TP : Neural Network

Simon JAXEL – Grégoire LEGRIS – Arthur MAGRON

December 4, 2025

1 Problèmes à la main

1.1 Régression logistique

1.1.1 Modélisation de Y

La variable Y étant binaire ($Y \in \{0, 1\}$), la loi la plus adaptée pour la modéliser est la **loi de Bernoulli**. Si l'on note $\alpha = \mathbb{P}(Y = 1)$ la probabilité a priori de la classe 1, alors :

$$Y \sim \mathcal{B}(\alpha)$$

1.1.2 Densité de probabilité de X

On connait les densités conditionnelles, car elles suivent des lois normales. On les utilise dans la loi des probas totales :

$$p(x) = p(x|Y = 0)\mathbb{P}(Y = 0) + p(x|Y = 1)\mathbb{P}(Y = 1)$$

En notant $\phi(x; \mu, \sigma)$ la densité gaussienne, on obtient un mélange de Gaussiennes:

$$p(x) = (1 - \alpha) \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu_0)^2}{2\sigma^2}} + \alpha \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}$$

1.1.3 Difficulté de calcul

La difficulté pour calculer $p(x)$ est que l'on ne connaît pas α (soit $\mathbb{P}(Y = 1)$).

1.1.4 Expression du Log-Odds Ratio

On cherche à exprimer le log-odds ratio sous la forme $\beta_0 + \beta_1 x$. En utilisant le théorème de Bayes et en simplifiant par $p(x)$:

$$\ln\left(\frac{\mathbb{P}(Y = 1|X = x)}{\mathbb{P}(Y = 0|X = x)}\right) = \ln\left(\frac{p(x|Y = 1)\mathbb{P}(Y = 1)}{p(x|Y = 0)\mathbb{P}(Y = 0)}\right)$$

On développe ensuite les logarithmes :

$$\begin{aligned}
\beta_0 + \beta_1 x &= \ln\left(\frac{\mathbb{P}(Y = 1)}{\mathbb{P}(Y = 0)}\right) + \ln\left(\frac{p(x|Y = 1)}{p(x|Y = 0)}\right) \\
&= \ln\left(\frac{\alpha}{1 - \alpha}\right) + \ln\left(\frac{\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma^2}\right)}{\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu_0)^2}{2\sigma^2}\right)}\right) \\
&= \ln\left(\frac{\alpha}{1 - \alpha}\right) - \frac{(x - \mu_1)^2}{2\sigma^2} + \frac{(x - \mu_0)^2}{2\sigma^2}
\end{aligned}$$

En développant les carrés, les termes en x^2 s'annulent:

$$\begin{aligned}
\beta_0 + \beta_1 x &= \ln\left(\frac{\alpha}{1 - \alpha}\right) - \frac{1}{2\sigma^2}\left(x^2 - 2x\mu_1 + \mu_1^2 - (x^2 - 2x\mu_0 + \mu_0^2)\right) \\
&= \ln\left(\frac{\alpha}{1 - \alpha}\right) + \frac{x(\mu_1 - \mu_0)}{\sigma^2} - \frac{\mu_1^2 - \mu_0^2}{2\sigma^2}
\end{aligned}$$

En identifiant avec l'expression $\beta_0 + \beta_1 x$, nous obtenons :

$$\boxed{\begin{cases} \beta_1 = \frac{\mu_1 - \mu_0}{\sigma^2} \\ \beta_0 = \ln\left(\frac{\alpha}{1-\alpha}\right) - \frac{\mu_1^2 - \mu_0^2}{2\sigma^2} \end{cases}}$$

1.1.5 Prise de décision

Maintenant qu'on connait β_0 et β_1, on peut calculer $\pi(x) = \mathbb{P}(Y = 1|X = x)$ avec la fonction sigmoïde :

$$\pi(x) = \text{Sigmoïde}(\pi(x)) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

On choisit donc d'utiliser la règle suivante : si $\pi(x) < 0.5$, alors on prédit la classe 1, si $\pi(x) > 0.5$, alors on prédit la classe 0. Cette règle est pratique puisqu'elle revient à étudier le signe du LogOdds :

- Si $\beta_0 + \beta_1 x > 0$, on prédit la classe **1**.
- Sinon, on prédit la classe **0**.

1.1.6 Représentation graphique

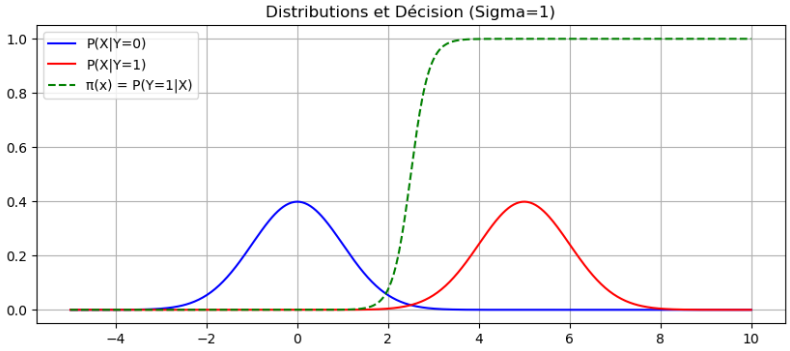

Figure 1: Représentation pour $\sigma = 1$

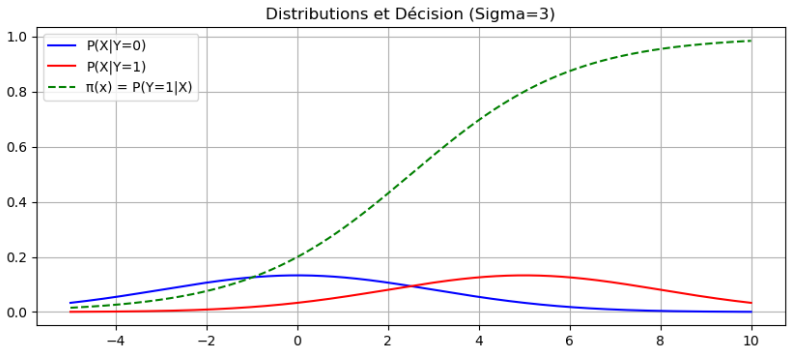

Figure 2: Représentation pour $\sigma = 3$

1.1.7 Commentaire sur la décision

L'analyse graphique ci-dessus permet de comparer deux situations :

- **Cas $\sigma = 1$ (Faible variance) :** Les distributions conditionnelles sont bien séparées. On ibserve une rupture nette sur la fontion $\pi(x)$, ce qui donne une confiance élevée.
- **Cas $\sigma = 3$ (Forte variance) :** Il y a un fort recouvrement entre les classes. La pente de $\pi(x)$ est plus douce, indiquant une zone d'incertitude plus large.

Même si les probabilités a priori sont égales, et même si la frontière de décision (le point où $\pi(x) = 0.5$) reste au même endroit ($x = 2.5$), la confiance du modèle loin de la frontière diminue lorsque la variance augmente.

1.1.8 Estimation sur un jeu de données

On a un jeu de données $\mathcal{D} = \{(x_i, y_i)\}_{1 \leq i \leq n}$.

Hypothèse i.i.d (indépendante et identiquement distribuée) :

- **Indépendantes :** Le résultat d'une observation i n'influence pas l'observation $i + k$, quel que soit k.
- **Identiquement Distribuées :** Tous les couples (x_i, y_i) proviennent de la même loi statistique utilisée avec les mêmes paramètres.

Estimation par Maximum de Vraisemblance :

La méthode consiste à trouver les paramètres β qui maximisent la probabilité d'avoir observé notre jeu de données. La formule de la vraisemblance est :

$$L(\beta) = \prod_{i=1}^{n} P(Y = y_i | X = x_i)$$

$$L(\beta) = \prod_{i=1}^{n} \pi(x_i)^{y_i} (1 - \pi(x_i))^{(1-y_i)}$$

On passe au logarithme pour pouvoir transformer le résultat en somme et mieux dériver (log-vraisemblance) :

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} [y_i \ln(\pi(x_i)) + (1 - y_i) \ln(1 - \pi(x_i))]$$

Ensuite, on utilise des méthodes itératives comme la descente de Gradients.

1.2 Perceptron

On a les données suivantes

$$f([0,0]) = 0 \quad f([0,1]) = 1$$
$$f([1,0]) = 1 \quad f([1,1]) = 0$$

1.2.1 Représentation graphique des données

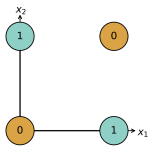

Figure 3: Représentation graphique des données

1.2.2 Fonction logique

La fonction logique à modéliser est XOR. Étant donné que ses données ne sont pas linéairement séparables, un perceptron simple ne peut pas la représenter

1.2.3 Réseau de neurones

La fonction XOR peut s'exprimer avec AND et OR :

$$\text{XOR}(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

On peut donc identifier une couche cachée avec deux neurones. Ces deux neurones réaliseront la fonction logique AND. Le neurone de sortie représentera la fonction logique OR. On notera que que les neurones de la couche cachée sont des percpetrons simples.

Ci-dessous les poids et les biais du réseau de neurones avce la fonction d'activation step :

Couche cachée

$$H_1 : \begin{cases} w_1 = 1, \\ w_2 = -1, \\ b = -0.5 \end{cases} \qquad \text{(implémente } x_1 \wedge \neg x_2\text{)}$$

$$H_2 : \begin{cases} w_1 = -1, \\ w_2 = 1, \\ b = -0.5 \end{cases} \qquad \text{(implémente } \neg x_1 \wedge x_2\text{)}$$

Couche de sortie

$$Y : \begin{cases} w_{H_1} = 1, \\ w_{H_2} = 1, \\ b = -0.5 \end{cases} \qquad \text{(implémente OR entre } H_1 \text{ et } H_2\text{)}$$

On utilise un biais de -0.5 afin d'ajuster le seuil du neurone. Sans biais, pour $x_1 = 1$ et $x_2 = 1$, on obtient :

$$H_1 = \text{step}(1 \cdot 1 + (-1) \cdot 1) = \text{step}(0) = 1.$$

Ce résultat est incorrect, car nous souhaitons que $H_1 = 0$ lorsque $(x_1, x_2) = (1, 1)$.

En ajoutant un biais négatif de -0.5, le neurone calcule :

$$H_1 = \text{step}(1 \cdot 1 + (-1) \cdot 1 - 0.5) = \text{step}(-0.5) = 0.$$

Le biais permet donc de déplacer le seuil du perceptron pour obtenir le comportement logique attendu.

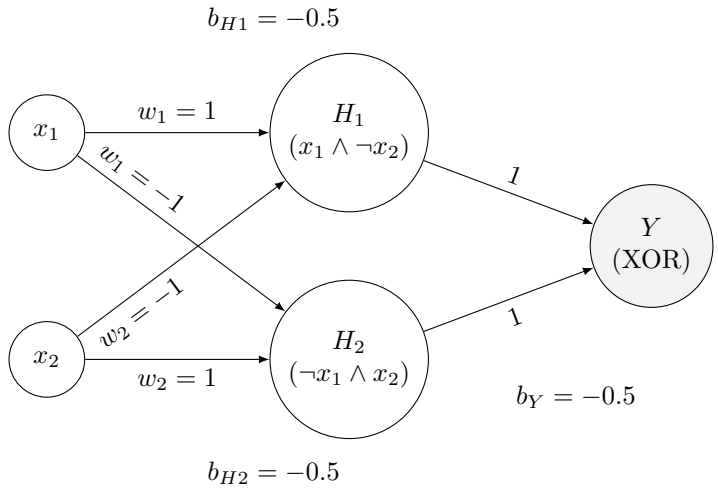

Figure 4: Perceptron à deux couches implémentant la fonction XOR

1.3 Régression ReLU

1.3.1 Fonction f et g

a. Représentation graphique de f

$$f(x) = \begin{cases} 2x - 1 & \text{si } x \leq 1, \\ 1 & \text{si } x > 1, \end{cases}$$

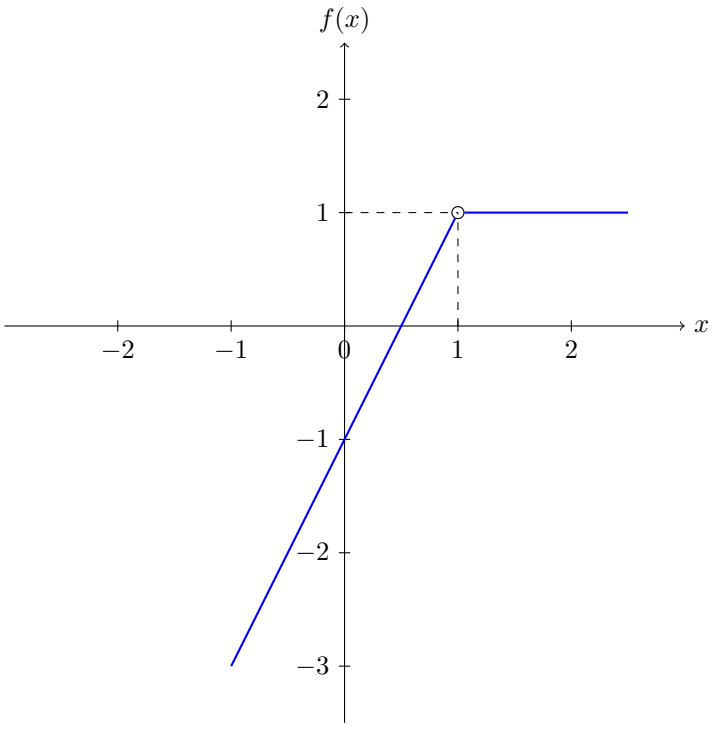

Figure 5: Graphe de la fonction $f(x)$ définie par morceaux

b. Composition de fonctions pour f

Nous souhaitons montrer que l'on peut écrire f sous la forme

$$f(x) = a_2(\text{ReLU}(a_1(x))),$$

où a_1 et a_2 sont des applications affines de la forme:

$$a_1(x) = \alpha_1 \cdot x + \beta_1, \qquad a_2(x) = \alpha_2 \cdot x + \beta_2.$$

On cherche les valeurs de x, tels que $a_1(x) < 0$:

$$\alpha_1 \cdot x + \beta_1 < 0 \quad \Longleftrightarrow \quad \begin{cases} x > -\dfrac{\beta_1}{\alpha_1}, & \text{si } \alpha_1 < 0, \\ x < -\dfrac{\beta_1}{\alpha_1}, & \text{si } \alpha_1 > 0. \end{cases}$$

Pour la suite, on gardera $\alpha_1 < 0$.

Ainsi,

$$\text{ReLU}(a_1(x)) = \begin{cases} \alpha_1 x + \beta_1 & \text{si } x \leq \frac{-\beta_1}{\alpha_1} \\ 0 & \text{sinon.} \end{cases}$$

En posant $\alpha_1 = -1$ et $\beta_1 = 1$, on obtient le seuil de la fonction RELU en $x = 1$.

Donc,

$$\text{ReLU}(a_1(x)) = \begin{cases} \alpha_1 x + \beta_1 & \text{si } x \leq 1 \\ 0 & \text{sinon.} \end{cases}$$

En composant avec $a_2(x) = \alpha_2 x + \beta_2$, on obtient

$$a_2(\text{ReLU}(a_1(x))) = \begin{cases} -\alpha_2 x + \alpha_2 + \beta_2 & \text{si } x \leq 1, \\ \beta_2 & \text{sinon.} \end{cases}$$

Nous souhaitons que cette expression coïncide avec $f(x)$, c'est-à-dire

$$\begin{cases} -\alpha_2 = 2, \\ \alpha_2 + \beta_2 = -1, \\ \beta_2 = 1. \end{cases}$$

Les dernières équations imposent $\alpha_2 = -2$ et $\beta_2 = 1$.

Ainsi,

$$a_1(x) = 1 - x, \qquad a_2(x) = -2x + 1.$$

On vérifie alors que

$$\text{ReLU}(a_1(x)) = \begin{cases} 1 - x & \text{si } x \leq 1, \\ 0 & \text{si } x > 1, \end{cases}$$

et donc

$$a_2(\text{ReLU}(a_1(x))) = \begin{cases} 2x - 1 & \text{si } x \leq 1, \\ 1 & \text{si } x \geq 1. \end{cases}$$

On obtient ainsi exactement la fonction désirée.

c. Représentation graphique de la fonction g

$$g(x) = \begin{cases} 0 & \text{si } x < 0, \\ x & \text{si } 0 \leq x \leq 1, \\ 1 & \text{si } x > 1. \end{cases}$$

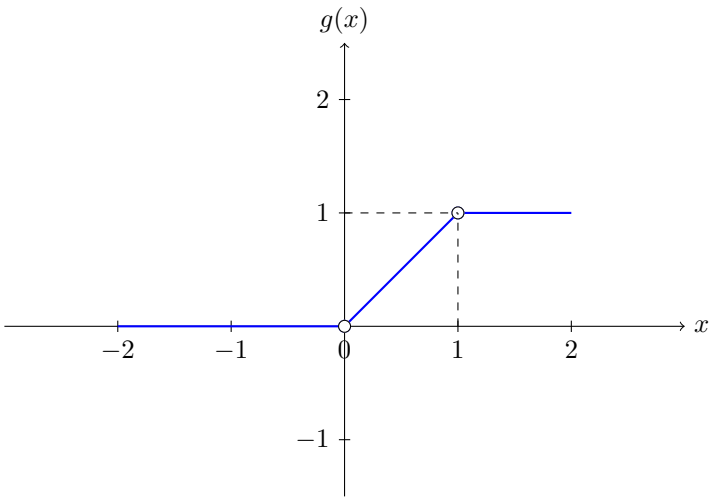

Figure 6: Graphe de la fonction $g(x)$ définie par morceaux

d. Composition de fonctions pour g

Notre intuition nous amène à penser qu'il faut poser $a_1(x) = x$ afin de créer le seuil de la fonction ReLU en $x = 0$.

Donc,

$$\text{ReLU}(a_1(x)) = \text{ReLu(x)} \begin{cases} 0 & \text{si } x \leq 0, \\ x & \text{si } x > 0, \end{cases}$$

On souhaite créer un deuxième seuil en $x = 1$.

En remarquant que,

$$\text{ReLU}(x+1) = \begin{cases} 0 & \text{si } x \leq 1, \\ x+1 & \text{si } x > 1, \end{cases}$$

et en utilisant l'indication de l'énoncée, on écrit :

$$g(x) = \alpha_{21} \cdot \text{ReLu}(\text{a}_1(\text{x})) + \alpha_{22} \cdot \text{ReLu}(\text{a}_1(\text{x}) - 1) + \beta_2$$

x	$z_1 = \text{ReLU}(x)$	$z_2 = \text{ReLU}(x-1)$	$g(x) = z_1 \cdot \alpha_{21} + z_2 \cdot \alpha_{22} + \beta_2$
$x < 0$	0	0	$0 \implies \beta = 0$
$0 \leq x \leq 1$	x	0	$x \implies \alpha_{21} \cdot x + \beta_2 = x \implies \alpha_{21} = 1$
$x > 1$	x	$x-1$	$1 \implies \alpha_{21} \cdot x + \alpha_{22} \cdot (x-1) + \beta_2 = 1 \implies \alpha_2 = -1$

Table 1: Détermination des coefficients $\alpha_{21}, \alpha_{22}, \beta_2$ pour $g(x)$

On en déduit que $a_2(x, y) = x - y$

$$g(x) = \mathbf{w} \cdot \begin{pmatrix} \text{ReLU}(x) \\ \text{ReLU}(x-1) \end{pmatrix} + b$$

où

$$\mathbf{w} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}^{\mathsf{T}}, \quad b = 0$$

1.3.2 Déterminer expression des fonctions

a. 1ère fonction

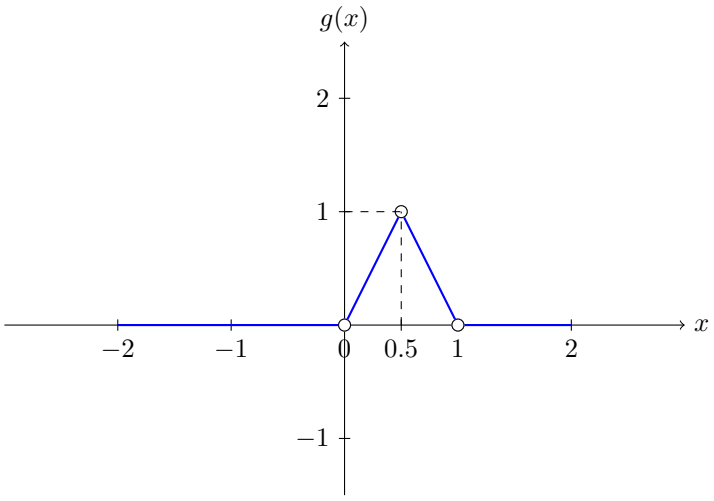

Figure 7: Graphe de la fonction $g(x)$ définie par morceaux

De la même manière, on pose $a_1(x) = x$ afin de créer le seuil de la fonction ReLu en $x = 0$

Cette fois-ci, on écrit :

$$g(x) = \alpha_{21} \cdot \mathrm{ReLu}(\mathrm{a_1(x)}) + \alpha_{22} \cdot \mathrm{ReLu}(\mathrm{a_1(x)} - \frac{1}{2}) + \beta_2$$

x	$z_1 = \mathrm{ReLU}(x)$	$z_2 = \mathrm{ReLU}(x - 1/2)$	$f(x) = z_1 \cdot \alpha_{21} + z_2 \cdot \alpha_{22} + \beta_2$
$x \leq 0$	0	0	$0 \implies \beta_2 = 0$
$0 < x \leq 1/2$	x	0	$2x \implies \alpha_{21} \cdot x + \beta_2 = 2x \implies \alpha_{21} = 2$
$1/2 < x \leq 1$	x	$x - 1/2$	$-2x + 2 \implies 2x + \alpha_{22} \cdot (x - 1/2) = -2x + 2 \implies \alpha_{22} = -4$
$x > 1$	x	$x - 1/2$	$0 \implies -2x + 2 \neq 0$

Table 2: Détermination des coefficients $\alpha_{21}, \alpha_{22}, \beta_2$ pour la fonction triangulaire $g(x)$

On remarque que pour $x > 1$, l'équation n'est pas vérifiée. Il suffit simplement de composer le tout par la fonction ReLu

$$g(x) = \mathrm{ReLU}(\mathbf{w} \cdot \begin{pmatrix} \mathrm{ReLU}(x) \\ \mathrm{ReLU}(x - \frac{1}{2}) \end{pmatrix} + b)$$

où

$$\mathbf{w} = \begin{pmatrix} 2 \\ -4 \end{pmatrix}^{\mathsf{T}}, \quad b = 0$$

b. 2ème fonction

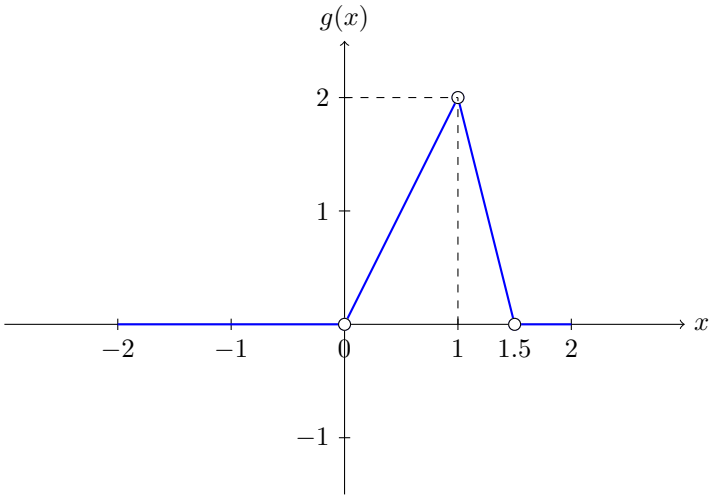

Figure 8: Graphe de la fonction $g(x)$ définie par morceaux

De la même manière, on obtient le tableau suivant :

x	$z_1 = \text{ReLU}(x)$	$z_2 = \text{ReLU}(x-1)$	$f(x) = z_1 \cdot \alpha_{21} + z_2 \cdot \alpha_{22} + \beta_2$
$x \leq 0$	0	0	$0 \implies \beta_2 = 0$
$0 < x \leq 1$	x	0	$2x \implies \alpha_{21} \cdot x + \beta_2 = 2x \implies \alpha_{21} = 2$
$1 < x \leq 1.5$	x	$x-1$	$-2x+2 \implies 2x + \alpha_{22} \cdot (x-1) = -4x + 6 \implies \alpha_{22} = -6$
$x > 1.5$	x	$x-1$	$0 \implies -4x + 6 \neq 0$

Table 3: Détermination des coefficients $\alpha_1, \alpha_2, \beta$ pour la fonction triangulaire (non symétrique)$g(x)$

On remarque que pour $x > 1.5$, l'équation n'est pas vérifiée. Il suffit une nouvelle fois de composer le tout par la fonction ReLu. On obtient :

$$g(x) = \text{ReLU}(\mathbf{w} \cdot \begin{pmatrix} \text{ReLU}(x) \\ \text{ReLU}(x-1) \end{pmatrix} + b)$$

où

$$\mathbf{w} = \begin{pmatrix} 2 \\ -6 \end{pmatrix}^\intercal, \quad b = 0$$

1.3.3 Approximation de la fonction h

a. Écrire g_α comme composition de fonctions ReLU et linéaires

De la même manière, on obtient :

x	$z_1 = \text{ReLU}(x)$	$z_2 = \text{ReLU}(x - 1/2)$	$f(x) = z_1 \cdot \alpha_{21} + z_2 \cdot \alpha_{22} + \beta_2$
$x \leq 0$	0	0	$\beta_2 = 0$
$0 < x \leq 1/2$	x	0	$\alpha_{21} \cdot x + \beta_2 = 2\alpha x \implies \alpha_{21} = 2\alpha$
$1/2 < x \leq 1$	x	$x - 1/2$	$2\alpha x + \alpha_{22} \cdot (x - 1/2) = -2\alpha x + 2\alpha \implies \alpha_{22} = -4\alpha$
$x > 1$	x	$x - 1/2$	$-2x + 2 \neq 0$

Table 4: Détermination des coefficients $\alpha_{21}, \alpha_{22}, \beta_2$ pour la fonction triangulaire $g_\alpha(x)$

On remarque que pour $x > 1$, l'équation n'est pas vérifiée. Il suffit simplement de composer le tout par la fonction ReLu

$$g_\alpha(x) = \text{ReLU}(\mathbf{w} \cdot \begin{pmatrix} \text{ReLU}(x) \\ \text{ReLU}(x - \frac{1}{2}) \end{pmatrix})$$

où

$$\mathbf{w} = \begin{pmatrix} 2\alpha \\ -4\alpha \end{pmatrix}^\mathsf{T}$$

b. Paramètre α optimal

On peut remarquer que si $\alpha \geq 0$:

$$g_\alpha(x) = \alpha \cdot \zeta(x)$$

avec

$$\zeta(x) = \text{ReLU}(\mathbf{v} \cdot \begin{pmatrix} \text{ReLU}(x) \\ \text{ReLU}(x - \frac{1}{2}) \end{pmatrix}), \quad \mathbf{v} = \begin{pmatrix} 2 \\ -4 \end{pmatrix}^\mathsf{T}$$

De plus, si $x \in [0, 1]$ on a :

$$\zeta(x) = \mathbf{v} \cdot \begin{pmatrix} \text{ReLU}(x) \\ \text{ReLU}(x - \frac{1}{2}) \end{pmatrix}$$

car $\forall x \in [0, 1]$, on a : $\mathbf{v} \cdot \begin{pmatrix} \text{ReLU}(x) \\ \text{ReLU}(x - \frac{1}{2}) \end{pmatrix} \geq 0$. Donc la fonction ReLU agit comme la fonction identité sur [0,1].

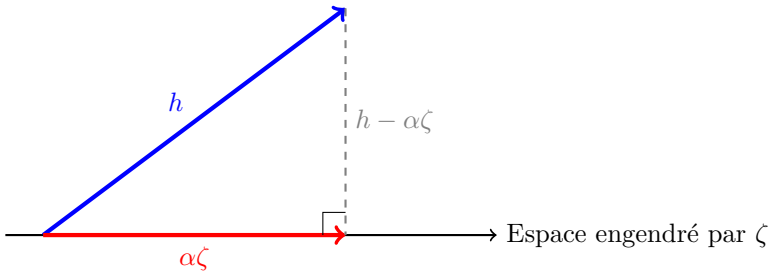

Figure 9: Interprétation géométrique de l'approximation au sens des moindres carrés

La distance la plus courte entre h et ζ est obtenue lorsqu'on projette h sur l'espace engendré par ζ. Cela revient à dire que le vecteur $h - \alpha \cdot \zeta$ est perpendiculaire à ζ

Autrement dit :

$$\langle h - \alpha \cdot \zeta, \zeta \rangle = 0 \Leftrightarrow \langle h, \zeta \rangle - \alpha \cdot \langle \zeta, \zeta \rangle = 0$$

Donc :

$$\alpha = \frac{\langle h, \zeta \rangle}{\langle \zeta, \zeta \rangle} = \frac{\int_0^1 h(x) \cdot \zeta(x)\, dx}{\int_0^1 \zeta(x) \cdot \zeta(x)\, dx}$$

On a :

$$\begin{aligned}
\int_0^1 \zeta(x) \cdot \zeta(x)\, dx &= \int_0^1 (2 \cdot \max(0, x) - 4 \cdot \max(0, x - \frac{1}{2}))^2\, dx \\
&= \int_0^{\frac{1}{2}} (2x)^2\, dx + \int_{\frac{1}{2}}^1 \left(2x - 4\left(x - \frac{1}{2}\right)\right)^2\, dx \\
&= 4 \cdot \left[\frac{x^3}{3}\right]_0^{\frac{1}{2}} - \frac{1}{6} \cdot \left[(-2x+2)^3\right]_{\frac{1}{2}}^1 \\
&= \frac{1}{3}
\end{aligned}$$

et

$$\begin{aligned}
\int_0^1 h(x) \cdot \zeta(x)\, dx &= \int_0^1 \sin(\pi x) \cdot \left(2 \cdot \max(0, x) - 4 \cdot \max\left(0, x - \frac{1}{2}\right)\right)\, dx \\
&= \int_0^{\frac{1}{2}} 2x \sin(\pi x)\, dx + \int_{\frac{1}{2}}^1 (-2x+2) \sin(\pi x)\, dx \\
&\quad \text{(En posant t = 1 - x pour la deuxième intégrale)} \\
&= \int_0^{\frac{1}{2}} 2x \sin(\pi x)\, dx + \int_0^{\frac{1}{2}} 2t \sin(\pi t)\, dt \\
&= 2 \times \int_0^{\frac{1}{2}} 2x \sin(\pi x)\, dx \\
&= 4 \int_0^{\frac{1}{2}} x \sin(\pi x)\, dx \\
&\quad \text{(Intégration par parties : } u = x, v' = \sin(\pi x)) \\
&= 4 \left(\left[-\frac{x}{\pi} \cos(\pi x)\right]_0^{\frac{1}{2}} - \int_0^{\frac{1}{2}} \left(-\frac{1}{\pi} \cos(\pi x)\right)\, dx \right) \\
&= 4 \left(\underbrace{\left(-\frac{1}{2\pi} \cos\left(\frac{\pi}{2}\right) - 0\right)}_{=0} + \frac{1}{\pi} \int_0^{\frac{1}{2}} \cos(\pi x)\, dx \right)
\end{aligned}$$

$$
\begin{aligned}
&= \frac{4}{\pi}\left[\frac{1}{\pi}\sin(\pi x)\right]_0^{\frac{1}{2}} \\
&= \frac{4}{\pi^2}\left(\sin\left(\frac{\pi}{2}\right) - \sin(0)\right) \\
&= \frac{4}{\pi^2}
\end{aligned}
$$

Finalement, on obtient $\alpha = \frac{12}{\pi^2}$

1.4 CNN

1.4.1 Opérateur linéaire

1. Matrice C_k

$$
C_k = \begin{bmatrix}
\alpha & \beta & \gamma & 0 & \cdots & 0 & 0 \\
0 & \alpha & \beta & \gamma & 0 & \cdots & 0 \\
\vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & \alpha & \beta & \gamma & 0 \\
0 & \cdots & 0 & 0 & \alpha & \beta & \gamma
\end{bmatrix}
$$

2. Noyau de convolution associé

Le noyau de convolution associé est :

$$
k = \left[-\tfrac{1}{h},\ \tfrac{1}{h}\right]
$$

$$
Dy = \begin{bmatrix}
\dfrac{f(h) - f(0)}{h} \\
\dfrac{f(2h) - f(h)}{h} \\
\vdots \\
\dfrac{f(nh) - f((n-1)h)}{h}
\end{bmatrix}
$$

Le vecteur Dx représente les taux d'accroissement de la fonction f sur chaque intervalle de longueur h.

Si $n \to +\infty$, alors $h \to 0$, ces taux d'accroissement convergent vers la dérivée $f'(x)$.

3. Matrice C_k tel que $y = C_k \cdot x$

$$
vec(x) = \begin{bmatrix}
x_{11} \\ x_{12} \\ x_{13} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{31} \\ x_{32} \\ x_{33}
\end{bmatrix}
$$

On a :

$$
vec(x) = \begin{bmatrix}
y_{11} \\ y_{12} \\ y_{21} \\ y_{22}
\end{bmatrix}
$$

Avec :

$$y_{11} = \alpha * x_{11} + \beta * x_{12} + \gamma * x_{21} + \delta * x_{22}$$
$$y_{12} = \alpha * x_{12} + \beta * x_{13} + \gamma * x_{22} + \delta * x_{23}$$
$$y_{21} = \alpha * x_{21} + \beta * x_{22} + \gamma * x_{31} + \delta * x_{32}$$
$$y_{22} = \alpha * x_{22} + \beta * x_{23} + \gamma * x_{32} + \delta * x_{33}$$

On a alors :

$$y = C_k x$$

Avec :

$$C_k \begin{bmatrix} \alpha & \beta & 0 & \gamma & \delta & 0 & 0 & 0 & 0 \\ 0 & \alpha & \beta & 0 & \gamma & \delta & 0 & 0 & 0 \\ 0 & 0 & 0 & \alpha & \beta & 0 & \gamma & \delta & 0 \\ 0 & 0 & 0 & 0 & \alpha & \beta & 0 & \gamma & \delta \end{bmatrix}$$

4. Avec un noyau de convolution 3*3 et n*n

On a un noyau de taille 3*3 et une matrice de taille n*n, la convolution est sans padding et avec un stride de 1, on obtiendra donc une matrice de taille (n-2)*(n-2)*n*n.

En notant le noyau de convolution :

$$k = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Pour $1 \leq i, j \leq n-2$, on a :

$$y_{i,j} = a * x_{i,j} + b * x_{i,j+1} + c * x_{i,j+2} + d * x_{i+1,j} + e * x_{i+1,j+1} + f * x_{i+1,j+2} + g * x_{i+2,j} + h * x_{i+2,j+1} + i * x_{i+2,j+2}$$

5. Sparcity index

On note s le *sparsity index* de la matrice de convolution C_k. La matrice est de taille $n^2 * (n-2)^2$, puisque le noyau de convolution est de taille 3*3, il y a 9 termes non nuls à chaque ligne, on a donc :

$$s = \frac{(n^2-9)(n-2)^2}{n^2(n-2)^2}$$
$$s = \frac{n^2-9}{n^2}$$
$$s = 1 - \frac{9}{n^2}$$

6. Nombre d'éléments et sparcity index

La matrice est composée de $3 * 4 = 36$ éléments. De plus, elle représente une couche dense de neurones donc elle ne contient aucun terme nul. On a donc :

$$s = \frac{0}{36} = 0$$

1.4.2 Équivariance

1. $(\varphi_y(f) * k)(x) = (f * k)(x - y)$

On veut montrer :

$$\forall x \in \mathbb{R}^2, (\varphi_y(f) * k)(x) = (f * k)(x - y)$$

On a :

$$(\varphi_y(f) * k)(x) = \int_{\mathbb{R}^2} \varphi_y(f(z))k(z-x)dz = \int_{\mathbb{R}^2} f(z-y)k(z-x)dz$$

On pose $u = z - y$ donc $z = u + y$

Alors :

$$(\varphi_y(f) * k)(x) = \int_{\mathbb{R}^2} f(u)k(u+y-x)du = \int_{\mathbb{R}^2} f(u)k(u-(x-y))$$

Et :

$$(f * k)(x-y) = \int_{\mathbb{R}^2} f(z)k(z-(x-y))dz$$

On a bien :

$$\varphi_y(f) * k = \varphi_y(f * k)$$

2. Rôle de la convolution et du pooling dans les CNN

CNN, Convolutional Neural Network

La convolution détecte et extrait les caractéristiques de l'image. Le pooling consiste à réduire la dimension, ce qui réduit le nombre de calculs pour les couches suivantes. Le polling permet également de réduire le surapprentissage, en réduisant la dimension les détails trop spécifiques disparaissent.

1. Calcul de $R_0, R_{2\pi}, R_\pi, R_{\theta_1} + R_{\theta_2} - R_{\theta_1+\theta_2}$

$$R_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R_{2\pi} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, R_\pi = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, R_{\theta_1} R_{\theta_2} - R_{\theta_1+\theta_2} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

2. Différents opérateurs

Avec l'opération $x \longmapsto \max_{0\leq k\leq 3}\big(f \diamond k\big)(x, k\pi/2)$ quatre convolutions sont calculées et on prend le maximum de ces convolutions. Cela permet de détecter l'orientation de l'emoji, par exemple si l'emoji est orienté à 90°, soit $\frac{\pi}{2}$. alors l'opération sera maximale pour $k = 1$, soit un angle de $\frac{\pi}{2}$. Avec une telle opération les motifs sont détectés si leur orientation est parmi ces quatres angles.

$x \longmapsto \frac{1}{2\pi}\int_0^{2\pi}(f \diamond k)(x, \theta)\, d\theta$, cette opération fait la moyenne sur tous les angles possibles et non plus seulement sur quatre angles comme l'opération précedente. De plus quelque soit l'orientation de l'emoji le résultat sera le même, les motifs sont détectés indépendemment de leur orientation.

Toutefois l'approximation de l'intégrale entraîne un coût en calculs élevé. Pour un réseau avec plusieurs couches le nombre total d'opérations peut exploser.

1.5 Softmax

1.5.1 Version modifiée du SOFTMAX

a. Somme des composantes

Calculons la somme des éléments du vecteur $S_T(z)$:

$$\sum_{i=1}^{n} s_i = \sum_{i=1}^{n} \frac{e^{z_i/T}}{\sum_{j=1}^{n} e^{z_j/T}}$$

Le dénominateur ne dépend pas de l'indice i de la somme, on peut donc factoriser :

$$\sum_{i=1}^{n} s_i = \frac{1}{\sum_{j=1}^{n} e^{z_j/T}} \times \left(\sum_{i=1}^{n} e^{z_i/T} \right)$$

Le numérateur est identique au dénominateur. Par conséquent :

$$\sum_{i=1}^{n} s_i = 1$$

b. Calcul numérique

On calcule les valeurs pour $z = (-2, 3, 4, 1, 0)$ avec différentes valeurs de T. On utilise python pour faire plus rapidement les calculs :

```python
import math as m

z = [-2, 3, 4, 1, 0]

def S(T):
    L = []
    sum = 0
    for i in range (len(z)):
        sum = sum + m.exp(z[i]/T)
    for i in range (len(z)):
        L.append(m.exp(z[i]/T)/sum)
    return L

print(S(1), S(0.2), S(10))
```

Listing 1: Calcul du Softmax avec température en Python

Résultats obtenus :

- **Pour T = 1 :**
  $s \approx [0.0017, 0.2557, 0.6952, 0.0346, 0.0127]$
- **Pour T = 0.2 :**
  $s \approx [0.000, 0.0067, 0.9933, 0.000, 0.000]$
- **Pour T = 10 :**
  $s \approx [0.1420, 0.2341, 0.2587, 0.1917, 0.1734]$

c. Limite quand $T \to +\infty$

Quand T devient très grand, pour tout i, le terme $\frac{z_i}{T}$ tend vers 0. Ainsi, $e^{z_i/T} \to e^0 = 1$.

$$\lim_{T \to +\infty} S_T(z)_i = \frac{1}{\sum_{j=1}^{n} 1} = \frac{1}{n}$$

La distribution converge vers une loi uniforme.

d. Limite quand $T \to 0^+$

On suppose qu'il existe un unique indice k tel que $z_k > z_j$ pour tout $j \neq k$. On écrit l'expression de s_k en séparant le terme k des autres termes dans la somme du dénominateur :

$$s_k = \frac{e^{z_k/T}}{e^{z_k/T} + \sum_{j \neq k} e^{z_j/T}}$$

On divise le numérateur et le dénominateur par $e^{z_k/T}$:

$$s_k = \frac{1}{1 + \sum_{j \neq k} e^{(z_j - z_k)/T}}$$

Puisque z_k est le maximum strict, pour tout $j \neq k$, on a $z_j - z_k < 0$. Lorsque $T \to 0^+$, le terme $\frac{z_j - z_k}{T}$ tend vers $-\infty$ donc l'exponentielle tend vers 0 :

$$\lim_{T \to 0^+} e^{(z_j - z_k)/T} = 0$$

Alors :

$$\lim_{T \to 0^+} s_k = \frac{1}{1 + 0} = 1$$

Pour les autres indices $i \neq k$, comme la somme des probabilités est toujours égale à 1 ($\sum s_j = 1$) et que le terme du maximum s_k tend vers 1, la somme des termes doit tendre vers 0. Comme les s_i sont positifs, ils tendent tous vers 0 :

$$\lim_{T \to 0^+} s_i = 0$$

e. Influence sur l'échantillonage

Si l'on tire un élément selon les probabilités $S_T(z)$:

- **T élevé :** Le choix est très aléatoire. Cela permet de générer de la diversité, mais risque de produire des résultats incohérents.
- **T faible :** On choisit presque systématiquement la valeur ayant le score le plus élevé. Le comportement est "sûr".

2 Application à un jeu de données

Les notebooks se trouvent à cet URL: `https://github.com/magarthur/Mines/tree/main`

2.1 Analyse exploratoire

2.1.1 Type des données

```
# On affiche les 5 premières lignes du dataset
data = pd.read_csv("Mines/bike_daily.csv")
data.head()
```

	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	6	0	2	0.34	0.36	0.81	0.16	331	654	985
1	2	2011-01-02	1	0	1	0	0	0	2	0.36	0.35	0.70	0.25	131	670	801
2	3	2011-01-03	1	0	1	0	1	1	1	0.20	0.19	0.44	0.25	120	1229	1349
3	4	2011-01-04	1	0	1	0	2	1	1	0.20	0.21	0.59	0.16	108	1454	1562
4	5	2011-01-05	1	0	1	0	3	1	1	0.23	0.23	0.44	0.19	82	1518	1600

Table 5: Aperçu des premières lignes du dataset

Dans la suite du TP, on supprimer la colonne 'Instant' car doublon avec l'indexation proposée par python.

```
for col in data.columns:
    print(f"Colonne: {col}")
    print(f"  Type: {data[col].dtype}")                    # Type de la colonne
    print(f"  Valeurs manquantes: {data[col].isna().sum()}")  # Nombre de valeurs manquantes
```

Colonne	Type	Valeurs manquantes
dteday	object	0
season	int64	0
yr	int64	0
mnth	int64	0
holiday	int64	0
weekday	int64	0
workingday	int64	0
weathersit	int64	0
temp	float64	0
atemp	float64	0
hum	float64	0
windspeed	float64	0
casual	int64	0
registered	int64	0
cnt	int64	0

Table 6: Structure du dataset

Notre dataset ne contient pas de données manquantes.

```
"""
on sépare les colonnes en trois types :
  -catégorielles
  -continues
  -Nombre de vélos utilisés
"""

L1 = ['season','yr','mnth', 'holiday','weekday','workingday','weathersit']
L2 = [('temp',41),('atemp',50),('hum',100),('windspeed',64)]
L3 = ['casual','registered','cnt']

#alcul des statistiques
tableau_stats = data[L3].describe()

# Affichage du tableau
print(tableau_stats.round(2))
```

2.1.2 Analyse statistique

Statistique	Casual	Registered	Total (cnt)
Count	731.00	731.00	731.00
Mean	848.18	3656.17	4504.35
Std	686.62	1560.26	1937.21
Min	2.00	20.00	22.00
25%	315.50	2497.00	3152.00
Médiane	713.00	3662.00	4548.00
75%	1096.00	4776.50	5956.00
Max	3410.00	6946.00	8714.00

Table 7: Statistiques descriptives des locations (Casual, Registered et Total)

La valeur min semble être une valeur atypique, voici la ligne correspondante :

	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
667	2012-10-01	4	1	10	0	1	1	3	0.44	0.44	0.88	0.36	2	20	22

Table 8: Aperçu ligne atypique casual et registered

En se renseignant sur internet, on apprend qu'un ouragan est passé à côté de Washington DC le 12 octobre 2012. Pour des raisons de sécurité, le service de vélo en libre service a été désactivé ce jour-là. L'objectif du TP n'étant pas de prédire un futur ouragan au dessus de Washington, nous décidons d'inputer la valeur moyenne à 'casual' et à 'registered' selon plusieurs critères :

- mnth = 10
- workingday = 1
- weathersit =3

In fine, on obtient :

- casual = 262
- registered = 2897

- cnt = 3159

```python
indices_anomalie = data1[data1['cnt'] == 22].index

# On cherche les jours similaires (Mois 10, Travaillé, Mauvaise Météo)
masque_reference = (
    (data1['mnth'] == 10) &
    (data1['workingday'] == 1) &
    (data1['weathersit'] == 3) &
    (data1['cnt'] != 22)
)

#Calcul des moyennes casual et registered
moyenne_casual = data1.loc[masque_reference, 'casual'].mean()
moyenne_registered = data1.loc[masque_reference, 'registered'].mean()

# Mise à jour de la valeur
# On remplace casual et registered par leurs moyennes respectives
data1.loc[indices_anomalie, 'casual'] = int(moyenne_casual)
data1.loc[indices_anomalie, 'registered'] = int(moyenne_registered)

# Calcul de cnt
data1.loc[indices_anomalie, 'cnt'] = data1.loc[indices_anomalie, 'casual'] +
↪  data1.loc[indices_anomalie, 'registered']
```

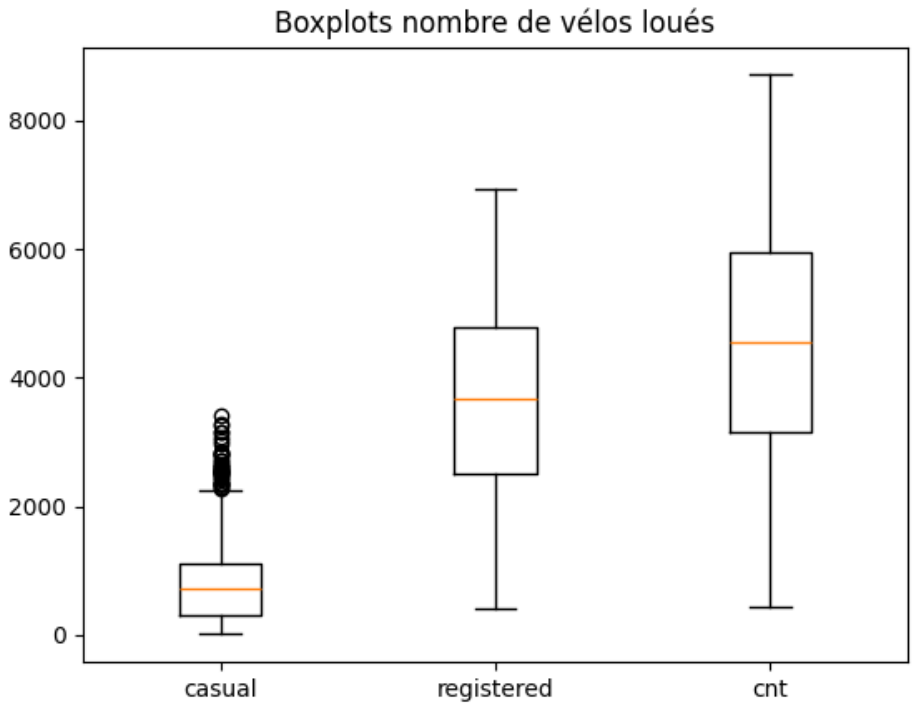

Figure 10: Boxplots nombre de vélos loués

On remarque de nombreux points atypiques sur le boxplot casual. Ces différents points correspondent à des jours de très beau temps. On ne peut donc pas les considérer comme aberrants.

2.1.3 Valeurs aberrantes

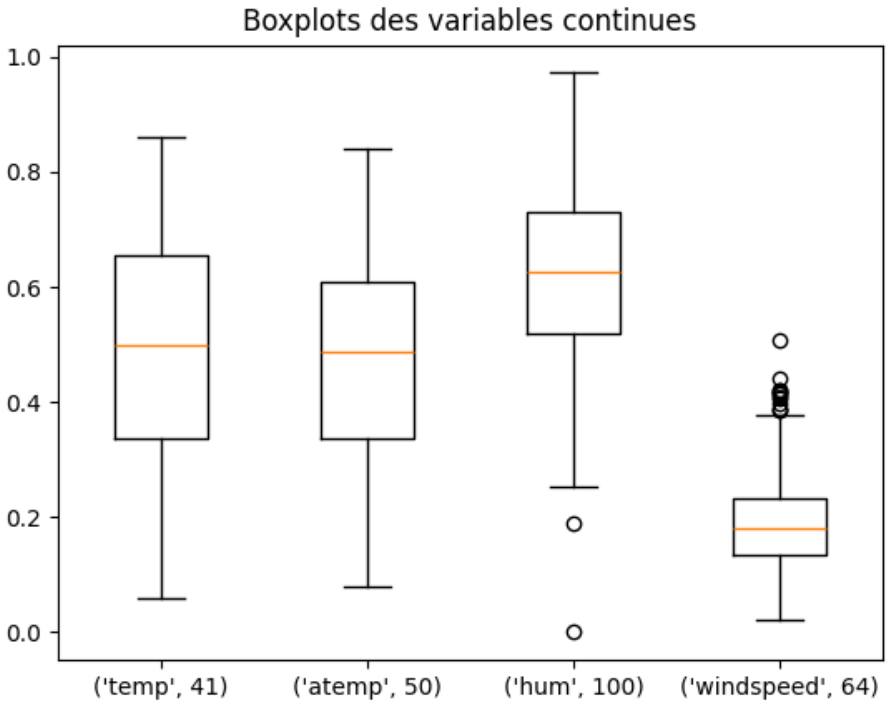

Figure 11: Boxplots des variables continues

L'analyse de ces boxplots nous amène à penser qu'il existe une valeur aberrante pour la variable humidité. Une humidité nulle à Washington est impossible. Ci-dessous la ligne correspondante :

	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
68	2011-03-10	1	0	3	0	4	1	3	0.39	0.39	0.0	0.26	46	577	623

Table 9: Aperçu ligne aberrante humidité

Nous choisissons donc d'imputer à 'hum' la valeur moyenne de l'humidité observée pour le même mois et la même année.

On obtient hum = 0.59

Pour 'windspeed', aucune valeur aberrante n'a été détectée, seulement des valeurs atypiques correspondant à des jours de fort vent.

2.1.4 Analyse par les graphiques

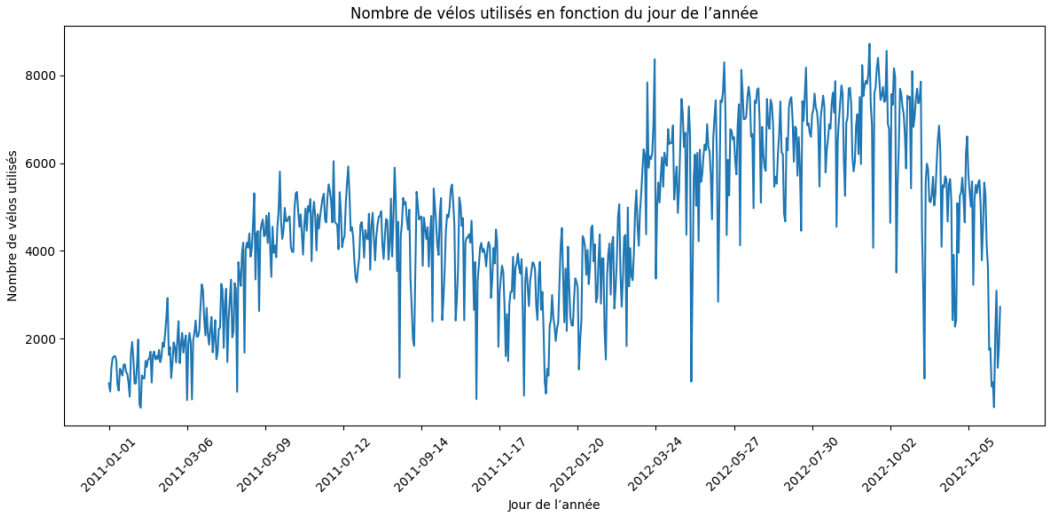

Figure 12: Nombre de vélos utilisés en fonction du jour de l'année

On commence notre analyse par la représentation graphique du nombre de vélos utilisés (figure 12). On remarque tout d'abord de nombreux pics de variation qui correspondent aux jours de très mauvais temps. La période assez courte des données ne nous permet pas d'identifier une réelle tendance, même si nous observons une légère croissance dans le nombre de vélos loués. Cette croissance s'explique en grande partie par l'installation de nouvelles stations à Washington DC. Les premières stations Capital Bikeshare ont été installées en septembre 2010 (environ 114). Aujourd'hui, on en dénombre environ 800, soit environ 8000 vélos.

En ce qui concerne la saisonnalité, il semble y avoir une forte corrélation positive entre le nombre de vélos loués et le beau temps. Les pics de fréquentation se situent en été et en automne (climat humide et subtropical → été chaud avec des hivers froids et neigeux)

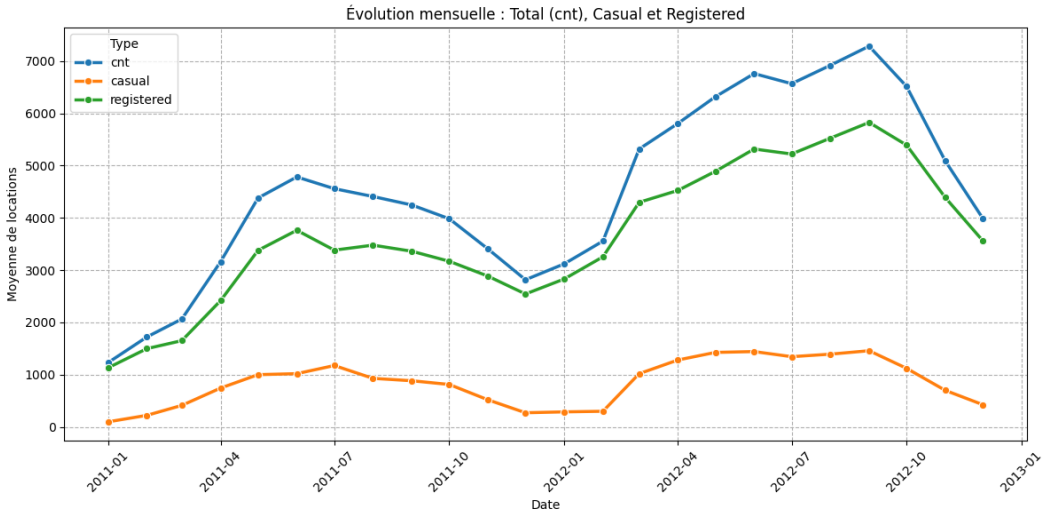

Figure 13: Moyenne mensuelle des vélos utilisés

La figure 13 permet de voir plus facilement la saisonnalité du jeu de données. Calculer la moyenne mensuelle permet de lisser les variations et donc de supprimer les nombreux pics.

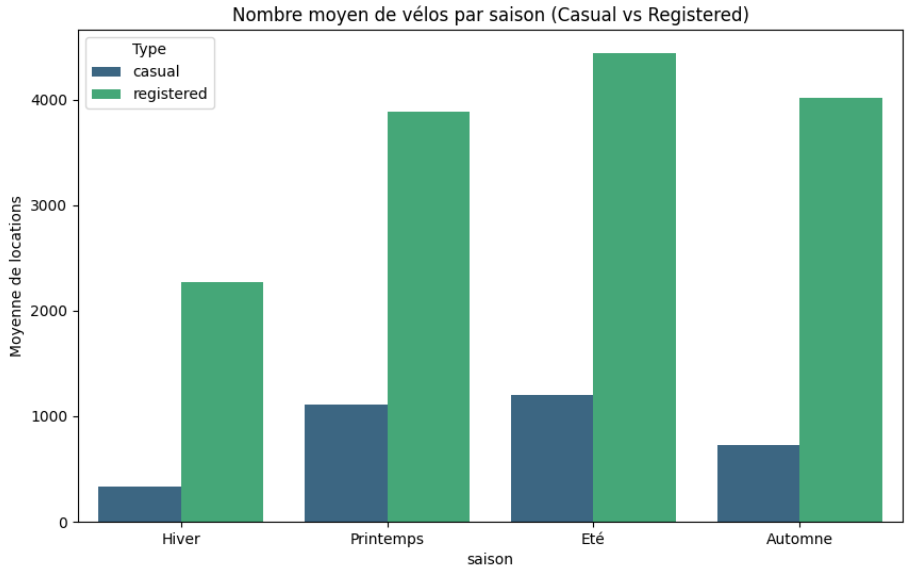

Figure 14: Moyenne de vélos par saison

Notre conclusion faite au-dessus semble juste. Le temps froid et neigeux fait baisser de manière significatif le nombre d'utilisateurs de vélos. De même, casual et registered suivent les mêmes variations sur une année civile.

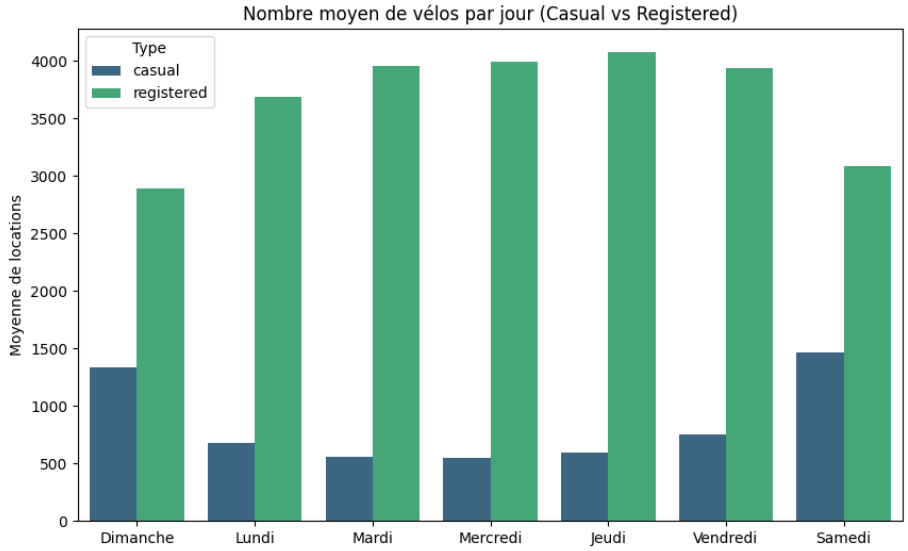

Figure 15: Moyenne de vélos par jour (casual vs registered)

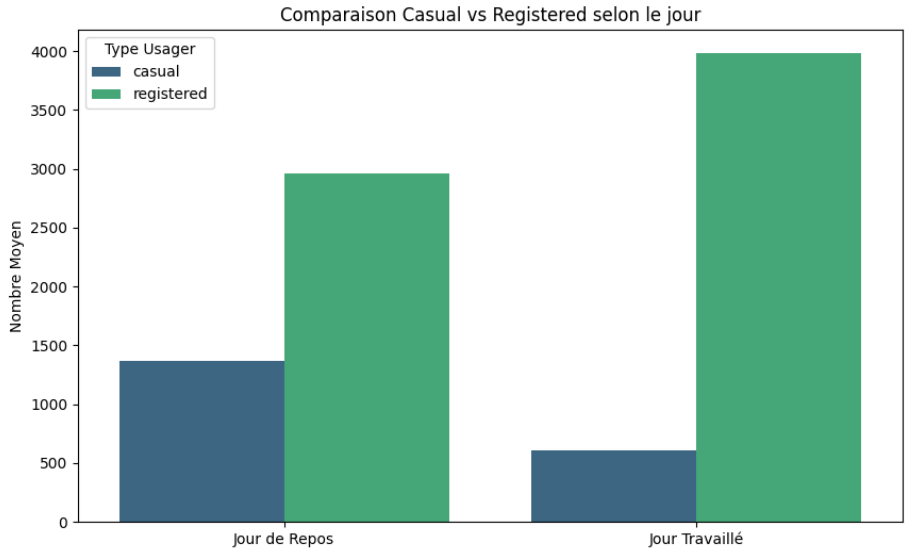

Figure 16: Moyenne de vélos utilisés selon le type de jour (casual vs registered)

Les figures 15 et 16 permettent d'identifier les types d'utilisateurs. Les "registered" sont des travailleurs, ils utilisent principalement le vélo du Lundi au vendredi (vélotaf). Les "casual" sont principalement des touristes qui utilisent le vélo le Samedi ou le Dimanche pour découvrir la ville.

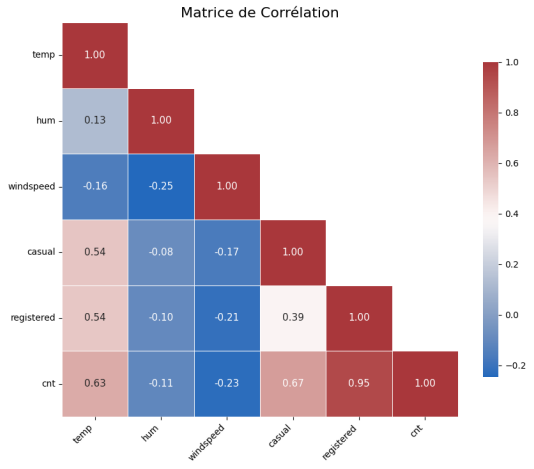

Figure 17: Matrice de corrélation 1

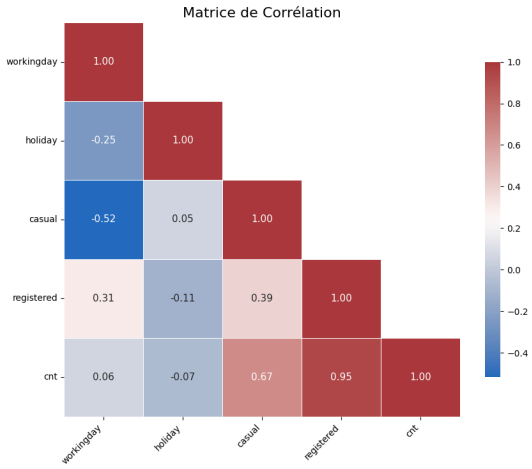

Figure 18: Matrice de corrélation 2

Ces deux matrices de corrélation confirment les conclusions précédentes. La variable "casual" présente une corrélation négative, tandis que "registered" est corrélée positivement. Par ailleurs, "casual", "registered" et donc "cnt" sont négativement corrélés avec "hum" et "windspeed". Une forte humidité ainsi que des vents soutenus se rencontrent plus fréquemment en hiver. Ceci nous conforte l'idée que les gens utilisent moins les vélos en hiver.

2.2 Classification

2.2.1 Choix des données

Les variables "atemp", "temp", "hum" et "windspeed" sont toutes des variables continues déjà normalisées, il n'y a donc pas besoin d'appliquer un nouveau traitement.

Le but est de prédire si le nombre de vélos dépassent le seuil de 4000, on classifie donc les journées en deux types. On crée pour cela une nouvelle colonne "class" qui vaut 1 lorsque "cnt" est supérieur à 4000 et 0 sinon.

2.2.2 Division des données

On divise nos données en données d'apprentissage et en données test avec une proportion 80/20.

La proportion de lignes qui appartiennent à la classe 1 est de 61% environ pour les données d'apprentissage tandis qu'elle est de 63% pour les données test. Les proportions sont semblables, les classes sont réparties équitablement entre les données d'apprentissage et les données test.

2.2.3 Fonction de coût

On est face à un problème de classification, on n'utilise donc pas de fonction de coût de type Mean Squared Error qui est réservée à la régression. Pour un problème de classification on utilise par exemple Binary Cross-Entropy ou Categorical Cross-Entropy, on a ici seulement deux classe donc on utilise Binary Cross-Entropy comme fonction de coût pour notre modèle.

2.2.4 Entraînement des modèles

On commence avec un modèle avec une seule couche cachée et 8 neurones. On fixe "epochs" à 400, on a une seule couche cachée et peu de données, on peut donc se permettre un nombre élevé. On fixe "batch-size" à 100. On entraîne ce modèle. On trace l'évolution de l'accuracy sur les données d'apprentissage et sur les données test, ainsi que l'évolution de la valeur de la fonction coût.

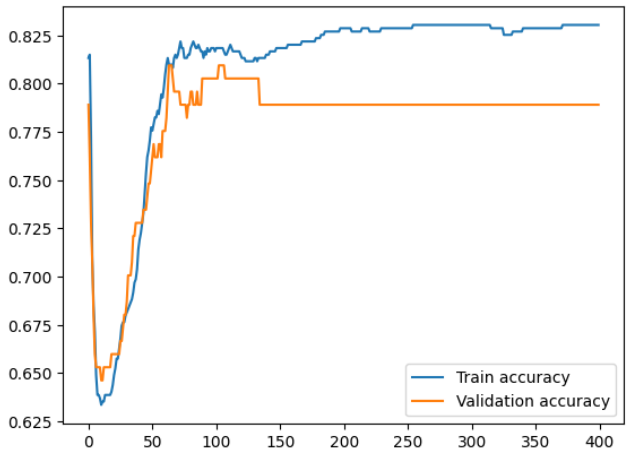

Figure 19: Evolution de l'accuracy du modèle 1

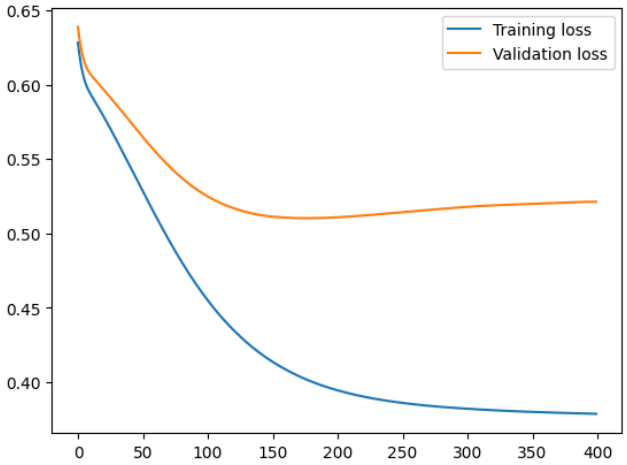

Figure 20: Evolution de loss du modèle 1

On remarque qu'après environ 115 epochs, l'accuracy sur les données de d'apprentissage stagne, de plus la loss augmente à nouveau pour les données de validation et s'éloigne de la loss des données d'apprentissag , c'est le signe

d'un possible surapprentissage. On entraîne donc le même modèle mais en fixant epochs à 115. On obtient alors la matrice de confusion suivante :

	Prédit 0	Prédit 1
Réel 0	34	20
Réel 1	8	85

La majorité des éléments sont bien classés, cependant on a beaucoup de faux positifs (20), on le voit également avec les différents scores, la précision est semblable pour les deux classes, environ 81% des éléments attribués à une classe le sont justement. Toutefois le modèle ne détecte que 65% des éléments de la classe 0.

Table 10: Rapport de Classification du Modèle 2

Classe	**Précision**	**Rappel**	**F1-Score**	**Support**
0	0.81	0.65	0.72	54
1	0.82	0.91	0.86	93

On essaie ensuite un modèle avec une seule couche cachée avec cette fois 16 neurones. On obtient :

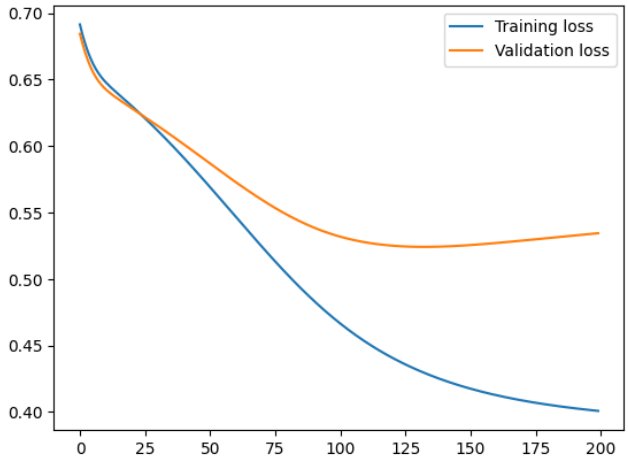

Figure 21: Evolution de l'accuracy du modèle 3

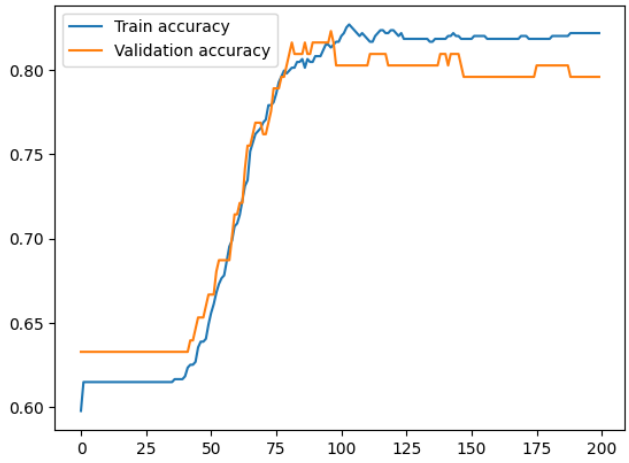

Figure 22: Evolution de loss du modèle 3

Pour les mêmes raisons que précédemment, on entraîne à nouveau le modèle avec seulement 110 epochs, on obtient les prédictions suivantes :

	Prédit 0	Prédit 1
Réel 0	37	17
Réel 1	13	80

On observe que le nombre de faux positifs a diminué, il y a maintenant plus d'éléments de la classe 0 correctement classés, mais cela s'est fait au détriment de la précision. On note aussi que le nombre d'éléments de la classe 1 correctement identifiés a diminué.

Table 11: Rapport de Classification du Modèle 4

Classe	**Précision**	**Rappel**	**F1-Score**	**Support**
0	0.74	0.69	0.71	54
1	0.82	0.86	0.84	93

On essaie maintenant un modèle avec deux couches cachées de neuronnes, une prmeière avec 8 neurones et une seocnde avec 4 neurones. On obtient :

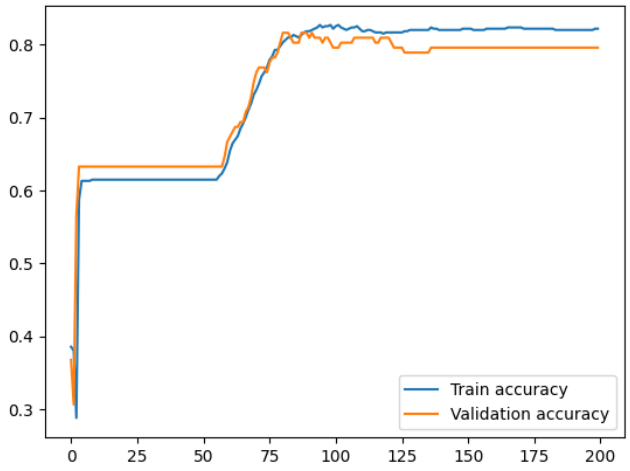

Figure 23: Evolution de l'accuracy du modèle 5

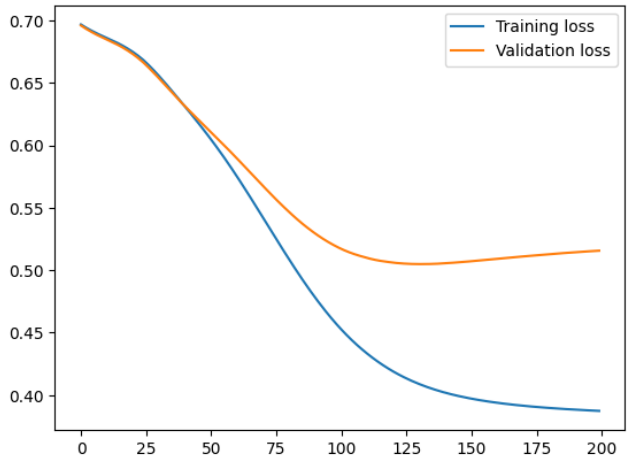

Figure 24: Evolution de loss du modèle 5

On entraîne de nouveau le modèle avec epochs = 115.

	Prédit 0	Prédit 1
Réel 0	37	17
Réel 1	12	81

Avec ce modèle un faux négatif s'est transformé en vrai positif, on a donc une très légère amélioration.

Table 12: Rapport de Classification du Modèle 6

Classe	**Précision**	**Rappel**	**F1-Score**	**Support**
0	0.76	0.69	0.72	54
1	0.83	0.87	0.85	93

On essaye donc de rajouter une troisième couche pour voir s'il on continuer à améliorer le modèle, pour cela on entraîne un modèle avec 16 neurones pour la première couche, 8 pour la deuxième et 4 pour la troisième. Aprsè observation des courbes d'accuracy et de loss, on fixe epochs à 50 et on obtient :

	Prédit 0	Prédit 1
Réel 0	37	17
Réel 1	14	79

La matrice de confusion nous montre que ce modèle prédit plus de faux négatif que le modèle précédent, augmenter le nombre de couches et de neurones conduit peut-être à un surapprentissage, on décide donc de ne pas continuer avec un tel modèle.

On revient plutôt au modèle 6, avec une couche de 8 neurones et une couche de 4 neurones et on change la batch-size à 50 au lieu de 100. Avec epochs=115 on obtient la matrice de confusion suivante :

	Prédit 0	Prédit 1
Réel 0	37	17
Réel 1	12	81

Soit exactement les mêmes résultats, la modification de batch-size n'a eu aucune incidence.

Le choix du modèle dépend maintenant du problème auquel on souhaite répondre, par exemple une compagnie de location de vélos en libre-service peut souhaiter savoir si plus de 4000 vélos vont être utilisés dans la journée afin d'ajouter des vélos à son réseau et contenter tous ses clients. Dans une telle situation, la compagnie est prête à accepter des faux positifs si cela permet de prédire plus de vrais positifs par exemple. Il faudrait alors préférer le modèle 2 qui prédit certes plus de faux positifs mais a un meilleur rappel pour la classe 1. On rappelle la matrice de confusion du modèle 2 :

	Prédit	
	0	1
Réel 0	34	20
Réel 1	8	85

On trace la courbe ROC de ce modèle pour évaluer sa capacité de prédiction.

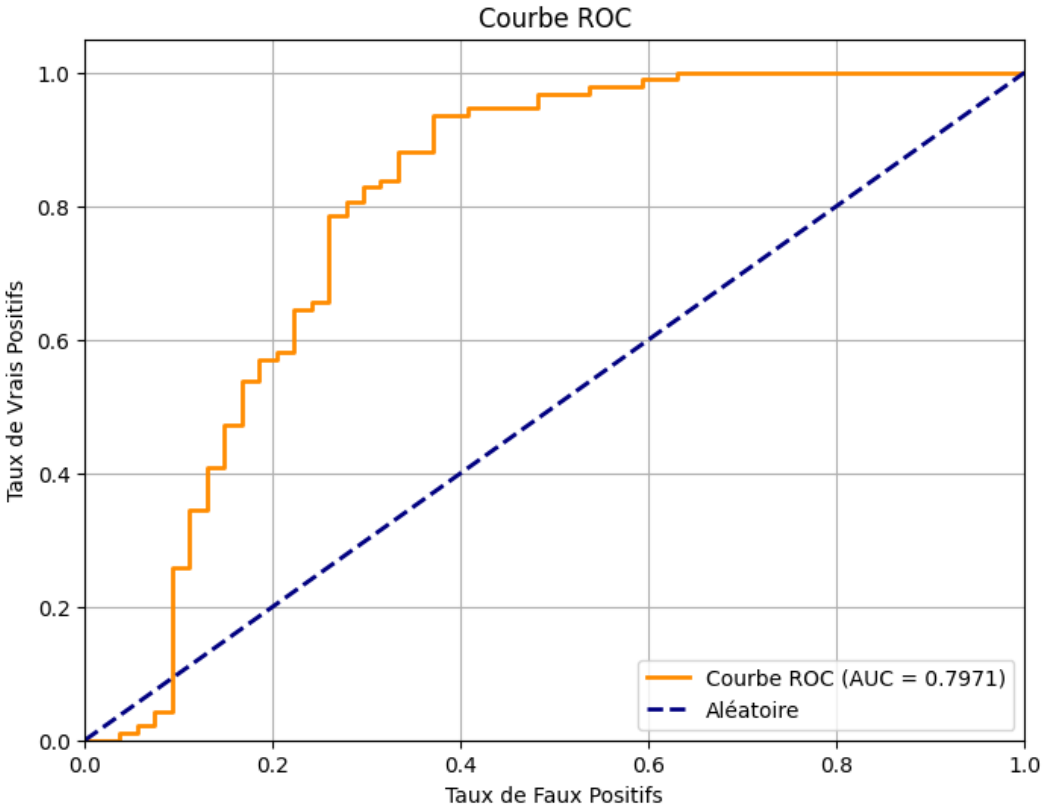

Figure 25: Courbe ROC du modèle choisi

On observe que la courbe ROC du modèle augmente assez rapidement et que celle-ci est bien au-dessus de la diagonale, de plus le modèle a une valeur d'AUC de 0.80. Le modèle a donc un bon pouvoir prédictif par rapport à un classement aléatoire mais celui-ci n'est pas parfait.

2.2.5 Régression logisitique

On utilise maintenant un modèle de régression logistique classique. On obtient la matrice de confusion suivante :

	Prédit	
	0	1
Réel 0	37	17
Réel 1	12	81

On remarque qu'on obtient des résultats semblables à ceux obtenus avec des modèles de réseaux de neurones. On regarde la courbe ROC :

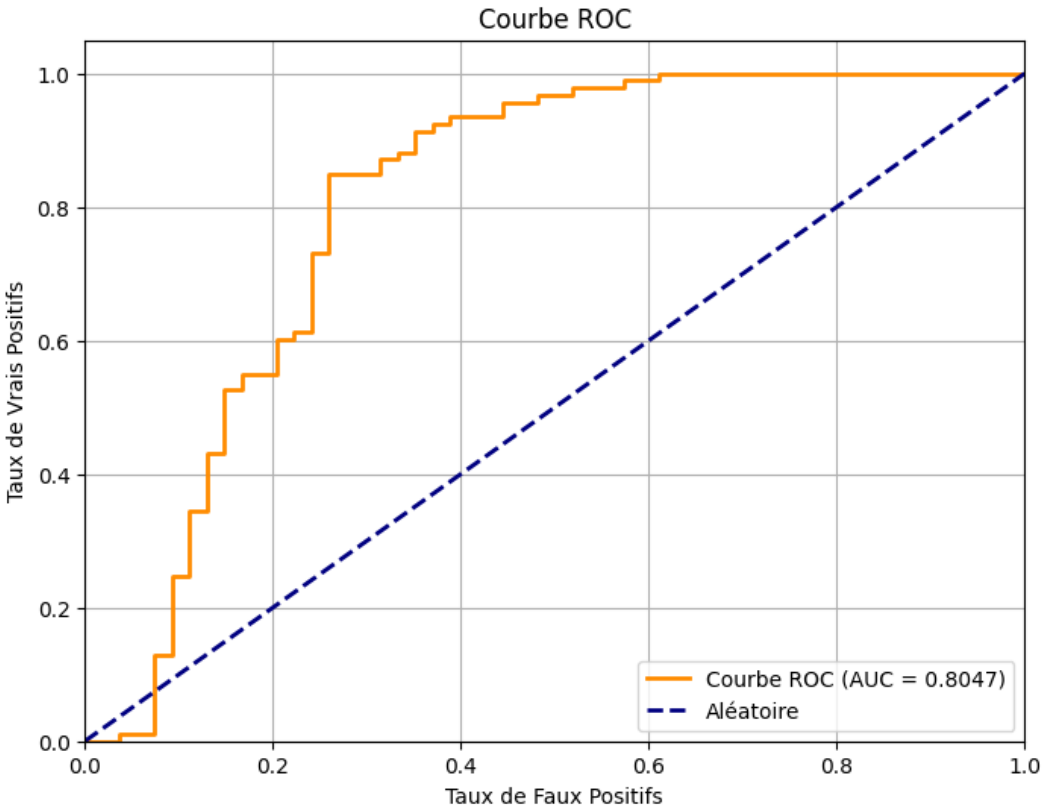

Figure 26: Courbe ROC du modèle choisi

La courbe ROC du modèle de régression logistique est semblable à celle du deuxième modèle de neurones, on note une valeur d'AUC légèrement plus faible pour la régression, 0.78 contre 0.80.

2.2.6 Conclusion

On a vu que la majorité des modèles de neurones entraînes donnaient des résultats de prédiction semblables, avec de nombreux faux positifs et faux négatifs. Il est intéressant de noter que l'on obtient un résultat très semblable avec la régression logistique. Il semble donc que des composantes non-linéaires ne sont pas utiles dans notre problème. Dans ce cas on privilégie la régression logistique qui est plus facile à entraîner et moins coûteuse. Toutefois, le deuxième modèle de neurones avec une seule couche de 8 neurones a donné un résultat un petit peu différent avec moins de faux négatifs et plus de vrais positifs, améliorer le rappel de la classe 1 peut être intéressant dans notre situation, de plus le modèle de neurones est simple, c'est pourquoi on peut justifier l'utilisation d'un modèle de neurones en fonction de ce que l'on préfère maximiser.

2.3 Régression de série temporelle (RNN)

2.3.1 Analyse et représentation graphique

Voir section 2.1 et plus particulièrement la section 2.1.4

2.3.2 Traitement des variables d'entrée

On utilise la méthode des **OneHot-Encoding** pour faire rentrer les variables catégorielles dans la prédiction. Ainsi, pour la variable *weathersit*, on crée 4 variables booléennes, chacune correspondante à une valeur prise par la variable originale (1, 2, 3 ou 4).

Pour les variables déjà booléennes, comme *workingday*, on se contente de bien leur faire avoir un type 'int' (1 ou 0) afin de pouvoir être lues par *numpy*.

On fixe également la cible *cnt*, et l'on normalise ses valeurs. On 'concatène' ensuite, c'est à dire qu'on assemble tout dans un seul dataset pour l'entraînement.

2.3.3 Séparation train/test

On sépare comme fait dans le notebook le dataset en 80% de train et 20% de test. On s'inspire ensuite des codes fournis sur *ecampus* pour la création de tensors et la création de DataLoaders.

2.3.4 Création des windows length

Le modèle RNN va utiliser les valeurs $i-1$ et i pour prédire une valeur $i+1$, comme cela est demandé par l'énnoncé ($windows_length = 2$). Pour faire cela, on crée un décalage dans le dataset de deux valeurs. On convertit le tout en tableaux numpy pour la suite.

2.3.5 Choix et entraînement d'un modèle de réseau récurrent

On choisit d'utiliser le modèle Lightning, dont un exemple de code est présent sur *ecampus*.

2.3.6 Choix faits, et statistiques durant l'entrainement

Les choix faits sur la structure du réseau :

- **Couche 0 :** RNN
- **Couche 1 :** Dropout (pour éviter que le modèle apprenne par coeur le "Train" et soit sur-entraîné)
- **Couches 2, 3 et 4 :** Reprise de l'architecture fournie dans les codes python que vous nous avez donnés, avec respectivement 64, 16 et 16 neurones
- **Couche 5 :** Sortie

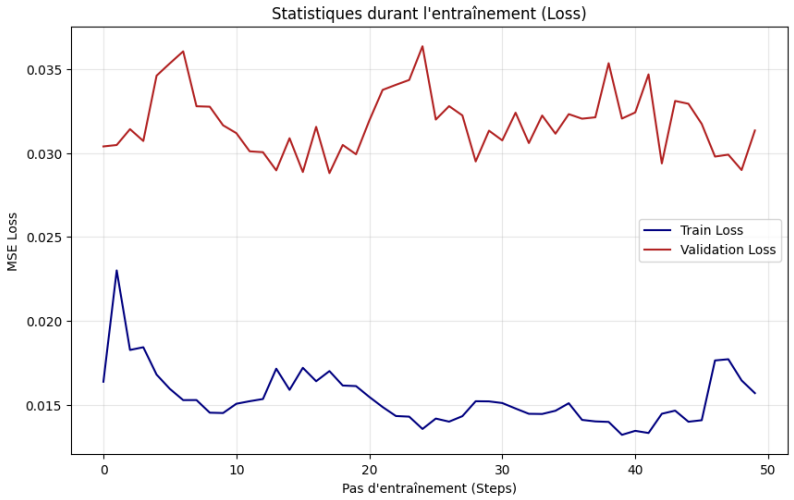

Figure 27: Courbes d'apprentissage : Loss

On analyse la LossCurve ci-dessus:

- Le train loss ne diverge pas, ce qui est bon signe.
- Le train loss semble avoir une tendance à diminuer, ce qui est un bon signe aussi.
- En revanche, la validation loss est nettement supérieure au train loss et ne semble ni diminuer, ni converger.respectivement 64, 16 et 16 neurones. Cela peut être un signe de léger surapprentissage -¿ on aurait tendance à trop généraliser avec ce modèle. Ce surapprentissage restant léger, on poursuit le TP avec ce modèle.

Par ailleurs, la RMSE du modèle est de 1177,77, ce qui semble énorme comparé aux valeurs de 'cnt', mais on voit ensuite que la courbe des prédictions suit plutôt bien la courbe réelle.

2.3.7 Performances du modèle, analyse et interprétations

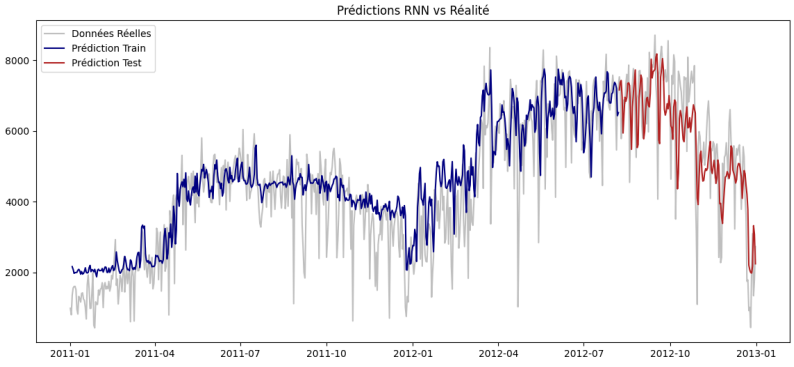

Figure 28: Prédictions RNN sur l'ensemble des données (Train + Test)

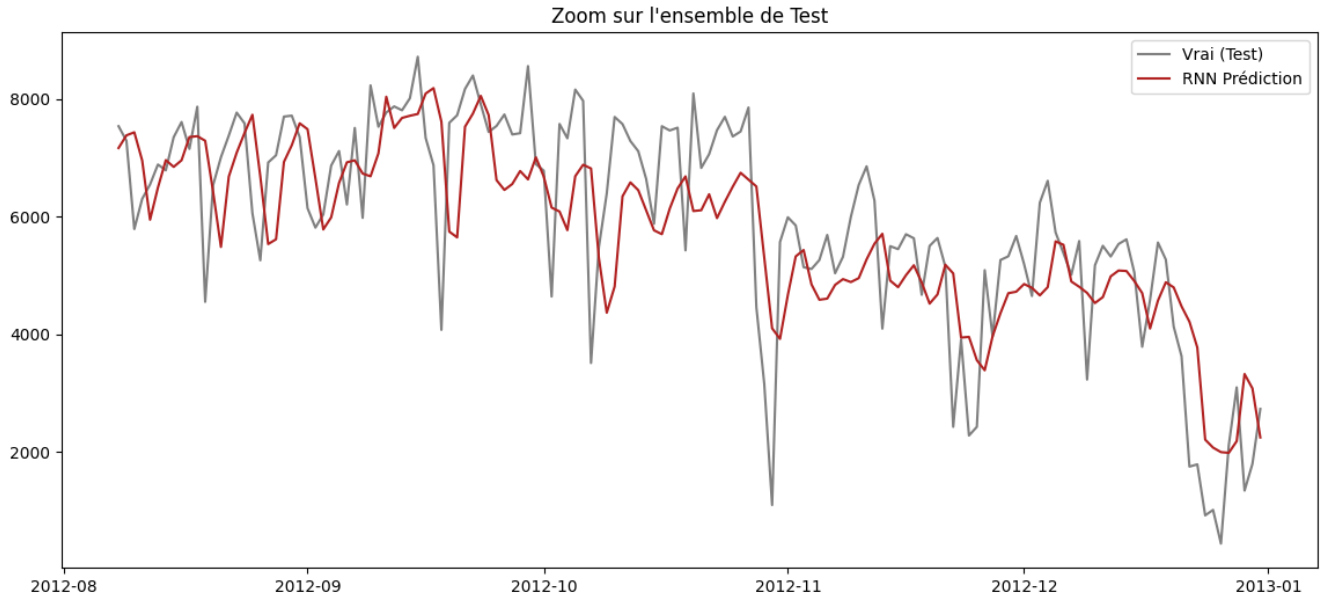

Figure 29: Zoom des prédictions sur l'ensemble de Test

2.3.8 Comparaison avec ARIMA

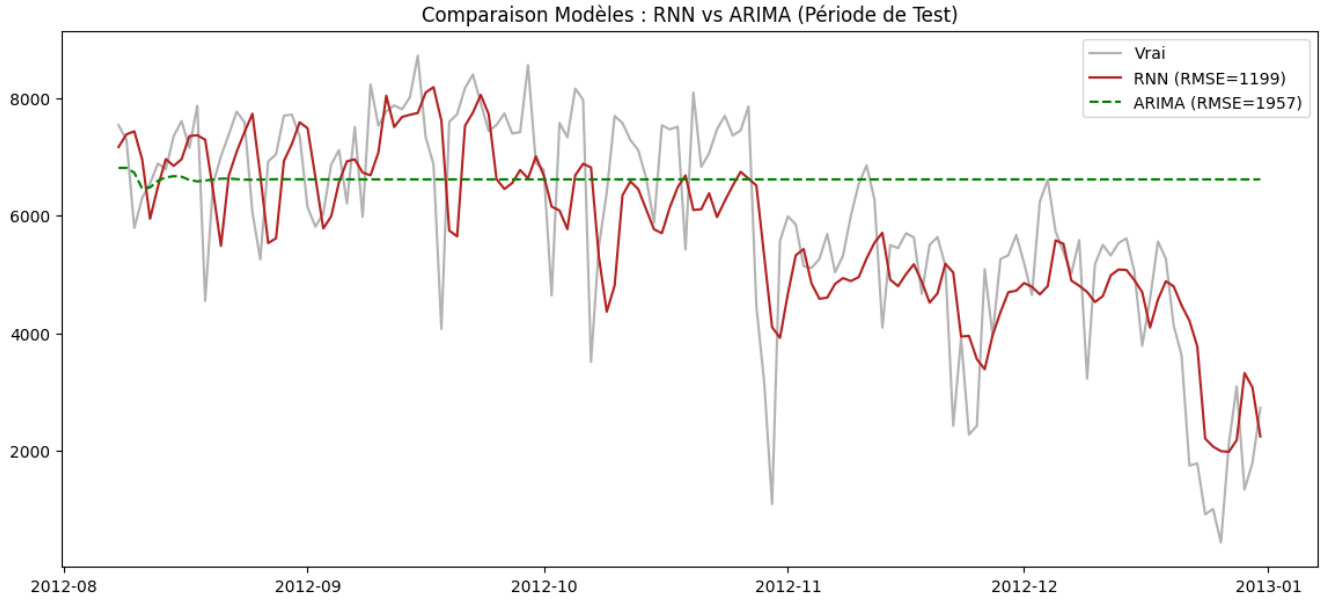

Figure 30: Comparaison finale : RNN (Rouge) vs ARIMA (Vert)

Problème : la prédiction ARIMA tend vers qqch de constant. En effet elle ne se base que sur le jour J pour prédire le jour J+1. Cela fonctionne au début, puis petit à petit ARIMA utilise ses propres prédictions pour prédire la suite, d'où un retour vers la moyenne. On essaye de corriger cela dans le code suivant :

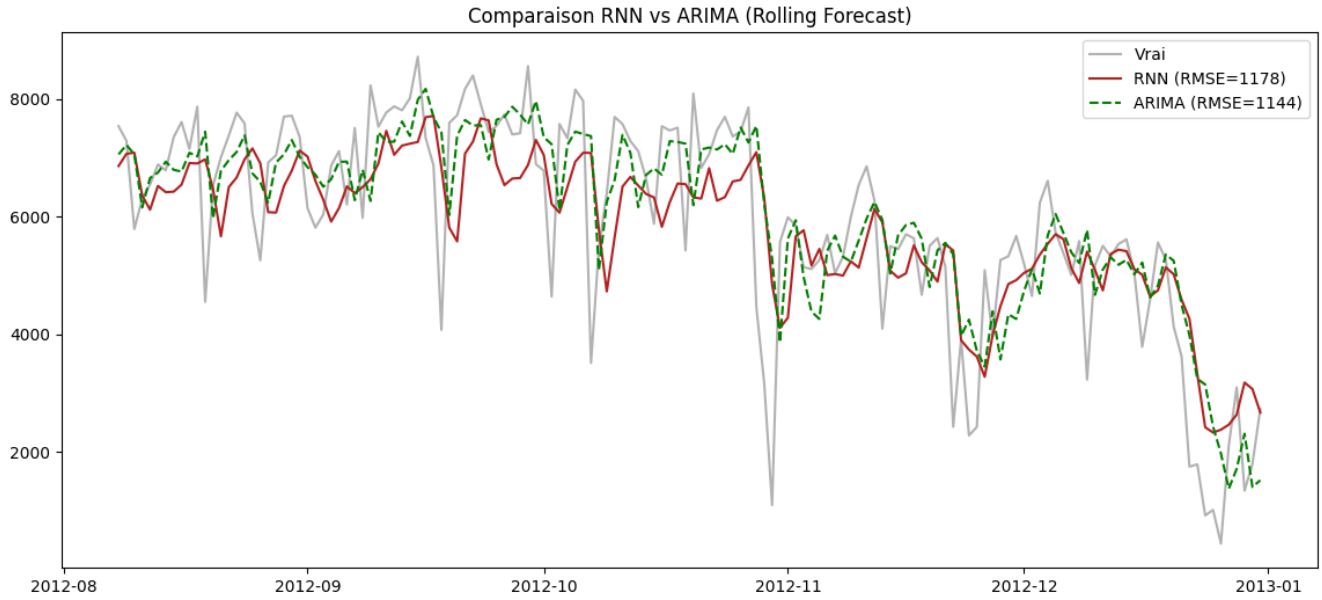

Figure 31: Comparaison finale : RNN (Rouge) vs ARIMA (Vert)

2.3.9 Commentaires

La différence entre le modèle RNN, qui est l'objet de cette partie, et le modèla ARIMA, est que le modèle RNN est dit multivarié. Il utilise en entrée pluieurs paramètres, et peut donc prendre en compte la météo, le jour de la semaine (travaillé ou non), et toutes les données du dataset, ce qui semble à première vue une amélioration énorme par rapport à un ARIMA qui se contente juste d'étudier le modèle comme une série temporelle. Cependant, on constate que le modèle ARIMA a une meilleure performance que le modèle RNN. Comment expliquer cela ? En réalité, le nombre de vélos débloqués dépend tellement de la météo et du jour que ces dernières variables en deviennent inutiles. De plus, ce sont des variables avec une très forte saisonnalité, ce qui les rend facilement prévisible, et les informations qu'elles donnent sont déjà intégrées dans les informations données par l'historique de 'cnt'. Ensuite, le modèle ARIMA choisi était simplement plus efficace et correspondait mieux à notre jeu de données qu'un RNN.