SORBONNE UNIVERSITE - FACULTE DES SCIENCES

Réseau de neurones DIY
UE Machine learning

Autheures :
Nolwenn PIGEON
Karima SADYKOVA

Master 1 Données Apprentissage et Connaissances
Semestre 2

Table des matières

1 Introduction

L'objectif de ce projet est de créer une implémentation en `Python` d'un réseau de neurones. Cette implémentation s'inspire des anciennes versions de PyTorch et d'autres implémentations similaires qui permettent de créer des réseaux modulaires et génériques. Chaque couche du réseau est considérée comme un module, et un réseau est constitué d'un ensemble de ces modules. De plus, les fonctions d'activation sont également considérées comme des modules à part entière.

Le fonctionnement général de la bibliothèque repose sur une classe abstraite appelée `Module`, qui représente un module générique du réseau de neurones. Cette approche modulaire offre une grande flexibilité dans la construction et la configuration des réseaux de neurones, permettant ainsi de les adapter à différents problèmes et architectures.

En utilisant cette implémentation, il sera possible de créer des réseaux de neurones personnalisés en combinant différentes couches et fonctions d'activation, tout en bénéficiant d'une structure claire et organisée. Ce projet vise donc à fournir une solution puissante et modulaire pour l'implémentation de réseaux de neurones en Python.

2 Mon premier est... linéaire

Le premier modèle que nous implémentons est une régression logistique, qui est un type de réseau de neurones à une seule couche linéaire. L'objectif de ce modèle est de minimiser le coût des moindres carrés (MSE) en ajustant les poids (ω) pour obtenir une meilleure frontière de décision. Cette frontière de décision est estimée à partir d'un ensemble de points **datax** et ces prédictions **datay**.

Pour évaluer les performances de notre modèle, nous avons ajouté un bruit aux données en utilisant les paramètres σ et ϵ pour contrôler respectivement la variance des données et leur mélange. En entraînant le réseau de neurones, nous pourrons tracer la frontière de décision et analyser les performances ainsi que les limites de ce modèle dans le contexte spécifique de notre problème.

Ces tests nous permettront de comprendre comment la régression logistique fonctionne dans notre cas d'étude et d'explorer les différents paramètres qui influencent les performances du modèle. Nous pourrons ainsi mieux comprendre les avantages et limitations de la régression linéaire.

2.1 Nombre d'itérations

Dans notre première expérience de régression logistique, nous avons alimenté notre réseau de neurones avec un ensemble de points de données. La droite tracée représente la frontière de décision qui sépare nos deux classes. Les résultats obtenus correspondent à nos attentes, comme en témoigne l'accuracy score. Nous avons également remarqué que la perte (loss) diminue rapidement : bien que notre réseau ait été entraîné sur 1000 itérations, il a seulement fallu 50 itérations pour atteindre la convergence.

Ces observations démontrent l'efficacité de notre modèle de régression logistique dans la classification de ces données. La rapidité de convergence et la précision des prédictions suggèrent que notre réseau de neurones est capable d'apprendre et de généraliser correctement à partir de ces exemples. Ces résultats encourageants nous motivent à poursuivre notre exploration des performances de la régression logistique dans d'autres scénarios et à évaluer son potentiel dans des problèmes plus complexes.

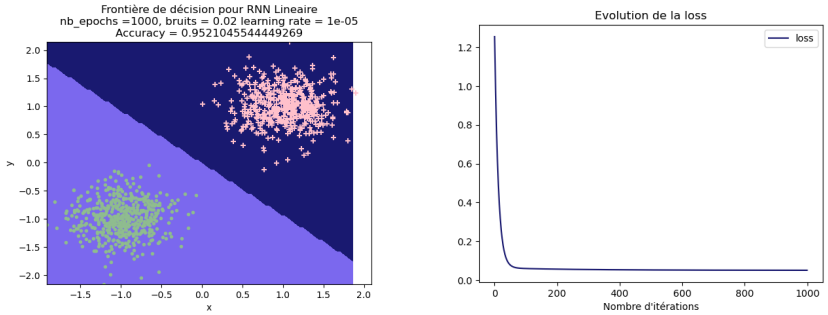


FIGURE 1 – Regression logistique avec 1000 itérations, Accuracy = 0.95

Voici les résultats obtenus après 50 itérations de notre réseau de régression logistique :

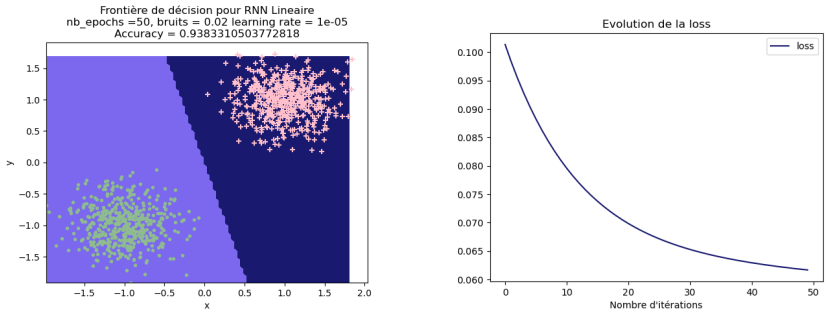


FIGURE 2 – Regression logistique avec 50 itérations, Accuracy = 0.94

2.2 Augmentation du bruit

Maintenant, nous explorons l'impact de la variation du bruit dans nos données en ajustant le paramètre ϵ et σ dans la fonction **gen_arti** .

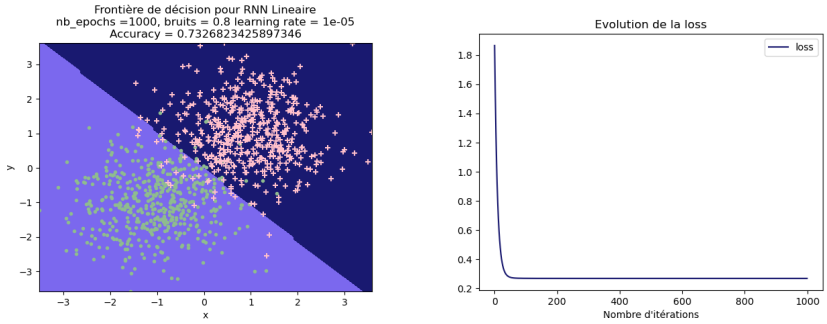


FIGURE 3 – Bruit = 0.5, , Accuracy = 0.73

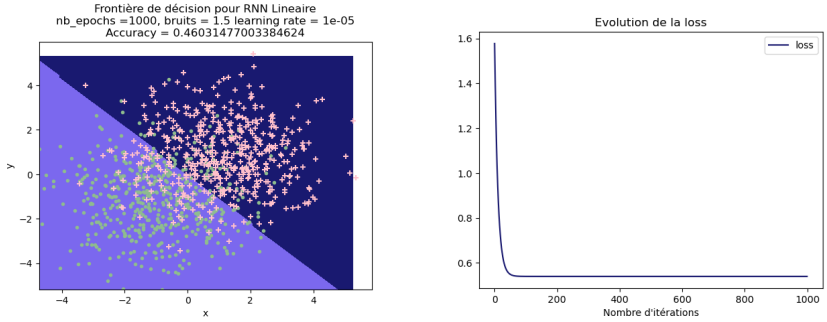


FIGURE 4 – Bruit = 1.5, Accuracy = 0.45

En analysant les résultats de nos tests, nous avons observé que l'accuracy score a tendance à diminuer à mesure que la valeur d'epsilon augmente. L'epsilon est un paramètre qui contrôle la tolérance de l'algorithme de régression logistique et détermine la sensibilité du modèle aux variations dans les prédictions.

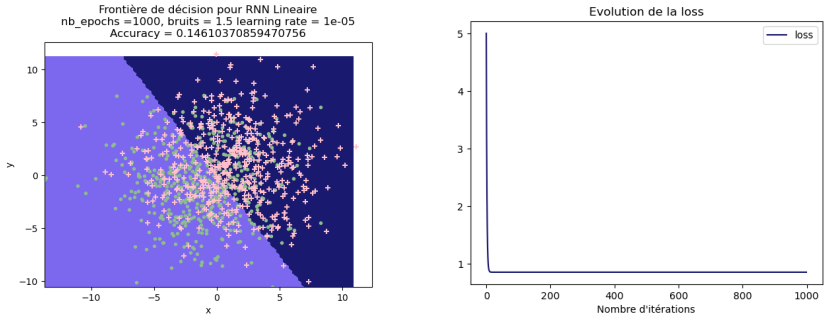


FIGURE 5 – Bruit = 1.5, Ecart = 10, Accuracy = 0.15

Finalement, nos résultats obtenus démontrent l'efficacité de notre modèle de régression logistique dans l'estimation des poids, malgré la présence de bruit dans les données. La convergence rapide de la perte témoigne de la capacité du réseau à apprendre et à s'ajuster aux données d'entrée.

3 Mon deuxième est ... non linéaire

Nous allons ensuite implémenter un réseau de neurones à deux couches linéaires avec une activation tangente entre les deux couches et une activation sigmoïde à la sortie. Cette implémentation sera réalisée au sein de la classe NonLineaire.

3.1 Variation du nombre de neurones

Lors de l'entraînement d'un modèle de descente de gradient sur un réseau de neurones avec deux neurones, nous avons effectué 300 itérations en utilisant un pas de gradient de 0.001. Les résultats obtenus ont une erreur quadratique moyenne (MSE) de 0.1575 et une précision (accuracy) de 0.97. Notre modèle a une performance élevée du modèle dans la tâche de classification. La classification se fait de manière parallèle.

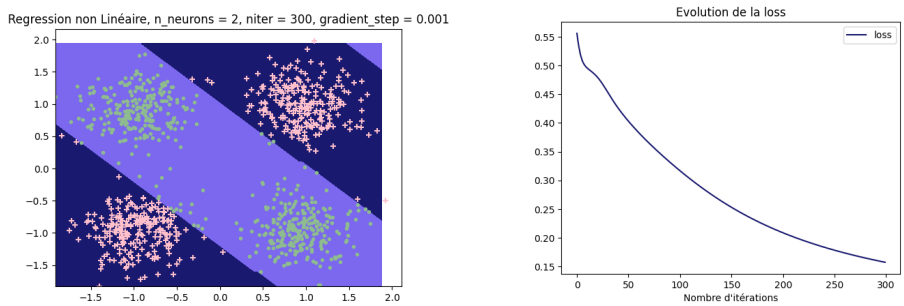


FIGURE 6 – MSE= 0.1575, Accuracy = 0.97

Comme les données sont générées aléatoirement nous avons refait tourné le test et cette fois nous avons obtenu des résultats très différents avec une erreur quadratique moyenne (MSE) de 0.2653 et une précision bien plus faible de 0.574.

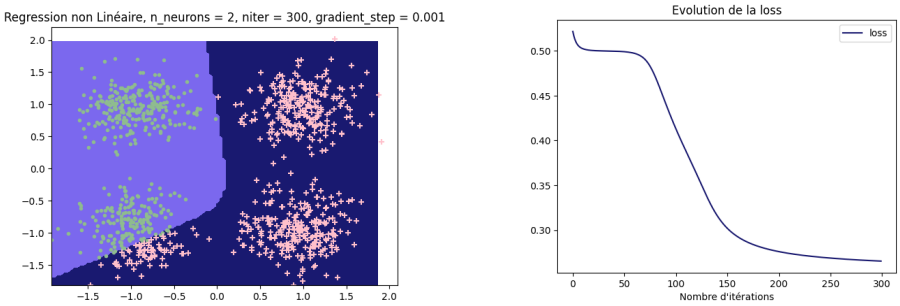


FIGURE 7 – MSE = 0.2653, Accuracy = 0.574

Lorsque nous avons testé notre réseau sur l'ensemble de données d'apprentissage avec 10 neurones, nous avons obtenu un score de performance de 100%. Cela suggère qu'en augmentant le nombre de neurones, la précision du modèle s'améliore. L'augmentation du nombre de neurones peut permettre au modèle de capturer des relations plus complexes dans les données d'entraînement, ce qui peut conduire à une meilleure performance en termes de précision. Avec plus de neurones, le modèle dispose d'une plus grande capacité d'apprentissage et peut être en mesure de représenter des motifs plus fins et des décisions plus précises. Cependant, il est important de surveiller le risque de surapprentissage et de trouver un équilibre approprié entre la capacité du modèle et sa capacité à généraliser correctement sur de nouvelles données.

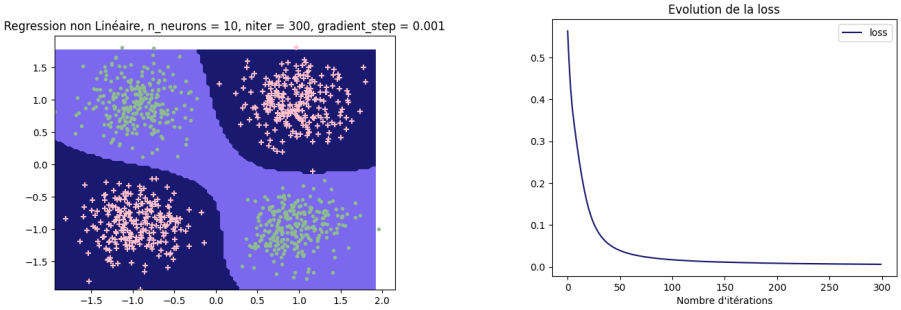


FIGURE 8 – MSE = 0.0061, Accuracy = 1

3.2 Variation du nombre d'itérations

Lorsque nous examinons l'impact des variations du nombre d'itérations sur un réseau non linéaire à 3 neurones, on remarque qu'en augmentant le nombre d'itérations, le modèle a plus d'opportunités d'ajuster ses poids et ses

biais pour minimiser l'erreur sur les données d'entraînement. La performance du modèle s'améliore jusqu'à obtenir un score d'accuracy à 0.988

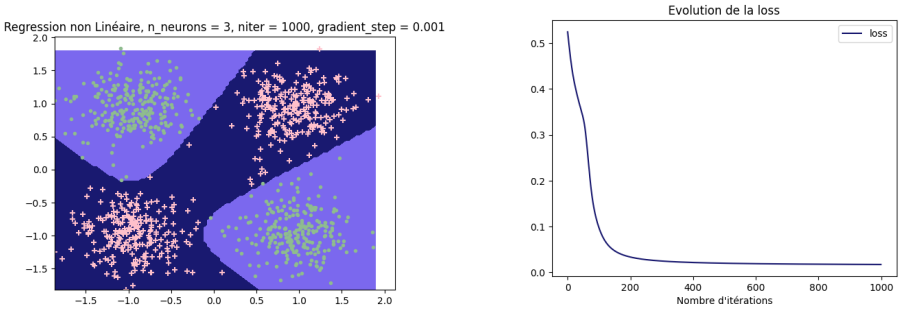


FIGURE 9 – MSE = 0.2617, Accuracy = 0.988

3.3 Variation du learning rate

Le learning rate (taux d'apprentissage) est un hyperparamètre qui détermine l'ampleur des mises à jour effectuées sur les poids et les biais du modèle à chaque itération de la descente de gradient. En augmentant le learning rate, les mises à jour sont plus importantes, on observe que notre modèle converge plus rapidement. Cependant, une augmentation significative du learning rate peut également entraîner des variations instables de la loss. Dans le cas où le learning rate passe de 0.001 à 0.1, une augmentation considérable est effectuée. Cela peut provoquer des oscillations instables de la loss, où la loss peut augmenter et diminuer de manière imprévisible à chaque itération.

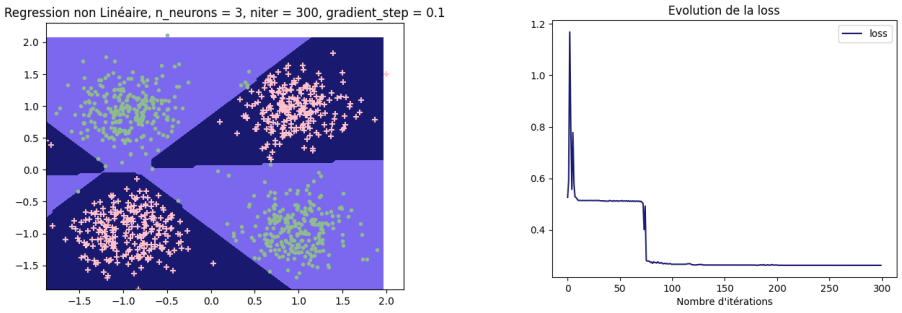


FIGURE 10 – MSE = 0.0172, Accuracy = 0.99

Notre implémentation d'un réseau à deux couches cachées non linéaires montre des performances prometteuses pour la classification binaire. Les résultats obtenus démontrent l'importance des fonctions d'activation non linéaires pour l'apprentissage de représentations complexes. En conclusion,

ces tests ont démontré l'importance du nombre de neurones, du nombre de batchs et du learning rate dans la performance d'un réseau de neurones non linéaire. Une recherche rigoureuse des meilleurs hyperparamètres est nécessaire pour obtenir des performances optimales. Cela est possible avec un GridSearch par exemple.

4 Mon troisième est un encapsulage

Nous avons rapidement réalisé que le processus d'inférence et de rétro-propagation pourrait devenir fastidieux à coder. Il serait également difficile d'imaginer créer un réseau plus complexe en suivant cette approche. Pour pallier ces difficultés, nous avons créé deux nouvelles classes : Sequentiel et Optim. La classe Sequentiel permet d'ajouter en série des modules à un réseau et d'appliquer la phase forward. Quant à la classe Optim, elle se concentre sur l'exécution de la passe backward. Nos expérimentations se sont déroulées sur des données artificielles caractérisées par la présence de quatre groupes gaussiens et deux classes distinctes.

Dans le but de simplifier notre démarche, nous avons élaboré une nouvelle version de notre réseau non linéaire, que nous avons nommée NonLineaire-Seq. Cette nouvelle version s'appuie sur les fonctionnalités offertes par les classes Sequentiel et Optim. La principale différence entre NonLineaireSeq et NonLineaire c'est que nous avons utilisé la descente de gradient stochastique (SGD) avec une approche mini-batch pour entraîner notre réseau de neurones. Cette fonction permet en outre de spécifier la taille des lots utilisés lors de l'apprentissage.

4.1 Variation du nombre de neurones

En augmentant le nombre de neurones dans notre modèle, nous avons constaté une amélioration des performances, ce qui indique que le modèle est capable de capturer des relations complexes entre les variables d'entrée. Cependant, nous avons également pris en compte le risque de surajustement en surveillant les performances du modèle sur les données de test. En trouvant un équilibre entre la complexité et les performances, on peut en deduire qu'il suffit 3 neurons pour que le modèle aprenne bien. De plus, nous avons noté que l'augmentation du nombre de neurones peut avoir un impact sur le temps d'entraînement, ce qui doit être pris en compte lors du choix du modèle final, car plus de neurones on a, plus de temps ça prendra pour l'apprentissage.

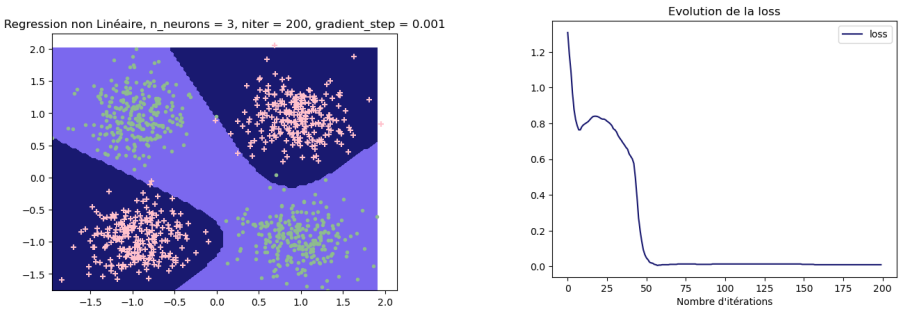


FIGURE 11 – Regression a 3 neurones, niter = 200, batch_size = 1(Descente Stochastique), Accuracy=0.995

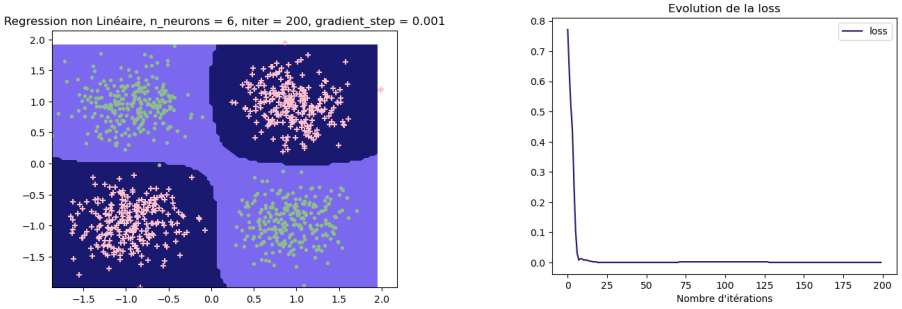


FIGURE 12 – Regression a 6 neurones, niter = 200, batch_size = 11(Descente Stochastique), Accuracy=1.0

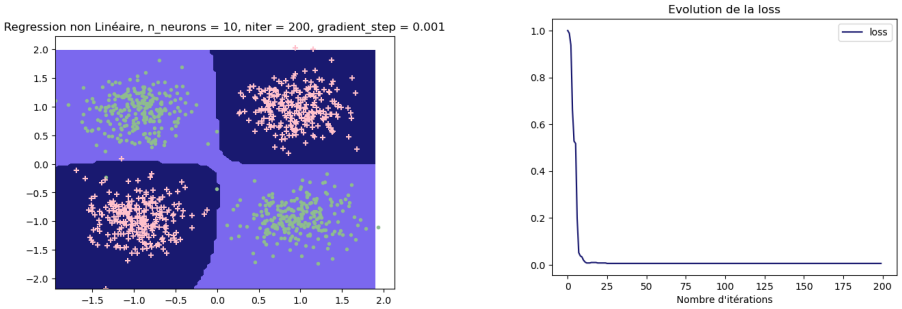


FIGURE 13 – Regression a 10 neurones, niter = 200, batch_size = 11(Descente Stochastique), Accuracy=1.0

4.2 Variation du nombre de batchs

Lorsque nous augmentons la taille du batch dans notre approche de mini-batch, nous observons que le réseau apprend plus lentement en raison du nombre plus élevé d'exemples qu'il doit analyser à chaque itération. Une conséquence de cette augmentation de la taille du batch est que la fonction de perte (loss) semble plus chaotique par rapport à la méthode SGD classique. Cela peut être dû au fait que le modèle effectue des mises à jour moins fréquentes des poids, ce qui entraîne des variations plus prononcées dans la fonction de perte lors de chaque mise à jour. Bien que cette loss chaotique puisse rendre le processus d'apprentissage moins lisse, le modèle continue néanmoins à converger vers une solution optimale. Il convient donc de considérer attentivement le compromis entre la vitesse d'apprentissage et la stabilité des mises à jour de poids lors du choix de la taille du batch pour l'entraînement du modèle.

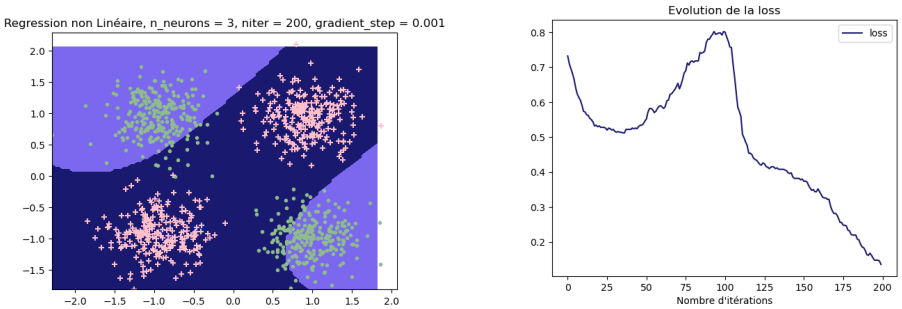


FIGURE 14 – Regression a 3 neurones, niter = 200, batch_size = 100, Accuracy=0.932

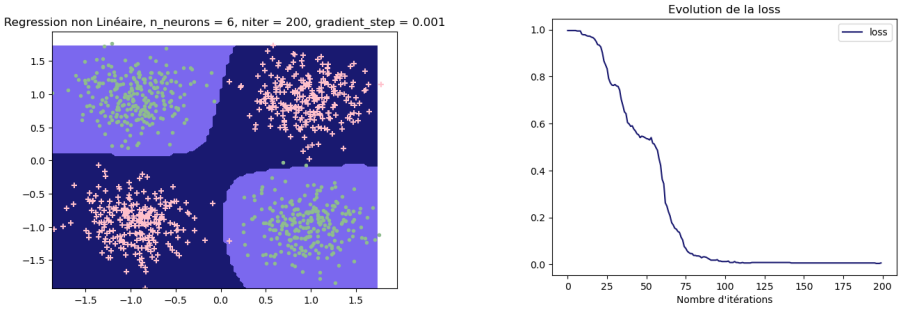


FIGURE 15 – Regression a 6 neurones, niter = 200, batch_size = 100, Accuracy=0.989

4.3 Variation du nombre d'itérations

Nous avons observé que 50 itérations ne sont pas suffisantes pour que ce réseau de neurones atteigne une convergence significative. Cependant, à partir de 200 itérations, nous avons constaté que les valeurs obtenues étaient plus cohérentes et se rapprochaient davantage des valeurs attendues. Cela souligne l'importance de la patience et de l'entraînement prolongé pour obtenir des résultats précis et fiables avec ce réseau. On en déduit qu'il faut prendre en compte le nombre d'itérations approprié en fonction de la complexité du problème et des données d'entraînement afin d'obtenir des performances optimales.

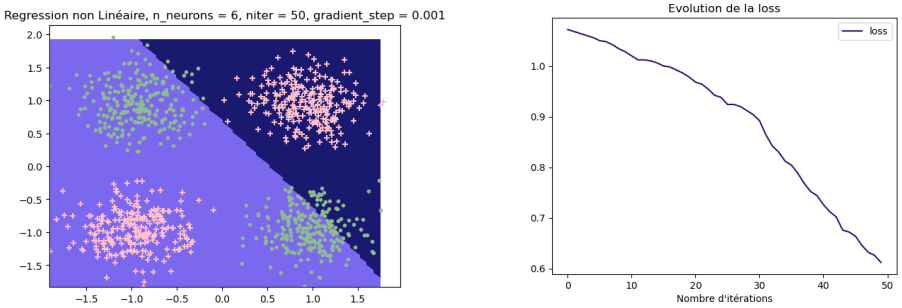


FIGURE 16 – Regression a 6 neurones, niter = 50, batch_size = 100, Accuracy=0.68

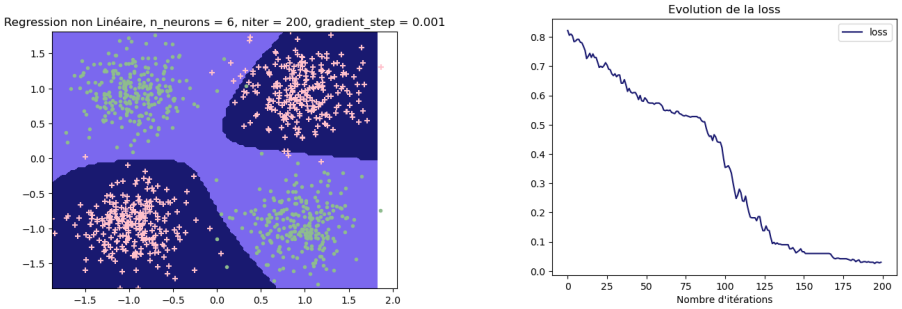


FIGURE 17 – Regression a 6 neurones, niter = 200, batch_size = 100, Accuracy=0.985

4.4 Analyse des résultats

En résumé, nos tests ont démontré l'importance de la sélection appropriée du nombre de neurones, de la taille du batch et du nombre d'itérations pour obtenir des performances optimales dans notre réseau non-linéaire. Ces résultats nous donnent des indications précieuses pour la configuration et l'entraînement de réseaux de neurones dans des problèmes similaires.

Par ailleurs, nos résultats démontrent l'efficacité de la descente de gradient stochastique avec mini-batch. Cette approche nous offre une meilleure vitesse de convergence et des performances améliorées par rapport à la descente de gradient stochastique classique. Elle constitue donc une méthode prometteuse pour l'entraînement et l'optimisation de réseaux de neurones dans des problèmes de régression logistique.

5 Mon quatrième est multiclasse

Dans cette partie, nous avons exploré l'utilisation des multiclasses, où chaque classe est représentée par une dimension de sortie pour indiquer la probabilité de chaque classe. Pour cela, nous avons utilisé une activation Softmax à la dernière couche pour transformer les entrées en une distribution de probabilités, et nous avons calculé la loss en utilisant l'entropie croisée. Ainsi, pour chaque exemple, nous avons été en mesure de prédire la classe avec la probabilité maximale.

Pour évaluer les performances de notre réseau, nous avons effectué des tests sur l'ensemble de données USPS composé d'images de chiffres manuscrits. Avec cet ensemble de données à 10 classes correspondant aux différents chiffres, notre objectif était de déterminer l'impact du nombre de neurones sur les performances du réseau.

Pour nos tests, nous avons créé un réseau non linéaire avec une activation tangente et une activation Softmax en sortie pour obtenir les probabilités de prédiction. Nous avons d'abord entraîné le réseau avec 10 neurones et obtenu une précision de 83%.

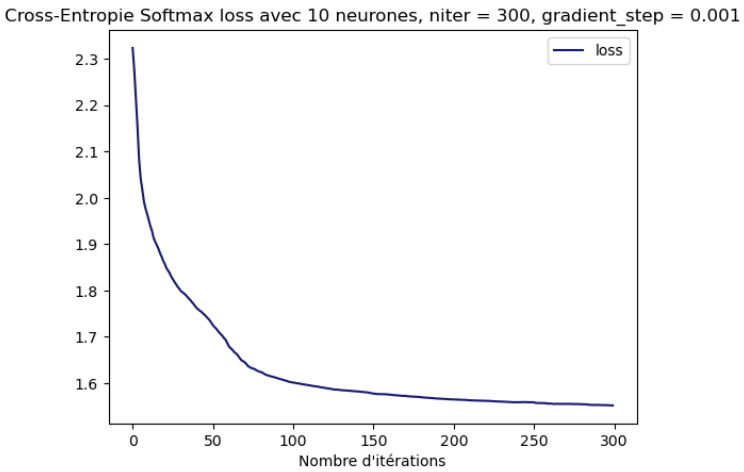
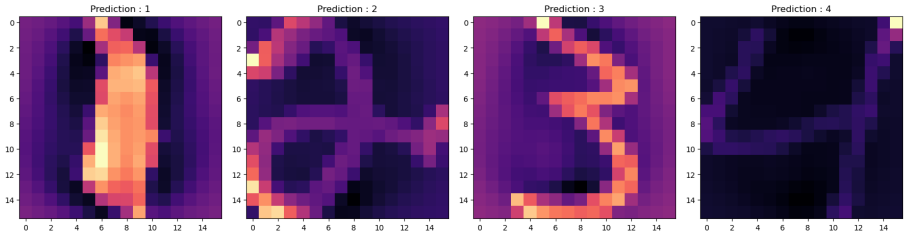
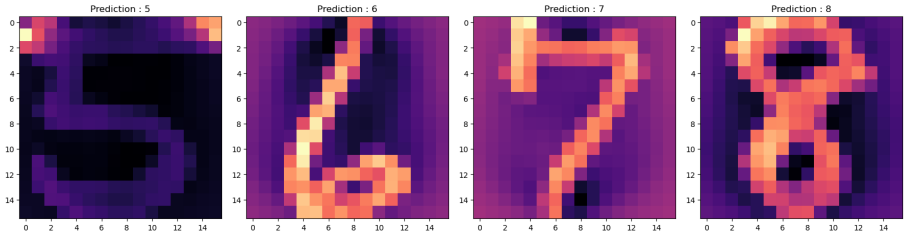


FIGURE 18 – Multi-classe avec 10 neurones, Accuracy=0.8315894369706028

Voici les prédictions obtenues par notre réseau de 10 neurones sur l'ensemble de test d'images de chiffres :



(a) yhat = 1 (b) yhat = 2 (c) yhat = 3 (d) yhat = 4



(e) yhat = 5 (f) yhat = 6 (g) yhat = 7 (h) yhat = 8

FIGURE 19 – Multi-classe avec 10 neurones

En augmentant le nombre de neurones, nous avons observé une légère amélioration du score de classification, mais cela a pris plus de temps à s'exécuter. Par exemple, en utilisant 100 neurones, nous avons obtenu une précision de 85%, mais cela a nécessité un temps d'exécution plus long.

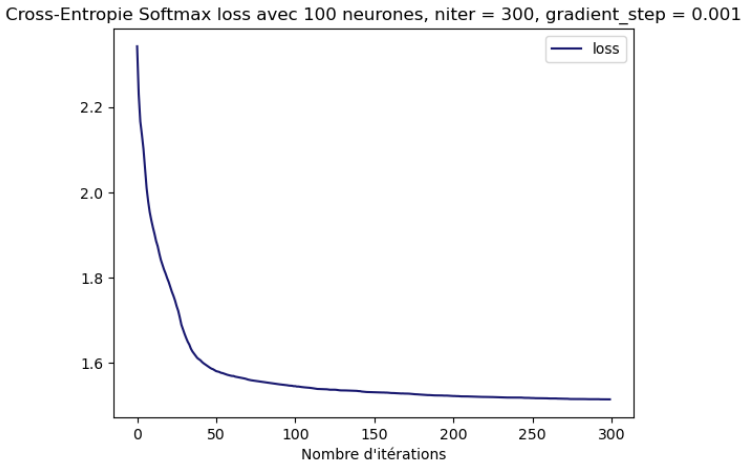


FIGURE 20 – Multi-classe avec 100 neurones, Accuracy=0.852017937219731

Nous avons également testé avec 200 neurones, et bien que cela ait accéléré la convergence du réseau, le gain de performance obtenu était relativement faible par rapport au nombre d'itérations supplémentaires nécessaires. Le

score de classification était de 85%, mais le temps d'exécution était d'environ 71 secondes.

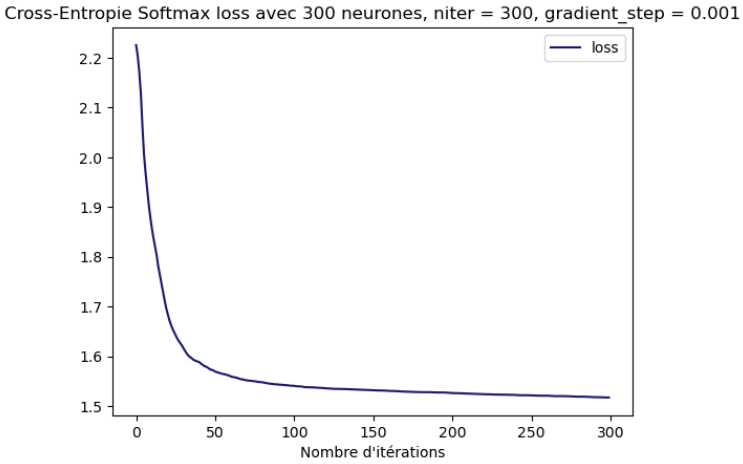


FIGURE 21 – Multi-classe avec 200 neurones, Accuracy=0.8535127055306427

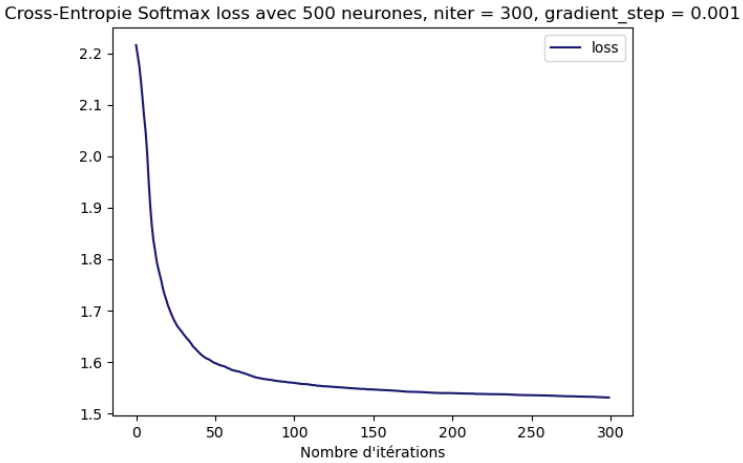


FIGURE 22 – Multi-classe avec 500 neurones, Accuracy=0.8420528151469856

En continuant nos expérimentations, nous avons augmenté le nombre de neurones à 500 pour évaluer son impact sur les performances du réseau. Cependant, nous avons observé une légère diminution de l'accuracy, qui est passée à 84%. De plus, le temps nécessaire pour effectuer les prédictions a doublé par rapport à celui du réseau avec 200 neurones (120 sec au lieu de 71 sec). Donc, on en deduit que l'ajout de plus de neurones ne conduit pas nécessairement à une amélioration continue des performances du réseau.

Nous constatons que l'augmentation du nombre de neurones entraîne une amélioration des performances et accélère la convergence du réseau. Cependant, le gain de performance obtenu est relativement faible par rapport au nombre d'itérations supplémentaires nécessaires.

6 Mon cinquième se compresse (et s'expérimente beaucoup)

6.1 Présentation de notre autoencoder

L'auto-encodeur est une technique d'apprentissage non supervisé (nous n'avons pas besoin de labels pour apprendre nos données) utilisée pour compresser et reconstruire des données. Il se compose de deux parties, un **encodeur** qui réduit la dimensionnalité des données en une représentation latente et un **décodeur** qui reconstitue les données originales à partir de cette représentation. En utilisant un ensemble de données, l'auto-encodeur apprend à capturer les caractéristiques les plus importantes et à les reconstruire avec une perte minimale. Cette approche permet d'explorer les structures intrinsèques des données, de visualiser les représentations apprises dans un espace réduit et d'évaluer les performances de l'auto-encodeur pour des tâches telles que le clustering, la détection d'anomalies ou la génération de nouvelles données.

Notre auto-encodeur sera composé de deux sous-réseaux distincts :
- L'encodeur, qui sera constitué de deux couches cachées utilisant des fonctions d'activation de type tangente hyperbolique (tanh)
- Le décodeur, qui sera également composé de deux couches cachées, mais avec des fonctions d'activation comprenant une tangente hyperbolique (tanh) suivie d'une sigmoïde

6.2 La Binary Crossentropy Loss

L'objectif principal de notre réseau sera de minimiser le coût de reconstruction de chaque exemple encodé, ce qui permettra de quantifier la perte d'information. En d'autres termes, l'auto-encodeur apprendra à reconstruire les données d'entrée à partir de leur représentation encodée, en minimisant les différences entre les données originales et les données reconstruites. Nous utilisons un coût de reconstruction :

$$L(X, \hat{X})$$

avec $\hat{X}$ = decodeur(encodeur(X)) la donnée reconstruite. Le coût de reconstruction peut être par exemple un coût aux moindres carrés ou - plus performant - une cross entropie binaire :

$$BCE(y, \hat{y}) = -(y \times log(\hat{y}) + (1 - y)log(1 - \hat{y})).$$

La cross-entropie binaire est plus "abrupte" que l'erreur quadratique moyenne (MSE) car elle attribue des poids plus élevés aux erreurs proches des extrêmes (0 ou 1). Cela signifie que la fonction de perte pousse les valeurs de sortie vers 0 ou 1, plutôt que vers une moyenne de valeurs, ce qui peut être souhaitable dans certains cas où une classification précise est importante. En ce qui concerne le rapport avec la vraisemblance, la fonction de perte basée sur la cross-entropie binaire est souvent utilisée pour les tâches de classification où les sorties sont des probabilités entre 0 et 1. La cross-entropie binaire mesure la différence entre la distribution de probabilité prédite par le modèle (dans ce cas, les valeurs de sortie $\hat{X}$) et la distribution réelle (les valeurs de X). Elle est liée à la notion de maximisation de la vraisemblance dans l'estimation de paramètres de modèles probabilistes.

Pour évaluer les performances de notre auto-encodeur, nous allons effectuer des tests sur les données chiffrées USPS, tout comme précédemment. La réduction de dimension obtenue grâce à l'auto-encodeur correspondra à une compression des données, et nous pourrons confirmer visuellement cette compression en analysant les résultats.

6.3 Variations du nombre d'itérations

Nous testons d'abord notre autoencodeur avec les hyper paramètres suivant : niter=500, neuron = 100 et gradient_step = 1e-4. Dans ce test, l'autoencodeur a été entraîné pendant 500 itérations, ce qui peut permettre d'atteindre une meilleure convergence et d'améliorer les performances de reconstruction. Le paramètre "gradient_step" contrôle la taille des pas de gradient lors de l'optimisation des poids du modèle. Une valeur plus petite, comme 1e-4 dans ce cas, indique des pas plus petits et plus précis lors de l'ajustement des poids. Cela peut aider à atteindre une meilleure convergence et à éviter des problèmes tels que des sauts de gradient importants.

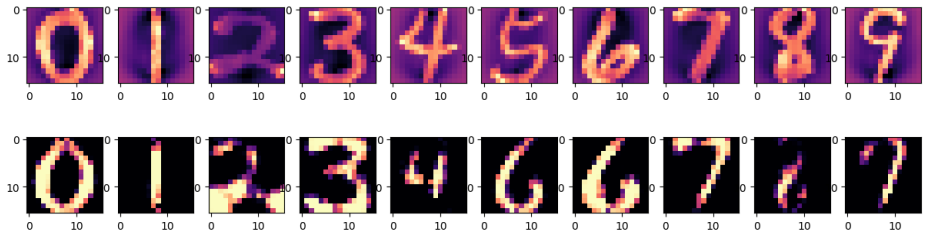


FIGURE 23 – Autoencoder pour 100 neurones, 500 iterations et un gradient step de 1e-4, Données originales sont en première ligne tandis que les données reconstruites par l'autoencoder sont en deuxième ligne

En diminuant le nombre d'iterations à 100 on remarque que l'autoencoder converge plus rapidement mais les images reconstruites sont de moins bonne qualité. Cela réduit aussi naturellement le temps d'exécution. En effet on voit bien que le 5, 8, et 9 reconstruits ne ressemblent pas vraiment aux données originales, ni au chiffres que nous connaissons. On pourrait même les confondre avec d'autres chiffres.

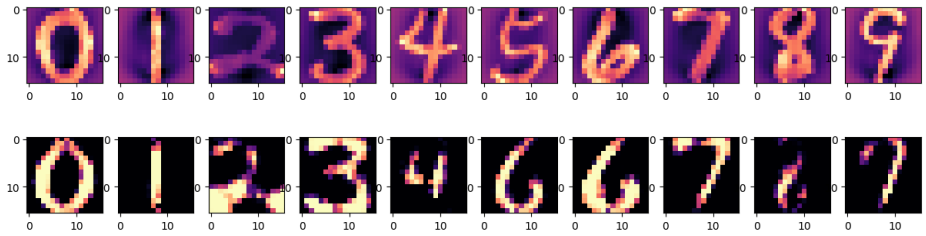


FIGURE 24 – Autoencoder pour 100 neurones, 100 iterations et un gradient step de 1e-4, Données originales sont en première ligne tandis que les données reconstruites par l'autoencoder sont en deuxième ligne

6.4 Variations du nombre de neurones

En augmentant le nombre de neurones à 500 dans l'autoencodeur, on constate que sa capacité à apprendre des représentations complexes augmente. Cela signifie que l'autoencodeur est capable de capturer des détails plus fins et des structures plus complexes des images d'origine. En conséquence, les reconstructions d'images obtenues à partir de la représentation latente de 500 neurones sont généralement de meilleure qualité. Les détails importants sont préservés, ce qui permet une reconstruction précise dans l'espace d'origine.

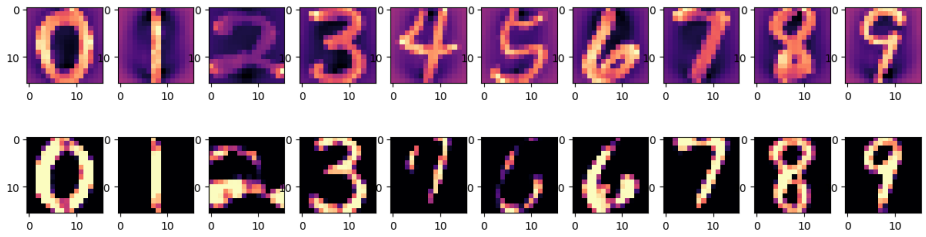


FIGURE 25 – Autoencoder pour 500 neurones, 100 iterations et un gradient step de 1e-4, Données originales sont en première ligne tandis que les données reconstruites par l'autoencoder sont en deuxième ligne

En revanche, si on diminue le nombre de neurones à 10, on observe que les images reconstruites sont de moindre qualité. Cela est dû au fait que la compression réduit considérablement le nombre de dimensions dans la représentation latente, ce qui conduit à une perte d'informations significative. En conséquence, les images reconstruites présentent une homogénéisation des caractéristiques, où les détails et les nuances de l'image d'origine sont moins bien préservés.

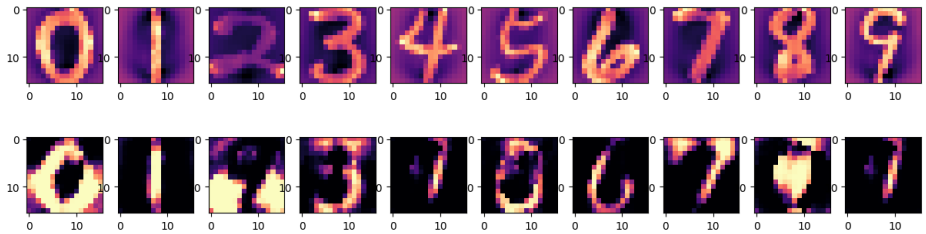


FIGURE 26 – Autoencoder pour 10 neurones, 100 iterations et un gradient step de 1e-4, Données originales sont en première ligne tandis que les données reconstruites par l'autoencoder sont en deuxième ligne

Le nombre de neurones dans l'autoencodeur joue donc un rôle crucial dans la capacité de l'autoencodeur à apprendre et à reconstruire des représentations précises des images. Une augmentation du nombre de neurones permet de mieux capturer les détails complexes, tandis qu'une diminution entraîne une perte d'informations et une détérioration de la qualité des reconstructions.

6.5 Clustering et TSNE

Après l'entraînement de l'autoencodeur, l'algorithme K-means est utilisé pour effectuer le clustering dans l'espace latent. On utilise les représentations latentes obtenues à partir de l'encoder. L'objectif est de regrouper les données similaires en clusters distincts. Si la compression réalisée par l'autoencodeur a réussi à conserver les informations discriminantes propres à chaque classe, on peut s'attendre à ce que les instances de la même classe soient regroupées dans la projection obtenue par le clustering. Puis à partir de TSNE (t-Distributed stochastic neighbor embedding) on visualise les représentations latentes en 2D avec différentes perplexités. La perplexité est une estimation du nombre de voisins proches de chaque point. Elle nous permet de mieux comprendre la structure des données dans l'espace latent. D'après nos analyses, si la compression garde assez bien les informations sur les dimensions discriminantes propres à chaque classe, alors celle-ci devraient se regrouper dans la projection. Si ces caractéristiques discriminantes sont perdues, les chiffres seront entremêlés entre eux.

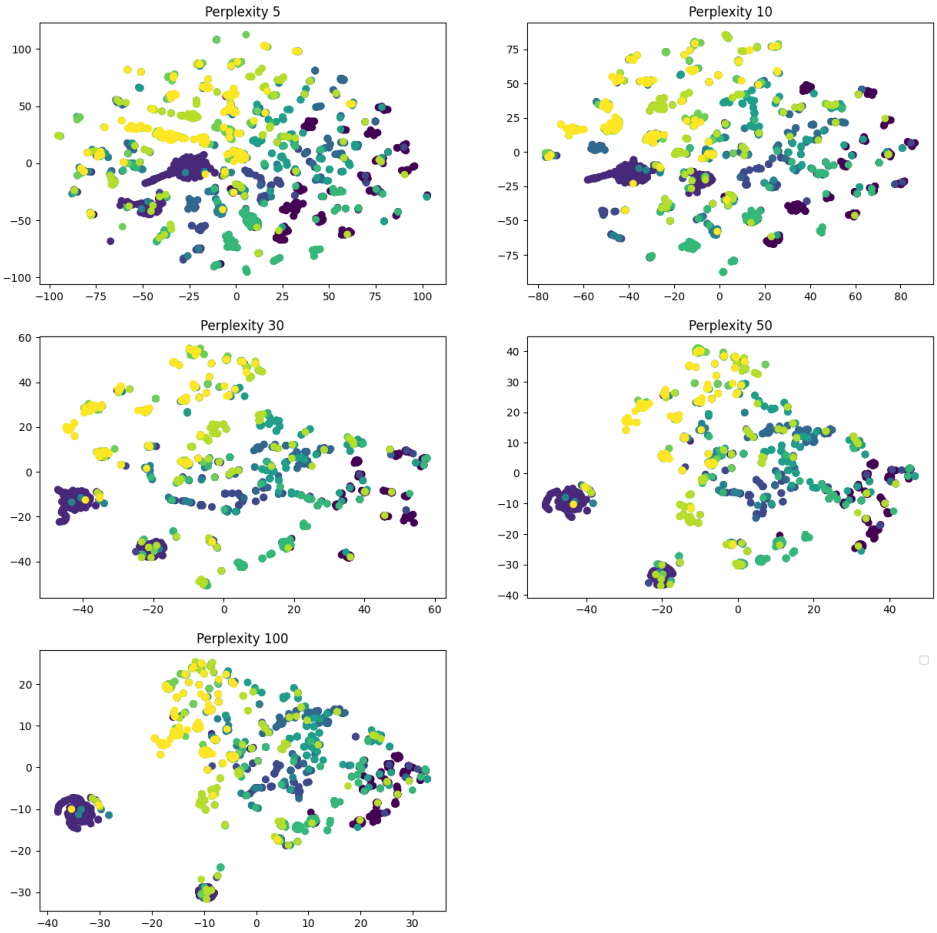


FIGURE 27 – TSNE avec différentes perplexité pour 100 neurones, 100 iterations et un gradient step de 1e-4

La visualisation t-SNE nous a permis de confirmer nos observations précédentes lors de l'affichage des images reconstruites. En projetant les données d'origine en 2D, nous pouvons clairement distinguer les différentes classes de chiffres, en particulier avec une perplexité (par exemple =5 pour 100 neurones). Cependant, en projetant les images reconstruites après compression dans un espace de dimension 100, nous avons constaté une perte d'informations discriminantes entre les classes. Malgré cela, nous avons remarqué une certaine similarité entre les données reconstruites de certaines classes. Par exemple, avec une perplexité de 5, les données de la classe jaune se regroupent vers le haut.

En réduisant ensuite le nombre de neurones à 10, la visualisation devient de moins en moins capable de distinguer les classes. Les classes ont subi une perte significative d'informations discriminantes, rendant de plus en plus

difficile la reconstruction précise d'une image appartenant à la bonne classe.

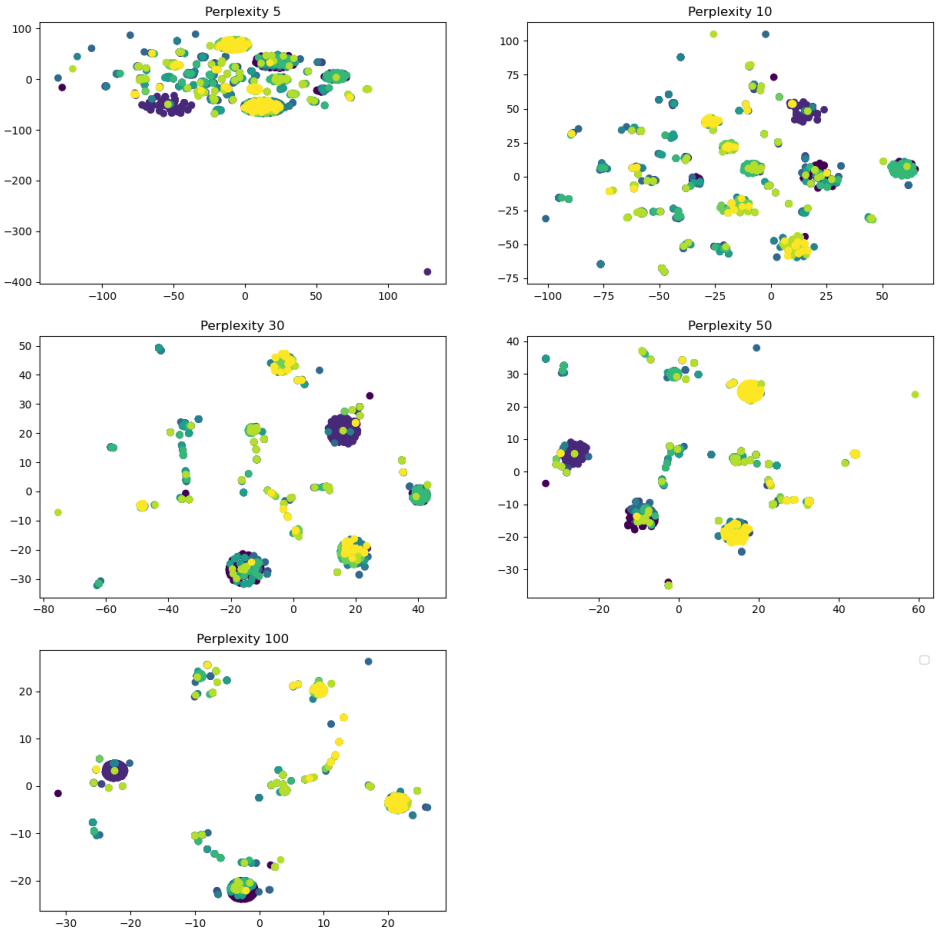


FIGURE 28 – TSNE avec différentes perplexité pour 10 neurones, 500 iterations et un gradient step de 1e-4

Pour finir, nous avons mis en évidence que la compression des données dans un espace de dimension réduite peut entraîner une perte d'informations discriminantes, ce qui affecte la distinction entre les différentes classes de chiffres. Cependant, malgré cette perte, certaines similarités entre les données reconstruites de certaines classes peuvent encore être identifiées.

7 Conclusion

En conclusion, notre projet de réseau de neurones a été une expérience enrichissante et instructive. Nous avons exploré divers aspects du fonctionnement des réseaux de neurones, tels que les couches, les fonctions d'activation et les différentes techniques d'optimisation.

Nos tests et expérimentations nous ont permis de mieux comprendre les performances et les limites de ces modèles. Nous avons pu observer l'importance du choix des hyperparamètres, tels que le nombre de neurones, le taux d'apprentissage et la taille du batch, sur les résultats obtenus. Nous avons également examiné l'influence du bruit dans les données sur les performances du réseau.

Les résultats obtenus ont été encourageants, montrant que nos modèles étaient capables d'apprendre à partir des données et de réaliser des prédictions précises. Nous avons également constaté l'importance de la convergence du réseau et la nécessité de trouver un équilibre entre le temps d'entraînement et les performances du modèle.

Ce projet nous a permis de développer nos compétences en programmation, en compréhension des réseaux de neurones et en analyse des résultats. Nous avons acquis une meilleure compréhension des concepts fondamentaux et des techniques avancées liées aux réseaux de neurones.

Enfin, ce projet ouvre la voie à de nombreuses possibilités d'amélioration et d'exploration future. Nous pourrions continuer à étudier d'autres architectures de réseaux de neurones, expérimenter avec des ensembles de données plus complexes et explorer des techniques d'optimisation plus avancées. Dans l'ensemble, ce projet a été une expérience gratifiante et nous a donné un aperçu concret du monde fascinant des réseaux de neurones.