ML

Machine Learning

Rapport de projet

Étudiants:
Balkis Bouthaina DIRAHOUI

Numéros:
21113733

Mai 2022

Table des matières

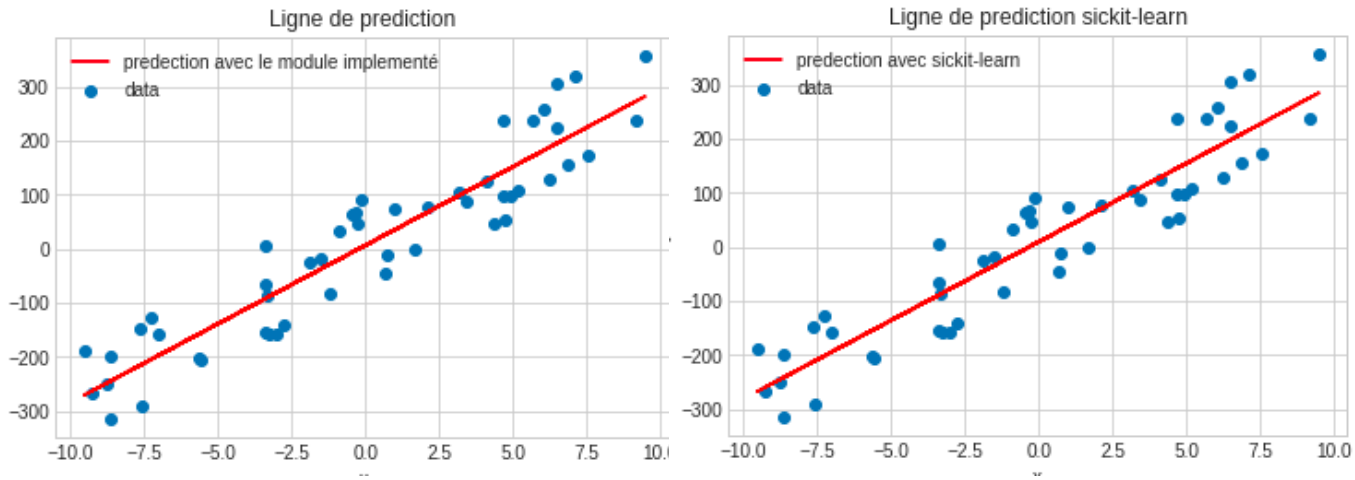
1 Introduction

Ce document décrit le travail réalisé dans le cadre d'un projet académique dans le but de concrétiser et améliorer nos connaissances pratiques et théoriques récoltées dans l'Unité d'Enseignement "Machine Learning" par le développement d'un réseau de neurones en version modulaire.Pour se faire, nous avons développés les modules suivant :

- ***Module linéaire*** : dans le but d'effectuer une tache d'apprentissage, celui-ci permet d'appliquer une transformation linéaire aux entrée. (e.g $y= \alpha\ x + \beta$).
- ***Module non-linéaire***: Afin d'appliquer une non-linéarité (données non linéairement séparables[1]), le module non-linéaire sera introduit grâce aux fonctions d'activations non-linéaires comme $TanH\ \sigma(x) = \frac{e^x - e^{-x}}{e^z + e^{-z}}$ ou $Sigmoid\ \sigma(x) = \frac{1}{1+e^{-x}}$.
- ***Module séquentiel***: Ensuite, dans l'objectif d'ajouter les modules (linéaires, non-linéaires) en série (séquence) le module séquentiel sera implémenté. Nous pouvons alors avoir par exemple:

  $Network = Sequential(Linear(100, 80), TanH(), Linear(50, 30), Sigmoid())$
- ***Module Multi-Classe***: Dans cette partie, comme son nom l'indique, la sortie des modèles (module) de classification ne seront plus que binaire. Ainsi, dans le but d'avoir plusieurs sortie du reseau, nous allons utiliser la fonction d'activation Softmax $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$
- ***Module Auto-encodeur*** Afin de compresser les données, nous avons implémenter un auto-encodeur (réseau de neurones qui réduit les dimensions des entrées).
- ***Module convolutionelle*** : Enfin, nous allons implémenter une couche convolutionnelle (très utilisé dans le traitement d'image).

[1] c.a.d il n'existe pas d'hyperplan qui sépare les données sans erreur.



(a) Nos Prédictions

(b) Prédictions sickit-learn

Figure 2.1: Comparaison du module developpé et du module sickit-learn

2 Module Linéaire

Dans cette section, nous testons notre implémentation du module linéaire.Nous effectuons ainsi les expérimentations nécessaires pour ce dernier.

2.1 Expérimentations

Notre module vs Sickit-learn Nous comparons ainsi notre module linéaire avec celui de la bibliothèque sickit-learn (Linear Regression) [2].

La figure 2.1 illustre la comparaisons effectué, nous pouvons alors remarquer que les deux droites sont parfaitement les mêmes.

Évolution du coût Dans cette partie, nous pouvons visualiser dans la figure 2.2 l'évolution du coût (MSELoss) selon le nombre d'itérations dans la descente de gradient.Nous pouvons ainsi remarquer que le module converge rapidement.

Néanmoins, le module linéaire ne peut être appliqué a tout types de données, notamment quand les données ne peuvent être séparées linéairement, nous détaillerons dans la prochaine section le module non-linéaire qui traite ce type de problèmes.

[2] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`

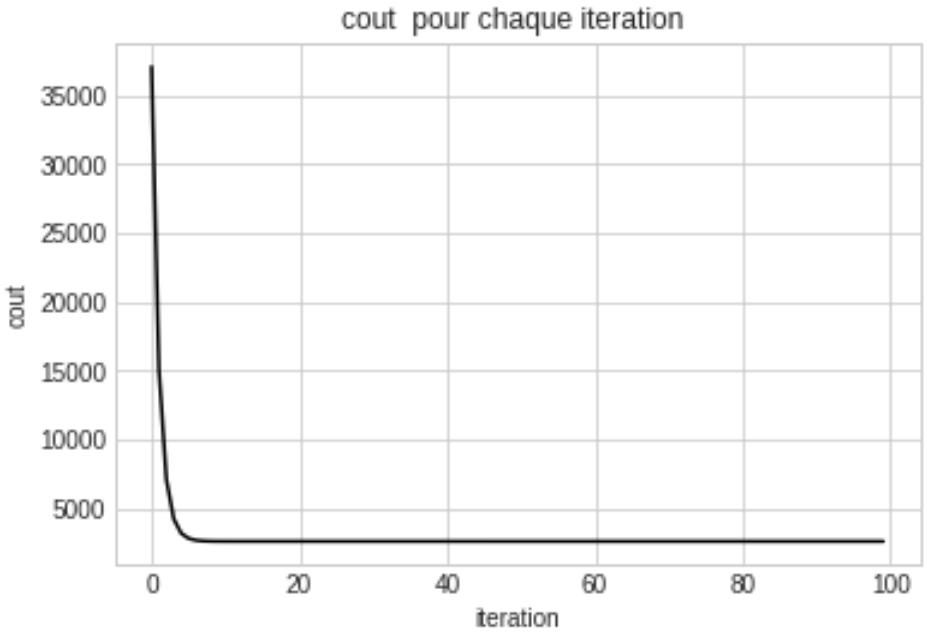


Figure 2.2: Variation du coût selon le nombre d'itérations

3 Module non-linéaire

3.1 Données linéairement séparables

Des données sont linéairement séparables, si nous pouvons trouver un hyperplan (une droite étant un hyperplan d'une dimension) séparateur tel qu'aucune donnée ne soit sur la frontière de décision [3].

La figure 3.1, représente un exemple de données séparables linéairement et des données non-séparables linéairement en classification binaire,ainsi, nous pouvons remarquer que la frontière n'est effectivement pas linéaire quand il s'agit de problèmes comme le problème du XOR (points ayant comme classe 1 les points $(1,1),(-1,-1)$ et comme classe 2 les points $(1,-1),(-1,1)$).

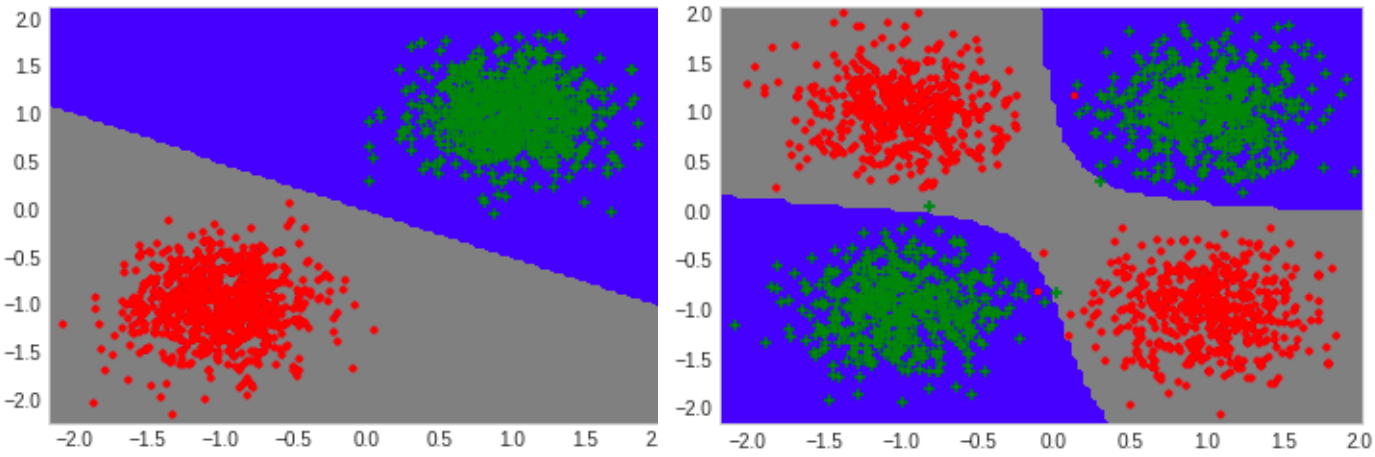
3.2 Expérimentations

Nous avons réalisé la figure 3.1 avec la fonction *gen_ arti* (pour plus de details voir l'annexe A).

Linéairement séparable La figure 3.1a a été généré grâce aux type=0 de la fonction, et a été génère avec le modèle : $model = (Linear(2,70), TanH(), Linear(70,1), Sigmoid())$ .Nous pouvons remarquer une frontière de décision qui sépare parfaitement les deux classes.

Non-linéairement séparable: La figure 3.1b illustre une frontière de décision pour des données non-séparables linéairement (généré avec la fonction cité précédemment pour type = 1) en ayant utilisé le modèle précèdent.

[3] la zone ou le classifieur est dis "confus", et qui sépare, dans le cas d'une classification binaire, les deux classes.



(a) Données linéairement séparables (b) Données non-linéairement séparables

Figure 3.1: Données non-linéairement séparable et linéairement séparable

4 Module Séquentiel

Dans cette section, nous implémentons un module Séquentiel permettant d'ajouter les modules en série. Nous implémentons également des fonctions d'optimisation (SGD,OPTIM (pour plus de détails voir l'annexe A)) qui pourrait nous permettre d'améliorer les performances du modèle. Les expérimentations de cette section seront fait comme suit :

- Nous allons dans un premier lieu visualiser les prédictions du modèle (frontières de décision).
- Dans un deuxième lieu, nous effectuerons des expérimentations sur les fonctions d'optimisation (variation du nombre d'exemples du batch,nombre d'exemples).

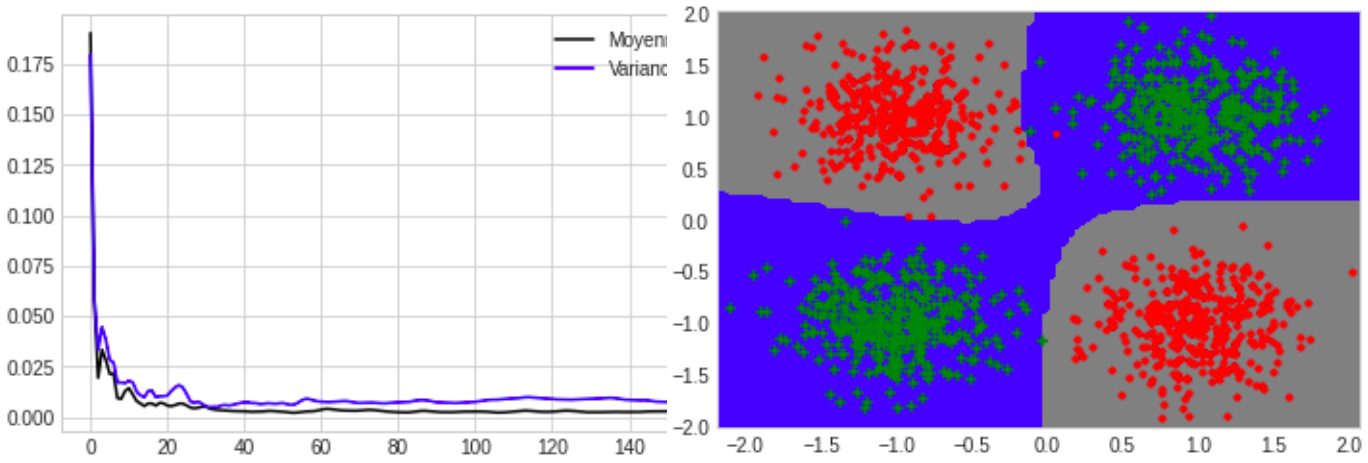
4.1 Expérimentations

4.1.1 Frontières de décisions et optimisations

Nous pourrons visualiser ici les différentes frontières de décisions générées selon les le nombre d'exemples total ainsi que le nombre d'exemple d'un batch (*batch size*).Nous pouvons ainsi faire des expérimentations sur des problèmes vu en cours :

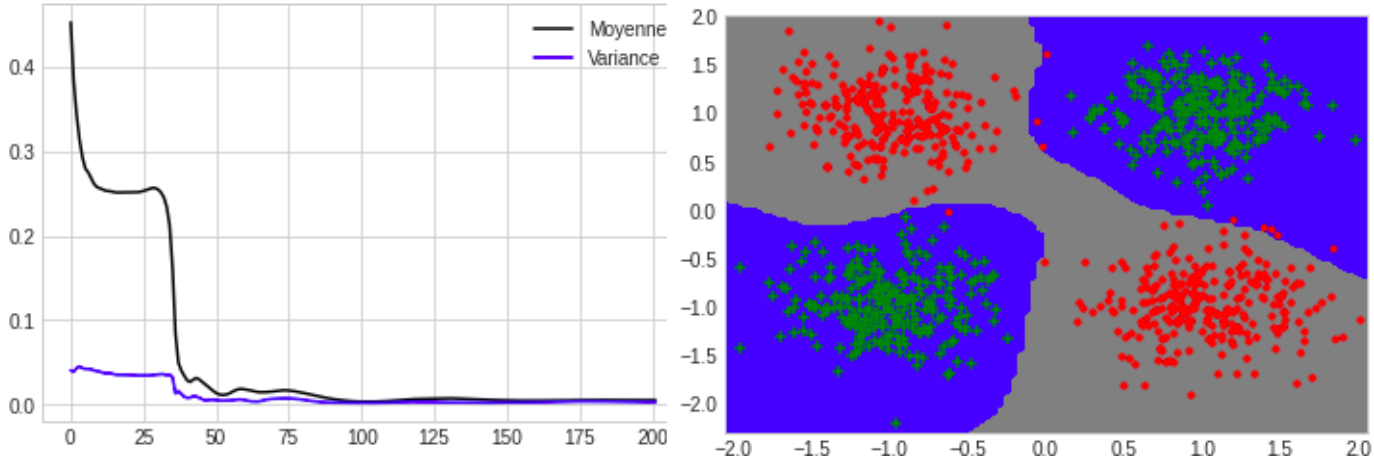
XOR Dans cette partie, les expérimentations sont sur des données de type = 1.Nous pouvons remarquer, comme illustré dans la figure 4.1, qu'une taille de batch égale a 10 améliore nettement les performances (et stabilise la moyenne et variance des coûts).Néanmoins, dans la figure 4.2 avec un taille de batch = 100, nous pouvons voir que la moyenne et la variance ne sont pas très proches dans le début des itérations, et une baisse de performances.

Échiquier Dans cette partie, les expérimentations sont sur des données de type =2. Les figures 4.3 et 4.4 illustrent les résultats du plot de notre modèle séquentiel, nous pouvons voir que quand la taille du batch est égale a 10, le modèle a de meilleurs performances.Nous pouvons également remarquer qu'il sépare les données d'une façon pas trop mal.



(a) Évolution de la moyenne et variance des coûts (b) Frontière de décision

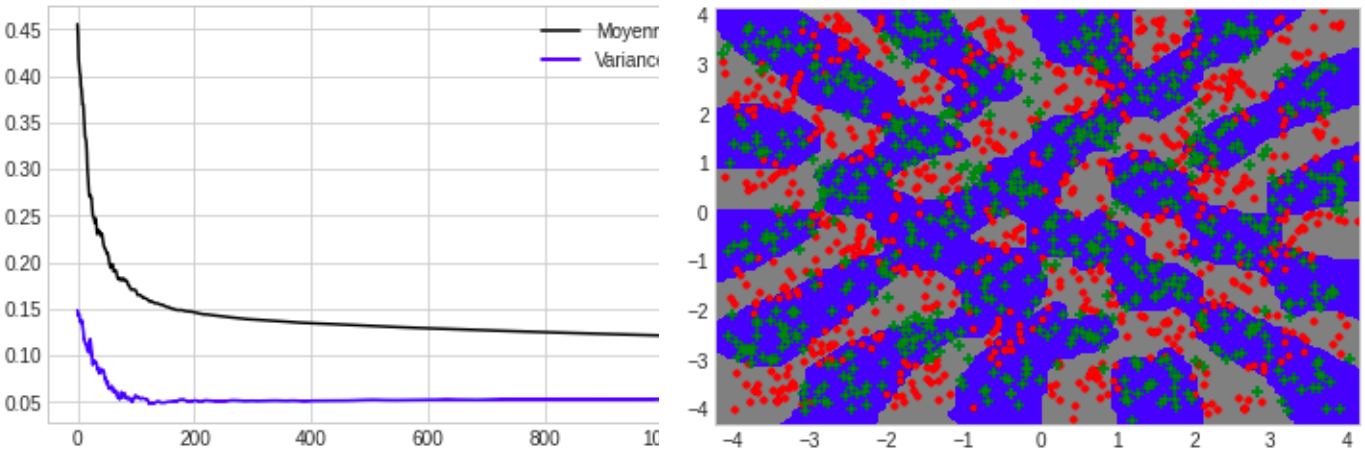
Figure 4.1: Frontière xor avec un nombre d'exemples du batch =10



(a) Évolution de la moyenne et variance des coûts (b) Frontière de décision

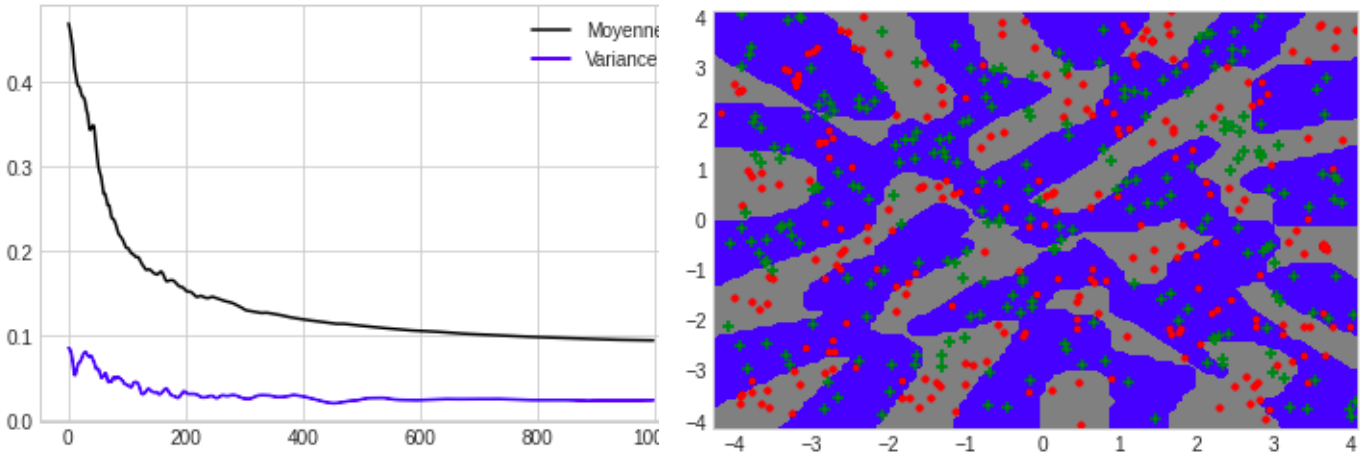
Figure 4.2: Frontière xor avec un nombre d'exemples du batch =100

Néanmoins, les problèmes de classification étudiés jusqu'à présent ne sont que des problèmes de classification binaire, nous pourrons voir dans la section suivante, un exemple de classification a plusieurs classes.



(a) Évolution de la moyenne et variance des coûts (b) Frontière de décision

Figure 4.3: Frontière avec un nombre d'exemples du batch =10



(a) Évolution de la moyenne et variance des coûts (b) Frontière de décision

Figure 4.4: Frontière avec un nombre d'exemples du batch =30

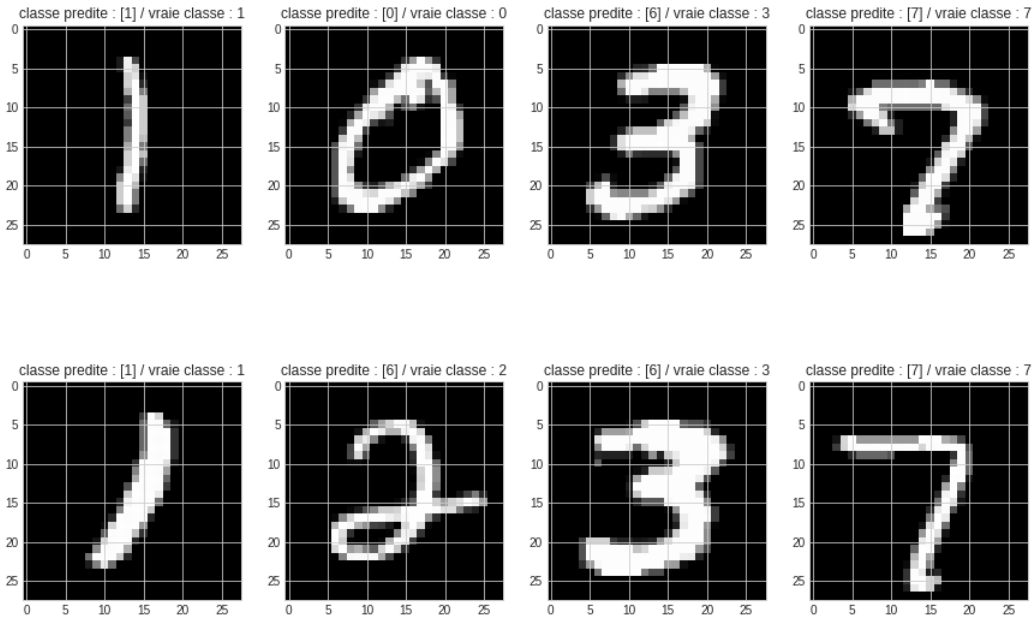


Figure 5.1: Prédictions sur données mnist avec maxiter=600 et pas=1e-4

Pas du gradient	**1e-1**	**1e-2**	**1e-3**	**1e-4**	**1e-5**	**1e-6**	**1e-7**	**1e-8**	**1e-9**	**1e-10**
Accuracy	0.69	**0.82**	0.73	0.75	0.48	0.40	0.29	0.19	0.16	0.08

Table 5.1: Performances du modèle selon le pas du gradient

5 Module multi-classe

Dans cette section, l'objectif est de classifier les données a plus de deux classes. Pour se faire, nous utiliserons comme fonction d'activation la fonction Softmax $\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K.$

Nous devons également utiliser une fonction de coût adapté, nous utiliserons alors, la fonction de coût *Cross-entropy* $CE = -\sum_{c=1}^{M} y_c \log(p_c)$.

5.1 Expérimentations

Les expérimentations effectués seront sur les données mnist[4] qui représentent un jeux de données de classification de chiffres manuscrits (pour plus détails voir le lien).Nous effectuerons dans nos expérimentations quelques variations d'hyperparametres, nous verrons dans un premier lieu l'impact du nombre d'itérations maximal sur les performances.Dans un deuxième lieu, nous ferons varier le pas du gradient.De plus, le réseau utilisé dans ce qui suit est le suivant (avec un batchsize fixé a 100.):

$Network = Sequential(Linear(input, 128), TanH(), Linear(128, 64), TanH(), Linear(64, output), SoftMax()))$

NB : la taille input de la la taille d'une image et output=10 le nombre de classes en sortie.

[4] `https://keras.io/api/datasets/mnist/`

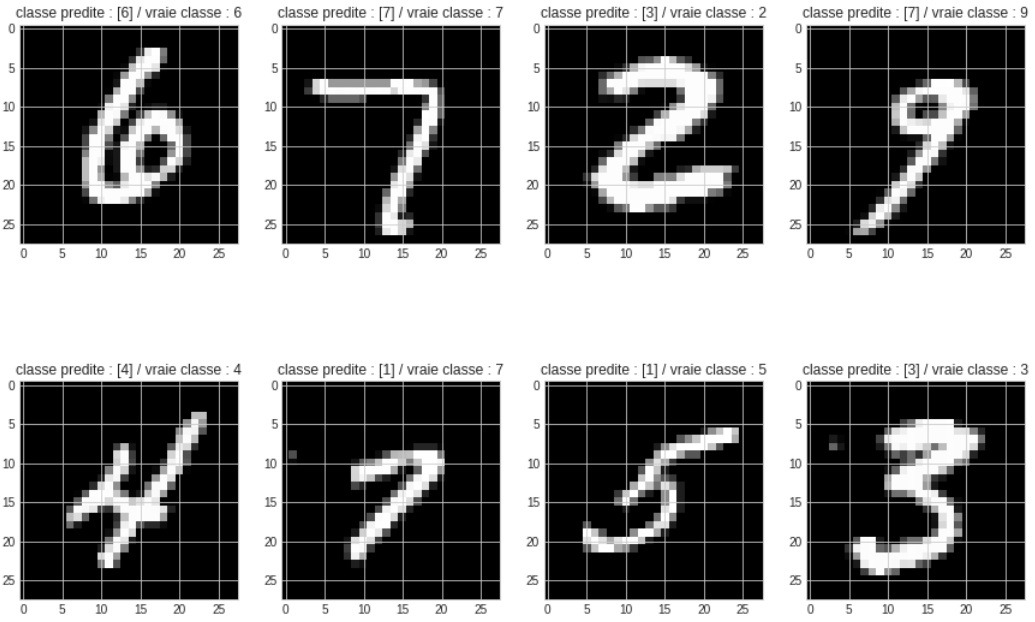


Figure 5.2: Prédictions sur données mnist avec maxiter=1500 et pas =1e-4

Variation du nombre d'itérations Dans cette section, nous étudions l'effet du nombre d'itérations maximal sur les performances de notre modèle.Les figures 6.1 et 5.2 illustrent, respectivement, les prédictions du modèle avec un nombre d'itérations égale a 600 et les prédictions de celui avec un nombre égale a 1500. Ces derniers ont pour performances (accuracy) respectivement 0.63 et 0.75, nous pouvons remarquer qu'augmenter le nombre d'itérations permet d'améliorer le modèle.Ceci, pourrait être expliqué par le fait qu'il ait plus de temps pour converger.

Variation du pas de gradient A présent, nous fixons le nombre d'itérations maximal a 1500, et faisons varier le pas du gradient.Comme nous pouvons le remarquer dans la table 5.1, un pas de gradient trop grand pourrait faire diverger le modèle.Nous pouvons également visualiser les prédiction du meilleur score pour un pas égal a 1e-2 dans la figure 5.4, ainsi que le pire score pour un pas égal a 1e-10 dans la figure 5.3.

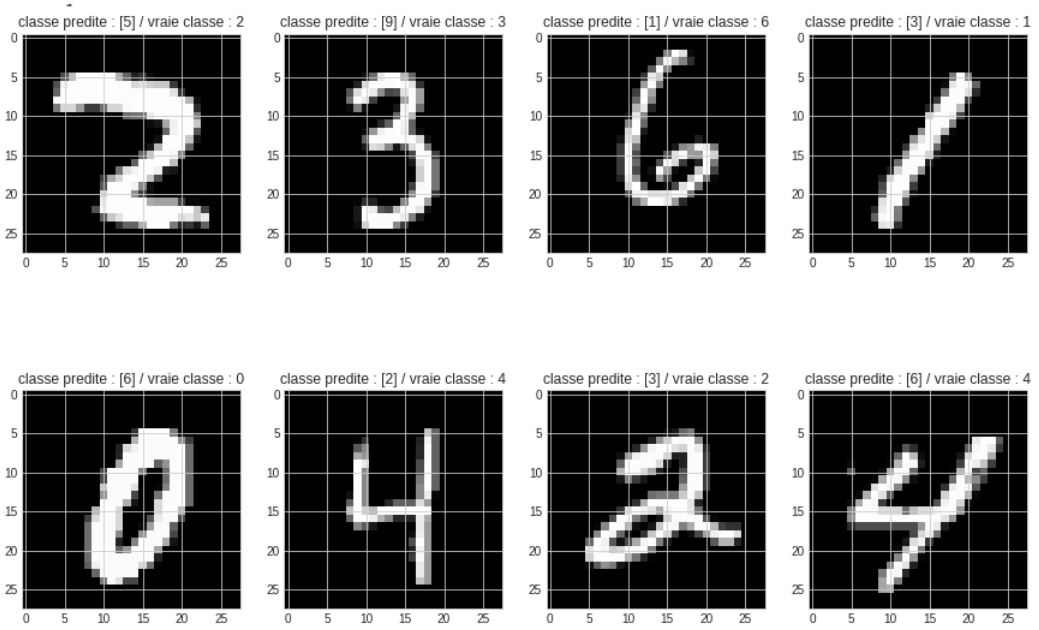


Figure 5.3: Prédictions sur données mnist avec maxiter=1500 et pas =1e-10 (pire score)

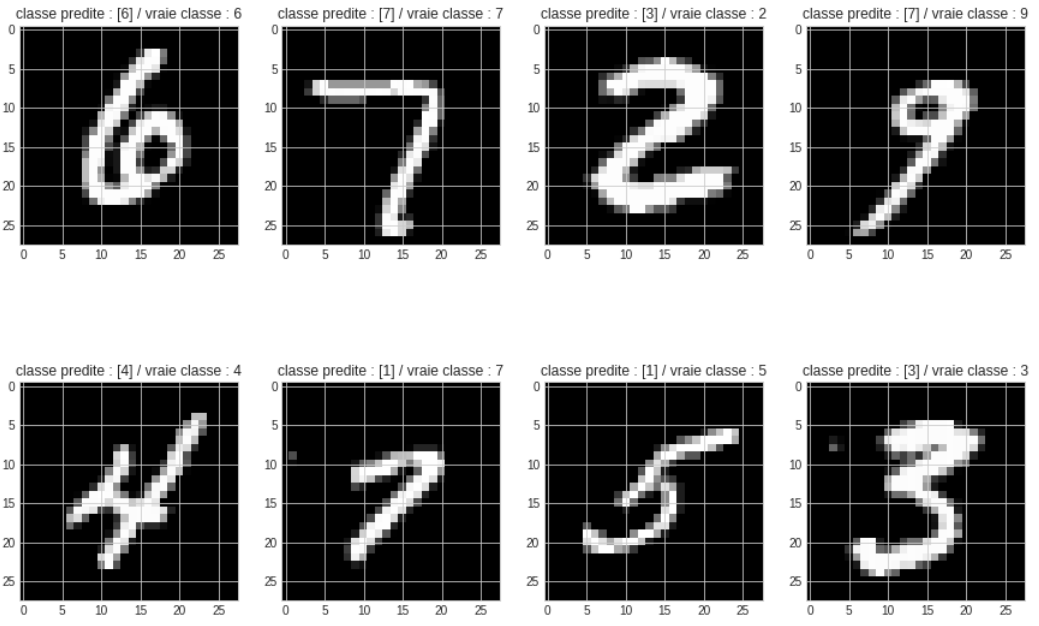


Figure 5.4: Prédictions sur données mnist avec maxiter=1500 et pas =1e-2 (meilleur score)

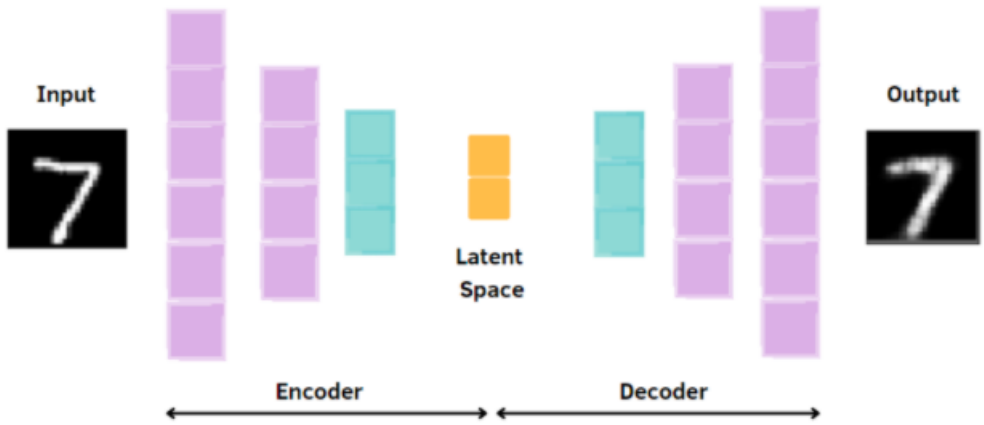


Figure 6.1: Exemple d'architecture d'auto-encodeur sur données mnist [3]

6 Module auto-encodeur

Le but de cette partie est de construire un auto-encodeur (un algorithme non-supervisé qui apprend la représentation des données) qui a pour objectif de compresser les données[5] puis les reconstruire (nous testerons sur des images).

Ça sera alors principalement formé de deux partie : un encodeur (transforme les données en une plus petite dimension aussi appelé espace latent) et un décodeur (décode l'exemple d'un encodeur vers sa représentation initiale).De plus, le but étant de minimiser l'erreur de reconstruction, nous utiliserons comme fonction coût la fonction Binary cross entropy (BCE) BCE=-(ylog(p) + $(1-y)\log(1-p)$).

Nous effectuerons dans cette partie 3 types d'expérimentations : La section 1 représente l'expérimentation sur la reconstruction des images, ensuite , la section 2 consiste à faire du débruitage de données (images) et enfin, la section 3 représente une visualisation de l'espace latent crée par a l'auto-encodeur (et d'étudier les performances du clustering effectué).

Reconstruction: Dans cette partie, nous visualisons les reconstructions en faisant varier la taille du batch, et cela en fixant le pas à $1e-4$ et le nombre d'itérations à 100.Les figures 6.2 et 6.3 représentent le résultat de la reconstruction avec un batch respectivement égal à 50 et 100. Nous pouvons remarquer que la reconstruction est plus précise avec un batch = 50,néanmoins, avec un batch = 100 c'est toujours acceptable.

Débruitage de données Cette section a pour objectif de débruiter les images. La figure 6.4 illustre le debruitage après un bruitage de 0.2, nous pouvons voir que nous arrivons à reconnaître parfaitement les 10 chiffres.Nous augmentons le bruitage dans les figures 6.5 ,6.6 et 6.7.Nous pouvons voir que pour un bruitage un peu plus augmenté, on arrive toujours a distinguer parfaitement les chiffres, mais même avec une forte compression (figure 6.7 ou la figure 6.8 qui est difficile a reconnaître à l'oeil), nous arrivons raisonnablement a les reconnaître.

[5]c.à.d réduire leur dimensions

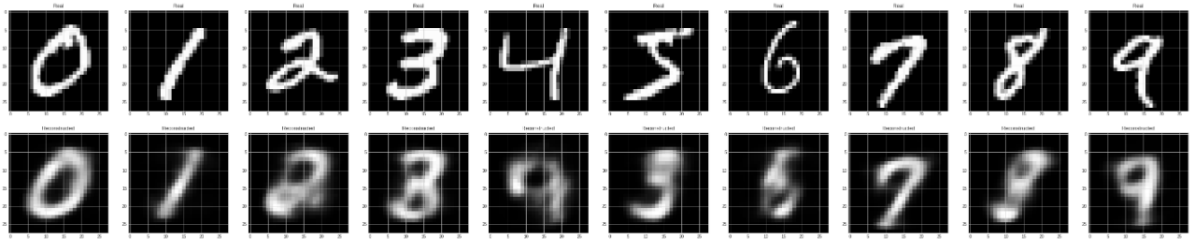


Figure 6.2: Reconstruction avec batch = 50

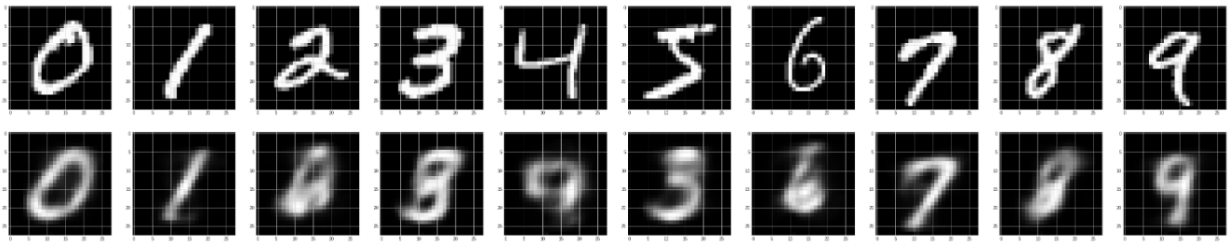


Figure 6.3: Reconstruction avec batch = 100

t-SNE et K-means: Afin de visualiser les resultats de notre auto-encodeur, nous utiliserons une tSNE et un Kmeans pour voir l'espace latent obtenu, nous normalisons les données et obtenons les figures suivantes: Les figures 6.9 et 6.10 représentent la représentation des données dans l'espace d'origine, quant a la figure 6.11, elle illustre la présentation K-means et t-SNE dans l'espace latent crée par l'auto-encodeur. De plus, nous pouvons retrouver l'évaluation des performances du clustering K-means dans la table 6.1.

Métrique	**Homogeneity**	**Completeness**	**V measure**
Score	0.48	0.48	0.48

Table 6.1: Évaluation des performances K-means

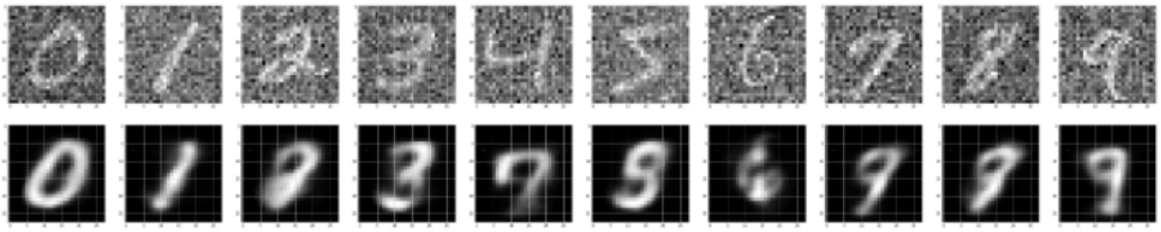


Figure 6.4: Debruitage avec bruit = 0.2

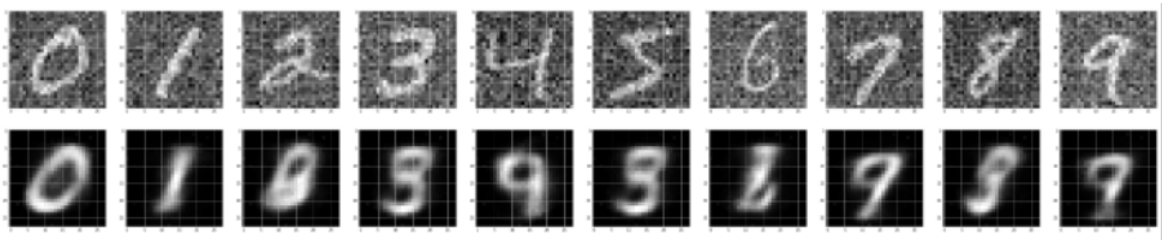


Figure 6.5: Debruitage avec bruit = 0.5

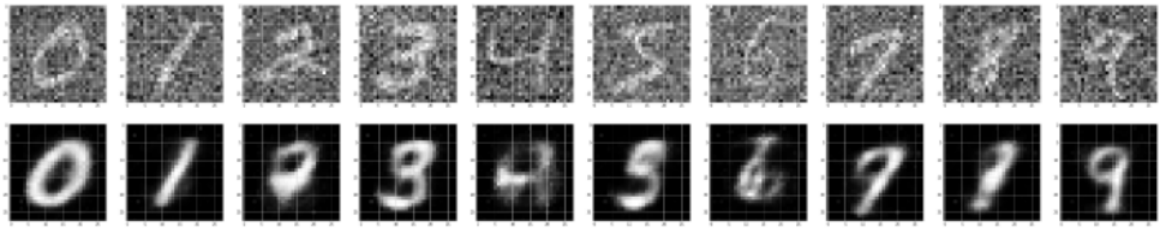


Figure 6.6: Debruitage avec bruit = 0.8

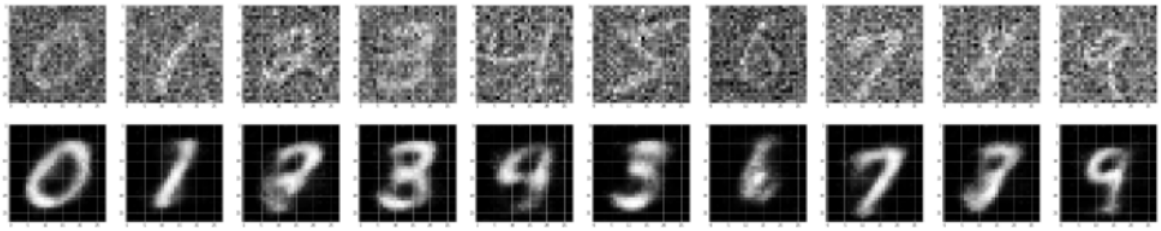


Figure 6.7: Debruitage avec bruit = 1

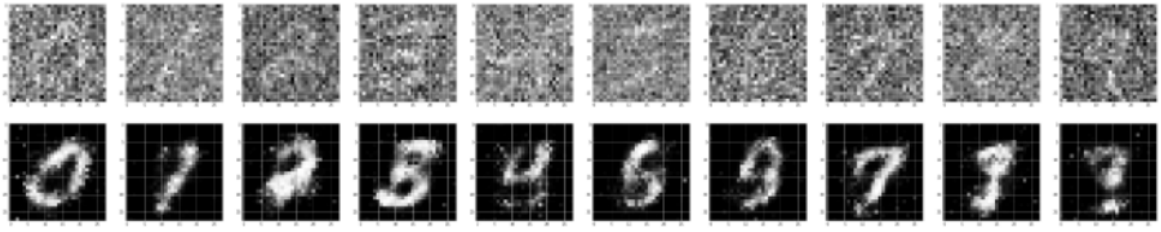


Figure 6.8: Debruitage avec bruit = 1.5

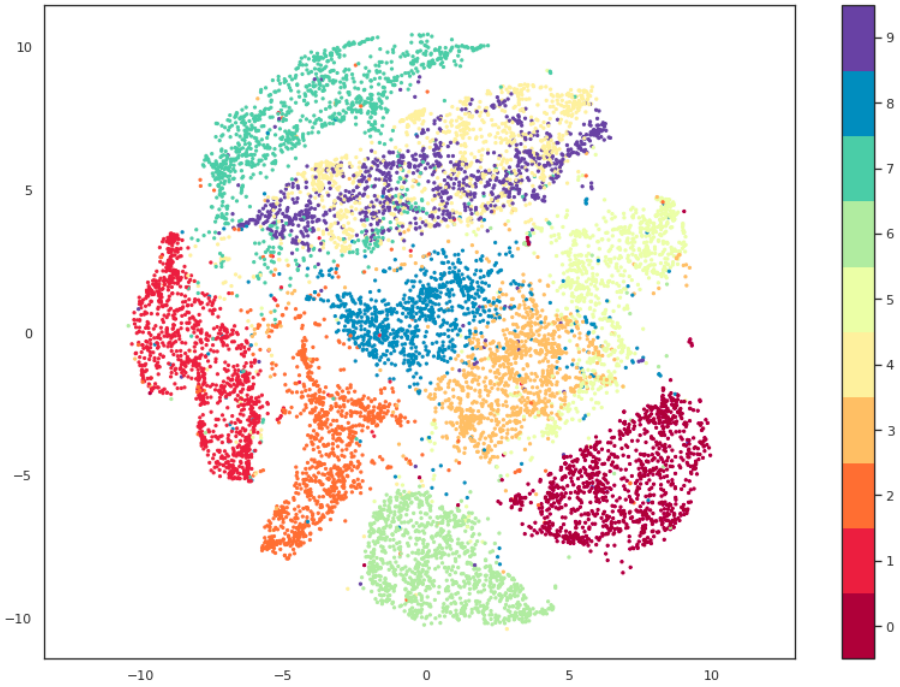


Figure 6.9: Visualisation tSNE de l'espace d'origine.

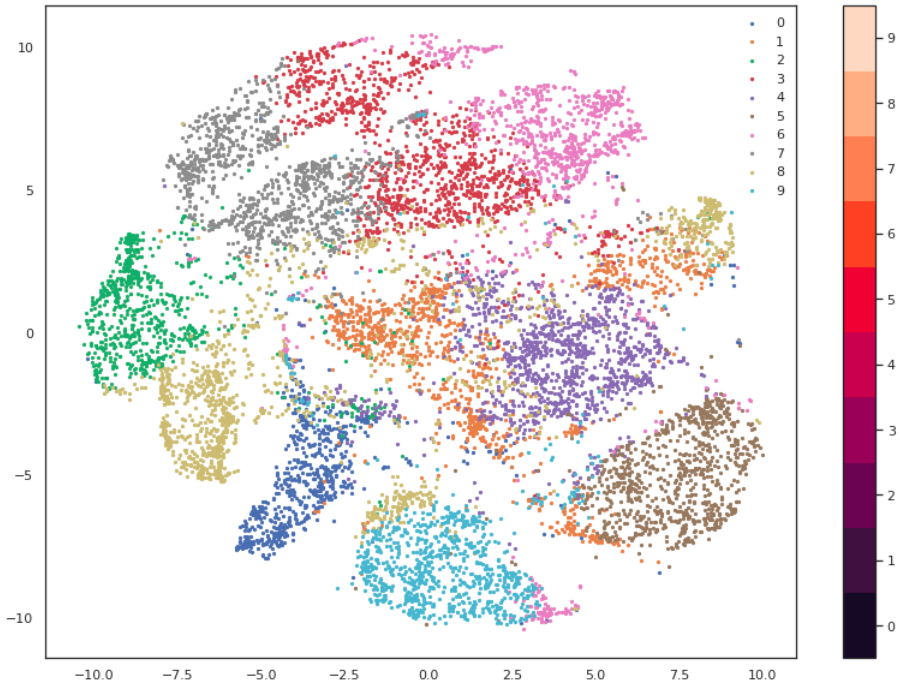


Figure 6.10: Visualisation K-means et tSNE de l'espace d'origine.

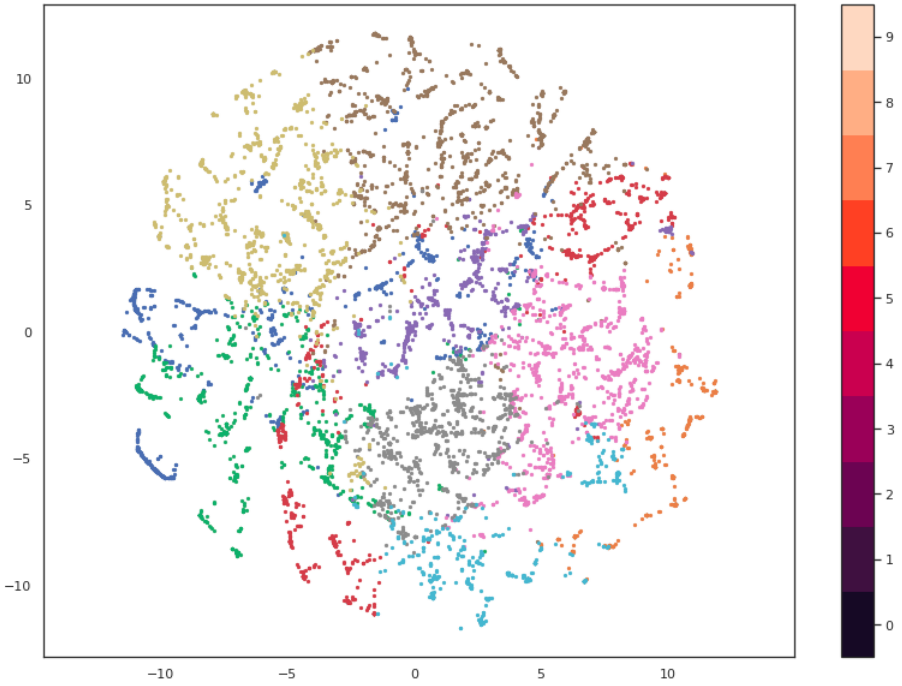


Figure 6.11: Visualisation K-means et tSNE de l'espace latent.

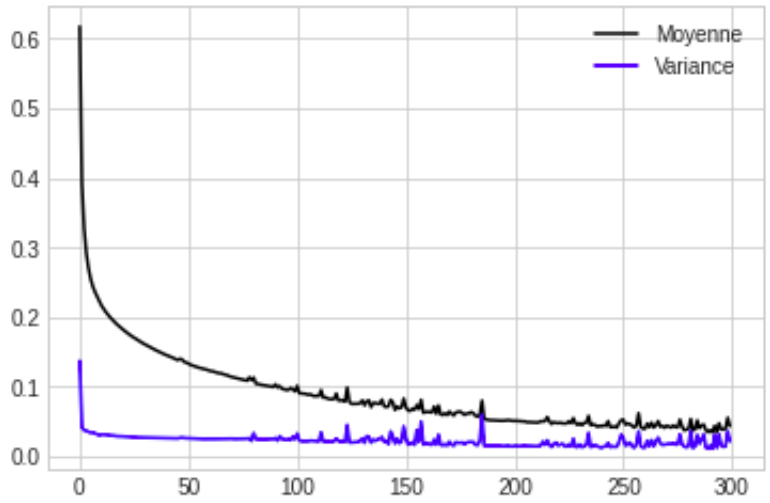


Figure 7.1: Conv1D : Évolution de la moyenne et de la variance durant le train

7 Module convolutionelle

Un réseau neuronal convolutionnelle, également connu sous le nom de CNN ou ConvNet, est une classe de réseaux neuronaux spécialisés dans le traitement de données présentant une topologie en forme de grille, notamment les images. Une image numérique est une représentation binaire de données visuelles.Un réseau convolutionelle comporte généralement trois couches : une couche convolutionnelle (permet principalement de faire le produit entre deux matrices) , une couche de pooling (permet de réduire la dimension de la représentation.) et une couche entièrement connectée (pour la classification).Cette section présente les résultats d'implémentation des modules Conv1D et Conv2D. Il est à noter que les modules convlutionelles consomment un temps remarquable (cela est dû aux différentes multiplication), et donc seraient plus rapide avec un GPU.

Conv1D Nous présentons dans cette section les résultats du module Conv1D sur les données USPS (tme4).Nous reprenons le reseau suivant

$Sequential(Conv1D(3,1,32), MaxPool1D(2,2), Flatten(), Linear(4064,100), ReLU(), Linear(100,10), SoftMax())$

Nous retrouvons avec ce modèle de très bonnes performances avec un score égal à 0.97. De plus, la figure 7.1 représente l'évolution de la moyenne et la variance selon les itérations, nous remarquons qu'elles diminuent.

Conv2D Nous présentons dans cette section les résultats du module Conv2D sur les données USPS (tme4).Nous reprenons le reseau suivant

$Sequential(Conv2D(3,1,32), MaxPool2D(2,2), Flatten(), Linear(1568,100), ReLU(), Linear(100,10), SoftMax())$

Nous retrouvons avec ce modèle a également de très bonnes performances avec un score égal à 0.95. De plus, la figure 7.2 représente l'évolution de la moyenne et la variance selon les itérations, nous remarquons qu'elles diminuent.

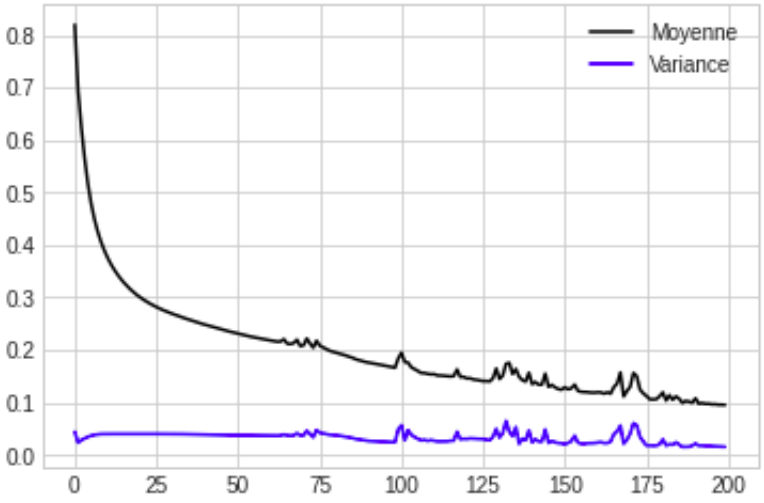


Figure 7.2: Conv2D : Évolution de la moyenne et de la variance durant le train

8 Conclusion

Nous avons implémenté et exprimenté les modules d'un réseaux de neurones inspiré des anciennes implémentations de la bibliothèque Pytorch [6].

Nous avons pu voir l'application du réseau sur différentes taches d'apprentissage, la classification sur des données linéairement séparables (avec des fonctions d'activations linéaires), des données non linéairement séparables (Sigmiod, TanH).

Ensuite, nous avons vu la classification binaire et multi-classe et avons eperimenté sur les application avec un auto-encodeur (debruitage,reconstruction,visualisation k-means, t-sne).

Enfin, nous avons vu les résultats de nos implementation de modules Conv1D,MaxPool1D, Flatten,MaxPool2D ainsi que Conv2D.

[6] `https://pytorch.org/`

Bibliographie

[1] https://cs229.stanford.edu/notes2020spring/cs229-notes-deep_learning.pdf

[2] Accessed at 10/04/2022. [Online]. Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

[3] E. Anello, « Convolutional Autoencoder in Pytorch on MNIST dataset », DataSeries, 28 mars 2022. [Online]. Disponible sur: https://medium.com/dataseries/convolutional-autoencoder-in-pytorch-on-mnist-dataset-d65145c132ac. [Consulté le: 22 avril 2022]

A Annexe

lien du drive contenant le code https://drive.google.com/drive/folders/1YRm2Mre9nLiaCd00oCP-HAwDx
usp=sharing