
PROJET RÉSEAUX NEURONAUX

DIANA CUERVO PALOMA

M2 MATHÉMATIQUES ET APPLICATIONS
INGÉNIERIE MATHÉMATIQUE
INGÉNIERIE STATISTIQUE ET DATA SCIENCE
2024 - 2025

1 Réseaux de neurones et l'apprentissage supervisé du PMC

1.1 Réseaux de neurones

Graphes orientés pondérés qui offrent des algorithmes performants pour la prédiction, l'interpolation, la classification, la segmentation ou le regroupement des données :

- N neurones avec états ou activations $a = [a_1, \dots, a_N] \in A^N$ et connexions définies par la matrice de poids $W = [W_{i,j}]$.
- Fonction d'entrée $h(i) = h(i, a, W)$:
 - Affine : $h(i) = W_i a - b_i$
 - Distance : $h(i) = \|a - W_i^\top\|$
- Activation pondérée $e_i(t) = h(i, a(t), W)$ du neurone i à l'instant t .
- Mise à jour par $a_i(t+1) = f_i(e_i(t))$, où f_i est une fonction d'activation. Exemples :
 - Seuil : $f(x) = \mathbb{1}_{x \geq \theta}$
 - ReLU : $f(x) = \max(x, 0)$
 - Sigmoïde : $f(x) = \frac{1}{1 + \exp(-x)}$
 - Gaussienne : $f(x) = \exp\left(-\frac{x^2}{\sigma^2}\right)$
 - Softmax : $f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$
 - Identité $f(x) = x$

2 modes de fonctionnement :

- **Reconnaissance** : Calcul des activations en temps discret (propagation) :

$$a(0) \rightarrow a(1) \rightarrow a(2) \rightarrow \dots \rightarrow a(t) \rightarrow a(t+1)$$
- **Apprentissage** : Trouver les poids W adaptés à l'application considérée à l'aide d'un corpus C d'exemples.

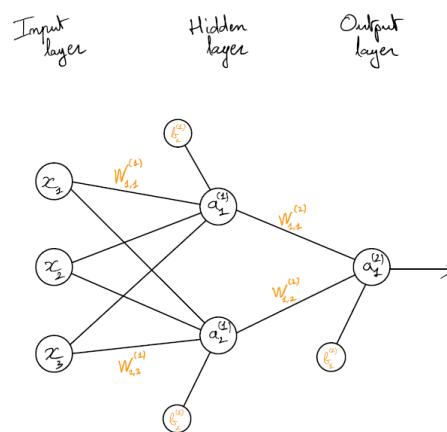


FIGURE 1 – Réseaux d'apprentissage. Source : [1].

Cadre supervisé

- $C = \{(x_i, d_i) \mid i = 1, 2, \dots, n\}$: corpus d'apprentissage avec $x \in \mathbb{R}^k$ (entrée) et $d \in \mathbb{R}^m$ (sortie désirée).
- $s = s(x)$: sortie du réseau après propagation de x .
- $E(s, d)$: fonction mesurant l'erreur entre s et d .

Objectif : Trouver W qui minimise l'erreur via les cycles d'apprentissage.

Initialisation : Définir $W = W(0)$.

Soit $W(t)$ l'état de W à l'instant t .

Cycle d'apprentissage : Pour $t \geq 0$, exécuter :

- Sélectionner aléatoirement $(x, d) \in C_{\text{apprentissage}}$.
- Calculer la sortie s du réseau et l'erreur $E(s, d)$.
- Règle d'apprentissage delta : Calculer $\frac{\partial E}{\partial W}$ et mettre à jour les poids :

$$W(t+1) = W(t) - \lambda \frac{\partial E}{\partial W}$$

où λ est le taux d'apprentissage.

- Incrémenter t : $t = t + 1$.

La séquence $W(t)$ pour $t \geq 0$ converge vers un minimum de E , qui est une fonction de Lyapunov du système dynamique définie par la règle delta.

Critères d'arrêt :

- **Seuil de tolérance** $\epsilon \geq 0$:

$$\|W(t+1) - W(t)\| \leq \epsilon$$

- **Durée maximale** $t_{\text{final}} = \text{Epochs} \times n$, où Epochs est le nombre total de passages du corpus d'apprentissage, et n sa taille.

Cadre non supervisé

- $C = \{x_i \mid i = 1, 2, \dots, n\}$: corpus d'apprentissage où la sortie $d \in \mathbb{R}^m$ est inconnue.
- **Objectif** : Ajuster W pour que les exemples x soient regroupés de manière cohérente.

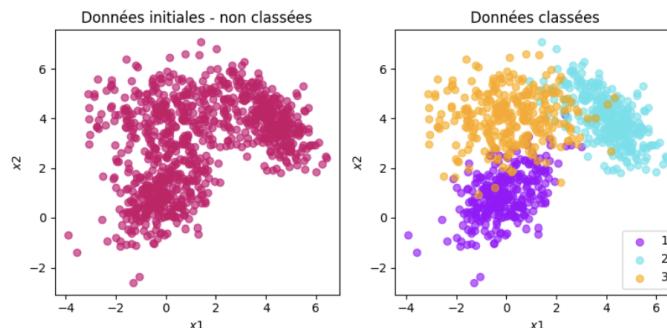


FIGURE 2 – Apprentissage non supervisée. Source : Travail personnel

1.2 Le Perceptron Multi-Couche (PMC)

- **Approximateur universel de fonctions** : Une seule couche cachée avec une activation non linéaire et une sortie linéaire peut approximer toute fonction continue sur un support borné.
- **Architecture** : Composé d'une couche d'entrée, de une ou plusieurs couches cachées, et d'une couche de sortie. L'activation se propage successivement de l'entrée vers les couches cachées, puis vers la sortie.

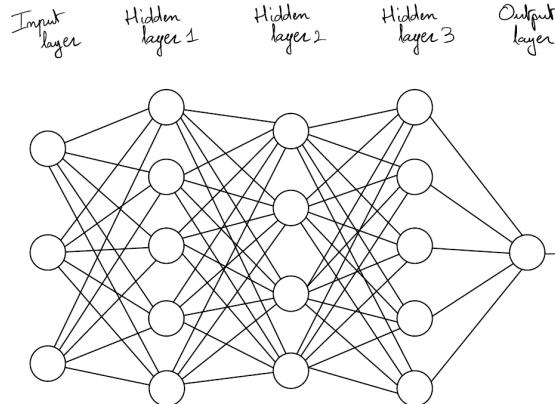


FIGURE 3 – Structure d'un Perceptron Multi-Couche (PMC) Source : [1].

- **Fonctions utilisées :**
 - **Entrée** : Affine.
 - **Activation cachée** : Sigmoïde, Relu.
 - **Sortie** :
 - Classification binaire : Sigmoïde.
 - Multi-classification : Softmax.
 - Régression : Identité.
- **Fonctions d'erreur :**
 - Régression : Erreur quadratique $E = \frac{1}{2} \sum_i (d_i - s_i)^2$.
 - Classification : Entropie croisée $E = -\sum_i d_i \log(s_i)$.
- **Règle d'apprentissage** : Ajustement des poids basé sur le gradient de l'erreur propagé en arrière à travers les couches (retropropagation) :

$$\Delta W_{i,j} = \lambda a_j \text{err}_i$$

$$\text{err}_i = \begin{cases} -\frac{\partial E}{\partial s_i} f'(h(i)) & \text{si } i \text{ est un neurone de sortie,} \\ f'(h(i)) \sum_k W_{k,i} \text{err}_k & \text{si } i \text{ est un neurone caché.} \end{cases}$$

2 Utilisation de R pour le PMC avec comparaison avec la méthode SVM

2.1 Problème de la classification

- **Objectif** : Assigner un exemple x à l'une des M classes possibles, où $M \geq 2$. Le corpus d'apprentissage est défini comme :

$$C = \{(x_i, d_i) \mid i = 1, 2, \dots, n\},$$

où $x_i \in \mathbb{R}^k$ représente les caractéristiques de l'exemple, et $d_i \in \{1, 2, \dots, M\}$ indique sa classe.

- **Sortie attendue** : Un vecteur $s \in \mathbb{R}^M$, souvent représenté sous forme *one-hot* :

$$s_m = \begin{cases} 1 & \text{si l'exemple appartient à la classe } m, \\ 0 & \text{sinon.} \end{cases}$$

2.2 Support Vector Machine (SVM)

Le classifieur SVM est une méthode d'apprentissage supervisé utilisée pour les problèmes de classification.

- **Contexte binaire ($M = 2$)** : Trouver un hyperplan de séparation $H : \langle w, x \rangle + b = 0$ avec $w \in \mathbb{R}^k$, $b \in \mathbb{R}$, tel que :

$$\begin{aligned} \langle w, x \rangle + b &> 0 & \text{si } d = 1, \\ \langle w, x \rangle + b &< 0 & \text{si } d = -1, \end{aligned}$$

et maximiser la marge entre les classes. Les points de données les plus proches de l'hyperplan sont appelés *vecteurs de support*.

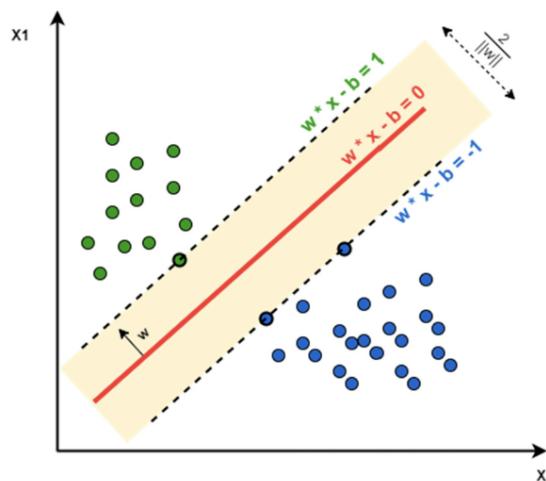


FIGURE 4 – SVM binaire. Source : [3]

- **Hyperplan optimal :** Résolution du problème d'optimisation :

$$\min_{w,b,s} \quad \frac{1}{2} \|w\|^2 + \alpha \sum_{i=1}^n s_i$$

sous les contraintes :

$$d_i(\langle w, x_i \rangle + b) \geq 1 - s_i, \quad s_i \geq 0, \quad \forall i \in \{1, \dots, n\}.$$

où :

- s_i : Variables pour tolérer les erreurs de classification.
- α : Paramètre qui contrôle le compromis entre la maximisation de la marge et la minimisation des erreurs.

- **Contexte multiclass ($M > 2$) :** Étendre le SVM binaire pour séparer M classes en construisant M hyperplans $H_m : \langle w_m, x \rangle + b_m = 0$ qui distinguent chaque classe m des autres.

La fonction objectif devient :

$$\min_{w,b,s} \quad \frac{1}{2} \sum_{m=1}^M \|w_m\|^2 + \alpha \sum_{i=1}^n s_i$$

sous les contraintes :

$$\langle w_{d_i}, x_i \rangle + b_{d_i} \geq \langle w_m, x_i \rangle + b_m + 1 - s_i, \quad \forall m \neq d_i, \quad s_i \geq 0.$$

La classe prédictive correspond à celle qui maximise la fonction de décision :

$$f(x) = \operatorname{argmax}_{m \in \{1,2,\dots,M\}} \langle w_m, x \rangle + b_m.$$

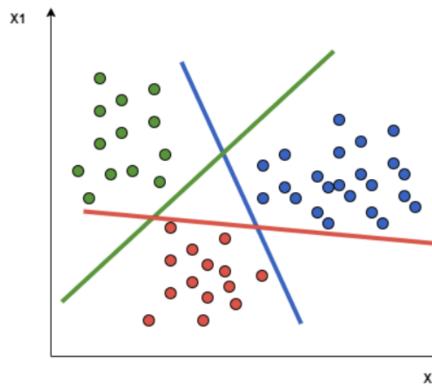


FIGURE 5 – SVM multiclass. Source : [4]

2.3 Implémentation en R

Nous allons entraîner le PMC et le SVM pour une classification multiclasse sur deux ensembles de données avec $M = 3$ classes simulées dans \mathbb{R}^2 .

- Le premier ensemble est une mixture gaussienne.
- Le deuxième ensemble est un ensemble de carrés.
- Chaque ensemble a été aléatoirement divisé en 70% pour l'entraînement des modèles et 30% pour les tests ou la validation *a posteriori*.

2.3.1 Mixture Gaussienne

Simulation de $n = 3000$ exemples des 3 distributions normales, avec 1000 points pour chacune en \mathbb{R}^2 , avec différentes moyennes (μ) et écarts-types (σ).

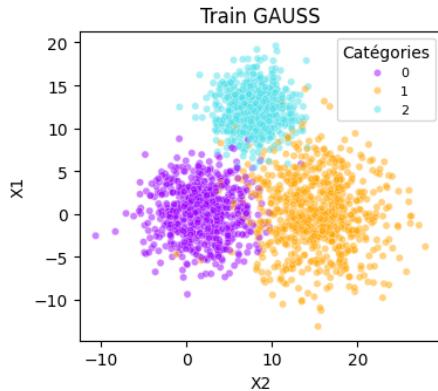


FIGURE 6 – Source : Travail personnel

SVM

Paramètres du modèle :

- Type de SVM : Classification C-SVM
- Noyau de SVM : Linéaire
- Coût : 1 (paramètre de pénalité)

Voici la sortie `summary(model_SVM)`

```
Call:
svm(formula = d ~ X1 + X2, data = sets$train_data, type = "C-classification",
     kernel = "linear")

Parameters:
  SVM-Type:  C-classification
  SVM-Kernel: linear
  cost:  1

Number of Support Vectors:  287
( 101 66 120 )

Number of Classes:  3

Levels:
  0 1 2
```

Le modèle a sélectionné 227 vecteurs de support. Ces points de données déterminent les frontières de décision entre les classes. Pour les classes 0, 1 et 2; 94, 52 et 81 vecteurs de support ont été sélectionnés respectivement.

À continuation, une visualisation des hyperplans qui séparent l'ensemble de données dans les prédictions du modèle.

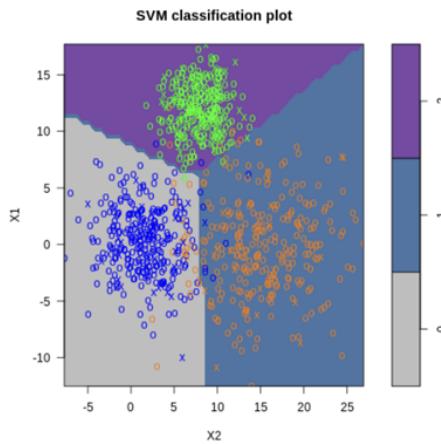


FIGURE 7 – Source : Travail personnel

PMC

Le réseau neuronal est ajusté à l'aide de la bibliothèque **caret**, en testant plusieurs combinaisons d'hyperparamètres (**size** et **decay**) pour trouver la meilleure configuration. La validation croisée est utilisée pour évaluer les combinaisons, et finalement, le modèle est entraîné avec les meilleurs paramètres trouvés, avec un résumé du modèle final. Cette stratégie est mise en œuvre dans les modèles utilisés, notamment le modèle PMC, avec la bibliothèque **nnet**.

Le modèle a été entraîné avec un réseau neuronal comportant 2 entrées, 3 neurones cachés et 3 sorties. Le processus d'entraînement a permis de minimiser la fonction d'erreur, se stabilisant après 100 itérations. Voici un résumé du modèle :

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec une couche cachée
Architecture		2 – 3 – 3
Fonction d'entrée	$h(i)$	Identité – Affine – Affine
Fonction d'Activation	$e(i)$	Identité - Sigmoïde – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.2
Époques		<code>maxit = 1000 ou convergence d'Erreur</code>
Ajustement des Hyperparamètres		# de neurones dans la couche cachée et taux d'apprentissage
Logiciel		R - nnet

Voici la visualisation de l'architecture en utilisant : `plotnet(model_PMC)`

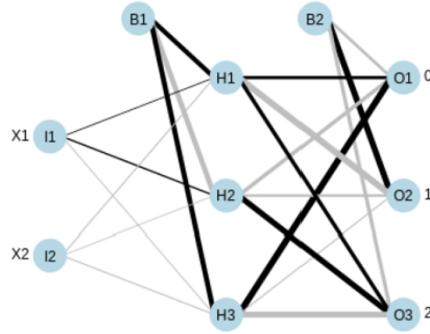


FIGURE 8 – Source : Travail personnel

Comparaison des résultats sur le corpus de validation

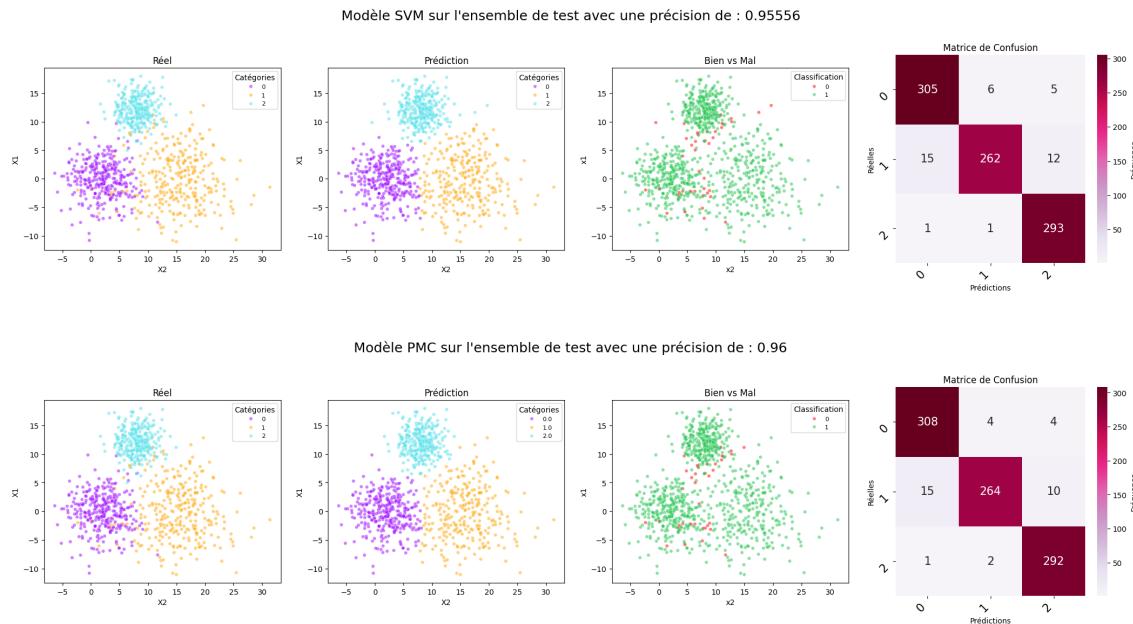


FIGURE 9 – Source : Travail personnel

Les deux modèles, SVM et PMC, obtiennent de très bonnes performances sur la mixture gaussienne, qui est un ensemble linéairement séparable.

2.3.2 Ensemble des Carrés

Simulation de $n = 2700$ exemples à partir des distributions uniformes entre 0 et 1 en \mathbb{R}^2 , déplacées dans le plan, avec 900 points par classe.

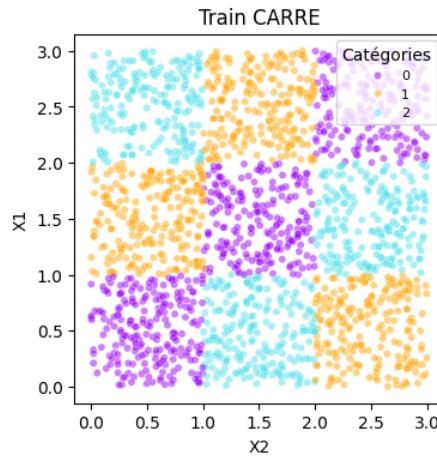


FIGURE 10 – Source : Travail personnel

SVM

Paramètres du modèle :

- Type de SVM : Classification C-SVM
- Noyau de SVM : Linéaire
- Coût : 1 (paramètre de pénalité)

Voici la sortie `summary(model_SVM)`

```
Call:
svm(formula = d ~ X1 + X2, data = sets$train_data, type = "C-classification",
     kernel = "linear")

Parameters:
  SVM-Type:  C-classification
  SVM-Kernel: linear
  cost:    1

Number of Support Vectors:  1880
( 626 634 620 )

Number of Classes:  3

Levels:
 0 1 2
```

Le nombre total de vecteurs de support est de 1880, répartis comme suit : pour la classe 0, il y a 626 vecteurs, pour la classe 1, 634 vecteurs, et pour la classe 2, 620

vecteurs. Bien que les classes soient équilibrées en termes de vecteurs de support sélectionnés, le modèle présente une complexité élevée, ce qui pourrait indiquer qu'il capture de nombreuses caractéristiques spécifiques des données d'entraînement, ou que le modèle n'est pas le plus approprié.

À continuation, une visualisation des hyperplans qui séparent l'ensemble de données dans les prédictions du modèle.

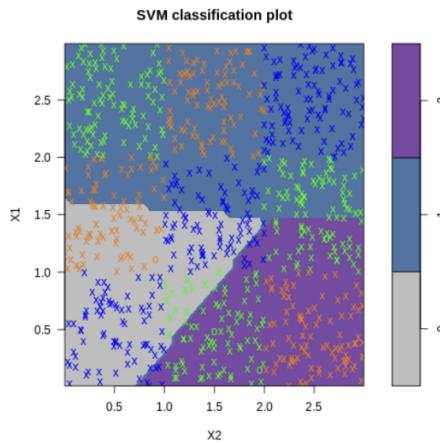


FIGURE 11 – Source : Travail personnel

PMC

La perte initiale est de 2728 et, après 100 itérations, la perte atteint 487, une diminution considérable de la perte. Le modèle a été entraîné avec un réseau neuronal comportant 2 entrées, 14 neurones cachés et 3 sorties. Voici un résumé du modèle :

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec une couche cachée
Architecture		2 – 14 – 3
Fonction d'entrée	$h(i)$	Identité – Affine – Affine
Fonction d'Activation	$e(i)$	Identité - Sigmoïde – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.001
Époques		<code>maxit</code> = 1000 ou convergence d'Erreur
Ajustement des Hyperparamètres		# de neurones dans la couche cachée et taux d'apprentissage
Logiciel		R - nnet

Voici la visualisation de l'architecture en utilisant `plotnet(model_PMC)` :

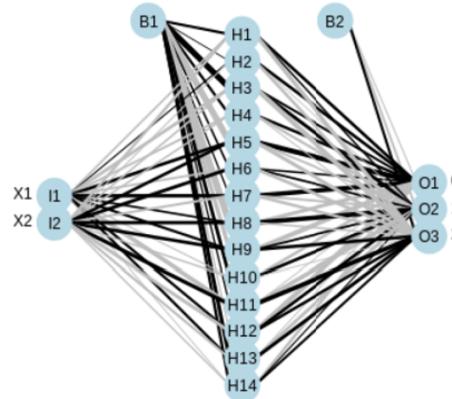


FIGURE 12 – Source : Travail personnel

Comparaison des résultats sur le corpus de validation

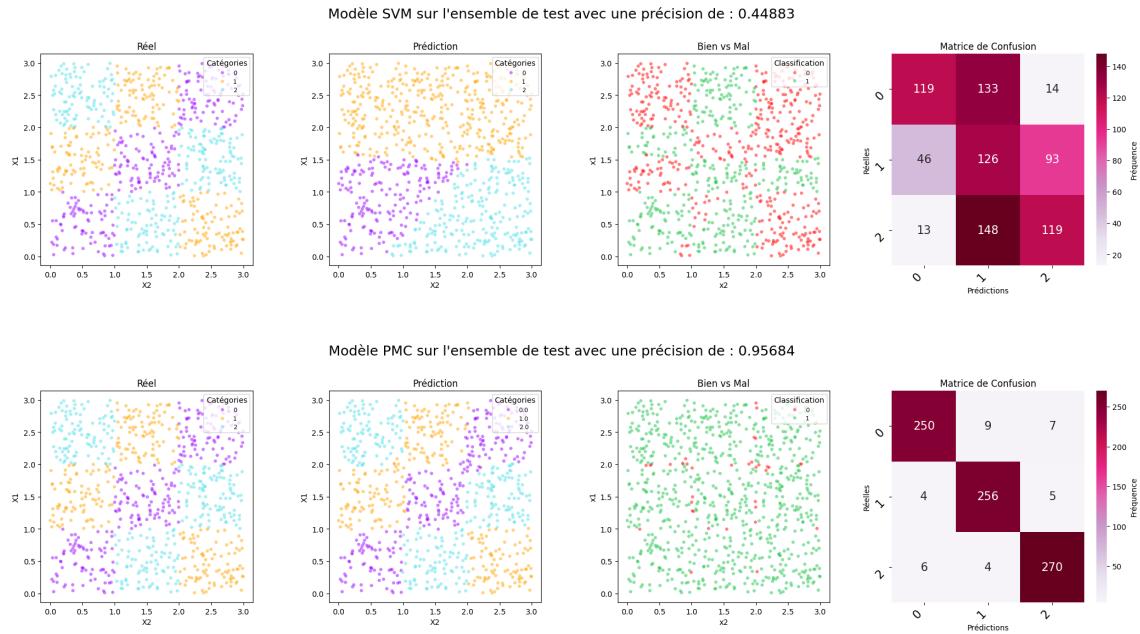


FIGURE 13 – Source : Travail personnel

La précision du SVM de 0,45 sur l'ensemble Carré s'explique par le fait que cet ensemble n'est pas linéairement séparable. En revanche, le PMC peut le séparer avec une précision de 0,95 en augmentant le nombre de neurones dans la couche cachée de 3 à 14, ce qui démontre sa capacité de généralisation et d'adaptation à de nouveaux corpus.

3 Démonstration des réseaux de neurones avec Keras

3.1 Corpus SPIRALE

Simulation de $n = 4000$ exemples à partir de coordonnées polaires, avec 2000 exemples dans chaque classe.

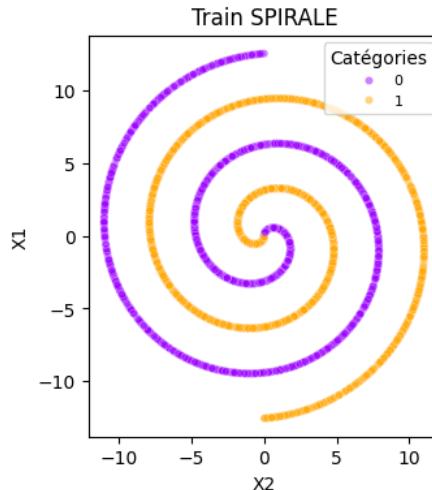


FIGURE 14 – Source : Travail personnel

3.2 SVM

Dans ce cas, le modèle SVM est un classificateur avec un noyau linéaire et un paramètre de coût (C) égal à 1, travaillant avec deux classes (0 et 1).

Voici la sortie `summary(model_SVM)`

```
Call:
svm(formula = d ~ X1 + X2, data = sets$train_data, type = "C-classification",
     kernel = "linear")

Parameters:
  SVM-Type: C-classification
  SVM-Kernel: linear
  cost: 1

Number of Support Vectors: 2536
( 1268 1268 )

Number of Classes: 2

Levels:
 0 1
```

Le nombre total de vecteurs de support est de 2536, répartis de la manière suivante : 1268 vecteurs de support pour la classe 0 et 1268 vecteurs de support pour la classe 1. Ce modèle présente un nombre notable de vecteurs de support, ce qui pourrait indiquer le modèle utilisé n'est pas adapté aux données.

À continuation, une visualisation des hyperplans qui séparent l'ensemble de données dans les prédictions du modèle.

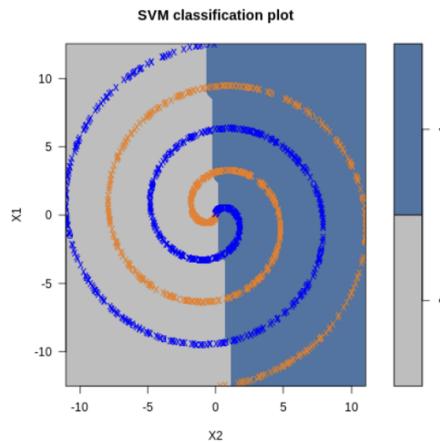


FIGURE 15 – Source : Travail personnel

3.3 PMC avec 1 couche caché

La perte initiale est de 2157 et, après 100 itérations, la perte atteint 1340, ce qui indique une diminution très lente de la perte. Cela suggère que le comportement du modèle sur l'ensemble d'entraînement n'est pas optimal. Voici un résumé du réseau implémenté.

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec une couche cachée
Architecture		2 – 20 – 3
Fonction d'entrée	$h(i)$	Identité – Affine – Affine
Fonction d'Activation	$e(i)$	Identité - Sigmoïde – Sigmoïde
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.001
Époques		maxit = 1000 ou convergence d'Erreur
Ajustement des Hyperparamètres		# de neurones dans la couche cachée et taux d'apprentissage
Logiciel		R - nnet

Voici la visualisation de l'architecture en utilisant `plotnet(model_PMC)` :

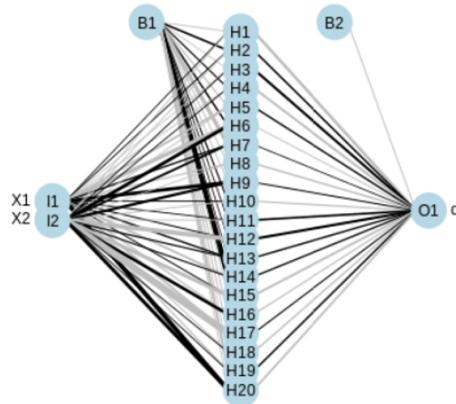


FIGURE 16 – Source : Travail personnel

3.4 PMC avec 2 couches cachées

Étant donné que la bibliothèque `nnet` est conçue pour des modèles PMC avec une seule couche cachée, la bibliothèque `Keras` de Python a été utilisée. Cette bibliothèque permet non seulement d'ajouter des couches supplémentaires, mais aussi de contrôler la fonction d'activation de chaque couche, ainsi que la fonction de perte à optimiser et l'algorithme d'optimisation.

Pour utiliser cette bibliothèque, il est nécessaire de standardiser les données. Dans ce cas, la méthode `sklearn.preprocessing` a été utilisée. Voici un résumé de l'architecture implémentée :

Caractéristiques	Notation	Valeur
Type de réseau		PMC avec deux couches cachées
Architecture		2 – 20 – 20 – 1
Fonction d'entrée	$h(i)$	Identité – Afine – Affine – Affine
Fonction d'Activation	$e(i)$	Identité – ReLu – ReLu – Sigmoïde
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Adam
Taux d'Apprentissage	λ	Valeur initiale par défaut (0.001)
Époques		epochs=50, batch_size=10
Ajustement des Hyperparamètres		Non
Logiciel		Python - Keras

En utilisant `plot_model(model, show_shapes=True, show_layer_names=True, dpi=100)`, voici la sortie fournie par Keras pour résumer l'architecture implémentée :

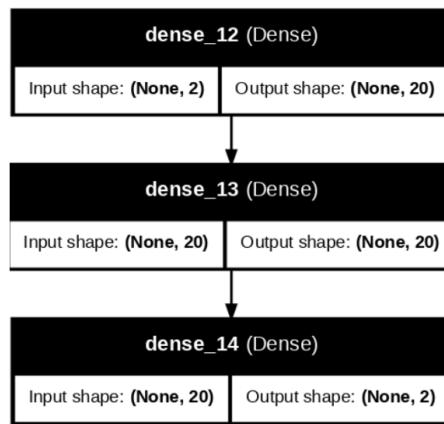


FIGURE 17 – Source : Travail personnel

3.5 Comparaison des résultats sur le corpus de validation

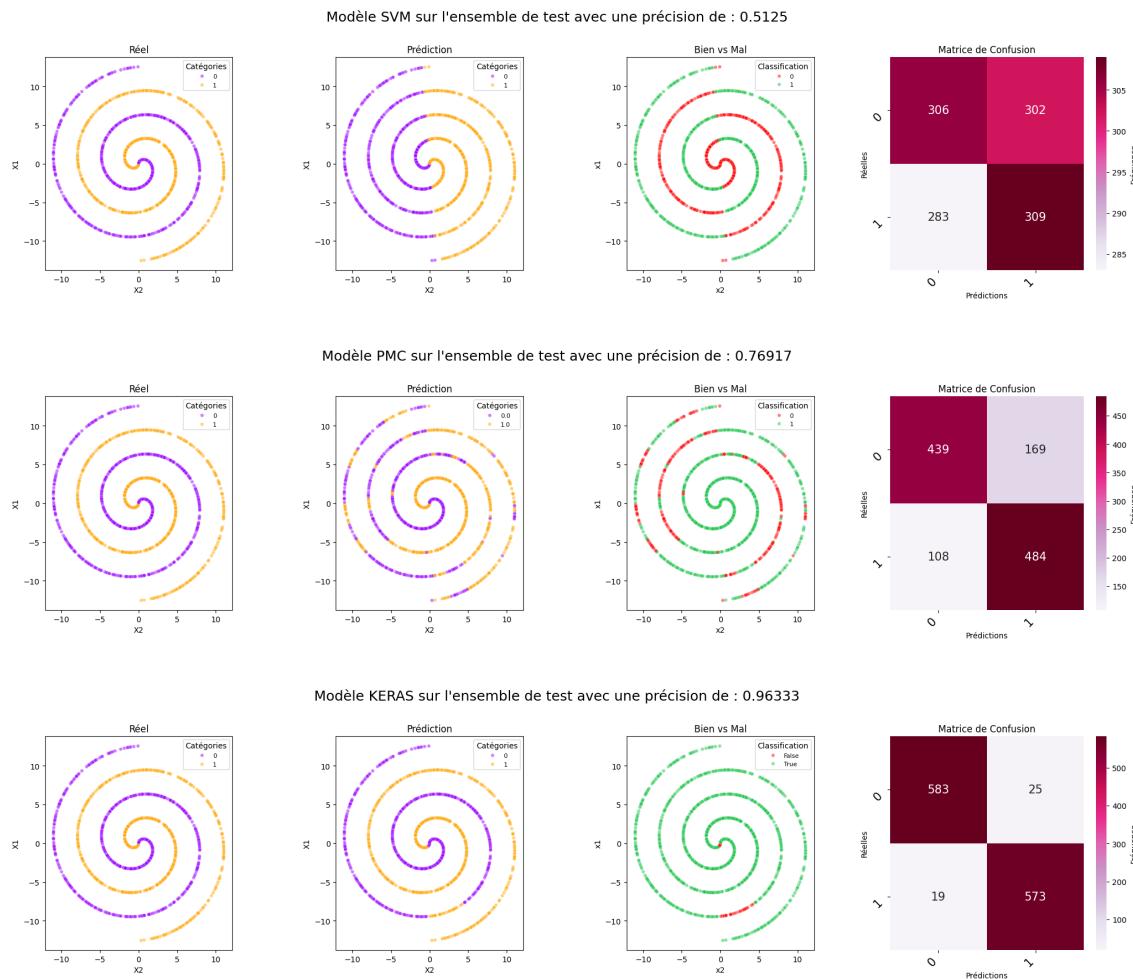


FIGURE 18 – Source : Travail personnel

Bien que la spirale challenge la précision du PMC avec une couche cachée (0.77), le PMC avec 2 couches cachées et activation ReLU est suffisant pour effectuer une bonne séparation de cet ensemble avec une précision de 0.96. Pour les SVM, il est évident que la spirale n'est pas linéairement séparable non plus.

4 Cartes auto-organisées de Kohonen - Self-Organizing Maps (SOM)

4.1 Definition

- Réseau de neurones non supervisé utilisé pour analyser la répartition des données d'entrée V et réduire leur dimensionnalité.
- Le réseau est constitué d'une grille A de noeuds ou neurones. Chaque neurone $r \in A$ est associé à un vecteur référent $w(r)$, responsable de représenter une zone dans V , définie par :

$$R_r = \{v \in V : \|v - w(r)\| \leq \|v - w(r')\|, \forall r' \in A\}.$$

- La carte A associe à chaque vecteur d'entrée $v \in V$ un neurone $r \in A$, tel que le vecteur référent $w(r)$ soit le plus proche de v :

$$r = \phi_w(v) = \arg \min_{r' \in A} \|v - w(r')\|.$$

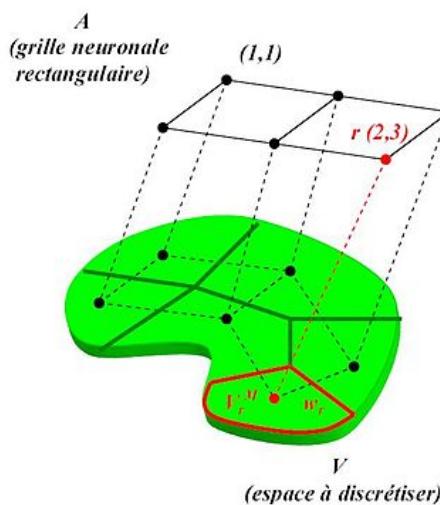


FIGURE 19 – Architecture d'une carte auto-organisée de Kohonen (SOM). Source : [13]

Algorithme d'apprentissage : Adapter les vecteurs référents à l'espace d'entrée.

Entrées :

- Données V , taille et forme de la carte A .
- Initialisation aléatoire $\{w_r(0) : r \in A\}$ des vecteurs référents.
- Nombre d'itérations T_{\max} , paramètre d'apprentissage initial ϵ_0 , paramètre de voisinage initial σ_0 .

Soit $\{w_r(t) : r \in A\}$ l'état des vecteurs référents à l'instant t .

Cycle d'adaptation : Pour $t \geq 0$, exécuter :

- Sélectionner un vecteur $v \in V$ aléatoirement.

- Chercher le neurone gagnant $r = \phi_w(v)$.
- Mise à jour des vecteurs référents :

$$w_{t+1}(s) = w_t(s) + \epsilon(t)h_t(s, r)(w_t(s) - v)$$

où :

- $\epsilon(t) = \epsilon_0 \exp\left(\frac{-t}{T_{\max}}\right)$ est le pas d'apprentissage. Sa valeur diminue au fil des itérations.
- $h_t(s, r) = \exp\left(\frac{-d^2(s, r)}{2\sigma^2(t)}\right)$ est la fonction de voisinage Gausienne, décrivant comment les neurones proches du vainqueur r sont entraînés.
- $d(s, r)$ est la distance entre les neurones s, r sur la carte A .
- $\sigma(t) = \sigma_0 \exp\left(\frac{-t}{T_{\max}}\right)$ est le coefficient de voisinage.

4.2 Implémentation en R

4.2.1 Cartographie de la mixture gaussienne

La carte de Kohonen a été appliquée à une mixture gaussienne (sur jeu d'entraînement) dans \mathbb{R}^2 , comprenant 3 classes et entraînée avec 15 vecteurs référents. La configuration de la carte est présentée en utilisant la bibliothèque **Kohonen** de R :

Carte de neurones : La carte de neurones est composée de 5 rangées et 3 colonnes. La topologie choisie est hexagonale. La fonction de voisinage utilisée est gaussienne.

Entraînement du modèle : Le modèle a été entraîné avec les paramètres suivants :

- Nombre d'itérations (rlen) : 1000
- Taux d'apprentissage adaptatif (alpha) : de 0.05 à 0.01

Pour une meilleure compréhension, le jeu d'entraînement et les vecteurs de référence sont analysés à l'aide des visualisations suivantes :

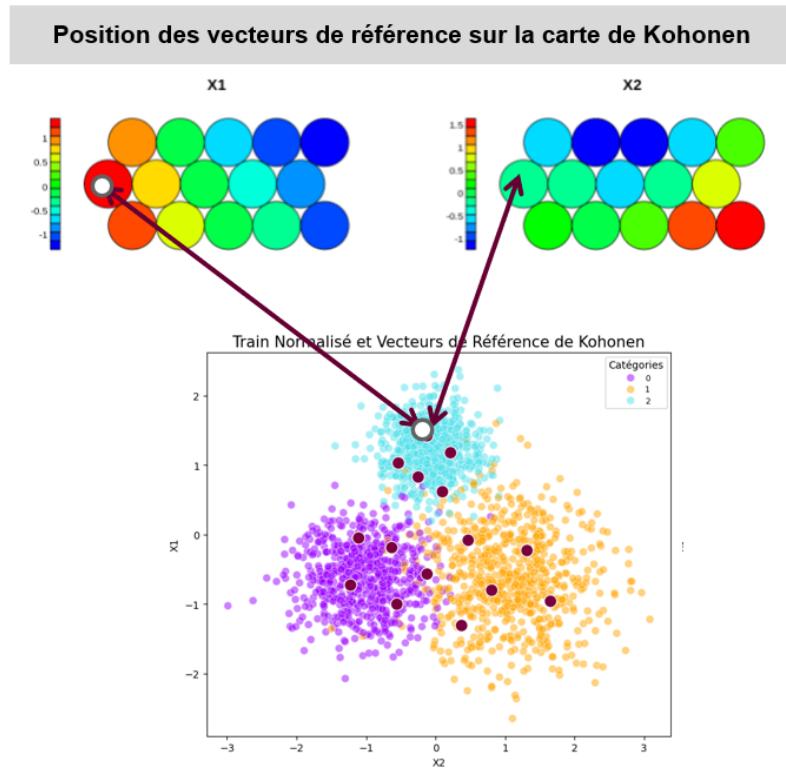


FIGURE 20 – Source : Travail personnel

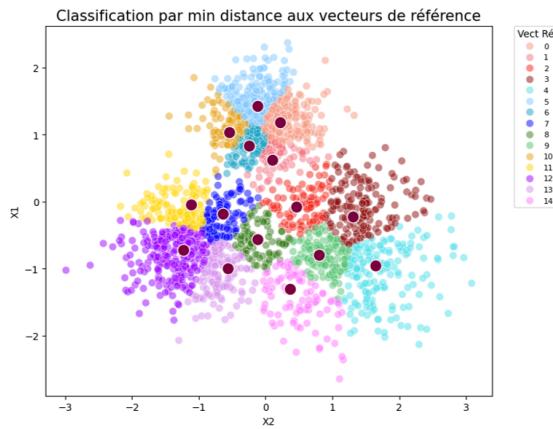


FIGURE 21 – Source : Travail personnel

- **Discrétisation de l'espace :** La carte de Kohonen a divisé l'espace \mathbb{R}^2 en 15 zones correspondant aux vecteurs référents. Chaque zone représente une approximation locale de la distribution des données dans cet espace.
- **Représentation des classes :** Les vecteurs référents se sont ajustés aux distributions des 3 classes de la mixture gaussienne, permettant une séparation visuelle des groupes de données dans l'espace.
- **Qualité de l'adaptation :** La position des vecteurs référents reflète les caractéristiques des données (moyennes et étendues des classes). Cependant, pour des classes proches dans l'espace, certains vecteurs référents peuvent se retrouver à la frontière.

4.2.2 Réseaux Radial basis Function RBF

Les vecteurs référents de la carte de Kohonen sont utilisés comme centres des neurones dans la couche cachée d'un réseau de fonctions de base radiale (RBF) avec activation gaussienne pour la classification supervisée. Cette approche permet de tirer parti de la représentation discrétisée de l'espace par la carte de Kohonen pour initialiser efficacement le modèle de classification.

Voici la programmation de la couche cachée à la couche de sortie en NumPy Python :

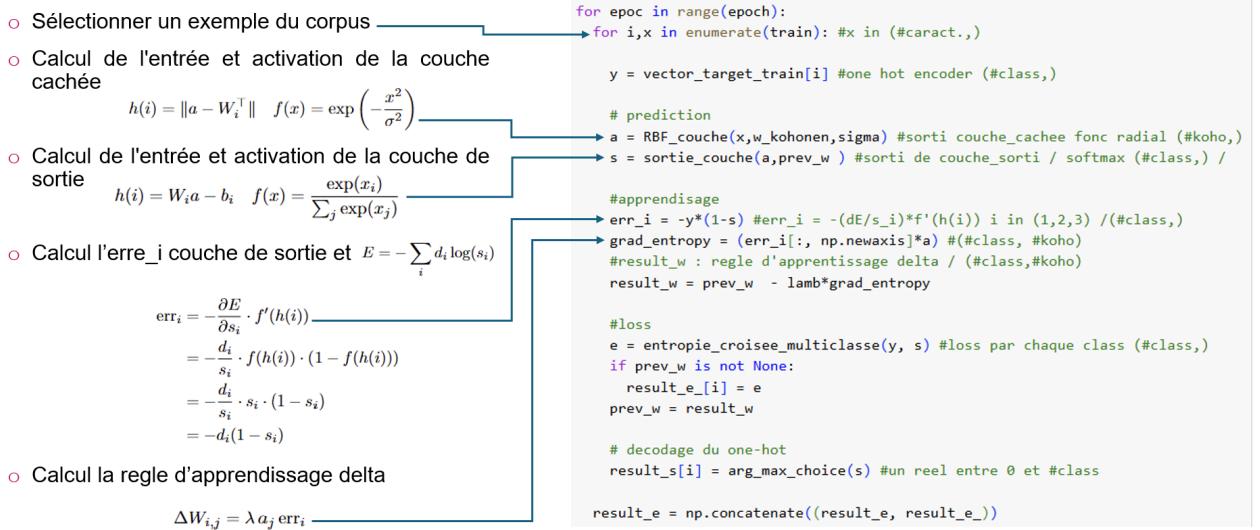


FIGURE 22 – Source : Travail personnel

Caractéristiques	Notation	Valeur
Type de réseau		RBF avec une couche cachée
Architecture		2 – 15 – 3
Fonction d'entrée	$h(i)$	Identité – Distance – Affine
Fonction d'Activation	$e(i)$	Standardisation – Gaussienne – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Itération règle du delta
Taux d'Apprentissage	λ	0.001
Époques		2
Ajustement des Hyperparamètres		Non
Logiciel		<ul style="list-style-type: none"> Couche cachée : R – kohonen Couche Sortie : Python - NumPy

Résultats sur le corpus de validation

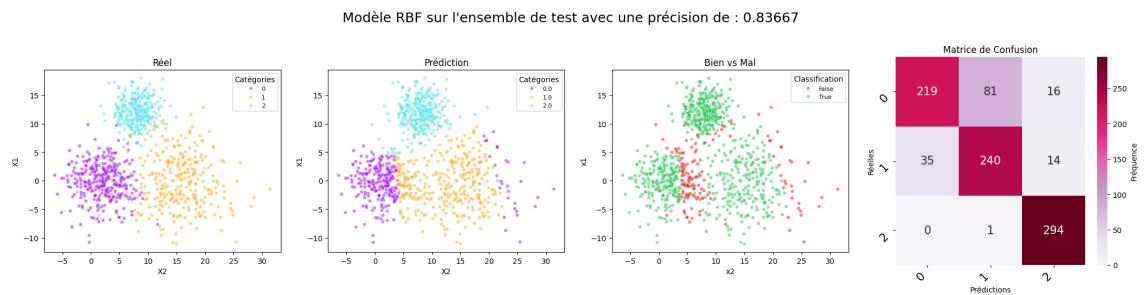


FIGURE 23 – Source : Travail personnel

La précision de la RBF, de 0.83, est bonne mais reste inférieure à celle du PMC avec une couche cachée et des SVM.

5 Machine de Boltzmann Restreinte (RBM)

Une Machine de Boltzmann Restreinte (RBM) est un modèle de réseau de neurones pour l'apprentissage non supervisé. Il est composé de deux couches :

- **Couche visible $\mathbf{x} = (x_1, x_2, \dots, x_N)$** : représente les données d'entrée (par exemple, une image ou un vecteur de caractéristiques).
- **Couche cachée $\mathbf{h} = (h_1, h_2, \dots, h_M)$** : apprend des représentations latentes des données visibles.

Les neurones visibles sont connectés aux neurones cachés, sans connexion au sein de chaque couche. L'objectif principal d'une RBM est de modéliser les caractéristiques des données.

Les RBM sont utilisées pour :

- **Réduction de dimension** : extraction de caractéristiques.
- **Initialisation** des poids dans les réseaux profonds.
- **Modélisation générative** : reconstruction et génération de données.

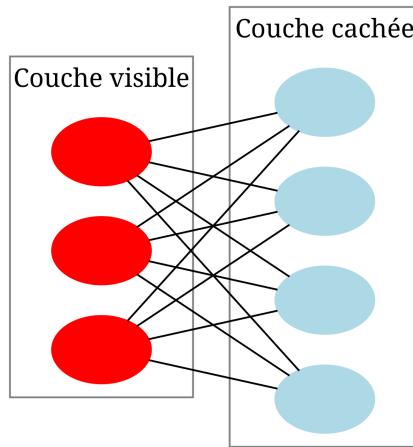


FIGURE 24 – Architecture RBM restreinte. Source : [18]

5.1 Fonction d'énergie et probabilités

L'énergie associée à un état (\mathbf{x}, \mathbf{h}) est définie par :

$$E(\mathbf{x}, \mathbf{h}) = - \sum_{i,j} W_{ij} x_i h_j - \sum_i b_i x_i - \sum_j c_j h_j$$

où W_{ij} est le poids entre x_i et h_j , et b_i , c_j sont les biais des neurones visibles et cachés.

La probabilité d'un état (\mathbf{x}, \mathbf{h}) est donnée par :

$$p(\mathbf{x}, \mathbf{h}) = \frac{\exp(-E(\mathbf{x}, \mathbf{h}))}{Z}, \quad Z = \sum_{\mathbf{x}, \mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h}))$$

Les probabilités conditionnelles :

$$p(h_j = 1|\mathbf{x}) = \sigma \left(\sum_i W_{ij} x_i + c_j \right), \quad p(x_i = 1|\mathbf{h}) = \sigma \left(\sum_j W_{ij} h_j + b_i \right)$$

où $\sigma(x) = \frac{1}{1+\exp(-x)}$ est la fonction sigmoïde.

5.2 Apprentissage avec la divergence contrastive

L'apprentissage dans une RBM vise à maximiser la vraisemblance des données d'entrée \mathbf{x} , définie comme :

$$P(\mathbf{x}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h}),$$

La maximisation de cette vraisemblance, équivalente à minimiser l'énergie du système, est approximée par la méthode de *Contrastive Divergence (CD)* avec les étapes suivantes :

1. **Initialisation** : Les paramètres W , b_i , et c_j sont initialisés aléatoirement ou selon une distribution gaussienne.
2. **Étape positive** : Pour chaque exemple \mathbf{x} du corpus :
 - Calculer $p(h_j = 1|\mathbf{x})$, les probabilités conditionnelles.
 - Échantillonner $\mathbf{h} \sim p(\mathbf{h}|\mathbf{x})$.
 - Obtenir $\langle x_i h_j \rangle_{\text{data}}$, l'attente des produits croisés sous les données.
3. **Étape négative** :
 - Générer $\mathbf{x}' \sim p(\mathbf{x}|\mathbf{h})$.
 - Calculer $\mathbf{h}' \sim p(\mathbf{h}|\mathbf{x}')$.
 - Obtenir $\langle x_i h_j \rangle_{\text{model}}$, l'attente des produits croisés sous le modèle.
4. **Mise à jour** :

$$\begin{aligned} \Delta W_{ij} &= \epsilon (\langle x_i h_j \rangle_{\text{data}} - \langle x_i h_j \rangle_{\text{model}}), \\ \Delta b_i &= \epsilon (\langle x_i \rangle_{\text{data}} - \langle x_i \rangle_{\text{model}}), \quad \Delta c_j = \epsilon (\langle h_j \rangle_{\text{data}} - \langle h_j \rangle_{\text{model}}), \end{aligned}$$

où ϵ est le taux d'apprentissage.

6 Démonstration logicielle pour un modèle de réseau profond (DNN)

Dans le cadre de la classification supervisée de la mixture gaussienne avec 3 classes en \mathbb{R}^2 , nous étendons le RBF de l'exercice 4 :

- Une couche cachée initiale RBF avec activation gaussienne basée sur la carte de Kohonen pour capturer les caractéristiques des données et fonction d'activation gaussienne.
- Un PMC avec deux couches denses activées par ReLU (couche cachée) et Softmax (couche de sortie).

6.1 Architecture

Ce modèle est un réseau profond (DNN) car :

- Il comporte plusieurs couches hiérarchiques : une couche Kohonen suivie de deux couches denses.
- Il extrait des représentations complexes grâce aux couches denses après un prétraitement initial.
- Il applique une propagation multi-étapes pour apprendre des niveaux d'abstraction successifs.

Caractéristiques	Notation	Valeur
Type de réseau		DNN avec deux couches cachées
Architecture		2 – 15 – 7 – 3
Fonction d'entrée	$h(i)$	Identité – Distance – Affine – Affine
Fonction d'Activation	$e(i)$	Standardisation – Gaussienne – ReLu – Softmax
Fonction d'Erreur	$E(s, d)$	Entropie croisée
Algorithme d'Optimisation		Adam
Taux d'Apprentissage	λ	Valeur initiale par défaut (0.001)
Époques		epochs=50, batch_size=10
Ajustement des Hyperparamètres		Non
Logiciel		<ul style="list-style-type: none"> ◦ Première couche cachée : R – kohonen ◦ Deuxième couche cachée et couche sortie: Python - Keras

6.2 Résultats sur le corpus de validation

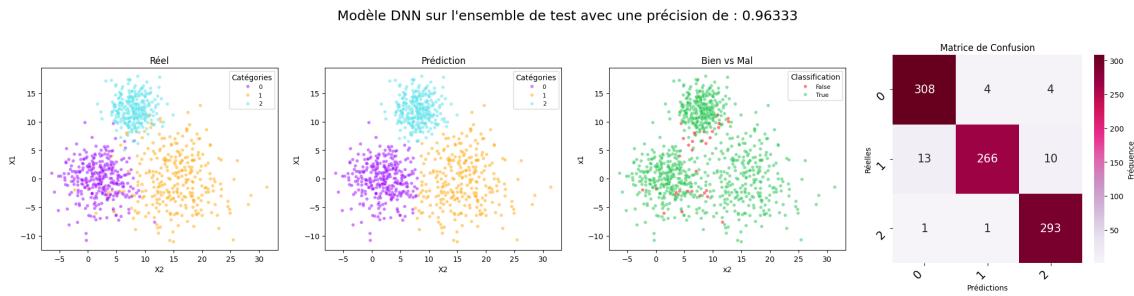


FIGURE 25 – Source : Travail personnel

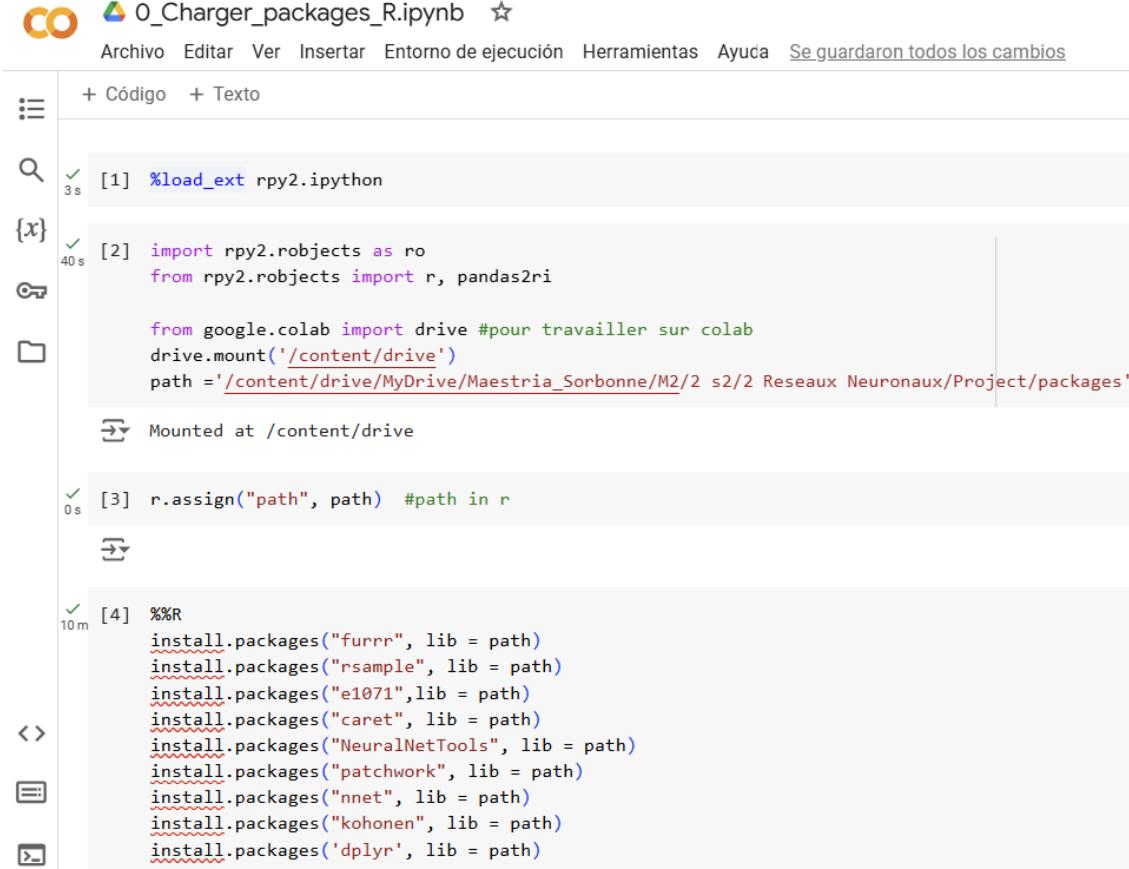
Ce modèle de DNN a une précision similaire à celle du PMC avec 2 couches cachées, implémenté avec Keras.

7 Conclusions

- Les deux modèles, SVM et PMC, obtiennent de très bonnes performances avec des précisions de 0,95 et 0,96 respectivement sur la mixture gaussienne, qui est un ensemble linéairement séparable.
- La précision du SVM de 0,45 sur l'ensemble Carré s'explique par le fait que cet ensemble n'est pas linéairement séparable. En revanche, le PMC peut le séparer avec une précision de 0,95 en augmentant le nombre de neurones dans la couche cachée de 3 à 14, ce qui démontre sa capacité de généralisation et d'adaptation à de nouveaux corpus.
- Bien que la spirale challenge la précision du PMC avec une couche cachée (0,77), le PMC avec 2 couches cachées et activation ReLU est suffisant pour effectuer une bonne séparation de cet ensemble avec une précision de 0,96. Pour les SVM, il est évident que la spirale n'est pas linéairement séparable non plus.
- Le PMC à une seule couche a montré un bon comportement sur les trois ensembles de données, montrant que les réseaux neuronaux peuvent s'adapter à différents ensembles de données, avec moins de restrictions que les autres modèles.
- La méthode de Kohonen est un outil puissant pour organiser des données ou réduire leur dimensionnalité, mais lorsqu'elle est implémentée dans le réseau RBF, elle n'a pas donné de résultats intéressants sur l'ensemble le plus simple (classification de 3 gaussiennes).
- Keras est une bibliothèque assez versatile pour la conception et le test de différentes architectures de réseaux neuronaux.

8 Code et Algorithmes Implémentés

Le code a été implémenté sur le service de Google appelé **Google Colaboratory** (ou **Colab**). Le code est hébergé sur Google Drive. Cette plateforme a été choisie en raison de sa capacité à permettre un travail fluide avec les langages R et Python dans un même environnement, facilitant ainsi la transition entre les deux langages. Pour travailler avec R, il est nécessaire de télécharger les bibliothèques correspondantes dans un dossier spécifique, comme cela est montré ci-dessous :



The screenshot shows a Google Colaboratory notebook titled "0_Charger_packages_R.ipynb". The code cell [2] contains the following R code:

```
[2] library(rpy2)
rpy2.robjects import r, pandas2ri
from google.colab import drive #pour travailler sur colab
drive.mount('/content/drive')
path = '/content/drive/MyDrive/Maestria_Sorbonne/M2/2 s2/2 Reseaux Neuronaux/Project/packages'
```

The code cell [3] contains:

```
[3] r.assign("path", path) #path in r
```

The code cell [4] contains:

```
[4] %%R
install.packages("furrr", lib = path)
install.packages("rsample", lib = path)
install.packages("e1071", lib = path)
install.packages("caret", lib = path)
install.packages("NeuralNetTools", lib = path)
install.packages("patchwork", lib = path)
install.packages("nnet", lib = path)
install.packages("kohonen", lib = path)
install.packages('dplyr', lib = path)
```

En ce qui concerne l'exportation du code au format PDF, cela n'est pas possible directement depuis Colab. Par conséquent, le code a été téléchargé au format .ipynb, puis exporté en Python au format PDF depuis Jupyter. Il se trouve à la fin de ce document en annexe.

9 Résumé "Analyzing Open-Source Chip - Design Ecosystem from an Environmental Sustainability Perspective"

9.1 Introduction

9.1.1 Contexte

Le secteur des TIC représente une part croissante des émissions mondiales de gaz à effet de serre (GES), entre 2 % et 4 %. Cette situation s'inscrit dans un contexte où plusieurs limites planétaires ont été dépassées.

- Les TIC contribuent significativement aux émissions globales de GES.
- Nécessité d'une réflexion sur leur durabilité dans un cadre planétaire.

9.1.2 Objectif

Ce rapport vise à explorer comment les innovations open-source peuvent répondre aux défis environnementaux liés à la durabilité des TIC.

- Analyser les impacts directs, indirects et structurels des TIC.
- Proposer des stratégies pour une transition technologique durable.

9.2 Analyse des Émissions de GES des TIC

9.2.1 Émissions Directes et Réduction Potentielle

Les TIC pourraient réduire les émissions d'autres secteurs :

- Une tonne de CO₂e émise par le secteur mobile pourrait éviter 10 tonnes de CO₂e dans d'autres secteurs.
- La numérisation pourrait réduire les émissions dans d'autres secteurs jusqu'à 20 % d'ici 2030.

Cependant, les études existantes présentent des limites méthodologiques (biais économique, hypothèses peu transparentes). L'effet rebond, qui peut être défini comme « une meilleure efficacité dans le processus de production d'un produit diminue les coûts énergétiques par unité produite, ce qui augmente la demande pour ce produit et donc la consommation totale », doit être pris en compte.

- Les gains d'efficacité énergétique ne suffisent pas à eux seuls.
- Importance d'intégrer des mécanismes de régulation pour éviter l'effet rebond.

9.2.2 Lois Empiriques et Limites Physiques

Bien que la loi de Moore et d'autres tendances technologiques améliorent l'efficacité énergétique, ces progrès ne suffisent pas à réduire l'empreinte totale des TIC.

- **La loi de Moore** : Le nombre de transistors par puce double environ tous les deux ans, améliorant l'efficacité mais augmentant aussi la complexité technologique.
- **La loi de Cooper** : Depuis 1900, la capacité de communication sans fil double tous les 2,5 ans.
- Les gains d'efficacité sont limités par des contraintes physiques (taille des atomes).
- Les coûts de conception des puces augmentent avec la complexité technologique.

Ainsi, par exemple, l'empreinte carbone des réseaux mobiles Internet a augmenté depuis la mise en œuvre de ces nouvelles tendances, car l'augmentation de l'utilisation des technologies compense largement les gains d'efficacité issus des innovations TIC.

9.3 Critique du Concept de "Croissance Verte"

9.3.1 Impossibilité de Découplage Absolu

Il n'existe aucune preuve de la "croissance verte" : un découplage absolu entre la croissance économique et l'empreinte écologique.

- L'empreinte écologique reste fortement liée à la croissance économique.
- La foi dans le progrès technologique et l'économie financière alimente la course à l'innovation.
- La course à l'innovation améliore l'efficacité des coûts, ce qui entraîne un effet rebond observé.
- Un découplage absolu n'a jamais été observé à grande échelle, ni dans les indicateurs socioéconomiques ni dans les indicateurs des systèmes terrestres.

9.3.2 Effets Structurels

Les stratégies comme l'obsolescence programmée augmentent l'impact environnemental global et nécessitent une régulation stricte.

- Obsolescence programmée : Produire intentionnellement des biens avec une durée de vie limitée pour encourager des achats répétés.
- Création de besoins artificiels accentuant la consommation.
- Réallocation des investissements et des ressources humaines vers des secteurs moins prioritaires pour la durabilité.

9.4 Écosystème de Conception Open-Source de Puces

Développement de puces électroniques accessibles via des outils open-source : outils de conception électronique assistée et bibliothèques.

9.4.1 Innovations Open-Source

- Transparence accrue dans les résultats de recherche.
- Conception au niveau système facilitée grâce à des blocs IP open-source.
- Démocratisation de la conception de puces avec des outils EDA et des PDK open-source.
- Création de start-ups.
- Réduction des coûts d'accès aux blocs IP et aux outils pour les petites entreprises.
- Augmentation du nombre d'ingénieurs disponibles pour le recrutement en attirant davantage d'étudiants dans la conception de puces.
- Augmentation de la demande de production de puces provenant de nouveaux produits et de puces plus grandes (avec davantage de blocs IP).

9.4.2 Impacts Environnementaux

- **Avantages** : Amélioration de l'efficacité énergétique, réduction de l'utilisation de silicium.
- **Inconvénients** : Risques d'augmentation des volumes de production et des applications inutiles.

9.5 Transition Socio-Écologique

Concevoir un modèle socio-technique qui équilibre le bien-être humain (plancher social) avec les limites écologiques (plafond écologique), afin d'éviter l'effondrement tel que défini par le Prof. Jared Diamond :

« Une diminution drastique de la taille de la population humaine et/ou de la complexité politique, économique et sociale, sur une zone considérable et pendant une longue période. »

9.5.1 Propositions

- Passer d'une simple recherche d'efficacité à une sobriété post-croissance.
- Restreindre les innovations aux applications ayant un bénéfice socio-écologique démontré.
- Développer une gouvernance participative pour établir un consensus sur les priorités technologiques.
- Intégrer ces principes dans les programmes éducatifs et les priorités de recherche.

9.5.2 Défis

Les comportements et valeurs culturels doivent évoluer pour éviter un effondrement écologique et économique global.

- Adoption de nouvelles valeurs éthiques adaptées à la durabilité.
- Nécessité d'actions rapides et coordonnées à l'échelle mondiale.

9.6 Conclusion et Question Ouverte

9.6.1 Résumé

La poursuite d'innovations technologiques sans restriction contribue à dépasser les limites planétaires. Une transition vers une sobriété technologique, combinée à des innovations ciblées à l'aide des innovations Open-Source, est essentielle.

- Les innovations doivent être évaluées pour leur impact socio-écologique.
- Importance de combiner efficacité technologique et réduction de l'usage inutile.
- Intégrer une perspective éthique et durable dans toutes les phases de la conception technologique.
- Développer des mécanismes participatifs pour guider les choix technologiques.
- Impliquer toutes les parties prenantes dans une gouvernance équitable et transparente.
- Encourager la coopération internationale pour des règles communes sur l'évaluation des technologies.

9.6.2 Question Ouverte

Comment parvenir à un consensus démocratique et informé sur les applications technologiques significatives ?

Références

- [1] Erwan Scornet. *Diapositives du cours de Machine Learning*. Sorbonne Université, 2024-2025.
- [2] Ismaël Castillo. *Cours de Machine Learning*. Sorbonne Université, suivi en 2024-2025.
- [3] Support Vector Machine. *ResearchGate*. 2025. Disponible à https://www.researchgate.net/figure/Support-vector-machine_fig1_383924619. Consulté : 2025-01-05.
- [4] Multiclass classification hyper-planes. *ResearchGate*. 2025. Disponible à https://www.researchgate.net/figure/Multiclass-classification-hyper-planes_fig4_373015698. Consulté : 2025-01-05.
- [5] Margaux Brégère. *Diapositives du cours Apprentissage séquentiel pour la prévision de séries temporelles*. Sorbonne Université, 2024-2025.
- [6] Annick Valibouze. *Polycopié de Réseaux Neuronaux*. Sorbonne Université, 2024-2025.
- [7] TensorFlow. *Keras : The Python deep learning library*. 2015. Disponible à <https://www.tensorflow.org/keras>. Consulté : 2025-01-05.
- [8] Brian D. Ripley. *nnet : Feed-Forward Neural Networks and Multinomial Log-Linear Models*. 1996. Disponible à <https://cran.r-project.org/web/packages/nnet/index.html>. Consulté : 2025-01-05.
- [9] D. Meyer, G. Dimitriadou, K. Hornik, F. Weingessel, and A. Leisch. *e1071 : Misc Functions of the Department of Statistics, Probability Theory Group (Formerly : E1071)*, TU Wien. 2015. Disponible à <https://cran.r-project.org/web/packages/e1071/index.html>. Consulté : 2025-01-05.
- [10] Ricco Rakotomalala. *Cartes auto-organisatrices de Kohonen*. Notes du cours Description et classification automatique, Université Lumière Lyon 2. Disponible à https://eric.univ-lyon2.fr/ricco/cours/slides/kohonen_som.pdf. Consulté : 2025-01-05.
- [11] W. N. Venables and B. D. Ripley. *kohonen : Self-Organizing Maps*. 2001. Disponible à <https://cran.r-project.org/web/packages/kohonen/index.html>. Consulté : 2025-01-05.
- [12] Meritis. *Cartes topologiques de Kohonen*. Disponible à <https://meritis.fr/blog/cartes-topologiques-de-kohonen/>. Consulté : 2025-01-05.
- [13] Wikimedia Commons. *Architecture SOM*. Disponible à https://commons.wikimedia.org/wiki/File:Architecture_SOM.JPG. Consulté : 2025-01-05.
- [14] Hugo Larochelle. *Neural networks [5.1] : Restricted Boltzmann machine - definition*. 2013. Disponible à https://www.youtube.com/watch?v=p4Vh_zMw-HQ. Consulté : 2025-01-05.
- [15] Hugo Larochelle. *Neural networks [5.2] : Restricted Boltzmann machine - inference*. 2013. Disponible à https://www.youtube.com/watch?v=lekCh_i32iE. Consulté : 2025-01-05.
- [16] Hugo Larochelle. *Neural networks [5.3] : Restricted Boltzmann machine - free energy*. 2013. Disponible à https://www.youtube.com/watch?v=eOTs_7Y6hZU. Consulté : 2025-01-05.

- [17] Hugo Larochelle. *Neural networks [5.4] : Restricted Boltzmann machine - contrastive divergence.* 2013. Disponible à <https://www.youtube.com/watch?v=MD8qXWucJBY>. Consulté : 2025-01-05.
- [18] Wikimedia Commons. *Restricted Boltzmann machine (fr).* Disponible à https://commons.wikimedia.org/wiki/File:Restricted_Boltzmann_machine-fr.svg?lang=fr. Consulté : 2025-01-05.

10 Annexes