TP : Réseau de Neurones

Module 3 : Machine Learning

Dupuis Octave

19 décembre 2022

Problème I : Problème de classification par un ou plusieurs « Simple Linear Perceptron »

1

Pour ce problème de classification des 4 points du graphique, nous allons donner pour chaque point un couple (X,Y). Le paramètre X (resp. Y) est une variable binaire associant à tout point 1 si le point est au-dessus de l'axe des abscisses (resp. à droite de l'axe des ordonnées) et 0 sinon. Dès lors, la classification demandée correspond à l'implémentation d'un "OU exclusif" ("XOR") qui peut être traitée de cette façon via un réseau de neurones, avec les blocs "AND", "NOT" et "OR" correspondant à des perceptrons simples :

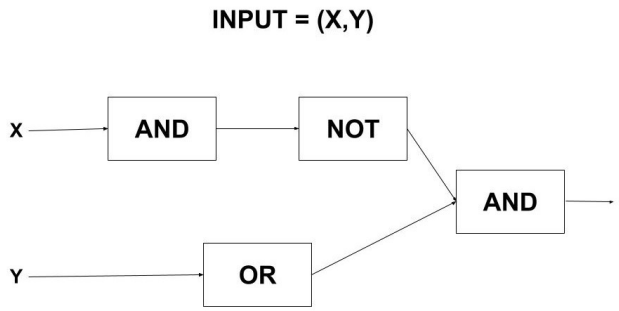


FIGURE 1 – Réseau de neurones avec 4 perceptrons simples

Sans utiliser des fonctions logiques, on pouvait tout simplement multiplier X et Y dans un perceptron simple, puis appliquer comme fonction d'activation la fonction -h , où h désigne la fonction d'activation "hardlim" :

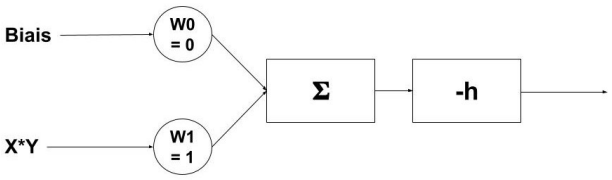


FIGURE 2 – Perceptron simple avec fonction d'activation linéaire

2

On va démontrer qu'un réseau de neurones avec plusieurs couches cachées dont toutes les fonctions d'activations sont linéaires est équivalent à un perceptron simple (e.g. sans couches cachées) ayant aussi une fonction d'activation linéaire par récurrence sur $n \in \mathbb{N}^*$ le nombre de couches cachées dans le réseau de neurones.

Initialisation : $n = 1$

Soit un réseau de neurones avec e entrée, une couche cachée de h neurones, et s neurones en sorties. On note $(w^0_{i,k})_{i\in[[0,e]],k\in[[1,h]]} \in \mathbb{R}^{(e+1)h}$ les poids associés à la couche d'entrée ($w^0_{0,k}$ correspond au biais), et $(w^1_k)_{k\in[[0,h]]} \in \mathbb{R}^{h+1}$ les poids associés à la couche cachée. Les deux fonctions d'activation associées sont linéaires de la forme : $\sigma_i : x \longmapsto A_i x + B_i$, avec $\forall i \in [[0,1]] : A_i, B_i \in \mathbb{R}$

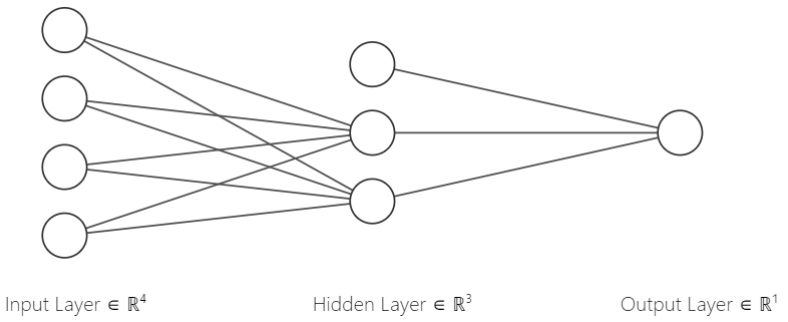


FIGURE 3 – Réseau de neurones avec $e = 3, h = 2$ et $s = 1$

Avec $x = (x_i)_{i\in[[1,e]]}$ l'entrée de notre réseau, et en notant a la sortie, on a :

$$a(x) = \sigma_1\left(w^1_0 + \sum_{k=1}^{h} w^1_k \sigma_0\left(w^0_{0,k} + \sum_{i=1}^{e} w^0_{i,k} x_i\right)\right)$$

On remplace les fonction σ_i :

$$a(x) = A_1\left(w^1_0 + \sum_{k=1}^{h} w^1_k\left(A_0\left(w^0_{0,k} + \sum_{i=1}^{e} w^0_{i,k} x_i\right) + B_0\right)\right) + B_1$$

On factorise l'expression par les x_i, ce qui nous donne :

$$a(x) = \underbrace{A_1}_{A_{eq}}\left(\sum_{i=1}^{e} \underbrace{\left(\sum_{k=1}^{h} w^1_k A_0 w^0_{i,k}\right)}_{w^{eq}_i} x_i + \underbrace{w^1_0}_{w^{eq}_0}\right) + \underbrace{A_1\left(w^1_0 + \sum_{k=1}^{h} w^1_k\left(A_0 w^0_{0,k} + B_0\right)\right) + B_1}_{B_{eq}}$$

En posant $\sigma_{eq} : x \longmapsto A_{eq}x + B_{eq}$, on a alors :

$$a(x) = \sigma_{eq}\left(w^{eq}_0 + \sum_{i=1}^{e} w^{eq}_i x_i\right) \tag{1}$$

Cette sortie correspond à un perceptron simple ayant une fonction d'activation linéaire, donc l'initialisation est validée.

Hérédité : En supposant qu'un réseau de neurones avec n couches cachées est équivalent à un perceptron, on s'intéresse à un réseau de neurones avec $n+1$ couches cachées. On va considérer le réseau sans sa sortie comme un réseau à n couches cachée (on considère la $n+1$ème couche cachée comme la sortie. On applique notre hypothèse de récurrence, ce qui nous donne un réseau de neurones équivalent à une couche cachée. On applique alors le cas $n = 1$ pour se ramener à un perceptron simple.

Conclusion : En vértue du principe de récurrence, on a montré qu'un réseau de neurones avec plusieurs couches cachées dont toutes les fonctions d'activations sont linéaires est équivalent à un perceptron simple (e.g. sans couches cachées) ayant aussi une fonction d'activation linéaire.

Problème II : Réseau de neurones « From Scratch »

Le code associé à cette question se trouve dans le fichier *Neural_Network_FS.py* (on a choisit de programmer notre réseau de neurones sur *Python*).

On implémente donc un réseau de neurones avec une couche cachée de 3 neurones et une fonction d'activation de type sigmoïde afin de résoudre un problème de classification. Chaque étape est détaillée, notamment les étapes de *FeedForward*, *Prédictions* et *BackPropagation*. Pour analiser les résultats, on représente la justesse de notre modèle (simplement le pourcentage de bonnes prédictions) à chaque pas de notre réseaux de neurones. On fait également varier le *learning rate*, ce qui nous donne la figure ci-dessous :

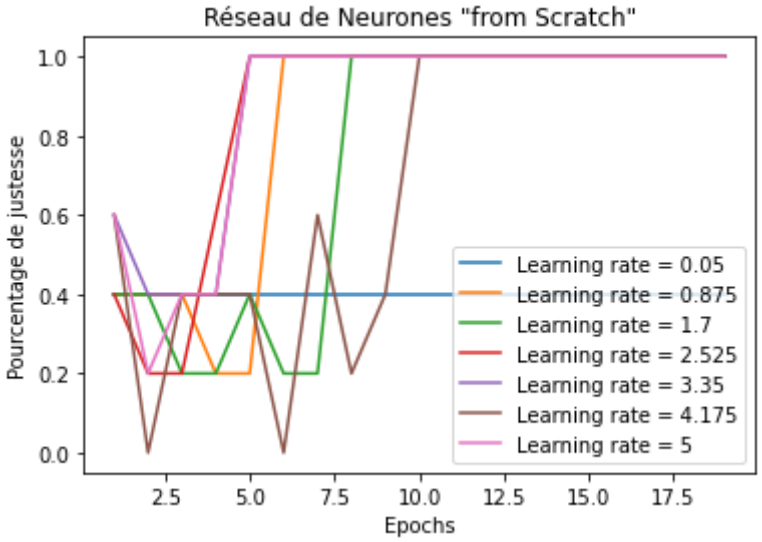


FIGURE 4 – Résultats du Réseau de Neurones "From Scratch"

Problème III : Classification

Le code associé à cette question se trouve dans le fichier *TP_NN_PART3.ipynb*.

On cherche à utiliser un réseau de neurones pour résoudre un problème de classification. Il s'agit de classifier des personnes, selon plusieurs caractéristiques, en risque élevé ("1") ou risque faible ("0") d'avoir une cardiopathie. Voici un aperçu des données sur lesquelles nous tavaillons :

	Age	Sexe	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up
...	...	...	...	...	...	...	...	...	...	...	...
913	45	M	TA	110	264	0	Normal	132	N	1.2	Flat
914	68	M	ASY	144	193	1	Normal	141	N	3.4	Flat
915	57	M	ASY	130	131	0	Normal	115	Y	1.2	Flat
916	57	F	ATA	130	236	0	LVH	174	N	0.0	Flat
917	38	M	NAP	138	175	0	Normal	173	N	0.0	Up

FIGURE 5 – Données *HD_Complete_Data*

On observe que certaines données sont de type catégorielles. Pour pallier ce problème, on choisit d'utiliser l'encodage *one hot* (ou *one hot encoding*). il s'agit de convertir chaque valeur catégorielle en une variable, et

d'attribuer une valeur binaire de 1 ou 0 à ces variables suivant la catégorie à laquelle appartient l'individu. Voici en exemple ce que devient la variable catégorielle "Sexe" après application du *one hot encoding* :

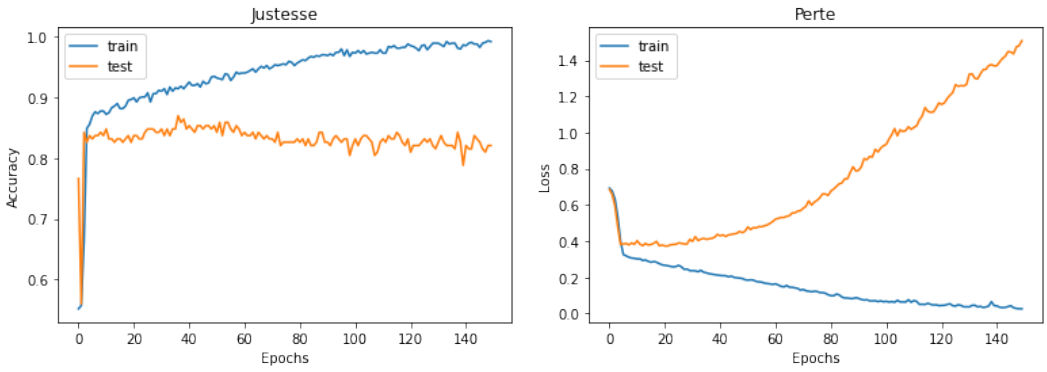
Sexe_F	Sexe_M
0	1
1	0
0	1
1	0
0	1
...	...
0	1
0	1
0	1
1	0
0	1

FIGURE 6 – Variable catégorielle "Sexe" après *one hot encoding*

Après plusieurs testes, le modèle retenue est le suivant : un réseaux de neurones avec deux couches cachées, une première de cinq neurones munit de la fonction *softmax* comme fonction d'activation et une seconde de dix neurones munit de la fonction **relu**. La fonction d'activation associée à la sortie est la fonction *sigmoid*. On utilise 150 epochs, et un batch_size de 40.

Pour éviter le surapprentissage, on peut faire plusieurs choses. Tout d'abord, limiter le nombre de couchées et de neurones par couches.

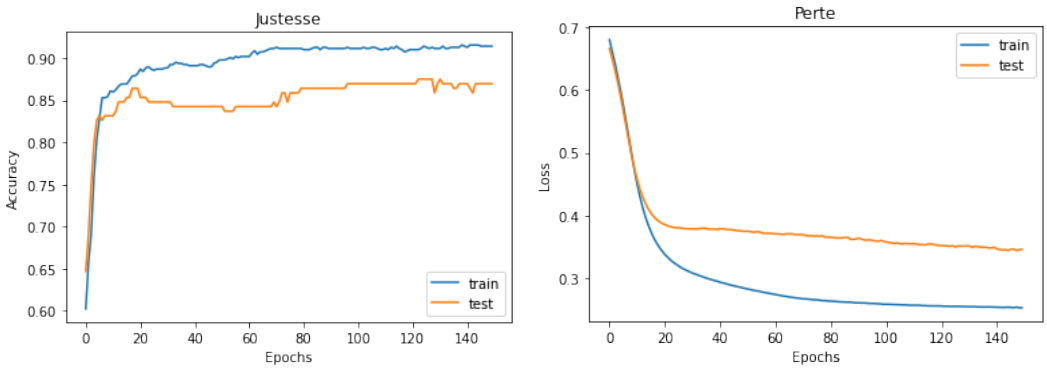
Voici par exemple les résultats en termes de justesse et de perte pour un réseau composé de quatres couches cachées, la première de cinquante neurones, les autres de cent, avec respectivements des fonctions d'activation *softmax*, *relu*, *sigmoid*, *relu* et *sigmoid*. Les autres paramètres sont similaires à ceux du modèle retenu.



(a) Justesse (b) Perte

FIGURE 7 – Résultats du réseau de neurones

On voit clairement sur ces figures qu'au fil des epochs, la justesse du modèle sur les données *train* augmente, tandis que celle des données *test* baissent. De plus, alors que la fonction de perte des données *train* baissent et se rapprochent de 0, celles des données *test* augmentent. Ces deux phénomènes temoignent de sur-apprentissage. Les autres paramètre que l'on doit contrôler sont le nombre d'epochs, qui ne doit pas être trop élevé, et le batch_size, qui lui non plus ne doit pas être trop élevé. Voici les résultats avec le modèle retenu :



(a) Justesse (b) Perte

FIGURE 8 – Résultats du réseau de neurones retenu

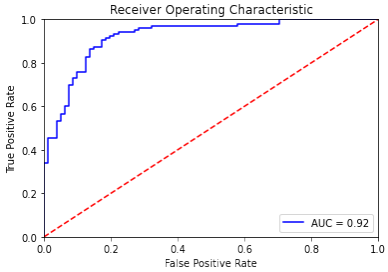
On observe que les problèmes soulevés par le modèle précédent ne sont pas présent. Le resultat final obtenu, en terme de justesse sur les données *test* est de 0.8696. On présente de plus d'autres caractérisques de notre modèle :

	precision	recall	f1-score	support
0	0.89	0.80	0.84	81
1	0.86	0.92	0.89	103
accuracy			0.87	184
macro avg	0.87	0.86	0.87	184
weighted avg	0.87	0.87	0.87	184

(a) Erreurs de classification

```
Matrice de confusion :
 [[65 16]
 [ 8 95]]
Sensitivité : 0.8904109589041096
Spécificité : 0.8558558558558559
```

(b) Matrice de confusion détaillée



(c) Courbe ROC

FIGURE 9 – Performances du modèle retenu

En conclusion, ce problème nécessitait de prêter attention au sur-apprentissage, comme nous l'avons vu précédemment. Effectivement, nous avons choisi nos paramètres pour limiter ce phénomène, tout en maximisant la précision de nos résultats.

Problème IV : Time series analysis

Le code associé à cette question se trouve dans le fichier *TP_NN_PART4.ipynb*.

Après avoir implémenté quatre modèles (RNN, LSTM, ARIMA, SARIMA) pour prédire le taux d'ammonium mensuel dans l'eau du Danube, on représentes les prédictions de chacuns de nos modèles sur le même graphique :

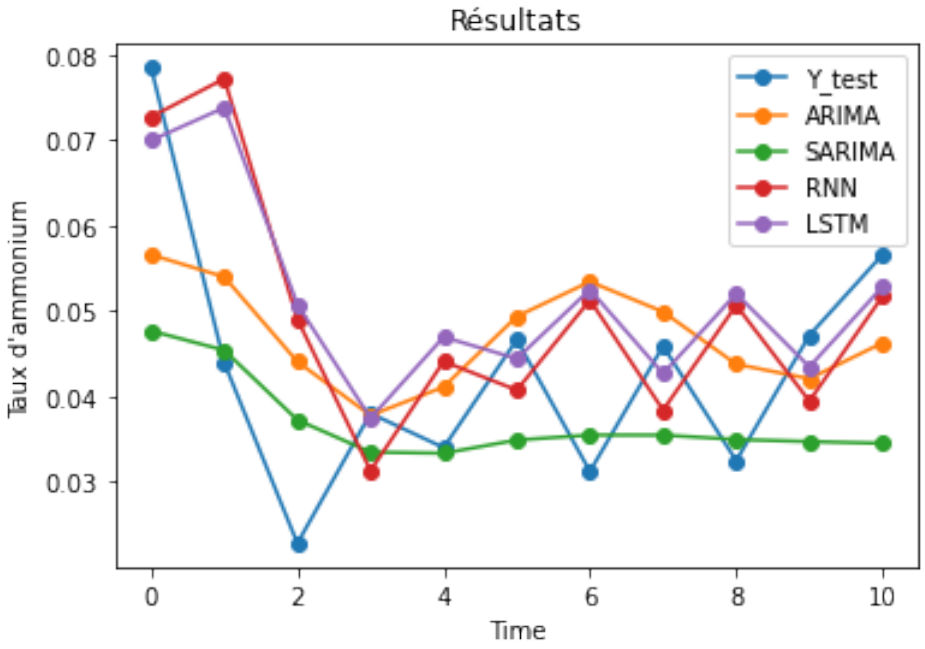


FIGURE 10 – Prédictions de nos quatre modèles, ainsi que valeurs réelles

On présente également la RMSE associée a chaque modèle :

```
RMSE RNN : 0.06377612732660458
RMSE LSTM : 0.06271038557546109
RMSE ARIMA : 0.05067685942411851
RMSE SARIMA : 0.05378046834830557
```

FIGURE 11 – RMSE de nos quatre modèles

On constate que nos résultats pour la méthode SARIMA ne sont pas meilleurs que ceux obtenus avec la méthode ARIMA, malgré la saisonnalité de nos valeurs du taux d'ammonium (période de 12 mois). En effet, la méthode SARIMA n'est pas complètement exploitée lors de nos tests, puisque nos données en entrée pour ce dernier (X_test) sont celles obtenues seulement sur un an.

Enfin, on utilise un autre réseau de neurones de type RNN utilisant toutes les données, ce qui nous donne les résultats suivants :

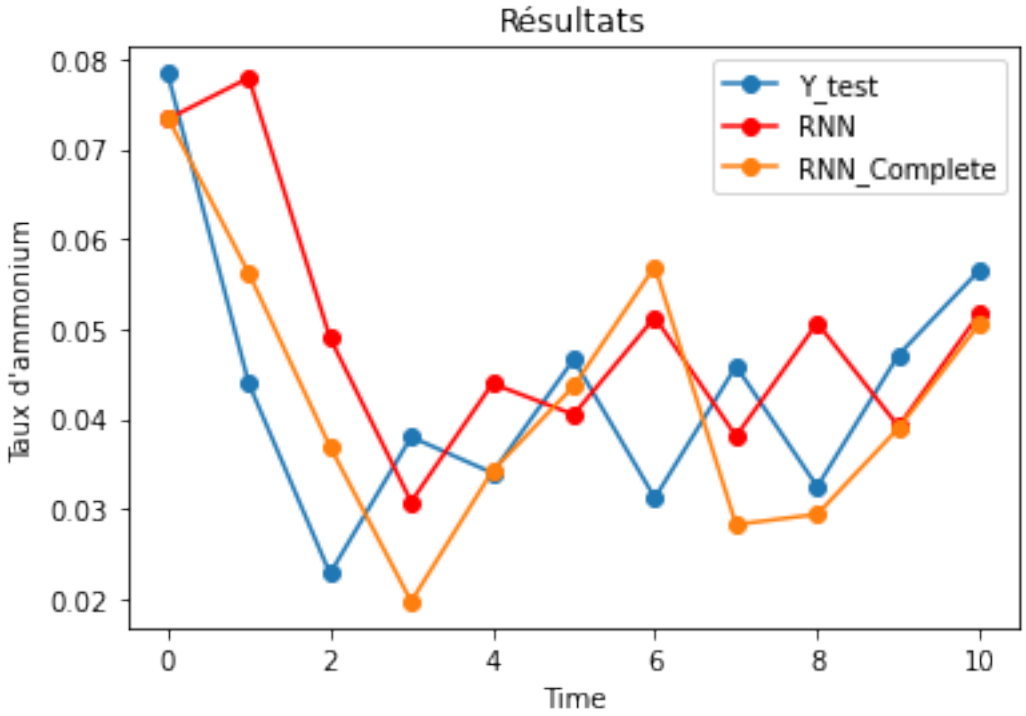


FIGURE 12 – Prédictions des deux RNN

La valeur RMSE obtenue est : 0.0499.