprimer les lignes avec NaN
df.fillna(df.mean())  # remplacer par la moyenne
df['col'].fillna('N/A')  # valeur par defaut
```

One-Hot Encoding

```
pd.get_dummies(df, columns=['ville'])
# 'ville' -> ville_Paris, ville_Lyon...
```

Pourquoi c'est crucial

La majorité du temps en ML est passée ici.

Un modèle entraîné sur des données mal nettoyées sera systématiquement biaisé — **garbage in, garbage out**.

Règle pratique

Toujours inspecter `df.info()` et `df.describe()` **avant** de toucher aux modèles.

Récapitulatif — Ce qu'on a vu

Mathématiques

- Vecteurs, matrices, produit, transposée
- Dérivée, gradient ∇f — direction de montée
- Espérance, variance, distributions $\mathcal{B}$, $\mathcal{N}$
- Vraisemblance $\leftrightarrow$ fonction de coût

Python

- Boucles `for` $\rightarrow$ opérations vectorisées
- `ndarray` : shape, rank, dtype, slicing, broadcasting
- Pandas : chargement, exploration, nettoyage