Rapport Projet – Réseau de neurones : DIY

YUAN Fangzheng, LI Haoran

Contents

Chapter 1

Introduction

L'objectif de ce projet est d'implémenter un réseau de neurones. Nous avons besoin d'écrire les fonctions à la main.
Chaque couche du réseau est vu comme un module et un réseau est constitué ainsi d'un ensemble de modules. En particulier, les fonctions d'activation sont aussi considérées comme des modules.

En utilisant NN, nous avons fait les applications simples en premier, comme les classifications de TME.
Nous résolvons la classification en multi-classe.
Après nous avons fait auto-encoder. Nous faisons les tests avec les chiffres MNIST, et ensuite les axes d'analyses choisis comme clustering et T-sne visualisation.
Enfin, nous attaquons CNN en 1D. C'est plus compliqué mais la performance est mieux.

Chapter 2

Linéaire

Dans cette partie, c'est simple qu'on ne utilise que le module linéaire et MSE loss fonction. Nous avons profité de données en TME ci-dessous.

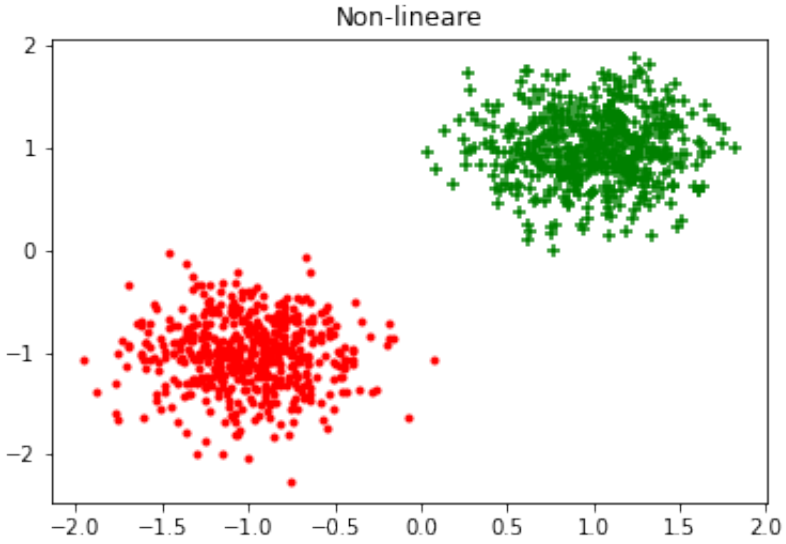

Figure 2.1: Données Linéaires

Si on utilise un module Lineaire Linear(2,1), on trouve sa performance:

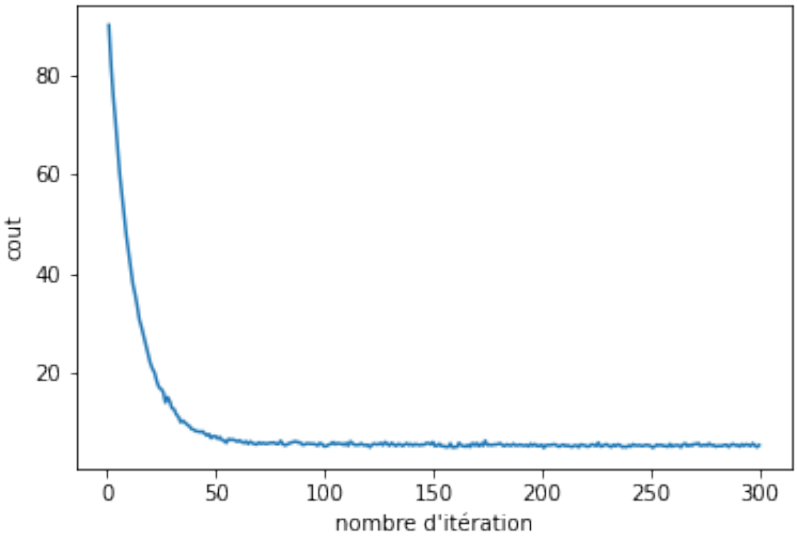

Figure 2.2: Graphe de Coût

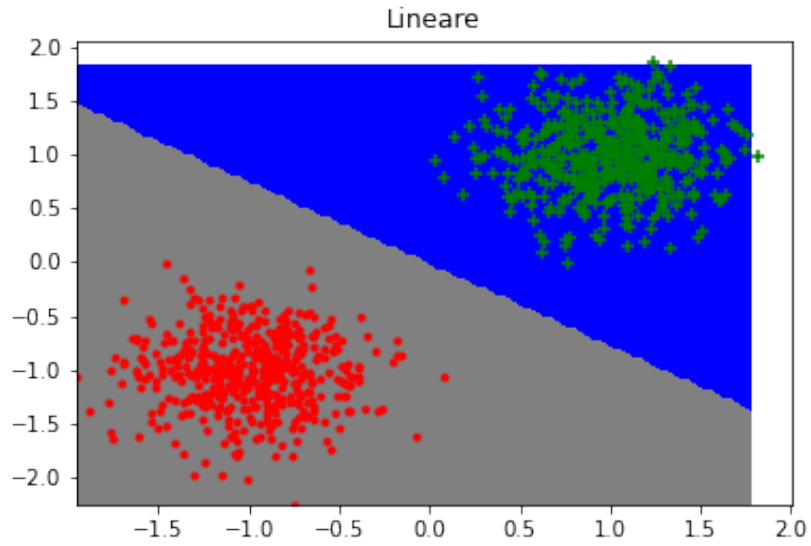

Figure 2.3: Graphe de Classification

Le score de prediction est : 100%
On regarde que pour cette simple tentative, nos résultats sont très bons.

Chapter 3

Non-linéaire

Ici, le cas est plus compliqué, on utilise aussi les données non-linéaires en TME. Voir ci-dessous:

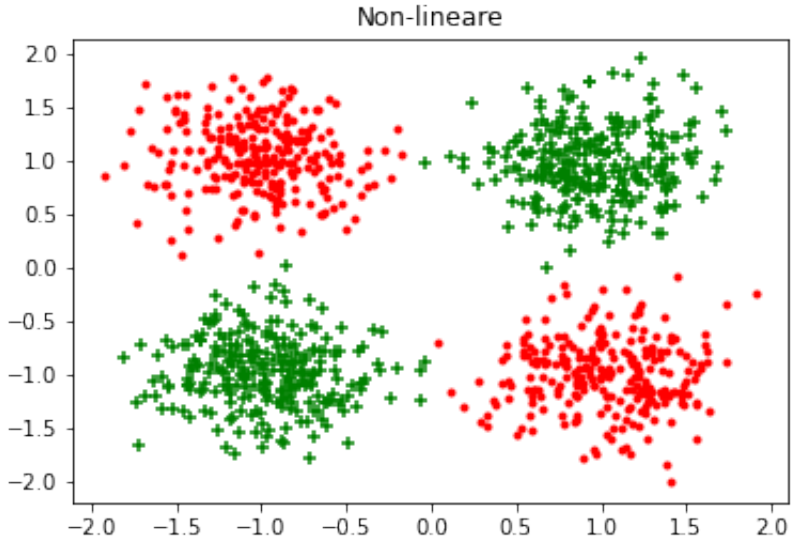

Figure 3.1: Données nonLinéaire

On trouve que si on n'ajoute pas **biais** sur datax, la performance est très male.

Donc on a besoin d'ajouter **biais** (une clonne de 1) sur datax.
On utilise le réseau ci-dessous:
$[Linear(3, 10), TanH(), Linear(10, 1), Sigmoid()]$ On trouve :

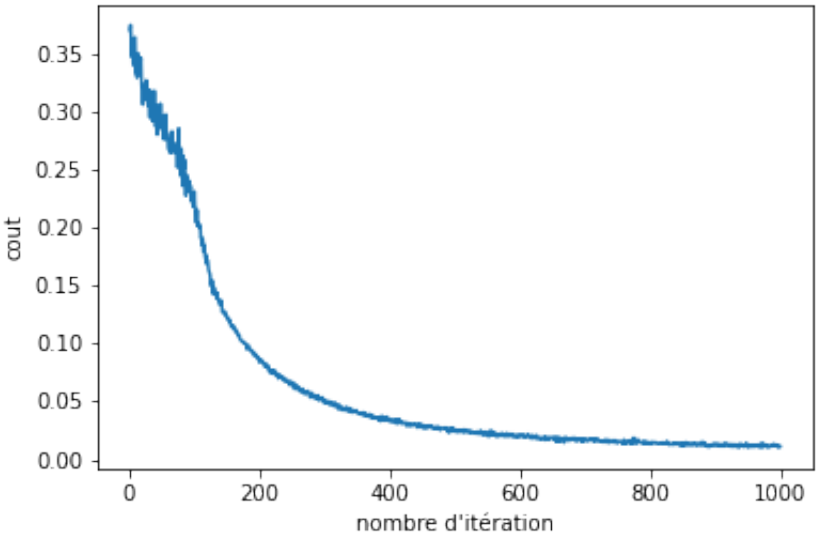

Figure 3.2: Graphe de Coût

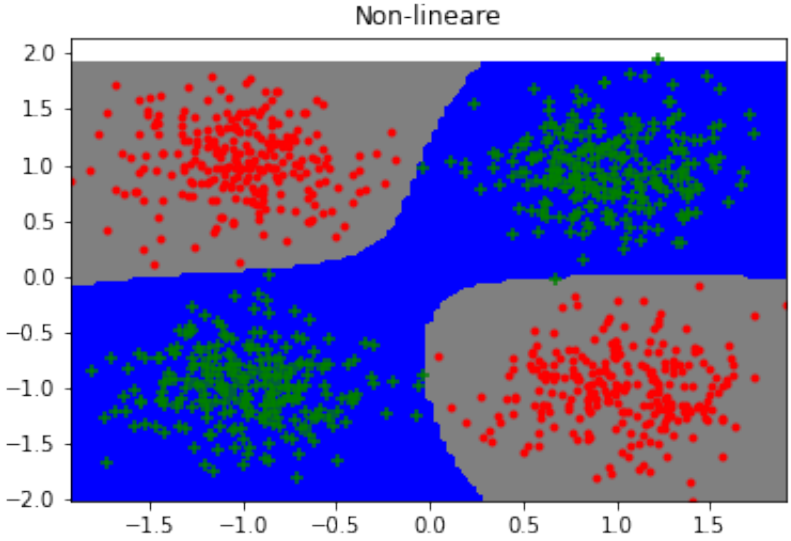

Figure 3.3: Graphe de Classification

Le score de prediction est : 99.9%

Chapter 4

Encapsulage

Ici, on essaye de simplifier nos fonctions par classe Sequentiel et Optim.

Sequentiel qui permet d'ajouter des modules en série et qui automatise les procédures de forward et backward quel que soit le nombre de modules mis à la suite.

Optim(net,loss,eps) condense une itération de gradient : elle prend dans son constructeur un réseau net, une fonction de coût loss et un pas eps.

Enfin, on a implémenté une fonction :
mini_SGD(seq, alltrainx, alltrainy ,batch_size = 1., nb_iteration = 300, loss_fonction = MSELoss(), eps = 1e-5)
nb_batch On le montre ci-dessous:

```
N = alltrainx.shape[0]
N_batch = int(N/batch_size)
# batch_size = 1 -> GD, batch_size = N -> SGD
```

Ensuite, on prend l'exemple non-linéaire ci-dessus pour montrer 3 méthodes : stochastique, mini batch et batch

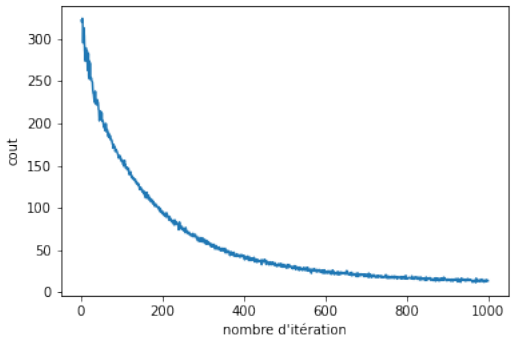

Figure 4.1: Coût Stochastique

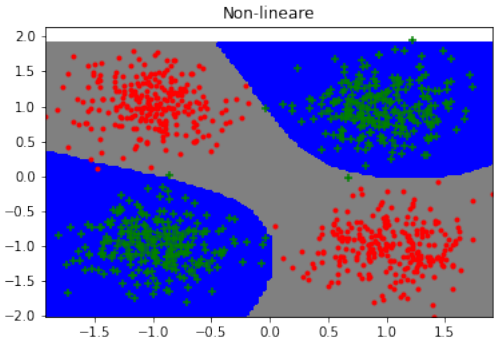

Figure 4.2: Classif Stochastique

Le score de prediction est : 99.6%

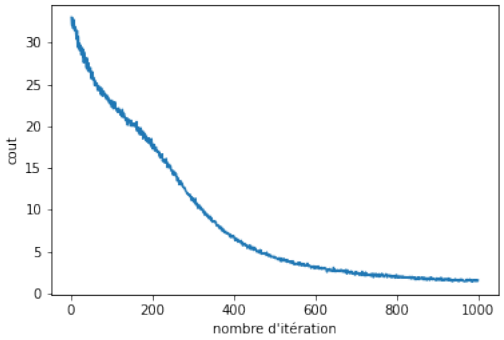

Figure 4.3: Coût Mini-batch

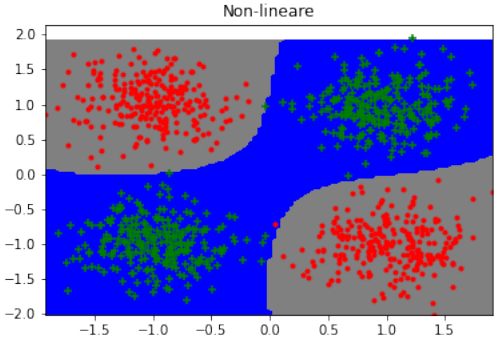

Figure 4.4: Classif Mini-batch

Le score de prediction est : 99.7%

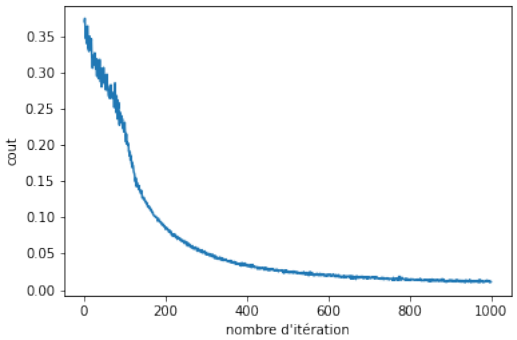

Figure 4.5: Coût Batch

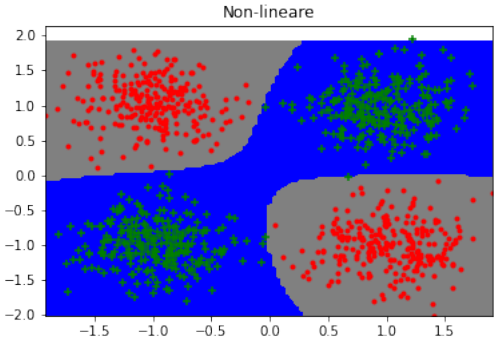

Figure 4.6: Classif Batch

Le score de prediction est : 99.9%

•Résumé
Il y a 3 façons, stochastique, minibatch et batch. On a bien montré ci-dessus. On trouve qu'ils nuancent la précision et l'efficacité.
Si on parcourt tous les batchs, la précision est meilleure. Mais la durée d'exécution du programme peut être trop longue. Par contre, si on choisit stochastique ou minibatch, c'est plus rapide mais la précision peut être pire.

On voit les graphes ci-dessus, il y a des différences sur la précisions entre les 3 méthodes. Mais SGD (Stochastic gradient descent) marche beaucoup beaucoup vite que GD(Gradient descent). Afin de gagner du temps et d'améliorer l'efficacité, cette approche est souvent très utilisée.

En conclusion, quand nous utilisons minibatch, nous devons choisir bien la taille de **batch size** en fonction de la situation spécifique.

Chapter 5

Multi-classe

Dans cette partie, on teste sur le jeu de données des chiffres manuscrits par exemple.
On utilise un réseaux **Sequentiel(Linear(256, 100),TanH(), Linear(100, 10), TanH())**
avec **OneHotEncoder, CELoss et Softmax à la dernière couche**

On trouve la performance ci-dessous:

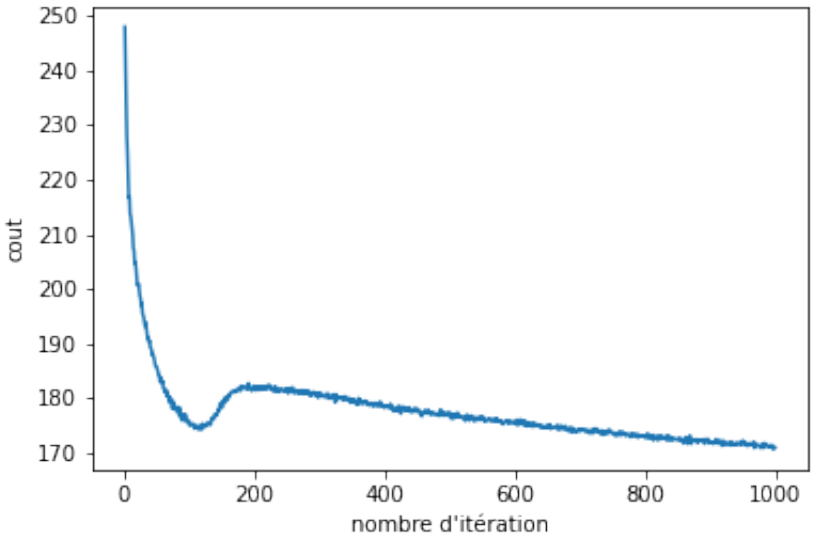

Figure 5.1: Graphe de Coût

Le score de prédiction est : 85.85%, c'est pas mal. On a choisi les bonnes fonctions d'activation.

Nous sélectionnons arbitrairement 4 images pour visualiser:

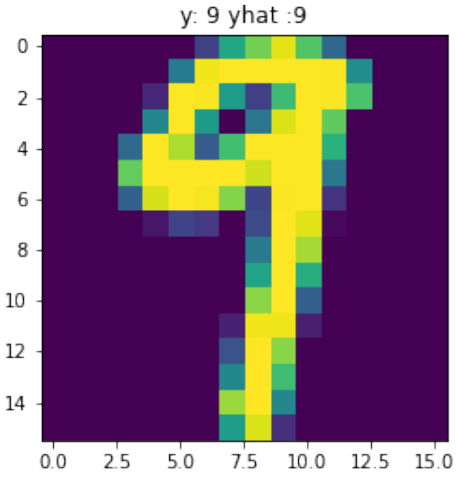

Figure 5.2: Image 1

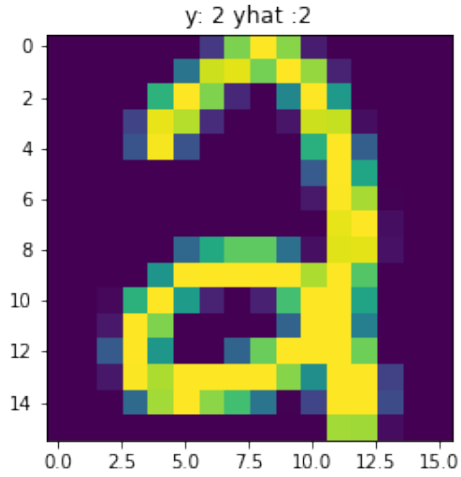

Figure 5.3: Image 2

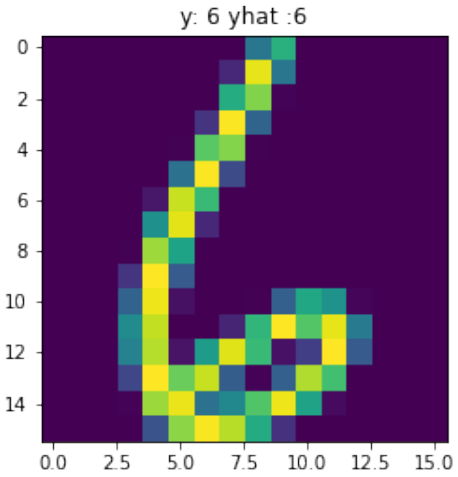

Figure 5.4: Image 3

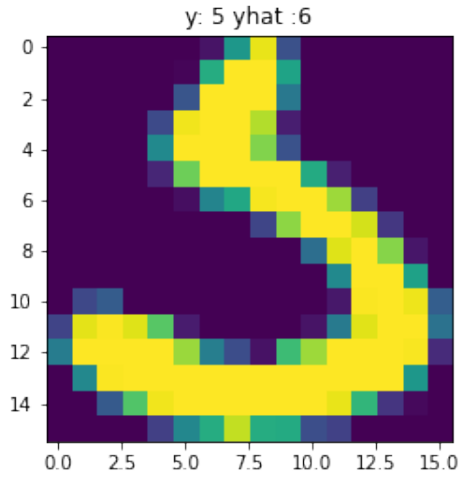

Figure 5.5: Image 4

Chapter 6

Compression—autocodeur

6.1 Original

Le but est de réduire les dimensions. Il s'agit de l'apprentissage non supervisé. Nous construisons l'architecture de notre auto-encodeur de la manière suivante:
•Encodage : Linear(256,100) → TanH() → Linear(100,10) → TanH()
•Décodage : Linear(10,100) → TanH() → Linear(100,256) → Sigmoide()

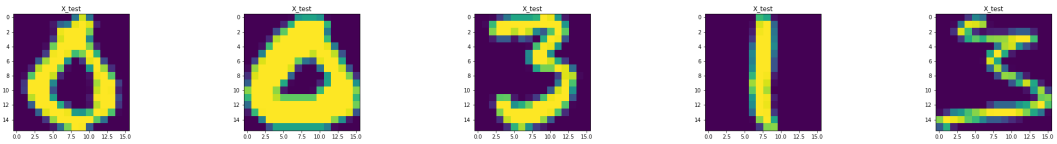

Figure 6.1: Minist original

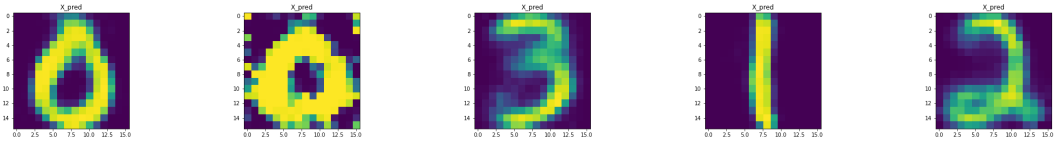

Figure 6.2: Après autocoder MSE

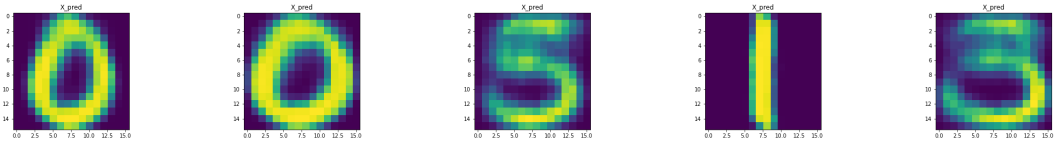

Figure 6.3: Après autocoder BCE

On a constaté que BCE marche mieux que MSE.
Pour le 5ème chiffre dans la figure, on voit que BCE peut identifier mais

MSE ne peut pas.

Si on fait clustering avec K-means, on a trouvé :
La pureté des clusters pour test : 0.6347032592373697

Si on fait T-Sne sur les préditions de KMEANS:

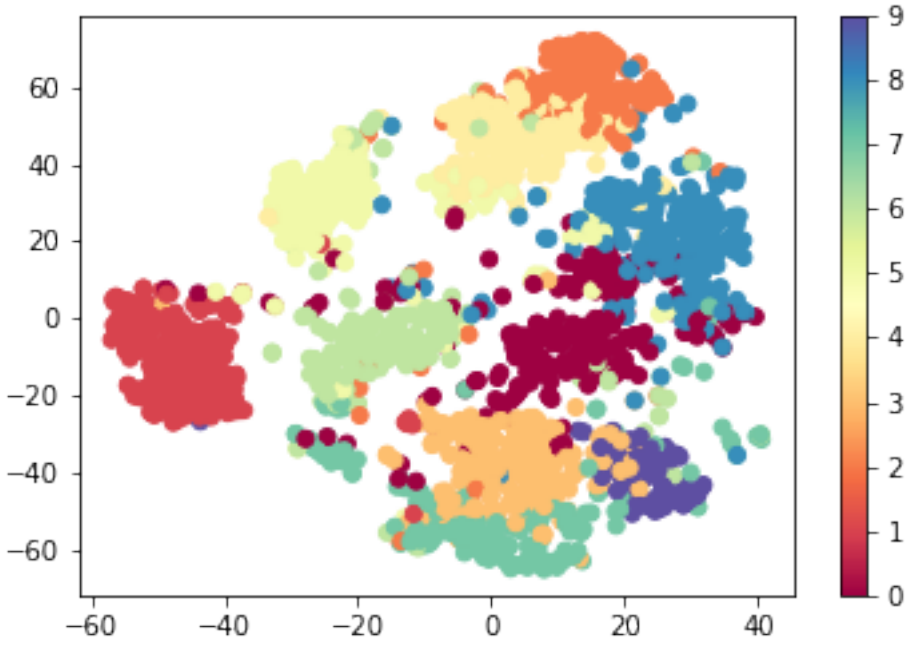

Figure 6.4: T-sne visualisation 1

6.2 Avec Bruit Gaussien

Le même processus qu'avant.
On ajoute des bruits gaussiennes dans Mnist comme ci-dessous:

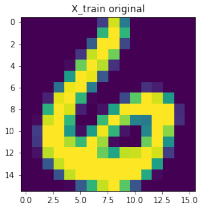

Figure 6.5: X train

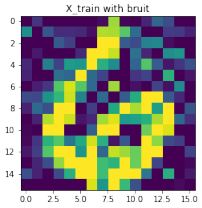

Figure 6.6: X train bruité

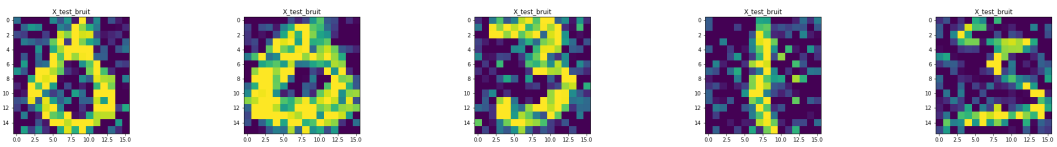

Figure 6.7: Minist avec bruit gaussien

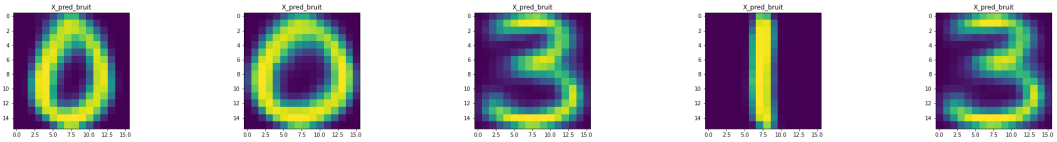

Figure 6.8: Après autocoder BCE

Si on fait clustering avec K-means, on trouve que :
La pureté des clusters pour test : 0.6243496315307603

Si on fait T-Sne sur les préditions de KMEANS:

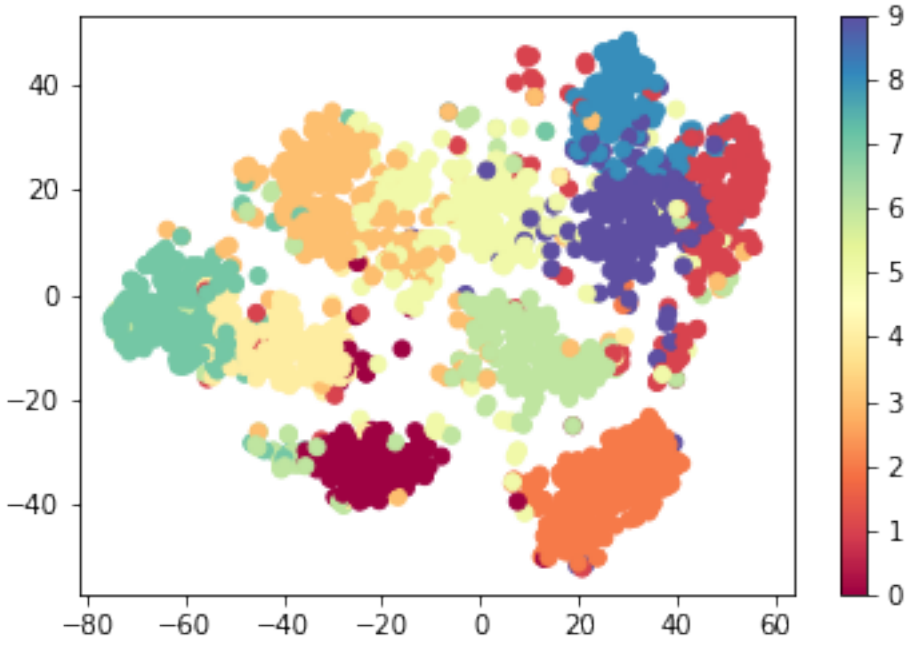

Figure 6.9: T-sne visualisation 2

Ça marche très bien. On voit que l'encodeur peut débruiter les images et qu'il est très performant.

6.3 Avec Binarisation

Le même processus que l'avant.
Mais on binarise nos données de Mnist comme ci-dessous:

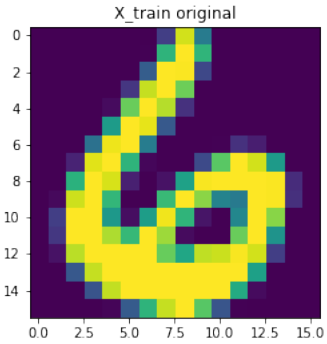

Figure 6.10: X train

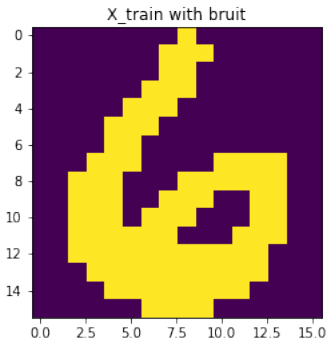

Figure 6.11: X train bruité

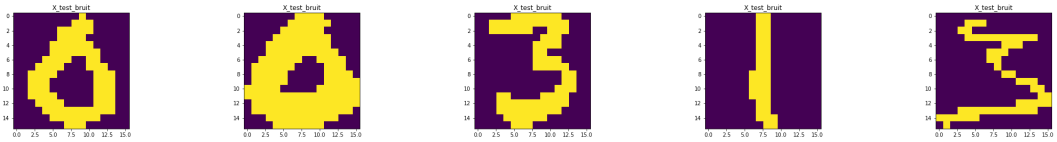

Figure 6.12: Minist avec bruit gaussien

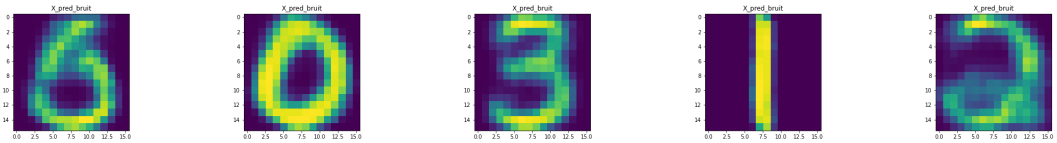

Figure 6.13: Après autocoder BCE

Si on fait clustering avec K-means, on trouve que :
La pureté des clusters pour test : 0.5941109359001544

Si on fait T-Sne sur les préditions de KMEANS:

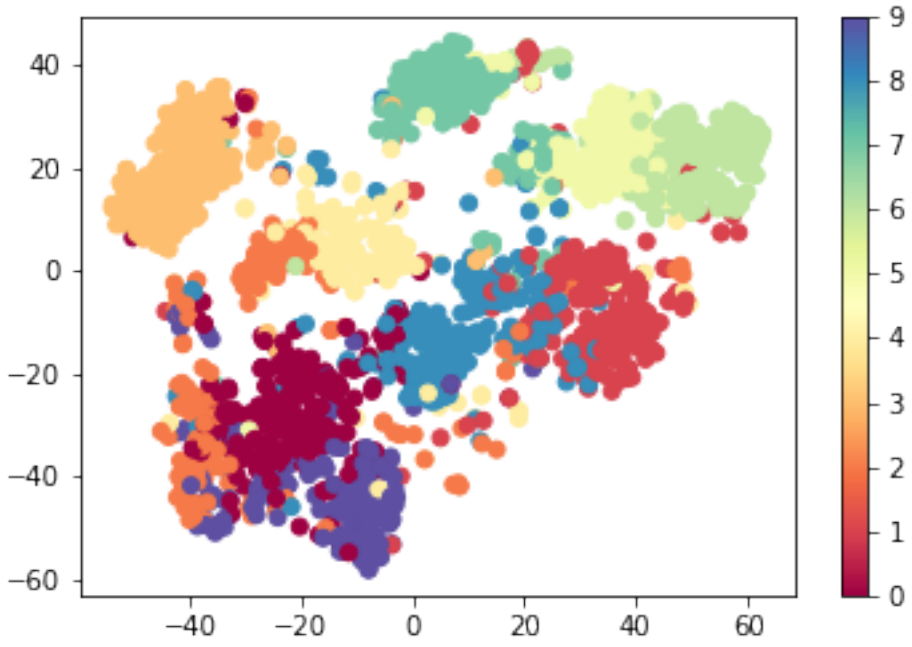

Figure 6.14: T-sne visualisation 3

On trouve que ce cas marche pire qu'avec le bruit gaussien.

6.4 Conclusion

Dans cette partie, on a essayé 2 fonctions de coût, MSE et BCE. On a trouvé que BCE marche mieux. On ajoute le bruit dans nos Mnists, on trouve que le performance est beaucoup mieux avec le bruit gaussien. On fait Kmeans et T-sne pour chacun, dans notre modèle, les puretés de clustering sont bonnes. Les visualisations de T-sne sont claires.

Chapter 7

Convolution(CNN)

Dans cette partie, on utilise CNN pour notre modèle qui est beaucoup compliqué. On utilise un réseaux ci-dessous :
Sequentiel(Conv1D(3,1,32,1), MaxPool1D(2,2), Flatten(), Linear(4064,100), ReLU(), Linear(100,10))
Comme la partie multi-classe qu'on a fait. On utilise CELoss avec Softmax et one-hot pour le résoudre.

En raison de la limitation des performances de notre ordinateur portable et de la complexité de notre algorithme est vraiment élevée comme nous utilisons les boucles For partout, nous n'avons sélectionné que 100 Mnist pour entraîner à la fin.

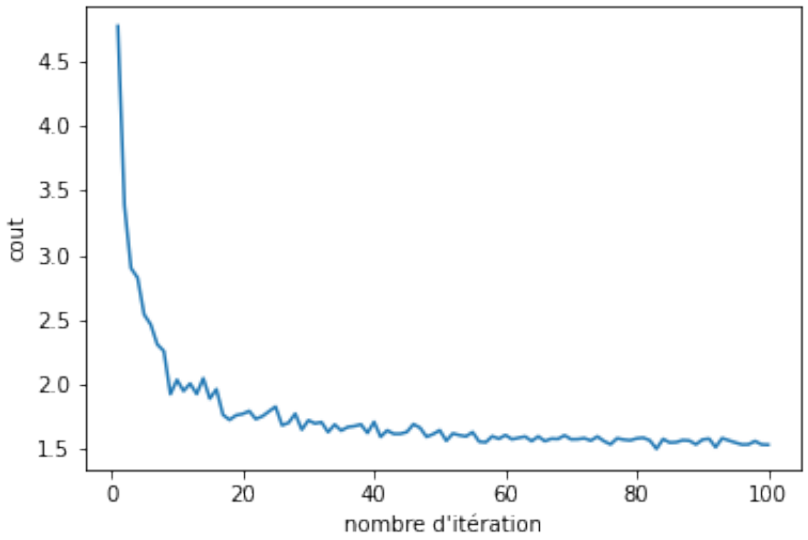

Figure 7.1: Coût de CNN

Les données en cours d'exécution sont trop faibles et trop lentes, nous ne mesurons donc que les scores à l'entraînement.

Le score de prédiction est 94%.
C'est très bien qu'il est beaucoup mieux que le cas sans convolution. Cela montre que notre réseau CNN fonctionne parfaitement, les dimensions des couches sont toutes correctes.

Amélioration possible:
Optimiser une partie du code de convolution
Reconnaître des motifs verticaux et horizontaux
Average Pooling plutôt que Max Pooling
Convolutions 2D

En raison des conditions matérielles (Laptop trop faible) et la limite de temps, on s'arrête là, nous vous remercions pour votre lecture.