

Mémoire d'informatique

Les intelligences artificielles et les mathématiques

**Quelle est la capacité d'une intelligence artificielle à
résoudre des problèmes mathématiques : l'exemple des
équation polynômiales**

- Baccalauréat International -

Nombre de mots : 3999

Table des matières

Résumé	3
Introduction	4
Les types d'intelligences artificielles.....	4
Problématique	4
Modèles théoriques.....	5
L'apprentissage automatique : les arbres de décisions.....	5
Présentation du programme Python pour les arbres de décision.....	6
Présentation du programme Python pour le <i>random forest</i>	7
L'apprentissage profond : les réseaux de neurones	8
Présentation du programme Python pour les réseaux de neurones.....	10
Présentation de l'expérience	12
I - Expérimentation : recherche les solutions d'un polynôme du second degré	14
Avec les arbres de décisions	14
Premier jeu de données.....	14
Deuxième jeu de données	15
Avec les réseaux de neurones.....	16
Premier jeu de données.....	16
Deuxième jeu de données	18
II - Expérimentation : recherche les solutions d'un polynôme de degré 5.....	20
Avec les arbres de décisions	20
Premier jeu de données.....	20
Deuxième jeu de données	21
Avec les réseaux de neurones.....	22
Premier jeu de données.....	22
Deuxième jeu de données	24
Discussions autour des résultats :.....	26
Conclusion.....	27
Bibliographie	28

Résumé

Cette étude porte sur l'évaluation comparative des capacités de modèles d'intelligence artificielle à résoudre des problèmes mathématiques tels que les équations polynomiales de degré 2 et 5. Ainsi, j'utilise deux ensembles de données distincts et explore l'influence de la taille de l'échantillon de test. Mes analyses mettent en évidence des tendances dans les performances des modèles. Enfin, j'explique l'importance de l'évaluation approfondie des compétences et des limites de l'IA pour lui déléguer des tâches.

Introduction

À la suite de l'ensemble de mes recherches préparatoire, je peux définir qu'une intelligence artificielle est un programme informatique élaboré qui repère des schémas répétitifs dans un jeu de données complexes et est capable de les utiliser pertinemment sur de nouvelles données. On peut dire de ce programme, par sa complexité et par cette phase « d'apprentissage », qu'il imite l'intelligence humaine, en tout cas qu'il s'en approche. Et peut même déduire des sensations si on lui fournit un jeu de données adéquat.

Les types d'intelligences artificielles

L'intelligence artificielle est divisée en plusieurs catégories : l'intelligence artificielle étroite, l'intelligence artificielle générale, et la super intelligence artificielle. Nous étudierons uniquement l'intelligence artificielle étroite.

L'intelligence artificielle étroite, est souvent associé à ce qu'on appelle l'apprentissage automatique ou *machine learning*. Globalement, ce type de programme est capable de réaliser une seule tâche : il faut le voir comme un modèle très spécialisé. Il existe énormément de modèle d'apprentissage automatique mais ici nous étudierons ceux qu'on appelle arbres de décision. Une partie dérivée du *machine learning* est le *deep learning* que nous étudierons également avec un système de réseaux de neurones.

Le but est de fournir un jeu de données qui soit organisé afin de réaliser des prédictions. Ainsi, lors de l'entraînement, suite à une prédiction, il va y avoir une rétropropagation pour modifier le système pour que la valeur d'une nouvelle prédiction soit plus proche de la valeur attendue.

Problématique

A partir des deux modèles cités, la question est de voir s'ils peuvent répondre à des problèmes mathématiques comme la résolution de polynômes du second degré. Mais, l'idée derrière serai plutôt de voir s'ils peuvent proposer des méthodes pour résoudre des problèmes mathématiques qu'on ne peut pas résoudre comme les polynômes de degré 5. Avant cela, l'objectif serait d'estimer les capacités de résolutions mathématiques d'un modèle et d'évaluer s'il permet un progrès dans le domaine. Si le modèles trouve une solution, en étudiant son organisation technique nous pourrions peut-être remonter à une nouvelle méthode de résolution.

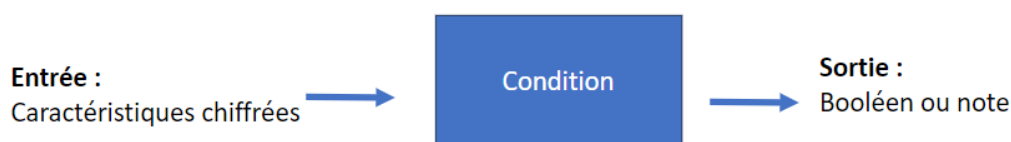
Modèles théoriques

L'apprentissage automatique : les arbres de décisions

J'ai parlé précédemment d'un modèle d'apprentissage automatique qu'on appelle : arbre de décision. Cependant, ce modèle connaît de nombreuses variantes. En ce qui me concerne, j'ai utilisé le « *decision tree regressor* ». Il s'agit d'un modèle d'apprentissage automatique disponible dans la bibliothèque « *scikit learn* » de Python.

Un arbre de décision est un système qui étudie plus ou moins de conditions et qui va cheminer en répondant par oui ou par non jusqu'à une décision finale. L'exemple le plus simple étant fourni sur la figure 1.

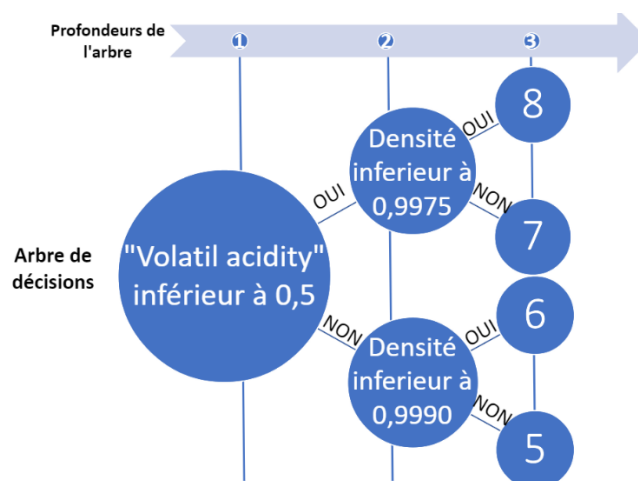
Figure 1 :



Ainsi, le principal paramètre de variation de ce modèle est la profondeur de l'arbre. L'arbre sur la figure 2 possède 3 niveaux, autrement dit une profondeur valant 3. La profondeur c'est la taille de l'arbre qui évolue selon les puissances de 2. Elle illustre aussi la complexité de l'arbre car plus il y a de profondeur plus il y a de chemins possibles.

Si nous prenons l'exemple de la notation des vins, admettons que nous donnions un certain nombre de paramètres et que nous lui demandions de prédire la note donnée au vin sur une échelle de zéro à 10. Par exemple, le programme va repérer, que lorsqu'il y a moins d'acidité, il y a une meilleure note.

Figure 2 :



Pour arriver à entrainer un modèle correctement, il faut donc un jeu de données appropriées à lui faire étudier. C'est la pertinence de ce jeu de données qui va influencer la pertinence générale du modèle. Enfin les conditions vont se réorganiser

par rétropropagation. C'est-à-dire que si j'arrive à une note de 7 au lieu de 8, le chemin va être parcouru dans l'autre sens pour adapter chaque condition afin de corriger la prochaine prédiction.

La première étape est de préparer les données. Elles sont composées de plusieurs valeurs pour des paramètres qu'on estime pertinents et des valeurs à prédire en fonction de ces paramètres. C'est là que l'arbre de décision se crée. Pour rentrer un peu plus dans la partie technique, l'ensemble des valeurs des paramètres sont appelées « X » et les valeurs à prédire « y ». Une fois le modèle entraîné, nous allons le tester sur de nouvelles données puis comparer ces prédictions avec les valeurs attendues. Pour cela, nous allons utiliser une fonction d'erreur. En ce qui me concerne, j'ai choisi d'utiliser la fonction d'erreur MAE (*Mean Absolute Error*). Cette fonction compare les valeurs prédites par le modèle et les valeurs que nous aurions voulu qu'il trouve. Puis, fait la moyenne des valeurs absolue des différences entre les valeurs et les prédictions ($MAE = moy(|prédiction - attendu|)$). Enfin, il est possible de modifier les paramètres comme la profondeur du modèle pour avoir plus d'exactitude sur les prédictions.

Présentation du programme Python pour les arbres de décision

On retrouve tout d'abord les imports autrement dit les bibliothèques que j'ai utilisées.

```
# Les imports
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
```

Ensuite, on retrouve le traitement des données. On retrouve conventionnellement en « X » les différentes colonnes de données qui servent à la prédiction. Puis en « y » les valeurs à prédire. Ensuite, on sépare en deux parties pour obtenir un échantillon d'entraînement et un échantillon de test. Avec le premier jeu de données (entraînement), le model va se former depuis un état aléatoire. Il faut gérer la taille de l'échantillon d'entraînement pour que le model ne soit ni trop entraîné ni pas assez.

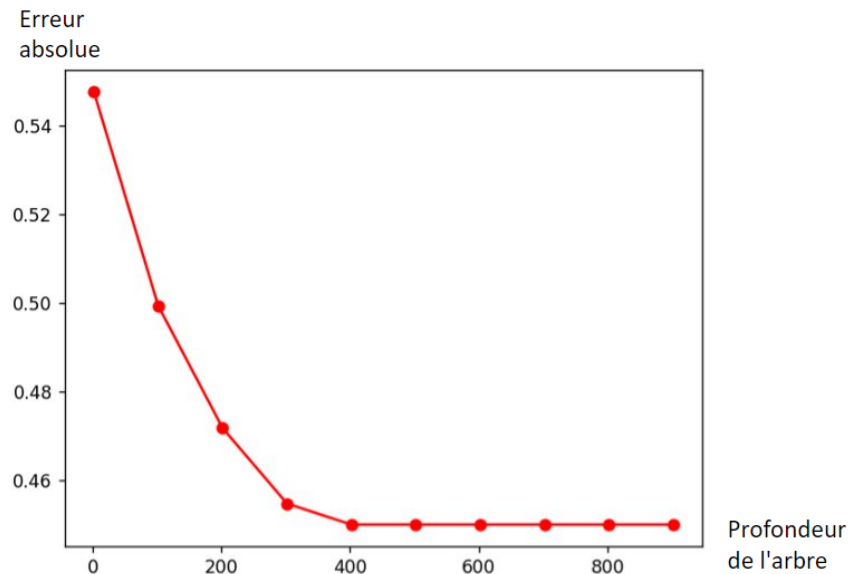
Le code pour le model étant celui-ci :

```
model = DecisionTreeRegressor(max_leaf_nodes=depth, random_state=0)
model.fit(train_X, train_y)
preds_val = model.predict(val_X)
mae = mean_absolute_error(val_y, preds_val)
```

Pendant l'entraînement, le model réalise les prédictions puis se corrige à l'aide des valeurs attendus. Pendant la phase de test, le but est de tester les capacités du modèle sur un échantillon suffisamment large de possibilité pour obtenir une moyenne de l'erreur commise par ce dernier. Ici, les valeurs sont les notes des vins allant de 0 à 10.

Enfin, j'ai bouclé l'opération pour étudier l'influence de la profondeur de l'arbre pour des profondeurs allant de 2 à 1002 avec un pas de 100. J'ai donc fait tracer le graphique suivant.

Graphique de l'erreur absolue en fonction de la profondeur de l'arbre de décision :



Sur ce graphique on observe un point de stagnation à partir d'une profondeur de 400 pour une erreur absolue de 0,45. C'est-à-dire qu'en moyenne la prédiction du modèle a un écart de 0,45 avec la valeur recherchée. Ici la courbe est décroissante mais il y aurait pu avoir un pic à un endroit aléatoire avec une courbe très irrégulière. Cela dépend du jeu de données. En conclusion, les notes allant de 0 à 10 cela fait environ 4,1% d'erreur systématique par rapport à la valeur réelle, ce qui n'est pas énorme mais reste conséquent.

C'est globalement ce même programme que je vais appliquer ensuite aux polynômes. La principale différence sera le jeu de données que je génèrerais aléatoirement dans un fichier CSV.

Présentation du programme Python pour le *random forest*

Il existe une variante plus complexe des arbres de décisions qu'on appelle *random forest*. Il s'agit d'une multitude d'arbres de décision aléatoire avec lesquels on réalise une moyenne. La plupart du temps les prédictions sont plus exactes. En code, cela se présente comme ceci :

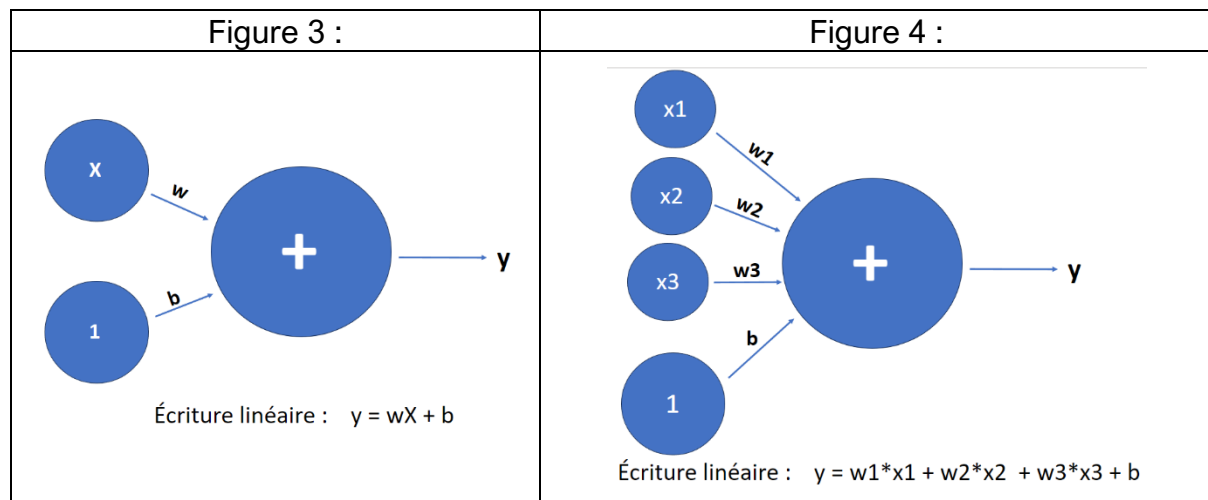
```
# Entraînement du modèle
forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)

# Calcul de l'erreur absolue
print(mean_absolute_error(val_y, preds))
```

Je n'ai pas remis les imports, ni le traitement des données qui sont les mêmes que pour les arbres de décision. Ici, je définis le modèle, je l'entraîne, je lui fais faire des prédictions et calculer l'erreur absolue qui est de 0,428. Nous retrouvons bien une erreur absolue plus faible sur ce modèle que sur les arbres de décisions précédent. Dans ce cas, le gain est minime mais cela varie selon les jeux de données.

L'apprentissage profond : les réseaux de neurones

Pour l'apprentissage profond (Deep Learning), c'est le réseau de neurones le modèle emblématique. Ce système est supposé ressembler au système neuronal humain. En informatique, on peut schématiser un neurone comme ceci :

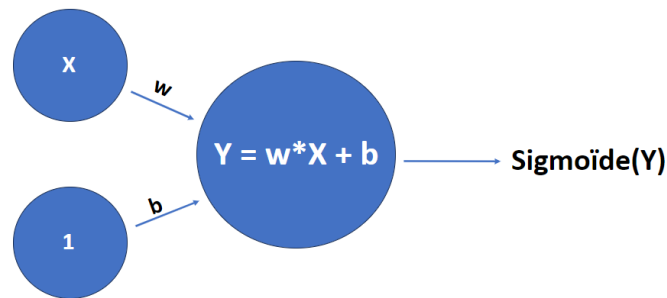


C'est un objet qui prend une entrée (X), lui applique un poids (w), et lui ajoute un biais (b). Enfin il a pour sortie « y » qui vaut donc : $X \times w + b$

Une fois de plus, le premier schéma est le modèle le plus simple possible. La plupart du temps un neurone a plusieurs entrées $x_1, x_2, x_3...$ auxquels il associe un poids $w_1, w_2, w_3...$ comme sur le deuxième schéma.

Une autre variable est la fonction d'activation du neurone. Ici, j'ai simplement noté un « + » dans les neurones pour simplement faire la somme des entrées multipliées par leurs poids et du biais. Mais il est possible de placer une fonction dans le neurone pour l'appliquer à la sortie afin de « manipuler » les valeurs et la plupart du temps pour perturber la linéarité du modèle. Nous pouvons par exemple utiliser la fonction Sigmoid, qui a pour effet de ramener tout $x \in \mathbb{R}$ entre 0 et 1 et à pour expression : $f(x) = \frac{1}{1+e^x}$. Ce qui donne schématiquement ceci :

Figure 5 :



$$\text{Ecriture linéaire : } L = \frac{1}{1+e^{wX+b}}$$

Ensuite, on construit un réseau qui sera organisé en niveaux comme ci-dessous. On trouve ici deux entrées et un biais distribué dans le premier niveau de neurones dit « caché » ou intermédiaire. Il est composé de 4 neurones. Puis, on distribue les sorties du premier niveau en entrées du deuxième niveau en ajoutant un biais par neurone. Enfin, on donne les sorties du deuxième niveau en entrée d'un unique neurone pour obtenir une sortie unique. Ici, la fonction d'activation est ReLu (Unité Linéaire Rectifiée) qui recherche le maximum entre 0 et y (expression : $f(x) = \max(0, x)$). Ci-dessous, il s'agit de connexions dites dense. En effet, on connecte chaque neurone du niveau précédent avec chacun des neurones du niveau suivant.

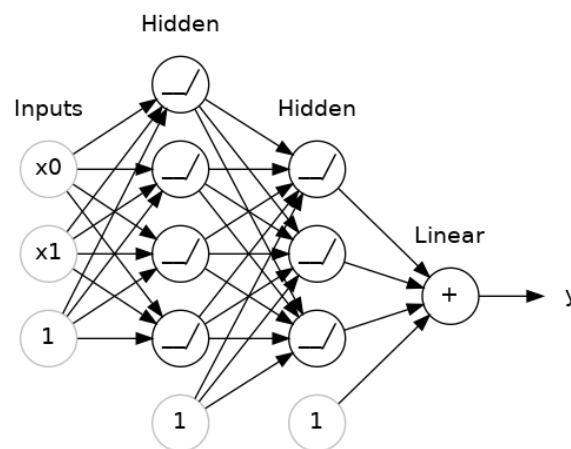


Schéma de : <https://www.kaggle.com/code/ryanholbrook/deep-neural-networks>

Il s'agit d'un réseau assez simple avec seulement deux niveaux intermédiaires de 4 et 3 neurones. Si on reprend l'exemple du jeu de données sur les vins : j'ai un réseau qui prends 11 entrées et contient 3 niveaux intermédiaires de 1024 neurones chacun. Soit avec N le nombre de niveaux total et a_n la taille du niveau (or biais) a et n étant des entiers naturels supérieur ou égaux à 1. Ainsi on a :

$$S = \sum_{n=1}^{N-1} (a_n + 1) \times a_{n+1}$$

Dans notre cas :

$$S = (11 + 1) \times 1024 + (1024 + 1) \times 1024 + (1024 + 1) \times 1024 + (1024 + 1) \times 1 = 2114563$$

Faisant un total de 2 114 563 biais et poids à moduler.

Le réseau de neurone est un système qui fonctionne également par rétropropagation, c'est à dire qu'au début, tout est paramétré aléatoirement. Puis en fonctions de la comparaison entre ses résultats et les résultats attendus, le réseau va corriger ses biais et ses poids, en progressant dans le sens inverse qu'il utilise pour calculer, dans le but de se rapprocher de la valeur qui était attendue lors de la prochaine prédiction.

Présentation du programme Python pour les réseaux de neurones

On retrouve ci-dessous les imports et donc les bibliothèques que je vais utiliser. J'utilise essentiellement le module « keras » de la bibliothèque « tensorflow » en Python.

```
# Les imports
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.callbacks import EarlyStopping
from tensorflow import keras
from keras import layers
from matplotlib import pyplot as plt
```

On arrive donc à la préparation du modèle. Il comporte quatre niveaux intermédiaires de 1024 neurones chacun. Ils ont pour fonction d'activation la fonction Sigmoïde. Le premier niveau prend 11 entrées qui sont les 11 caractéristiques de chaque vin, associées à des valeurs chiffrées. Également, avant chaque niveau intermédiaire excepté le premier, il y a un niveau dit de « Batch Normalisation » qui permet de normaliser les données : de toutes les mettre sous la même forme en quelque sorte. Enfin, une fonction d'erreur et une fonction d'optimisation sont associées au modèle. Pour la fonction d'erreur, il s'agit de la même fonction que pour les arbres de décision : la fonction MAE. La fonction d'optimisation est la fonction Adam.

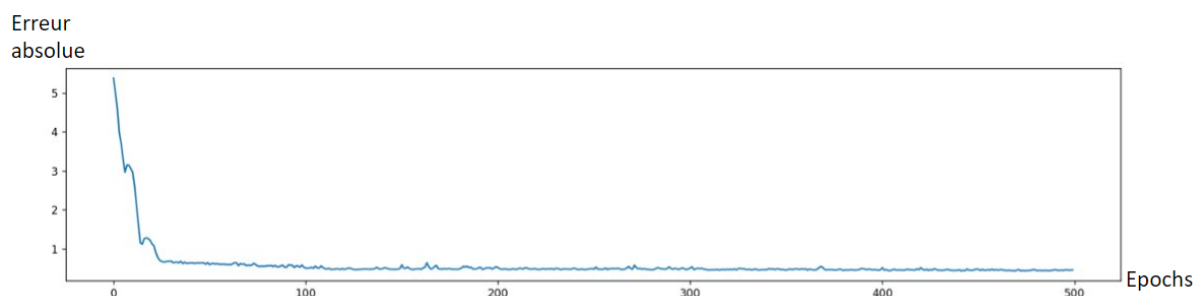
```
# Préparation du modèle
model = keras.Sequential([
    layers.Dense(1024, activation='sigmoid', input_shape=[11]),
    layers.BatchNormalization(),
    layers.Dense(1024, activation='sigmoid'),
    layers.BatchNormalization(),
    layers.Dense(1024, activation='sigmoid'),
    layers.BatchNormalization(),
    layers.Dense(1024, activation='sigmoid'),
    layers.BatchNormalization(),
    layers.Dense(1),
])

model.compile(
    optimizer="adam",
    loss="mae",
)
```

Les lignes ci-dessous, permettent d'entraîner et de tester le modèle. Deux nouveaux paramètres apparaissent : la taille du « batch » et le nombre « d'epochs ». La « batch size » c'est le nombre de lignes avant que le modèle se mette à jour. Il y a 1600 lignes dans mes données sur les vins et toutes les 256 lignes, le modèle va se mettre à jour. Le nombre d'epochs est le nombre de fois où le modèle va parcourir les données. Pour 1600 lignes, et 500 epochs, le modèle va se mettre à jour 3125 fois ($500 \text{ parcours} \times 1600 \text{ lignes} = 800\,000$ mises à jour, soit $\frac{800\,000}{256} = 3125$ mises à jour). Soit : $\frac{1600 \times 500}{256} = 3125$.

```
# Entraînement du modèle
history = model.fit(
    X_train, y_train,
    validation_data=(X_valid, y_valid),
    batch_size=256,
    epochs=500,
    verbose=0,
)
```

Enfin, une dernière partie de code va me renvoyer ce graphique d'erreur absolue en fonction du nombre de parcours du jeu de données (epochs). On observe sur le graphique que, dans ce cas, on a une faible évolution au-delà de 50 epochs. On retrouve également l'erreur absolue minimale qui, pour ce modèle, vaut environ 0,427. A jeu de données équivalent, le réseau de neurones est donc à peine plus efficace que l'arbre de décision et le *random forest*.



En guise de conclusion pour cette partie, je vais préciser la différence entre apprentissage profond et apprentissage automatique. En effet, ce sont déjà des systèmes différents mais ils reposent sur les mêmes principes. Alors finalement, la différence se fait surtout au niveau des données d'apprentissage. En effet, comme je l'ai dit dans l'introduction, l'apprentissage automatique va être plus spécialisé mais il va donc falloir lui fournir des données complètes et traitées. Tandis que dans le cas de l'apprentissage profond, le réseau de neurone peut recevoir des données plus brutes et en faire ressortir les données qui l'intéressent pour les prédictions. Pour l'apprentissage automatique, il faudra sélectionner au préalable les données importantes. C'est pour cela qu'on parle d'intelligence artificielle générale pour les réseaux de neurones et l'apprentissage profonds car ils sont polyvalents et plus performants.

Présentation de l'expérience

Je vais réaliser plusieurs séries de tests avec les trois programmes que je viens de présenter. Je vais réaliser une première partie sur les polynômes de degré 2, pour lesquels, je vais chercher le nombre de solutions, puis les solutions elles-mêmes. Ensuite, je réaliserais les mêmes étapes pour les polynômes de degré 5.

Pour les deux types de polynômes, je vais générer des polynômes aléatoires de formes $(x - a)(x - b)$ et $(x - a)(x - b)(x - c)(x - d)(x - e)$. Ainsi, les entiers : $a, b \in [-5060 ; 5060]$ et dans l'autre cas les entiers : $a, b, c, d, e \in [-20 ; 20]$. Puis, je développe pour obtenir un polynôme de degré 2 ou 5 ayant pour solutions a, b ou a, b, c, d, e . Concernant les étendues de valeurs aléatoires, elles sont arbitraires mais équivalentes. En effet, $(5060 \times 2 + 1)^2$ est proche de $(20 \times 2 + 1)^5$. Ainsi, j'obtiens environ le même nombre de possibilité pour les deux degrés de polynôme (avec un écart d'environ +15% pour le degré 5 dû à une erreur de calculs).

$$(5060 \times 2 + 1)^2 = 102\,434\,641 \text{ Possibilités pour le second degré}$$

$$(20 \times 2 + 1)^5 = 115\,856\,201 \text{ Possibilités pour le degré 5}$$

Pour revenir au fonctionnement de mon algorithme, admettons a, b, c, d et e les solutions

Si on génère le polynôme :

$$(x + 2)(x - 7)(x + 5)(x - 1)(x + 4)$$

On obtient la forme développée :

$$x^5 + 3x^4 - 43x^3 - 187x^2 - 54x + 280$$

La liste des solutions attendues en sortie est : $[-2, 7, -5, 1, -4]$

Enfin, je vérifie que le polynôme développé n'est pas déjà dans les données, pour éviter d'avoir deux fois les mêmes données, ou de tester l'intelligences artificielle sur les mêmes données que celle d'entraînement, ce qui n'a aucun intérêt et peut même être problématique (crée des déséquilibres). Une fois tous mes tableaux remplis avec 100 000 valeurs d'entraînement et de test, je passe aux essais que vous allez voir ci-dessous avec les différents modèles d'intelligences artificielles et avec des variations de paramètres. Par exemple, les réseaux de neurones étant très longs à entraîner, je fais des essais avec seulement 10 000 valeurs. Puis, j'entraîne le meilleur sur l'ensemble des données.

Enfin, je présenterai uniquement les résultats, les algorithmes étant similaires à ceux présentés précédemment.

Je réaliserai l'ensemble des tests sur deux jeux de données générés aléatoirement pour pouvoir observer l'impact de la pertinence de ce jeu de données. Ainsi, je

présenterai pour chaque partie des essais avec une quantité de données de test équivalent à 1%, 10%, 50% et 100% de la quantité de données d'entraînement. Données qui sont différentes de celles d'entraînement il est uniquement question de leur quantité.

Le premier modèle de réseaux testé aura la forme suivante : 5 niveaux intermédiaires de 1024 neurones systématiquement précédés d'un niveau de normalisation, le tout terminé par un niveau de sortie composé d'un unique neurone également précédé d'un niveau de normalisation.

Un deuxième modèle testé comprendra en plus, des niveaux de « Drop Out ». On abandonne une partie des données de manière aléatoire à la sortie de chaque niveau intermédiaire. L'intérêt de ce deuxième modèle est de casser un petit peu la linéarité du premier model en apportant une part d'aléatoire à l'intérieur du modèle un peu comme le *random forest*.

Enfin, tous ces systèmes prendront toujours en entrée les facteurs des puissances de x , des polynômes, et devront ressortir la liste des solutions. Ainsi seront affichés des graphiques de l'évolution de l'erreur absolue en fonction de la profondeur pour les arbres de décision et en fonction du nombre d'epochs pour les réseaux de neurones. Ces graphiques seront accompagnés de tableaux que j'ai rempli avec les valeurs numériques renvoyées dans mon terminal et d'une brève analyse.

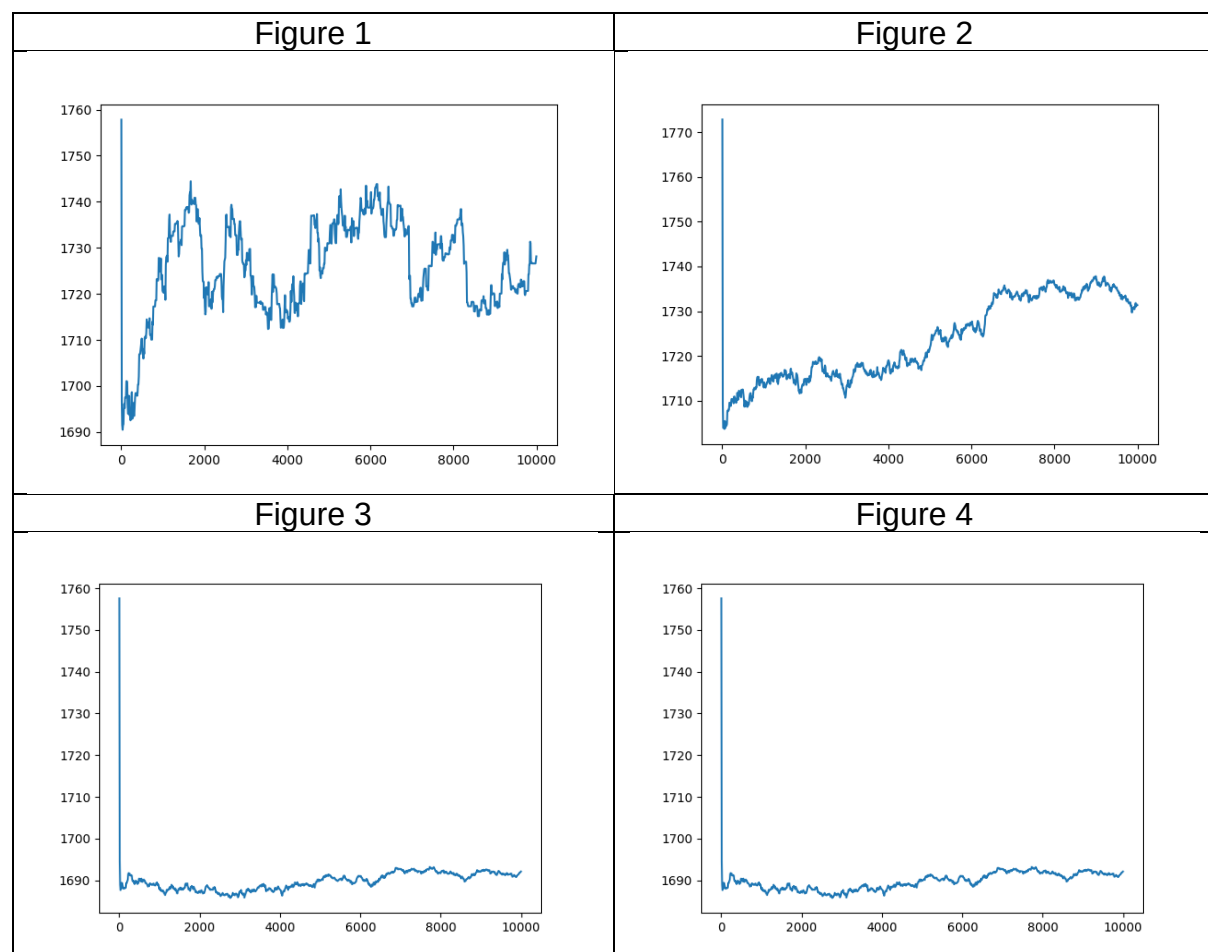
I - Expérimentation : recherche les solutions d'un polynôme du second degré

Avec les arbres de décisions

Premier jeu de données

Graphique	Nombre de données de test	Meilleure profondeur de l'arbre de décision	Meilleure MAE arbre de décision	MAE <i>random forest</i>
Figure 1	1 000 (1%)	35	1690.48	1706.46
Figure 2	10 000 (10%)	55	1703.70	1718.21
Figure 4	50 000 (50%)	2765	1685.80	1687.70
Figure 4	100 000 (100%)	2765	1685.80	1687.70

Graphiques de l'évolution de MAE (axe : OY) en fonction de la profondeur (axe : OX) pour les arbres de décisions :

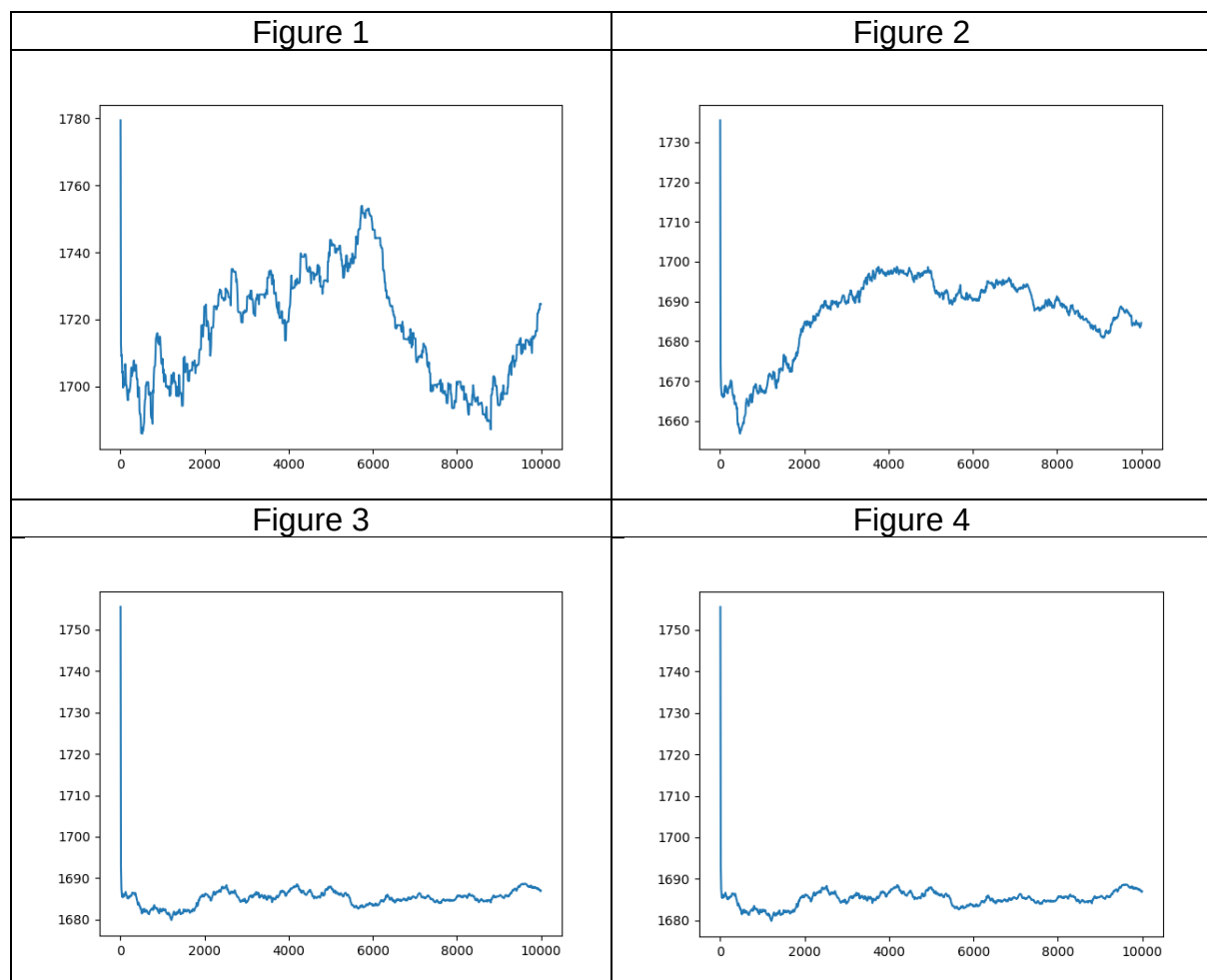


On peut observer que plus il y a de données de test plus la courbe est régulière et lisse. La meilleure erreur est celle des arbres de décision testés sur un grand nombre de valeurs (taille supérieure à 50% de l'échantillon). On peut penser que les fortes erreurs sont régulées par la masse. Enfin les modèles plus linéaires sont légèrement meilleurs.

Deuxième jeu de données

Graphique	Nombre de données de test	Meilleure profondeur de l'arbre de décision	Meilleure MAE arbre de décision	MAE <i>random forest</i>
Figure 1	1 000 (1%)	525	1685.91	1711.95
Figure 2	10 000 (10%)	475	1656.83	1676.80
Figure 4	50 000 (50%)	1215	1679.86	1688.47
Figure 4	100 000 (100%)	1215	1679.86	1688.47

Graphiques de l'évolution de MAE (axe : OY) en fonction de la profondeur (axe : OX) pour les arbres de décisions :



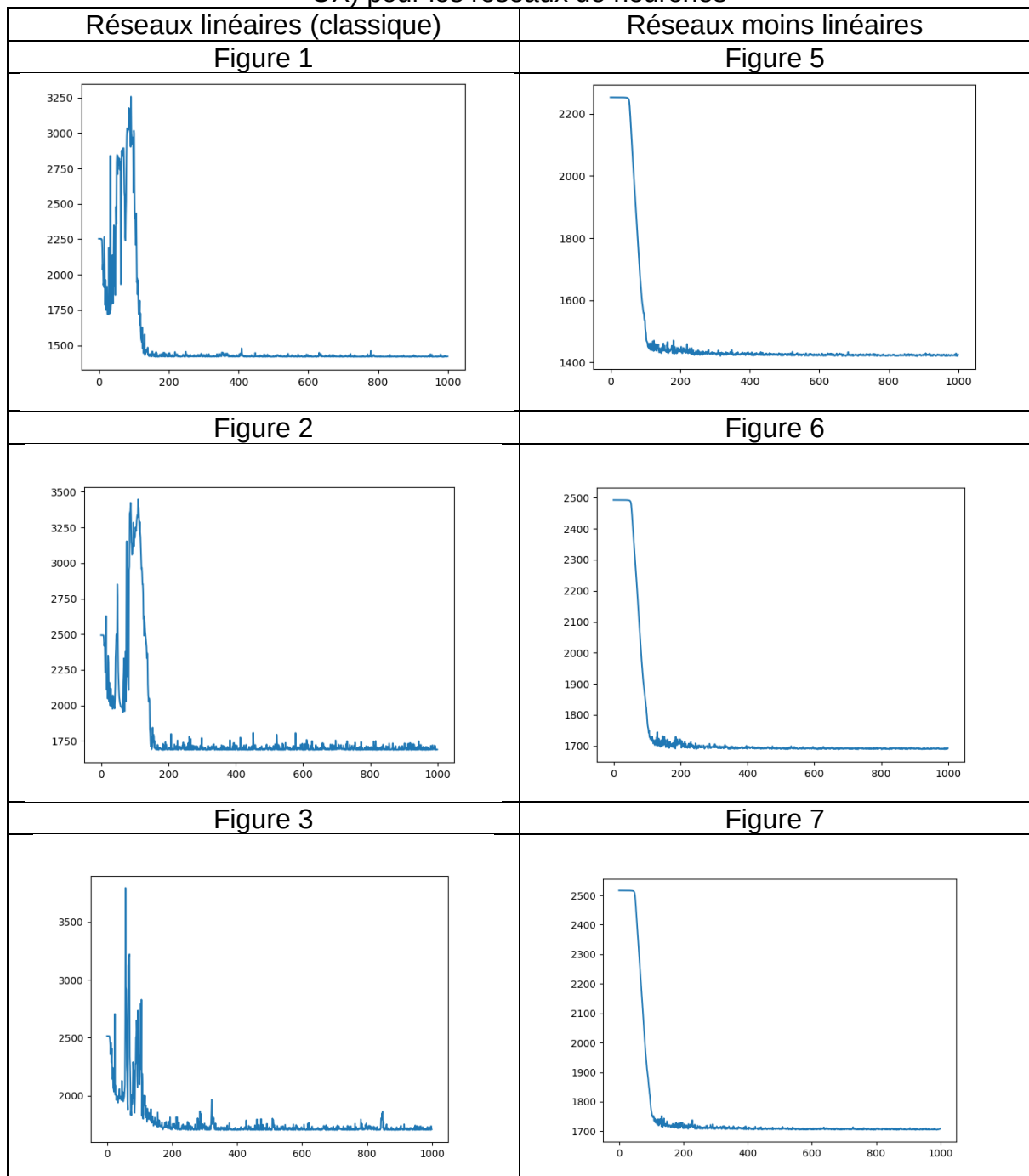
Pour le deuxième jeu de données, les remarques sont les mêmes. On constate simplement que d'un jeu de données à l'autre l'erreur minimale peut varier. Le programme est légèrement meilleur sur ce jeu ci. Une fois de plus, les modèles plus linéaires sont meilleurs.

Avec les réseaux de neurones

Premier jeu de données

Nombre de données de test	Graphique	MAE réseau classique	Graphique	MAE réseaux moins linéaire
100 (1%)	Figure 1	1419.07	Figure 5	1420.04
1 000 (10%)	Figure 2	1686.65	Figure 6	1688.35
5 000 (50%)	Figure 4	1703.22	Figure 7	1704.75
10 000 (100%)	Figure 4	1700.98	Figure 8	1701.43

Graphiques de l'évolution de MAE (axe : OY) en fonction du nombre d'epochs (axe : OX) pour les réseaux de neurones



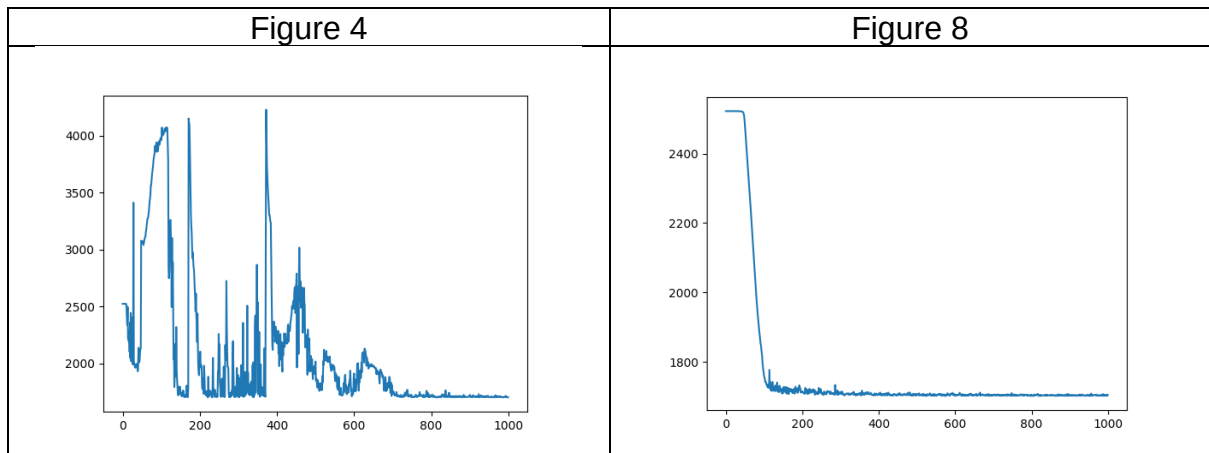
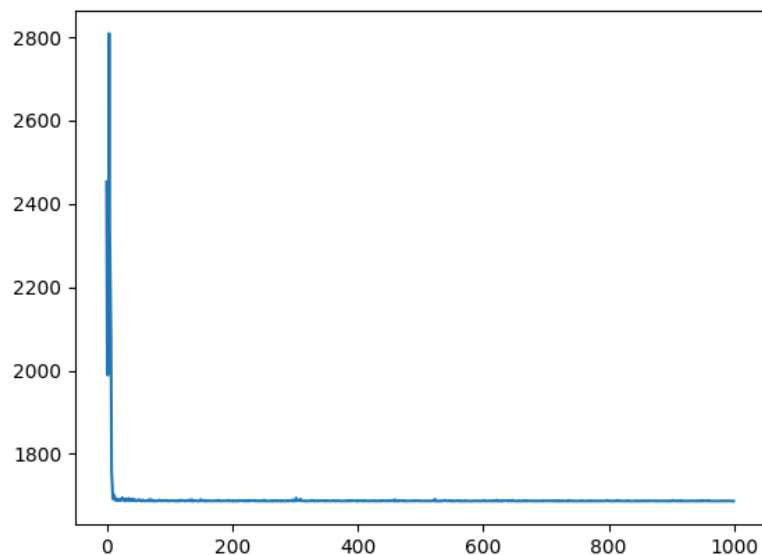


Figure 9 : Test avec 100 000 données d'entraînement et 1000 de test (1%) sur un réseau linéaire : 1686.55



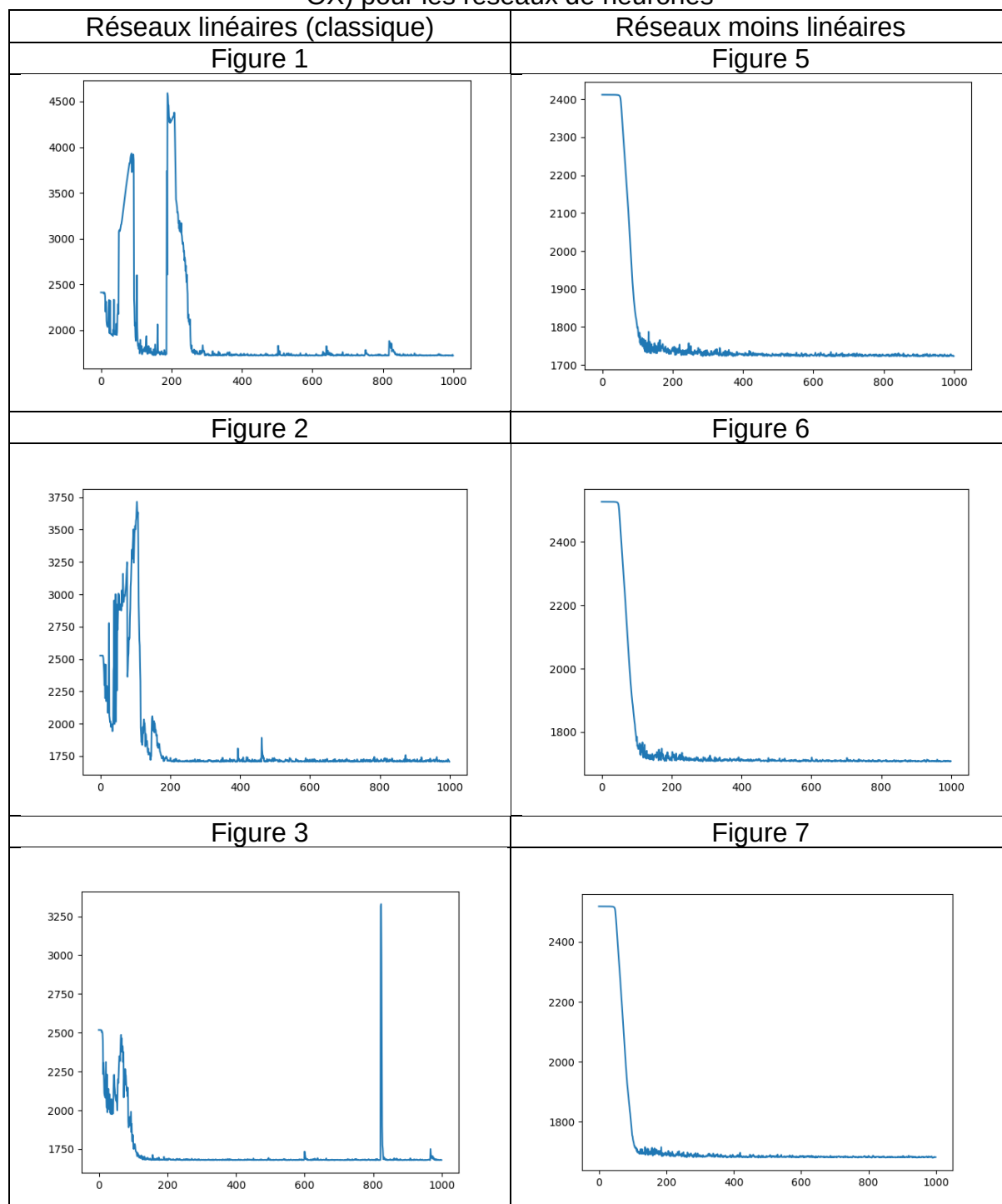
Pour ce premier jeu de données, avec les réseaux de neurones, on peut voir une erreur bien plus faible avec très peu de données de test, un résultat probablement « aberrant » : la quantité faible de données introduit un résultat plus aléatoire qui favorise une moyenne plus basse.

On notera également une courbe très irrégulière pour les réseaux simples : ils sont peut-être trop linéaires bien qu'ils aient des performances légèrement meilleures que les modèles moins linéaires. Cela se passe comme si les modèles linéaires cherchaient dans quel sens progresser alors que les autres sont toujours décroissants.

Deuxième jeu de données

Nombre de données de test	Graphique	MAE réseau classique	Graphique	MAE réseaux moins linéaire
100 (1%)	Figure 1	1721.89502	Figure 5	1721.925293
1 000 (10%)	Figure 2	1704.91150	Figure 6	1706.044556
5 000 (50%)	Figure 4	1679.06897	Figure 7	1680.382202
10 000 (100%)	Figure 4	1760.61597	Figure 8	1666.134155

Graphiques de l'évolution de MAE (axe : OY) en fonction du nombre d'epochs (axe : OX) pour les réseaux de neurones



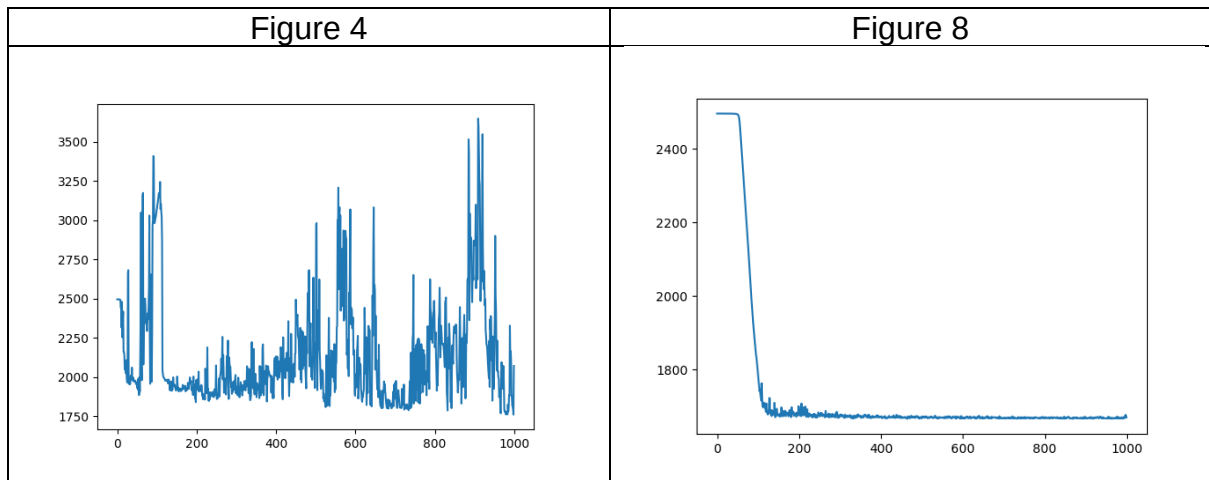
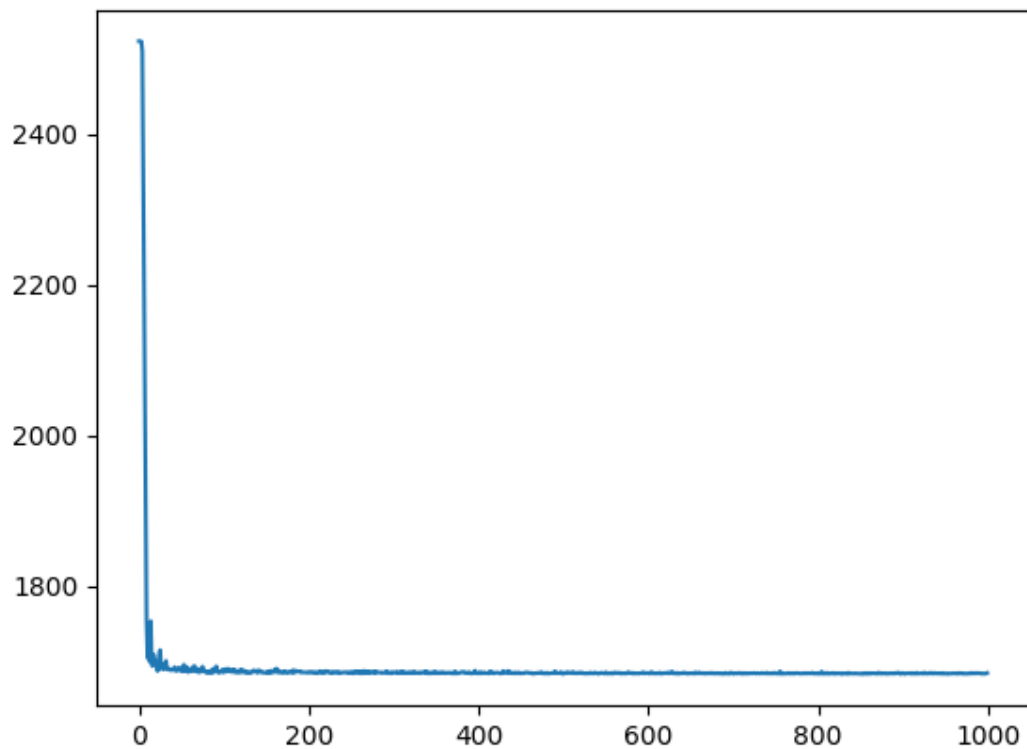


Figure 9 : Test avec 100 000 données d'entraînement et 100 000 de test (100%) sur un réseau moins linéaire : 1683.33



Sur ce jeu de données, les erreurs sur l'échantillon de faible quantité sont plus importantes, ce qui confirme l'hypothèse du point aberrant pour le jeu de données précédent. Étonnamment cette fois-ci, c'est le réseau le moins linéaire qui a de meilleurs résultats, le réseau linéaire avec autant de données de test que d'entraînement perd en précision.

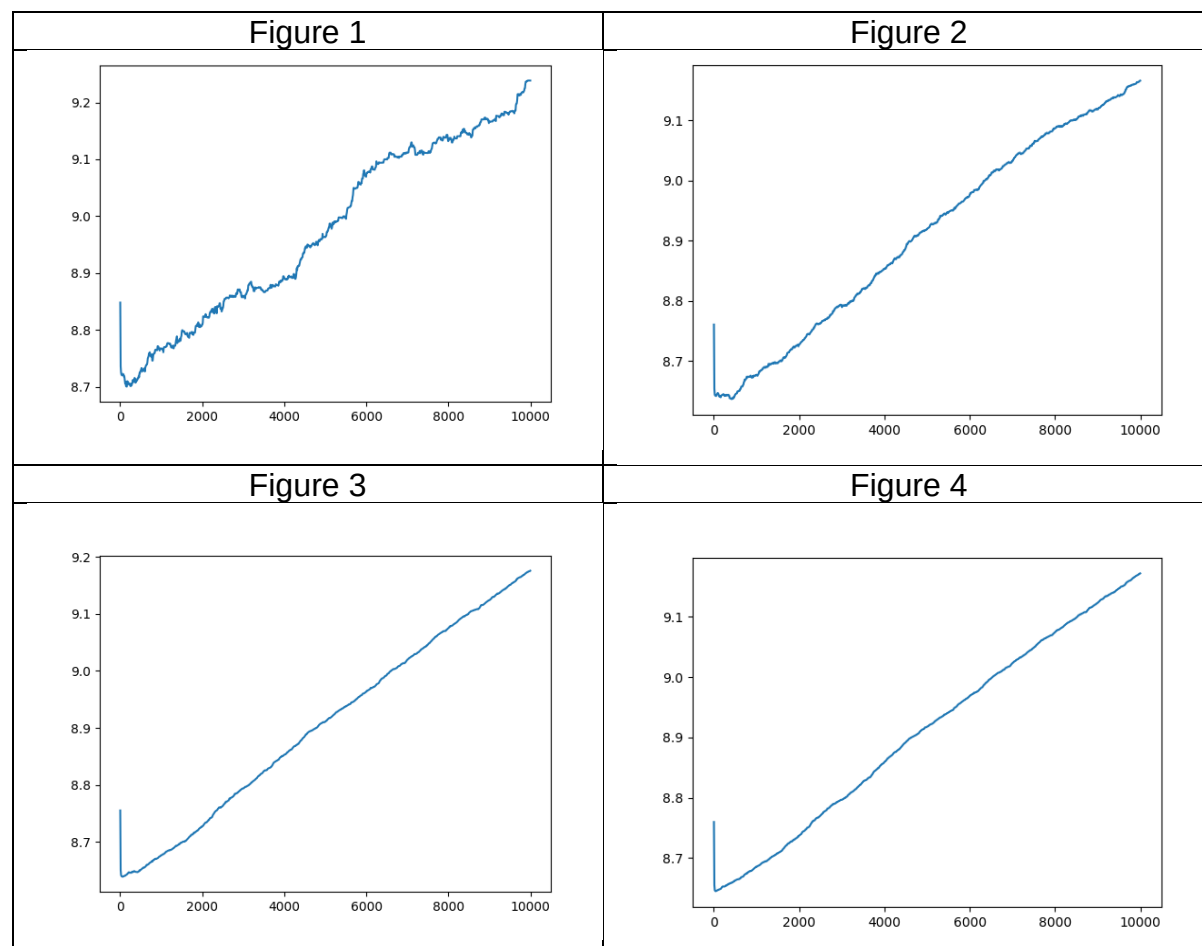
II - Expérimentation : recherche les solutions d'un polynôme de degré 5

Avec les arbres de décisions

Premier jeu de données

Graphique	Nombre de données de test	Meilleure profondeur de l'arbre de décision	Meilleure MAE arbre de décision	MAE <i>random forest</i>
Figure 1	1 000 (1%)	165	8.701	9.044
Figure 2	10 000 (10%)	425	8.637	8.916
Figure 4	50 000 (50%)	55	8.639	8.912
Figure 4	100 000 (100%)	45	8.645	8.918

Graphiques de l'évolution de MAE (axe : OY) en fonction de la profondeur (axe : OX) pour les arbres de décisions :

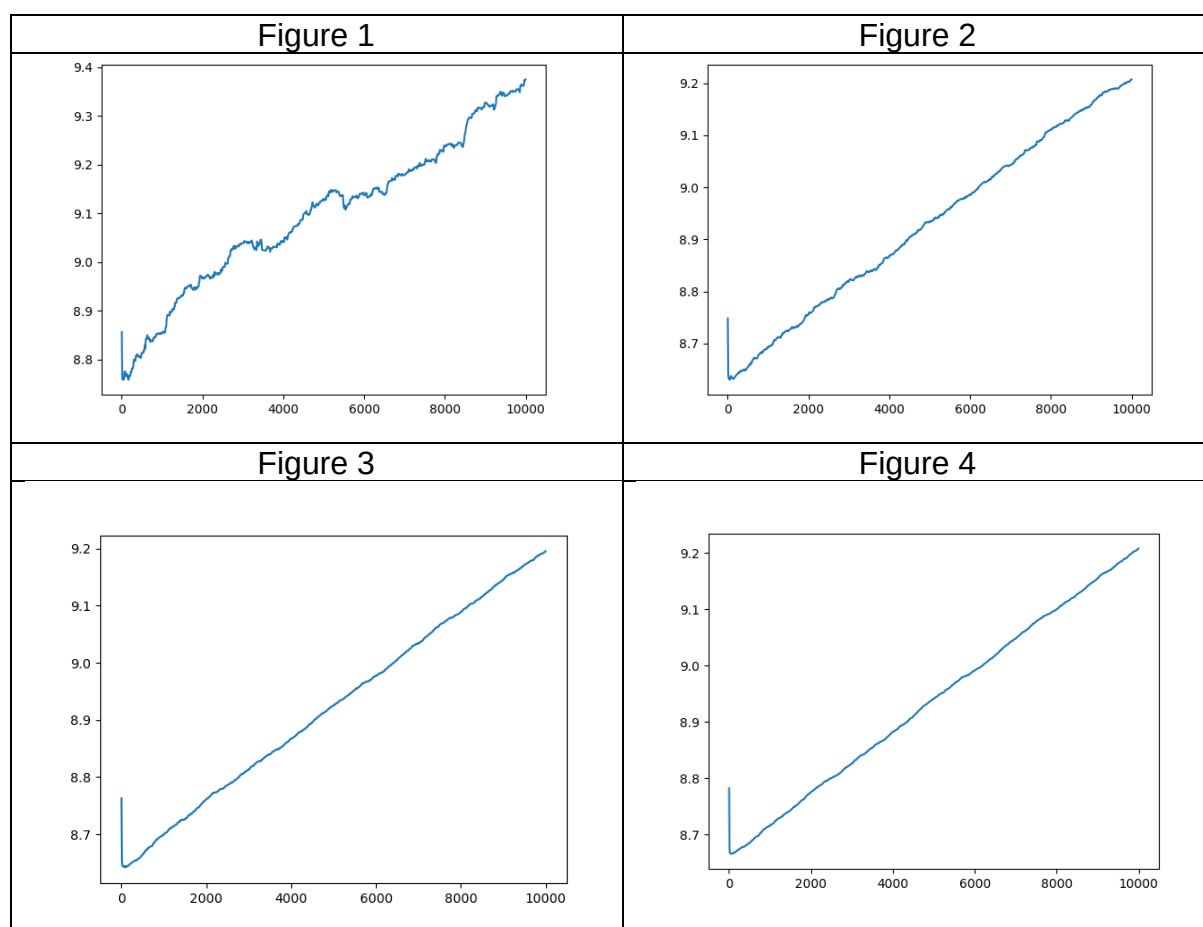


C'est le modèle le plus linéaire qui est le plus efficace.

Deuxième jeu de données

Graphique	Nombre de données de test	Meilleure profondeur de l'arbre de décision	MAE a cette profondeur d'arbre de décision	MAE <i>random forest</i>
Figure 1	1 000 (1%)	175	8.759	9.086
Figure 2	10 000 (10%)	55	8.630	8.926
Figure 4	50 000 (50%)	105	8.642	8.933
Figure 4	100 000 (100%)	55	8.666	8.945

Graphiques de l'évolution de MAE (axe : OY) en fonction de la profondeur (axe : OX) pour les arbres de décisions :



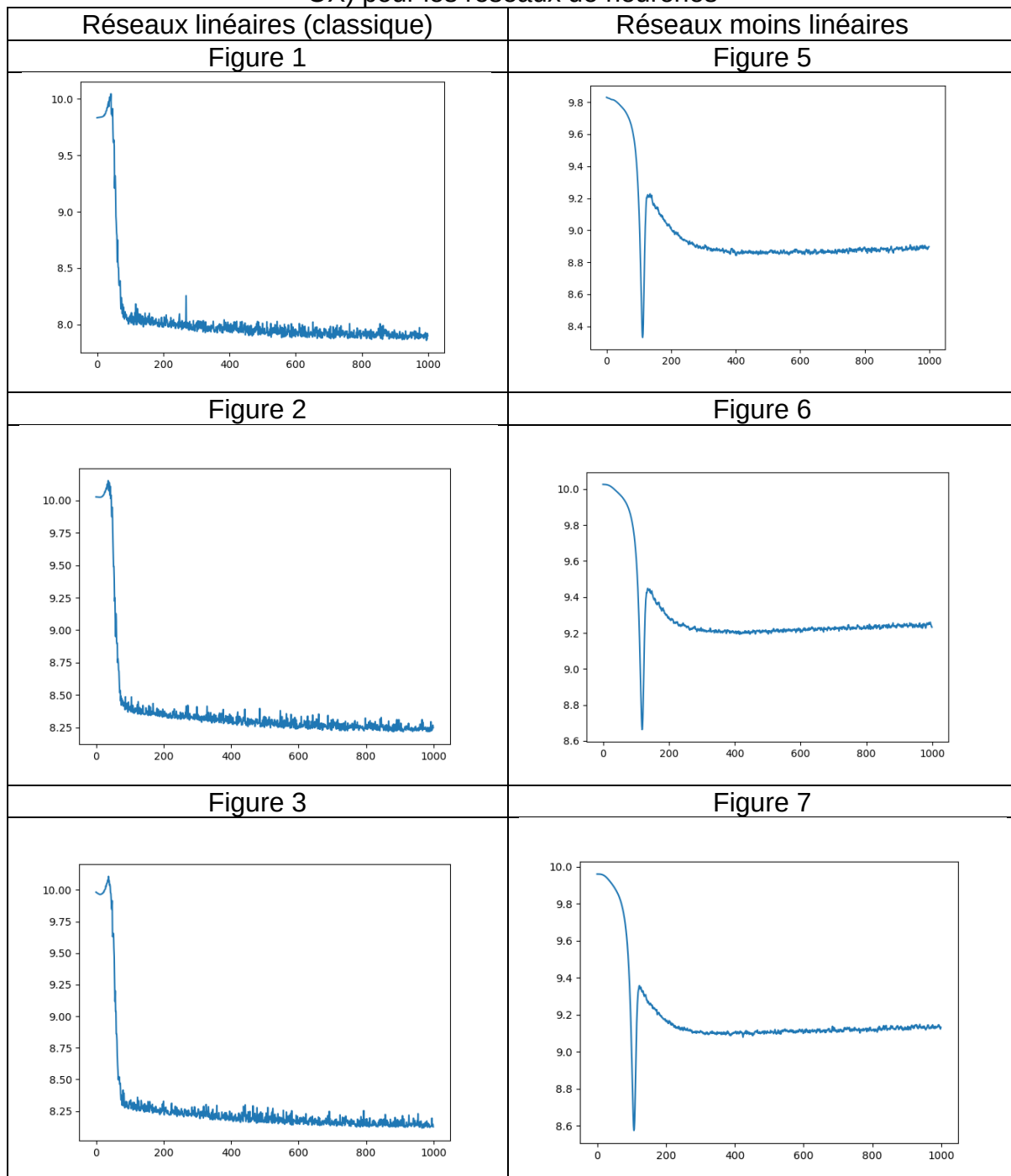
Une fois de plus, plus les données de test sont nombreuses, plus la courbe est régulière. On observe également des optimums très tôt puis une courbe globalement croissante montrant que l'arbre ne nécessite pas de trop grande complexité sinon il perd en précision.

Avec les réseaux de neurones

Premier jeu de données

Nombre de données de test	Graphique	MAE réseau classique	Graphique	MAE réseaux moins linéaire
100 (1%)	Figure 1	7.858	Figure 5	8.330
1 000 (10%)	Figure 2	8.217	Figure 6	8.667
5 000 (50%)	Figure 4	8.115	Figure 7	8.574
10 000 (100%)	Figure 4	8.121	Figure 8	8.583

Graphiques de l'évolution de MAE (axe : OY) en fonction du nombre d'epochs (axe : OX) pour les réseaux de neurones



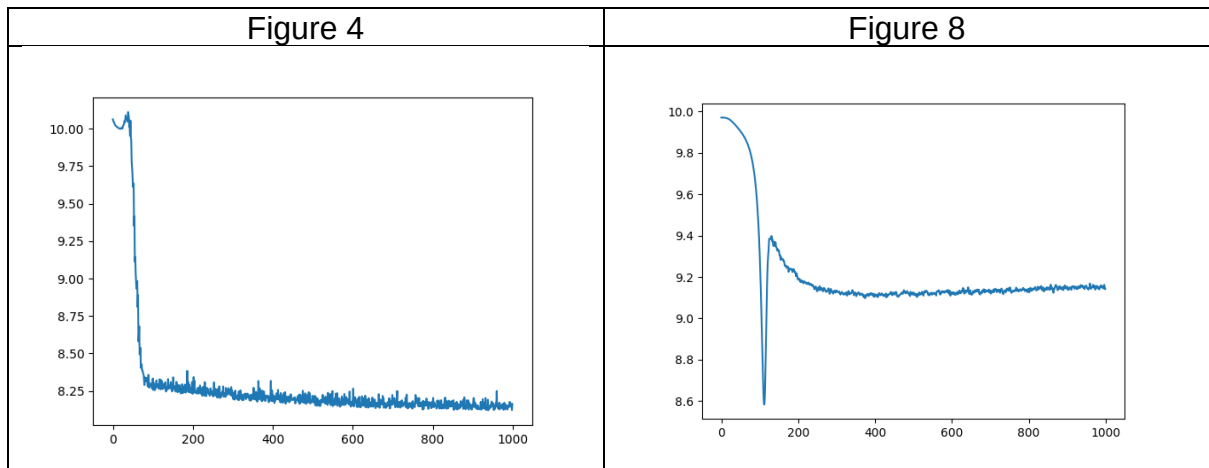
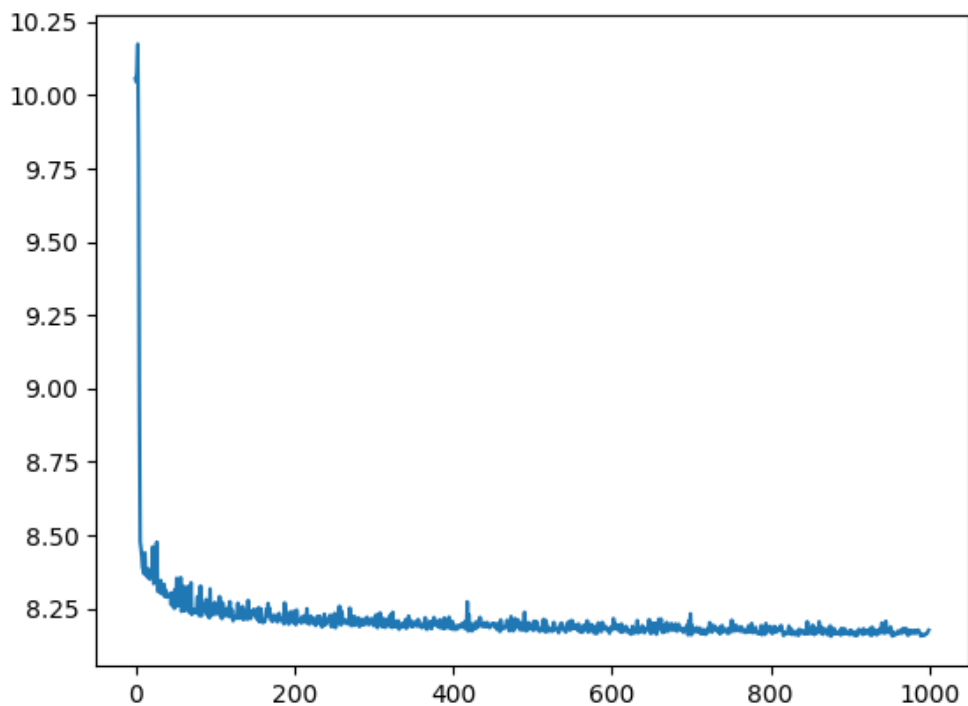


Figure 9 : Test avec 100 000 données d'entraînement et 1 000 de test (1%) sur un réseau linéaire : 8.155

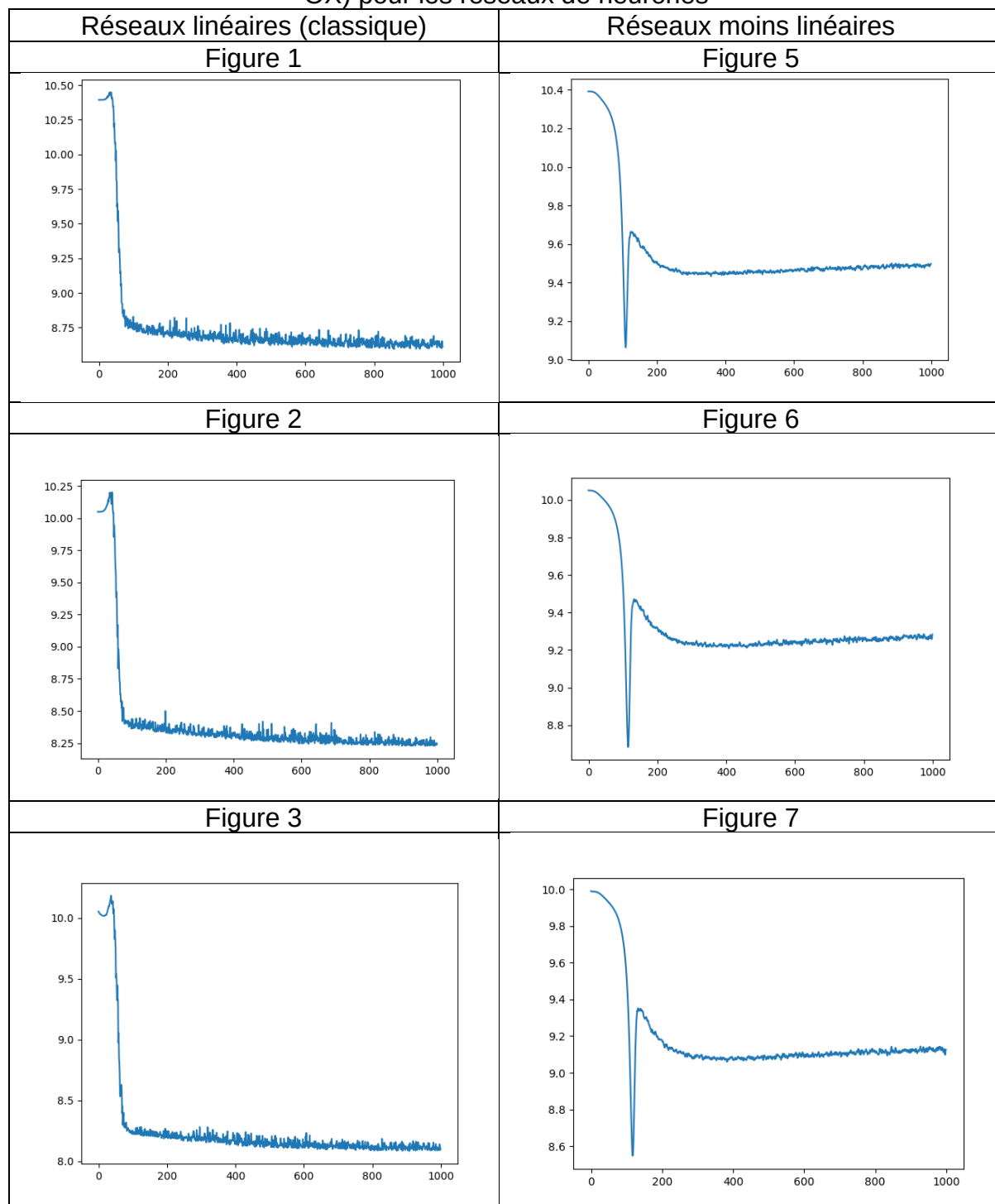


Pour ce deuxième jeu de données, on obtient encore une erreur étonnamment plus faible que les autres lorsqu'on teste le modèle sur très peu de données. On note également que le modèle linéaire est plus efficace que celui moins linéaire et même par rapport aux arbres de décision. Le modèle linéaire est même plus régulier sur ces polynômes contrairement au degré 2.

Deuxième jeu de données

Nombre de données de test	Graphique	MAE réseau classique	Graphique	MAE réseaux moins linéaire
100 (1%)	Figure 1	8.595	Figure 5	9.063
1 000 (10%)	Figure 2	8.227	Figure 6	8.683
5 000 (50%)	Figure 4	8.084	Figure 7	8.547
10 000 (100%)	Figure 4	8.104	Figure 8	8.554

Graphiques de l'évolution de MAE (axe : OY) en fonction du nombre d'epochs (axe : OX) pour les réseaux de neurones



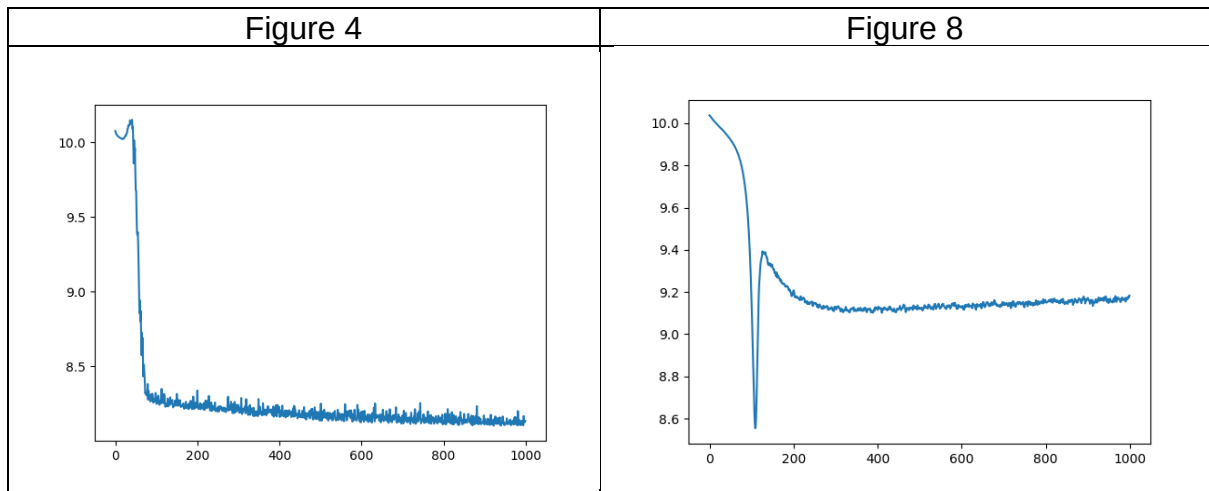
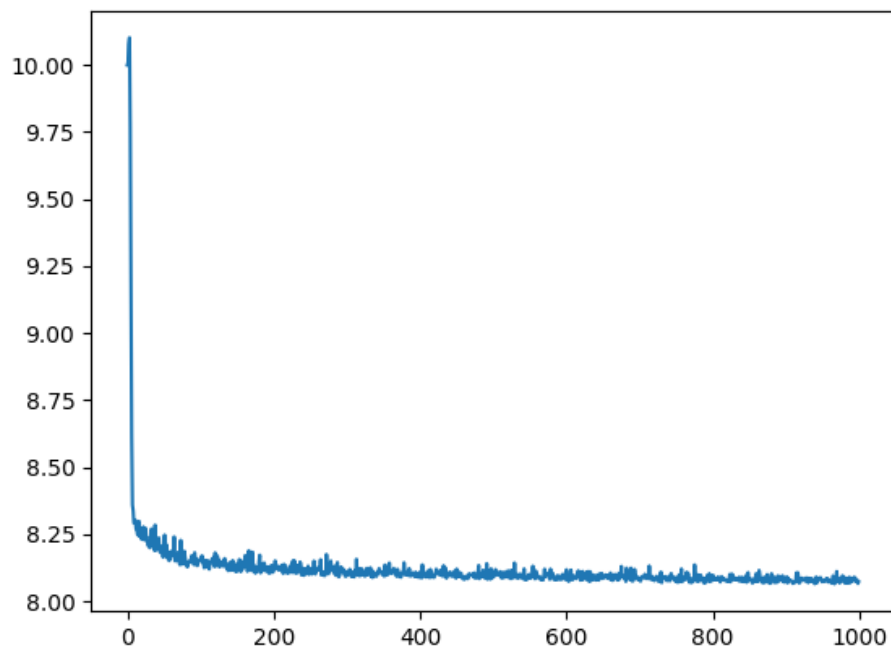


Figure 9 Test avec 100 000 données d'entraînement et 50 000 de test (50%) sur un réseau linéaire : 8.064



La valeur étonnante disparaît quand on prend le second jeu de données, une fois encore le modèle linéaire est plus performant et le modèle entraîné sur l'ensemble des données ressort avec le meilleur résultat.

Discussions autour des résultats :

J'ai éliminé les valeurs aberrantes et considéré le meilleur résultat possible que j'ai ramené en pourcentage sur l'étendue de possibilités.

Pour le degré 5 les possibilités allaient de -20 à 20 et les modèles tournait autour de 8.5 d'écart moyen avec la valeur qu'ils devaient trouver ce qui nous fait 850/41 d'écart moyen soit environ 20.73%. Pour le degré 2, c'est la même chose avec une étendue allant de -5060 à 5060.

	Solutions degré 2	Solutions degré 5
Arbre de décision	16.60%	21.58%
<i>random forest</i>	16.68%	22.28%
Réseaux de neurones linéaires	16.59%	20.16%
Réseaux de neurones moins linéaires	16.46%	21.37%

Ainsi, on peut constater que les réseaux de neurones sont légèrement plus efficaces et que cet écart se creuse plus la tâche est complexe, ce qui colle avec la théorie. De plus, on constate que la non-linéarité du modèle prônée pour améliorer l'efficacité ne fonctionne pas toujours. Il faut adapter cette linéarité selon chaque problème.

Nous retiendrons également l'importance du jeu de données de test qui influe beaucoup sur les résultats. Ainsi, plus il est petit, plus les résultats sont aléatoires. C'est pourquoi il est important de prendre un jeu de données assez large d'au moins 50% de la quantité de données d'entraînement, avec l'analyse que j'ai faite de ces différents cas. Peut-être faudrait-il procéder par dichotomie sur les tailles d'échantillon comprise entre 10% et 50% pour affiner cette indication.

Nous avons pu observer graphiquement la forme typique du résultat de modèles d'arbres de décisions : la courbe propose un optimum très rapidement sans nécessiter des profondeurs très importantes pour être efficaces. On observe également qu'au-delà de cette profondeur optimale les modèles perdent en précision.

En ce qui me concerne, je ne suis pas du tout satisfait des résultats pour les réseaux de neurones en rapport avec le temps de calcul nécessaire (alors que j'ai réduit l'échantillon à 10 000 cela me prenait 10 fois plus de temps qu'un modèle à arbre de décision à 100 000). A l'origine, j'attendais des résultats bien meilleurs pour ces derniers en raison du temps d'entraînement bien plus long, et que j'avais donc supposé bien plus complet.

Enfin, mon sujet étant d'étudier la capacité de modèles informatique de type intelligence artificielle à résoudre un problème mathématique, je constate que ces résultats sont beaucoup trop imprécis, ce qui n'est pas acceptable mathématiquement parlant.

Conclusion

En conclusion, selon moi, ces modèles en l'état ne sont pas capables de résoudre des polynômes. Cependant, ce sont des modèles classiques de l'intelligence artificielle, et ils ne sont pas exploités à leurs maximums pour des raisons de matériel (par exemple, la puissance d'un ordinateur permettant de réaliser ces calculs n'est aujourd'hui pas accessible à un étudiant). De plus, j'ai décidé d'étudier les modèles les plus classiques mais il existe probablement des modèles optimisés spécialement pour les mathématiques, auquel je n'ai pas eu accès.

Enfin, j'aimerais conclure concernant l'intelligence artificielle en elle-même. En ce moment, nous entendons beaucoup parler de Chat GPT par exemple ainsi que d'autres intelligences artificielles qui réalisent des tâches surprenantes. Après cette étude, bien que des IA telle que chat GPT soient infiniment plus puissantes que celles que j'ai fait fonctionner, je ne pense pas que confier tous nos travaux à ce type de programmes, qui travaillent par approximation, soit une bonne idée. Nous commettons souvent l'erreur de penser que tout ce qui est informatique est plus juste, plus exacte, et plus précis qu'un travail humain surtout en mathématique, mais la preuve en est que cela ne fonctionne pas toujours et qu'il faut faire attention au pouvoir que l'on accorde à un outil. Selon moi, ces IA doivent rester des outils de travail comme des logiciels. Je pense qu'elles ne doivent pas nous remplacer dans nos tâches intellectuelles, mais nous servir d'assistants pour nous soulager sur les tâches répétitives, être des outils et seulement des outils. Le problème éthique sera de décider où l'on place la limite.

Bibliographie

Livres :

NIELSEN, Michael A., 2015. Neural Networks and Deep Learning. [en ligne]. 2015.

Disponible à l'adresse : <http://neuralnetworksanddeeplearning.com/>

Sites internet :

KELDENICH, Tom, 2023. Fonction d'activation, comment ça marche ? Une explication simple. *Inside Machine Learning*. [en ligne]. 27 juin 2023. Disponible à l'adresse :

<https://inside-machinelearning.com/fonction-dactivation-comment-ca-marche-une-explication-simple/>

Deep Learning ou Apprentissage Profond : qu'est-ce que c'est ? 2023. *Formation Data Science | DataScientest.com*. [en ligne]. Disponible à l'adresse :

<https://datascientest.com/deep-learning-definition>

USER, 2023. Réseau de neurones : définition et fonctionnement. *Formation Data Science | DataScientest.com*. [en ligne]. 3 avril 2023. Disponible à l'adresse :

<https://datascientest.com/fonctionnement-des-reseaux-neurones>

Cours en lignes :

Learn Intro to Deep Learning Tutorials. [en ligne]. Disponible à l'adresse :

<https://www.kaggle.com/learn/intro-to-deep-learning>

Learn Intro to Machine Learning Tutorials. [en ligne]. Disponible à l'adresse :

<https://www.kaggle.com/learn/intro-to-machine-learning>

Vidéos :

3BLUE1BROWN, 2017a. *But what is a neural network ? | Chapter 1, Deep learning* [en ligne]. Vidéo. 5 octobre 2017. Disponible à l'adresse :

<https://www.youtube.com/watch?v=aircAruvnKk>

3BLUE1BROWN, 2017b. *Gradient descent, how neural networks learn | Chapter 2, Deep learning* [en ligne]. Vidéo. 16 octobre 2017. Disponible à l'adresse :

<https://www.youtube.com/watch?v=IHZwWFHWa-w>

3BLUE1BROWN, 2017c. *What is backpropagation really doing ? | Chapter 3, Deep learning* [en ligne]. Vidéo. 3 novembre 2017. Disponible à l'adresse :

<https://www.youtube.com/watch?v=Ilg3gGewQ5U>

3BLUE1BROWN, 2017d. *Backpropagation calculus | Chapter 4, Deep learning* [en ligne].

Vidéo. 3 novembre 2017. Disponible à l'adresse :

<https://www.youtube.com/watch?v=tIeHLnjs5U8>

JEDHA BOOTCAMP, 2020. *Quelles différences entre Machine Learning et Deep Learning ?* [en ligne]. Vidéo. 4 avril 2020. Disponible à l'adresse :

<https://www.youtube.com/watch?v=c5XLjUBEOa8>

Documentations :

1.11. Ensemble methods, [sans date]. *scikit-learn*. [en ligne]. Disponible à l'adresse :

<https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees>

3.11.4 Documentation, [sans date]. [en ligne]. Disponible à l'adresse :

<https://docs.python.org/3/>

Module : tf.keras | TensorFlow v2.12.0, [sans date]. [en ligne]. Disponible à l'adresse :

https://www.tensorflow.org/api_docs/Python/tf/keras

sklearn.tree.DecisionTreeRegressor, [sans date]. *scikit-learn*. [en ligne]. Disponible à

l'adresse : [https://scikit-](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html)

[learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html)