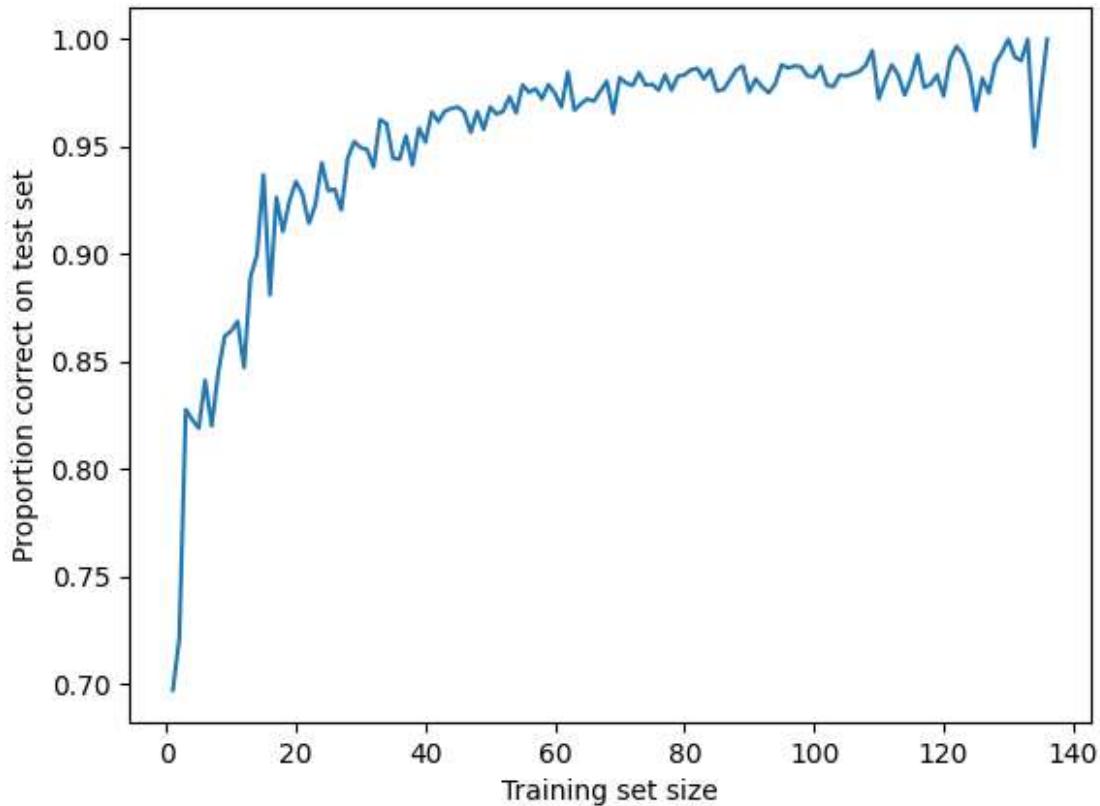


# Arbre de décision

## Courbe d'apprentissage



On constate que plus on a de données en entrainement plus on performe bien. \ Aussi, on observe deux grandes phases sur ce graphe:

- Une augmentation fulgurante pour les premières tailles de notre jeu de données;
- Et une phase où les performances n'évoluent que très peu.

Cette 2ème phase nous permet donc de déduire qu'on pourrait réduire le jeu d'entraînement pour gagner en temps sans toute fois perdre en terme de généralisation sur le jeu de test.

## Comparaison avec le classifieur scikit-learn

Nous obtenons des résultats similaires sur le jeu de données iris. \ Sur le de données wine, le classifieur de sklearn perfome mieux que le notre. \ Tandis que sur le jeu de données abalone, c'est notre classifieur qui a les meilleures performances.

Les résultats obtenus sont regroupés dans les tableaux ci-bas:

Custom

Accuracy	Precision	Recall	F1-score
----------	-----------	--------	----------

<b>iris</b>	[100, 91.67, 91.67]	[100, 75, 100]	[100, 100, 80]	[100, 85.71, 88.89]
<b>wine</b>	85.43	79.89	86.14	82.90
<b>abalone</b>	[93.54, 84.05, 90.51]	[69.80, 80.25, 62.79]	[74.29, 88.54, 66.26]	[71.97, 89.38, 64.48]

## Sklearn

	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>iris</b>	[100, 91.67, 91.67]	[100, 75, 100]	[100, 100, 80]	[100, 85.71, 88.89]
<b>wine</b>	88.15	87.62	88.49	87.91
<b>abalone</b>	[94.25, 83.65, 89.39]	[72.67, 90.28, 58.43]	[77.86, 87.90, 63.80]	[75.17, 89.08, 61.00]

## Comparaison avec élagage

Les résultats ci-bas montrent l'évaluation de notre classifieur après élagage.

## Prune

	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>iris</b>	[100, 91.67, 91.67]	[100, 75, 100]	[100, 100, 80]	[100, 85.71, 88.89]
<b>wine</b>	85.43	79.89	86.14	82.90
<b>abalone</b>	[92.98, 83.25, 90.35]	[66.25, 90.05, 62.65]	[75.71, 87.59, 63.80]	[70.67, 88.89, 63.22]

On constate, qu'on a une dégradation des performances sur le jeu de test. Ce qui signifie qu'on a probablement mal élagué notre arbre. Mais on n'a pas eu assez de temps pour vérifier d'où vient le problème

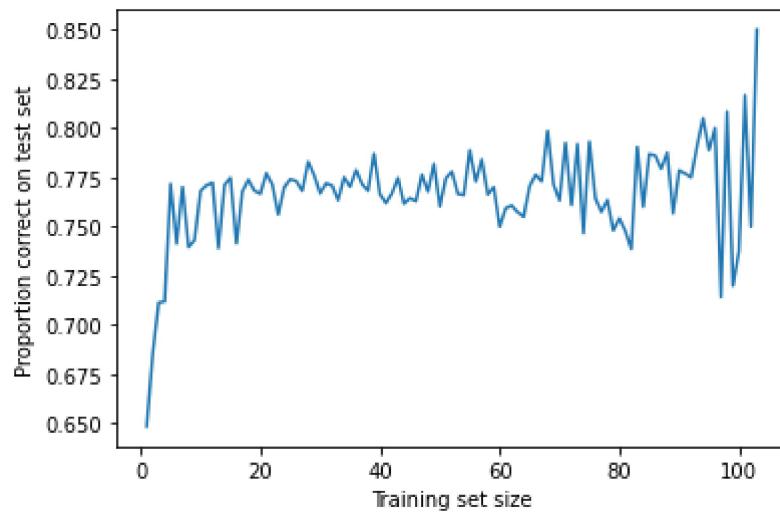
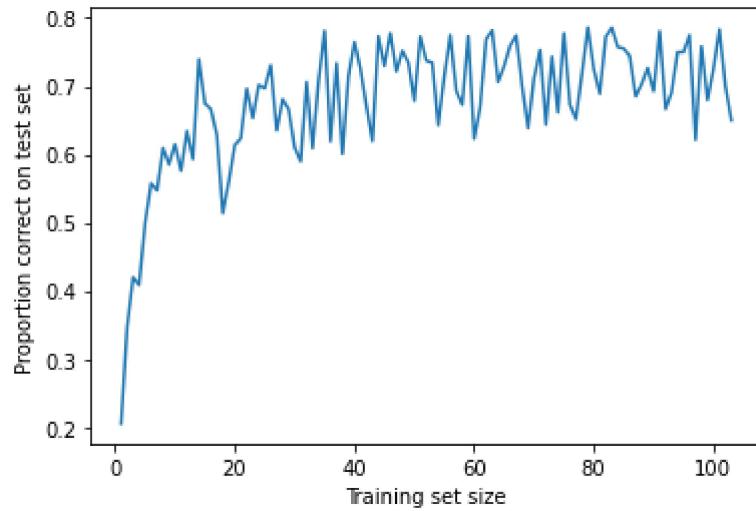
# Réseaux de Neurones Artificiels

## Initialisation des poids du réseau de neurones

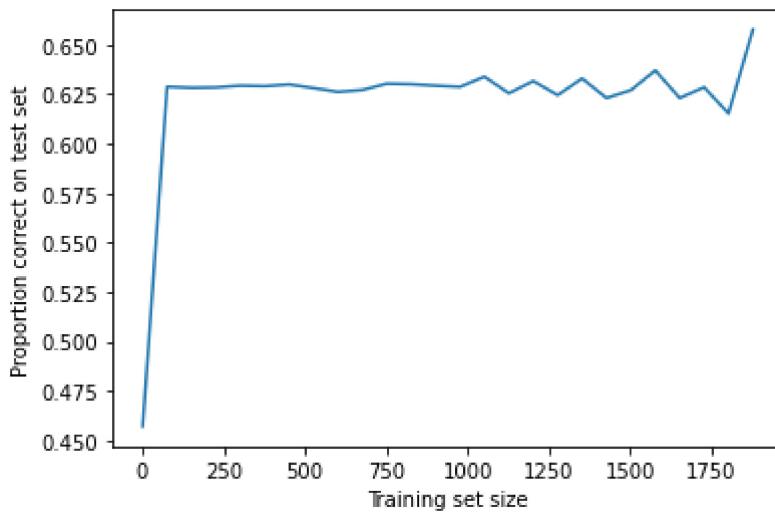
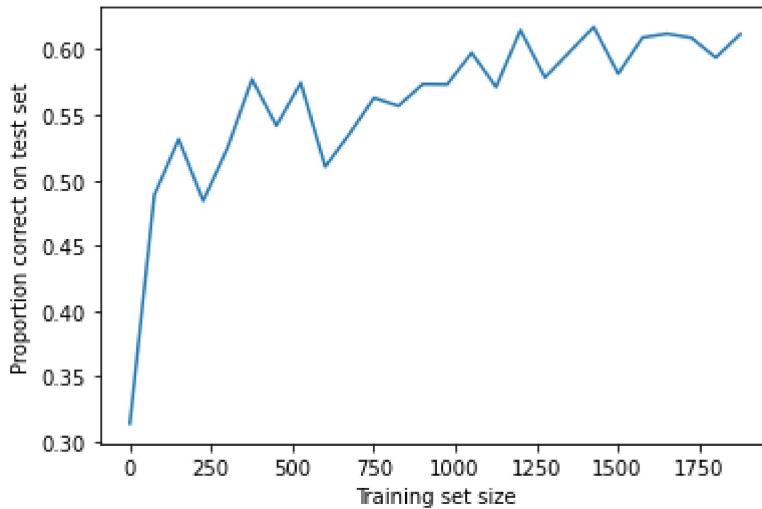
Nous avons choisi d'utiliser la technique d'initialisation Normal de Xavier. Elle consiste à initialiser les poids de façon aléatoire suivant une loi Normal avec une moyenne de 0 et un écart-type de

$$\sigma = \sqrt{\frac{2}{\text{Nombre d'input} + \text{Nombre d'output}}}.$$

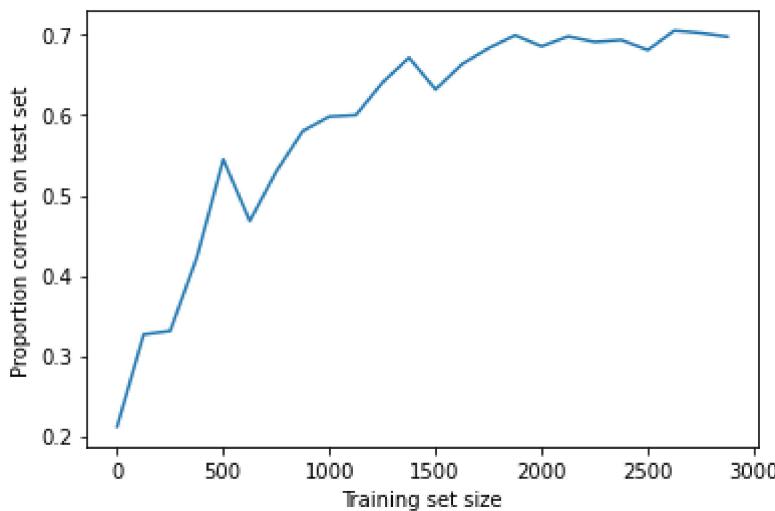
Toutes les figures suivantes représentent les courbes d'apprentissage des réseaux neurones en utilisant l'initialisation Normal de Xavier et l'initialisation des poids à zéros pour les trois jeux de données.

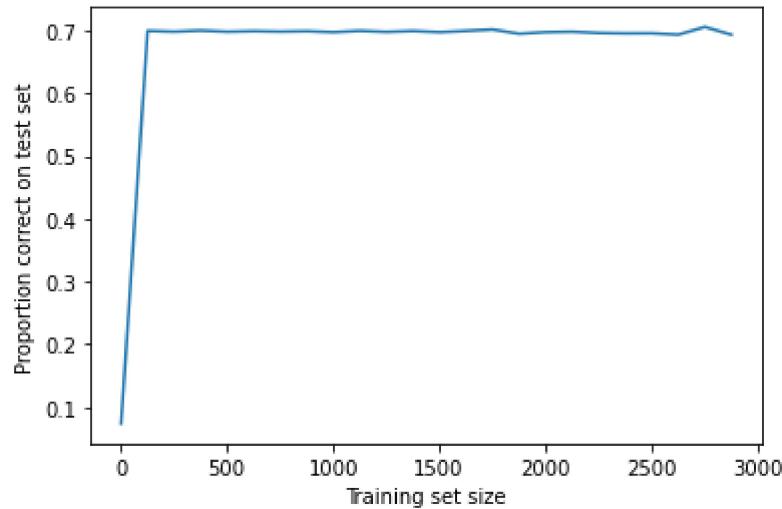


En comparant les figures 1 et 2, on remarque que l'apprentissage avec l'initialisation de Xavier est beaucoup plus constant qu'avec l'initialisation à zéro. Cependant, l'initialisation à zéro commence avec une meilleure prédiction. Le modèle semble ensuite rester constant sans s'améliorer, mais il y a beaucoup d'inconstance lorsque le jeu de donnée devient plus grand. Pour ce qui est de l'initialisation de Xavier, il semble toujours y avoir une certaine amélioration du modèle même si elle est de moins en moins grande.



Les apprentissages pour les figures 3 et 4 utilisent le même learning rate, mais on voit que l'initialisation à zéro converge beaucoup plus rapidement tandis que l'initialisation de Xavier s'améliore constamment. L'initialisation à zéro risque donc de converger à un taux plus faible.





Sur le jeu de données Abalone, on observe la même chose que sur le jeu de donnée Wine. C'est-à-dire que l'initialisation à zéro converge trop rapidement donc elle pourrait mal converger.

## Évaluation

Pour les jeux de données Iris, Wine et Abalone, on utilise respectivement 3, 8 et 6 couches cachées. On utilise un learning rate de 0.01 pour les 3 modèles.

	<b>Accuracy</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>	<b>Temps d'entraînement</b>	<b>Temps d'évaluation</b>
<b>iris</b>	[100, 58, 58]	[100, 37.5, nan]	[100, 100, 0]	[100, 54.54, 0]	18.79	0.00523
<b>wine</b>	71.48	72.25	83.89	77.63	347.356	0.13299
<b>abalone</b>	[87, 79.27, 91.79]	[50, 78.62, 97.435]	[2.45, 99.79, 27.14]	[4.678, 87.95, 42.46]	491.70	0.1595

Matrices de confusion Iris:

$$\begin{bmatrix} 8 & 0 \\ 0 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 5 \\ 0 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 0 \\ 5 & 0 \end{bmatrix}$$

Matrice de confusion Wine:

$$\begin{bmatrix} 178 & 154 \\ 77 & 401 \end{bmatrix}$$

Matrices de confusion Abalone :

$$\begin{bmatrix} 1087 & 4 \\ 159 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 45 & 258 \\ 2 & 949 \end{bmatrix}$$

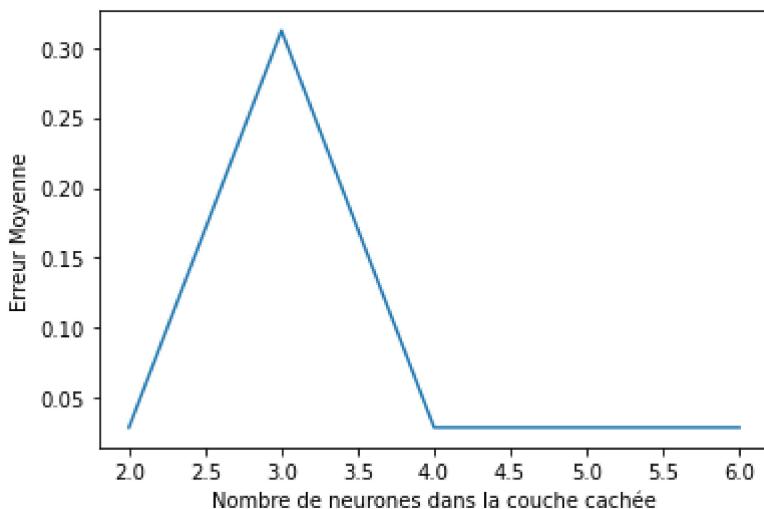
$$\begin{bmatrix} 1113 & 1 \\ 102 & 38 \end{bmatrix}$$

## Recherche d'hyperparamètres

### Choix du nombre de neurones dans la couche cachée

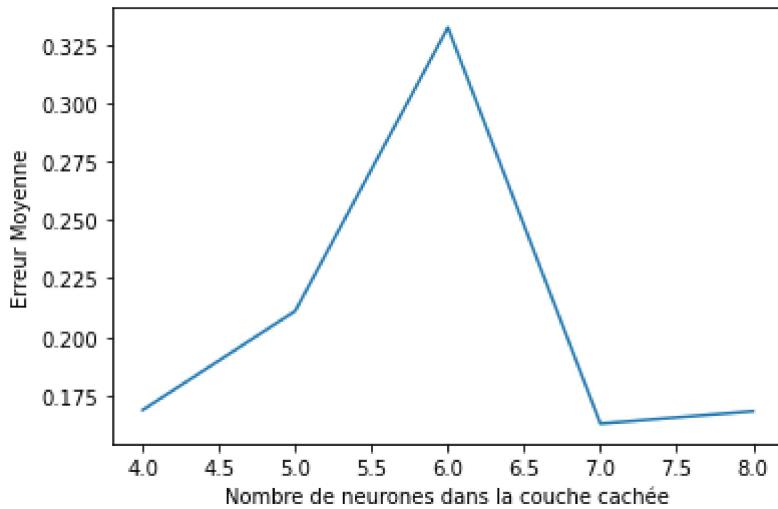
#### Iris

La figure ci-dessous montre que, pour le jeu de données Iris, l'erreur est minimale lorsqu'on utilise 2, 4, 5 ou 6 neurones. Nous avons décidé d'en utiliser 4.



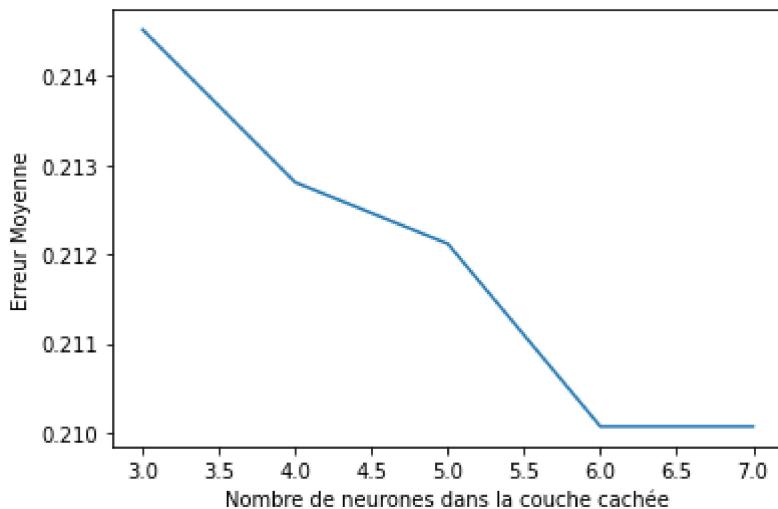
#### Wine

La figure ci-dessous montre que, pour le jeu de données Wine, l'erreur est minimale lorsqu'on utilise 7 neurones. C'est donc le nombre de neurones que nous avons utilisé.



## Abalone

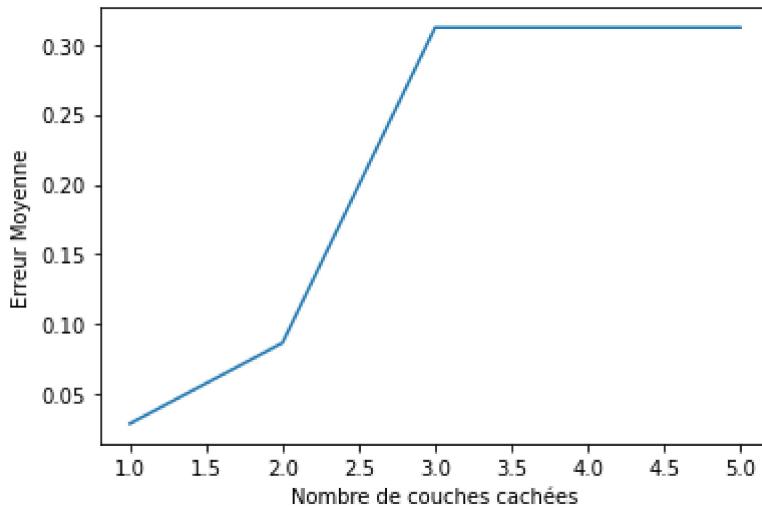
La figure ci-dessous est pour le jeu de donnée Abalone. On remarque que plus il y a de neurones, plus l'erreur est petite. Nous utilisons 6 neurones selon ce graphique.



## Choix du nombre de couches cachées

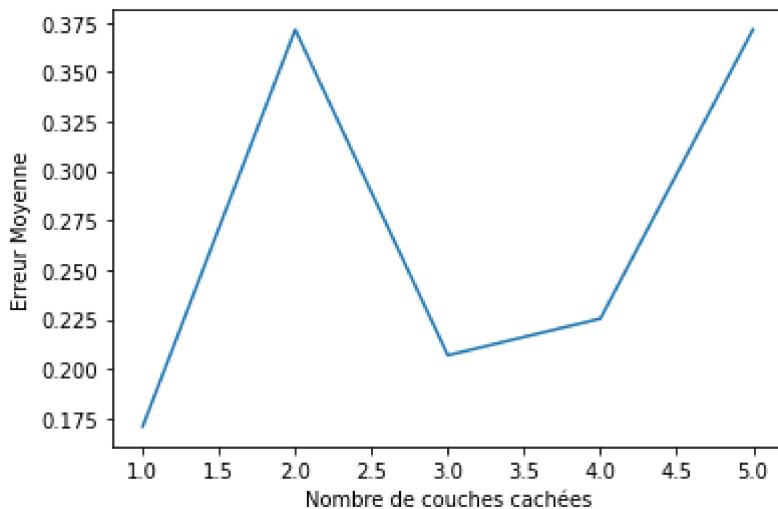
### Iris

Selon la figure ci-dessous, on utilise une seule couche cachée pour le jeu de donnée Iris. En effet, plus il y a de couches, plus l'erreur est grande. On utilise donc un réseau ayant une couche cachée et 4 neurones.



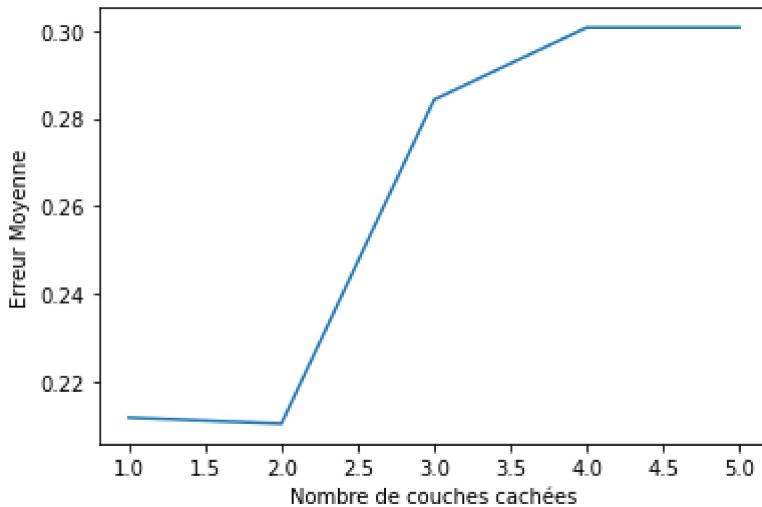
## Wine

Encore une fois, la figure semble montrer que le meilleur réseau utilise une seule couche cachée. On utilise donc un réseau ayant une couche cachée et 7 neurones.



## Abalone

On observe la même tendance par rapport au fait que plus il y a de couche, plus l'erreur est grande. On utilise donc un modèle avec 2 couches cachées et 6 neurones par couche.



## Phénomène de Vanishing Gradient

Le Vanishing Gradient est un phénomène qui se produit lorsqu'il y a beaucoup de couches dans un réseau de neurones. Ce phénomène rend l'apprentissage beaucoup plus lent. En effet, plus il y a de couches, moins la variation dans les premiers poids est grande. C'est le cas car, pour obtenir le gradient d'un poids, il faut passer par chacun des autres noeuds. On a donc une chaîne de multiplication de plusieurs petites valeurs. Donc plus un poids est loin de la dernière couche, moins il change rapidement.

Dans les figures pour les choix de nombres de couches, on a pu observer qu'un plus grand nombre de couches apportait généralement plus d'erreur. C'est peut-être dû au fait que nous avons utiliser le même nombre d'itérations maximum pour tous les nombres de couches. Les modèles avec plus de couches auraient probablement nécessité plus d'entraînement.

## Entraînement et test

Avec les choix de nombres de neurones et de nombres de couches, on obtient les résultats suivant :

	Accuracy	Precision	Recall	F1-score
<b>iris</b>	58.33	45.83	66.66	51.51
<b>wine</b>	81.73	81.28	80.70	80.95
<b>abalone</b>	82.70	74.96	64.79	68.72

Matrices de confusion Iris :

$$\begin{bmatrix} 8 & 0 \\ 0 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 5 \\ 0 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 7 & 0 \\ 5 & 0 \end{bmatrix}$$

Matrice de confusion Wine :

$$\begin{bmatrix} 249 & 83 \\ 65 & 413 \end{bmatrix}$$

Matrices de confusion Abalone:

$$\begin{bmatrix} 1089 & 25 \\ 55 & 85 \end{bmatrix}$$

$$\begin{bmatrix} 151 & 152 \\ 65 & 886 \end{bmatrix}$$

$$\begin{bmatrix} 1051 & 40 \\ 97 & 66 \end{bmatrix}$$

## Comparaison entre les algorithmes expérimentés dans le cours

	Iris				Wine				Abalone			
	KNN	CNB	Tree	Neural Net	KNN	CNB	Tree	Neural Net	KNN	CNB	Tree	Neural Net
Accuracy	91,30%	89,41%	94,44%	58,33%	84,94%	80,74%	85,43%	82,72%	89,31%	61,37%	89,37%	82,46%
Training time	5ms	3ms	179ms	499ms	130ms	7ms	157ms	1684ms	100ms	28ms	251ms	3884ms
Prediction time	78ms	32ms	6,97ms	0,75ms	31667ms	318ms	25ms	42ms	82249ms	648ms	38ms	53ms

L'arbre de décision a les meilleurs résultats pour tous les jeux de données tout en ayant des temps d'entraînement et de prédiction raisonnablement petits. Le KNN performe également très bien dans tous les jeux de données, mais il a un temps de prédiction beaucoup plus long. L'arbre est donc préférable pour nos jeux de données.

Pour le jeu de données Iris, le réseau de neurones a très mal performé. C'est probablement parce que ce genre de modèle nécessite beaucoup de données pour bien s'entraîner. Ce modèle est également très long à entraîner et il aurait peut-être performé mieux avec davantage d'entraînement.

Contrairement aux réseaux neurones, le classifieur naïf de Bayes semble être meilleur sur les plus petits jeux de données comme Iris. Il performe très mal sur abalone.

## Conclusion

Ce projet nous a permis d'apprendre comment fonctionne les algorithmes de KNN, de CNB, d'arbres de décisions et de réseaux neurones.

Nous avons également appris des techniques qui nous permettent de combattre l'overfitting, comme l'élagage des arbres et le choix du nombre d'itérations dans les réseaux neurones.

Nous avons appris à utiliser la validation croisée pour faire la sélection d'hyperparamètres de nos modèles.