ULB UNIVERSITÉ LIBRE DE BRUXELLES

INFO-F311 - Projet d'IA 4

Réseaux de neurones

Auteur:
Manuel Rocca - 000596086

Professeurs:
Tom Lenaerts
Assistants:
Axel Abels
Martin Colot
Yannick Molinghen
Pascal Tribel

Année académique 2025-2026

Contents

1 Introduction

Ce quatrième et dernier projet du cours INFO-F311 d'Intelligence Artificielle nous amène à étudier et implémenter en pratique un réseau de neurones, en particulier un *auto-encoder*. Nous présentons ci-dessous notre démarche scientifique.

2 Cadre expérimental

Comme exprimé dans l'introduction, nous nous penchons ici sur un *auto-encoder*. Un réseau neuronal tel que celui-ci est composé de deux partie: une qui encode et une qui décode. En particulier, il apprend, sur base d'un ensemble de données d'entraînement, à réduire la dimension, encoder la donnée en entrée et à la décoder.

Le jeu de données est un ensemble de chiffres de format 28x28 pixels appartenant à la base de donnée MNIST. Il est composé de deux fichiers:

- Un fichier d'entraînement de 60_000 éléments.
- Un fichier de test de 10_000 éléments.

La valeur de chaque pixel varie entre 0 et 255 mais sont normalisées à une valeur entre 0 et 1 dans notre programme pour améliorer l'efficacité et la précision du modèle (notamment en empêchant les valeurs comme 255 de dominer une petite valeur comme 2 lors d'une multiplication).

Le réseau neuronal a besoin de plusieurs paramètres en entrée:

- La dimension des données d'entrée, 784 dans notre cas (matrices 28x28 linéarisées).
- La dimension des données encodées $\hat{x}$, paramètre variable étudié dans les sections suivantes.
- Le *learning rate*, paramètre déjà étudié dans les autres projets, en particulier le 3e. Il nous semble pertinent de faire l'analogie avec les pas d'itération dans les simulations numériques.

Finalement, pour lancer l'entraînement de cet *auto-encoder*, il nous faut:

- Les données d'entraînement contenues dans le fichier *mnist_train.csv*.
- L'*epoch* correspondant au nombre de passages complets de toutes les données dans le réseau.
- Le *batch size* correspondant au nombre de paramètres/d'échantillons traités avant une mise à jour des valeurs du réseau.

3 Résultats

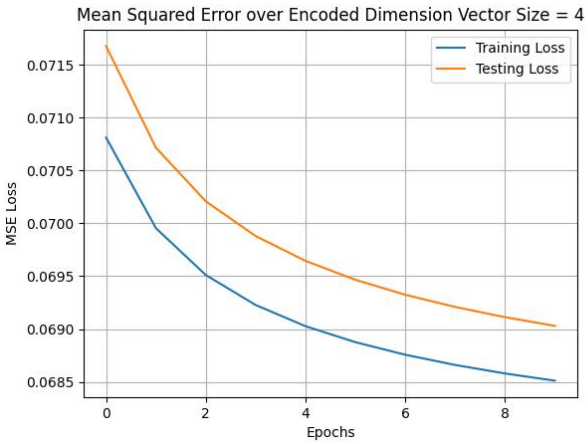
Une fois les notions posées dans les sections précédentes, il est maintenant temps pour nous de présenter et d'expliquer nos résultats expérimentaux. Différents paramètres pertinents sont analysés afin d'atteindre, de converger vers une compréhension plus exhaustive de ce qu'est une *auto-encoder*.

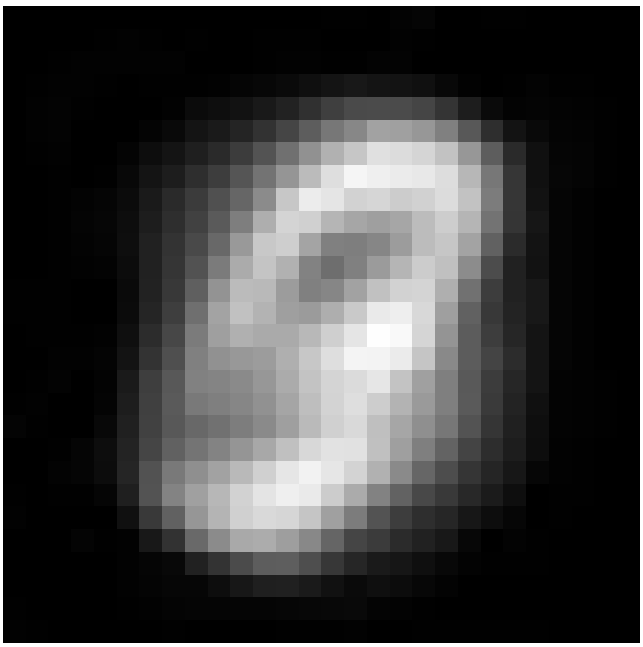
3.1 Impact de la taille du vecteur compressé $\hat{x}$

Le vecteur compressé est l'étape intermédiaire de l'*auto-encoder*. En effet, comme dit plus haut, il est divisé en deux parties: l'encodage et le décodage. La taille de ce vecteur est un paramètre crucial à régler de manière optimale pour obtenir une reconstruction de l'image initiale d'une qualité optimale.

Un vecteur compressé de petite dimension force le réseau de neurone à se focaliser sur les aspects importants de la figure. À l'opposé, un vecteur trop grand permettrait une reconstruction 100% fidèle à la réalité. Mais ce n'est pas le cas!

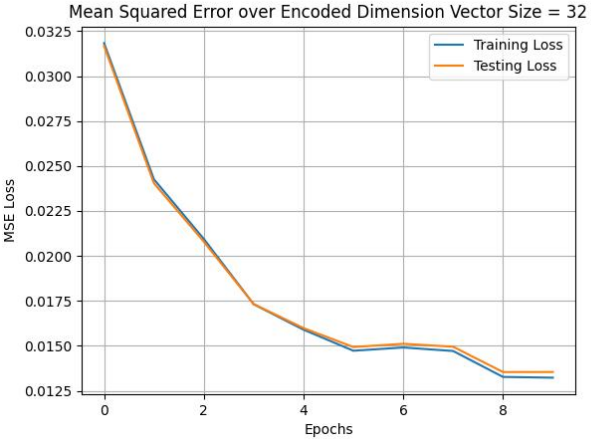
En effet, nos expériences prouvent le contraire. Avec un *learning rate* $\mu = 0.03$ et un nombre d'*epochs* $e = 10$, nous obtenons les résultats suivants:

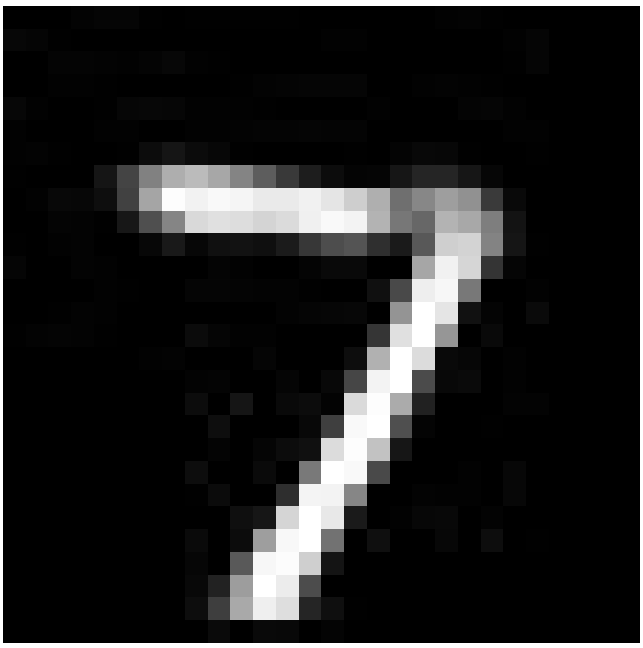




(a) Image d'un 7 reconstruit

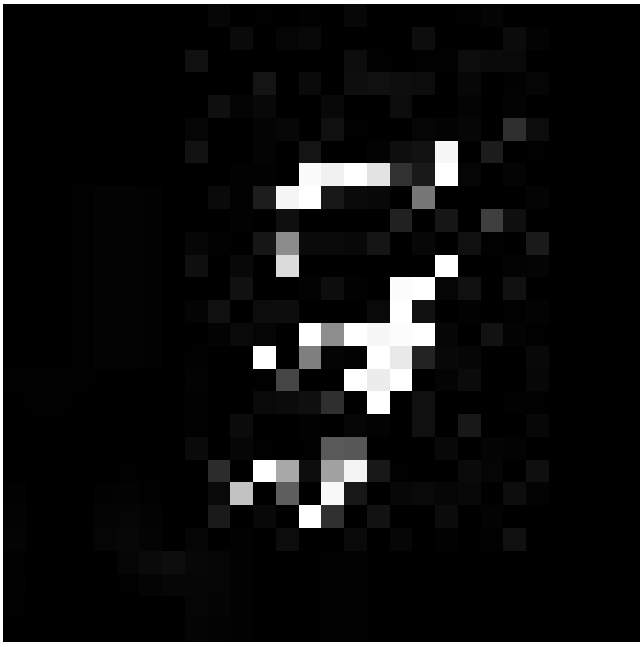
Figure 1: Taille du vecteur encodé $|\hat{x}| = 4$





(a) Image d'un 7 reconstruit

Figure 2: Taille du vecteur encodé $|\hat{x}| = 32$



(a) Image d'un 7 reconstruit

Figure 3: Taille du vecteur encodé $|\hat{x}| = 128$

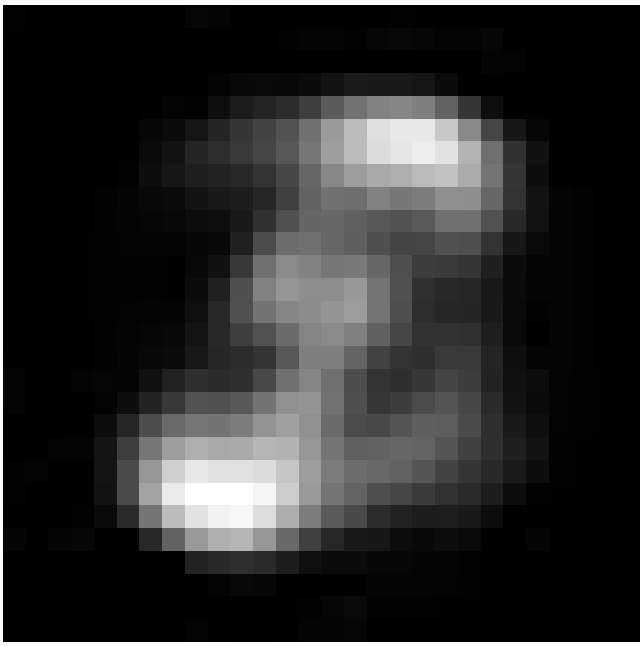
Nous observons que pour une dimension d'encodage trop petite, la *Minimum Squared Error* atteint une valeur minimale vers environ 0.0685 après 10 epochs. En augmentant la taille du vecteur, cette erreur diminue pour atteindre un minimum d'environ 0.0130 permettant une reconstruction de l'image nettement meilleure. Finalement avec une dimension de 128 (et au-dessus), l'erreur ne diminue pas en dessous de 0.1, environ dix fois plus élevée qu'avec une dimension plus petite. Ceci s'explique par le phénomène d'*overfitting*. En clair, cela signifie que le modèle apprend tellement bien les données d'entraînement que les performances diminuent

sur des données de test jamais vues [1].

3.2 Impact de la valeur du learning rate μ

L'impact du learning rate μ est le même (ou du moins très similaire) à son rôle dans l'apprentissage par renforcement. En effet, ce paramètre détermine la valeur du "saut" à chaque étape; à quel point les nouvelles valeurs écrasent les précédentes. Une analogie intéressante est celle de la balle lachée dans un bocal. Plus elle est rapide, plus elle se rapprochera du centre (qui serait alors le point de convergence) rapidement et continuerait son chemin pour remonter. La balle oscille ainsi jusqu'à arriver à un état de repos. La balle peut même sortir du bocal si elle est lancée à une vitesse trop importante (divergence). À l'opposé, une balle avec une vitesse initiale plus faible prend plus de temps à atteindre ce centre (converger) mais une fois au centre, elle s'y arrêtre et ne le dépasse pas.

Pour concrétiser ces propos, voici les résultats de nos expériences avec une dimension de vecteur econdé $\hat{x} = 32$ et un nombre d'*epochs* $e = 10$:

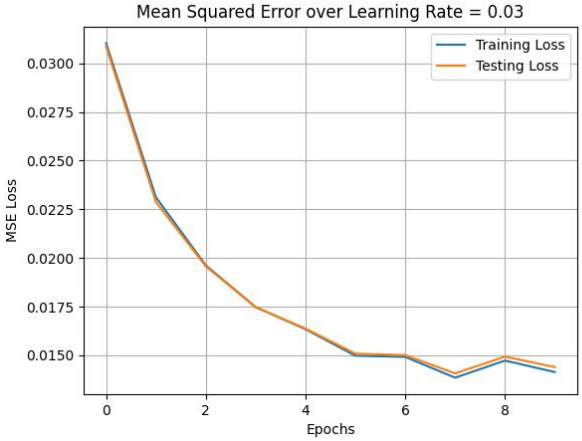


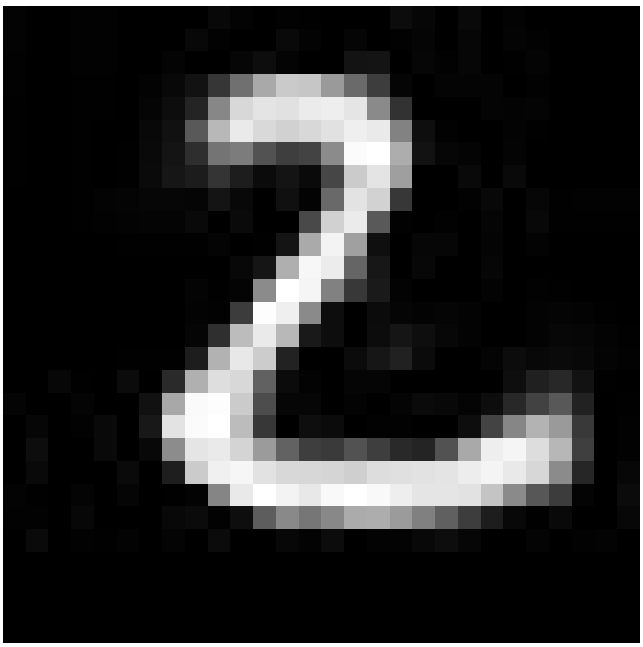
(a) Image d'un 2 reconstruit

Figure 4: Learning rate $\mu = 0.001$

Nous observons que les courbes descendent mais qu'il n'y a pas eu convergence. Avec plus d'*epochs*, la valeur de l'erreur devrait atteindre une valeur satisfaisante permettant une reconstruction optimale.

[1] Source: https://developers.google.com/machine-learning/crash-course/overfitting/overfitting

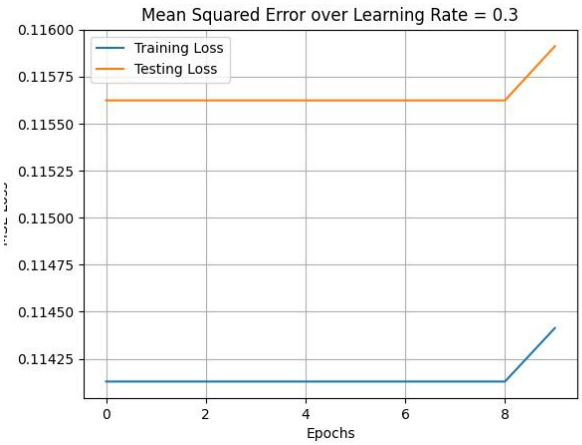


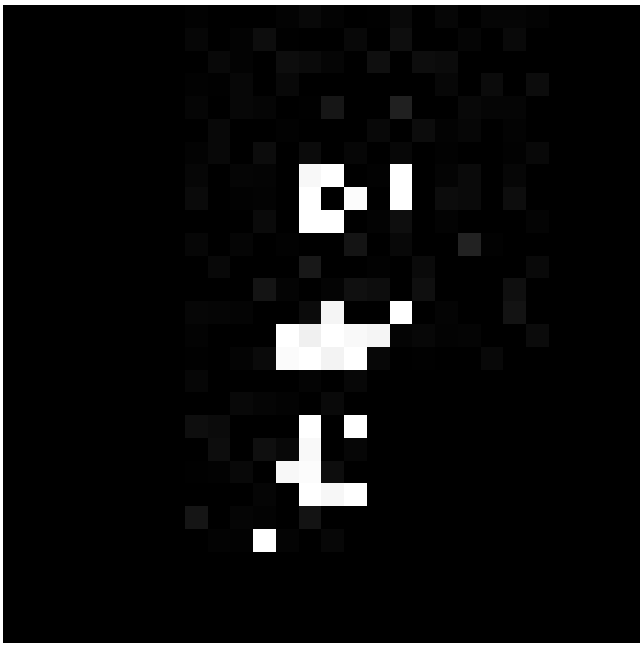


(a) Image d'un 2 reconstruit

Figure 5: Learning rate $\mu = 0.03$

Le learning rate permet ici d'atteindre une valeur d'erreur suffisament faible en 10 *epochs* pour une reconstruction optimale.





(a) Image d'un 2 reconstruit

Figure 6: Learning rate $\mu = 0.3$

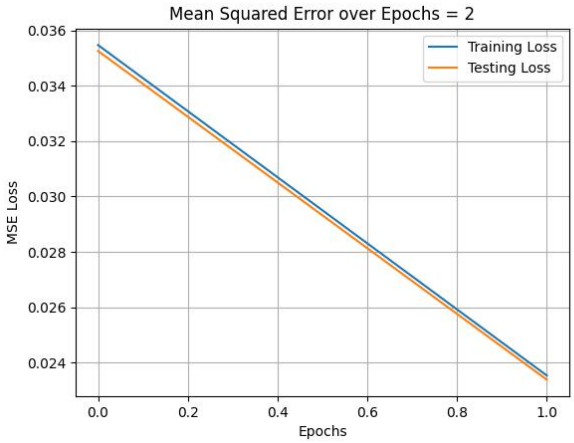
Avec une learning rate trop élevé, nous observons une divergence. En effet, les courbes ne descendent pas du tout.

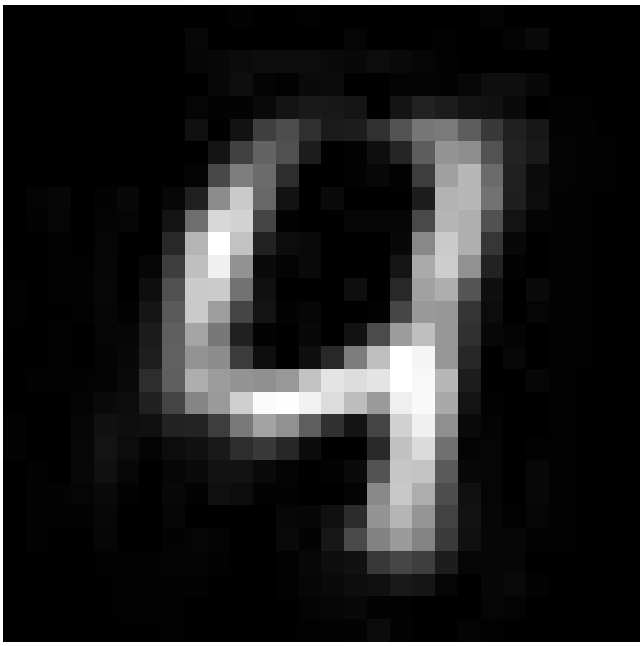
3.3 Impact du nombre d'epoch e

Finalement, ce dernier paramètre, le nombre d'*epochs* reflète grossièrement le nombre d'itérations. En effet, une *epoch* désigne un passage complet à travers l'ensemble des données

d'entraînement, permettant au modèle d'apprendre et de mettre à jour ses paramètres en fonction des données.

Voici donc nos résultats avec un learning rate $\mu = 0.03$ et une une dimension de vecteur econdé $\hat{x} = 32$:

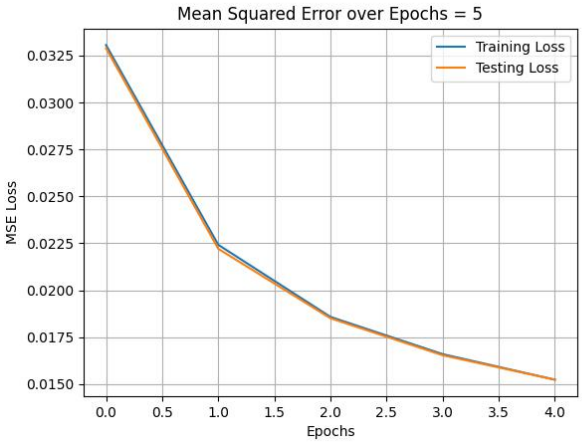


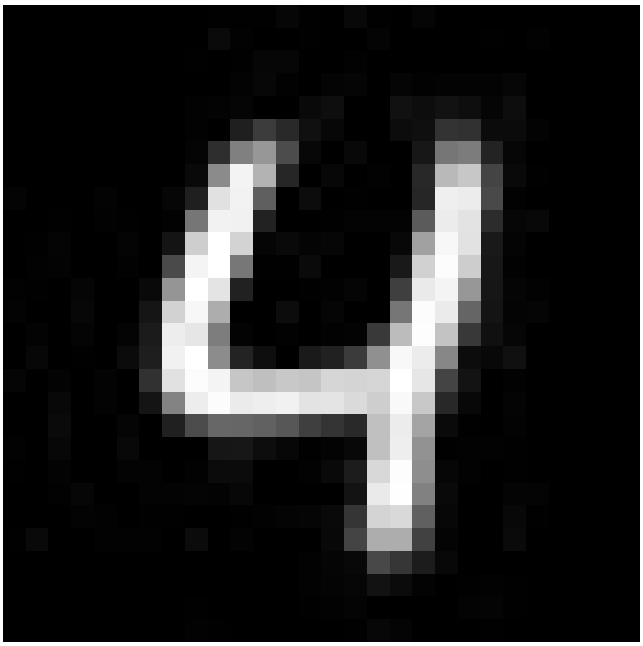


(a) Image d'un 4 reconstruit

Figure 7: Nombre d'*epochs* $e = 2$

Après deux *epochs*, la valeur de l'erreur est déjà relativement faible. Ceci nous permet donc de déjà distinguer le chiffre sur l'image reconstituée.

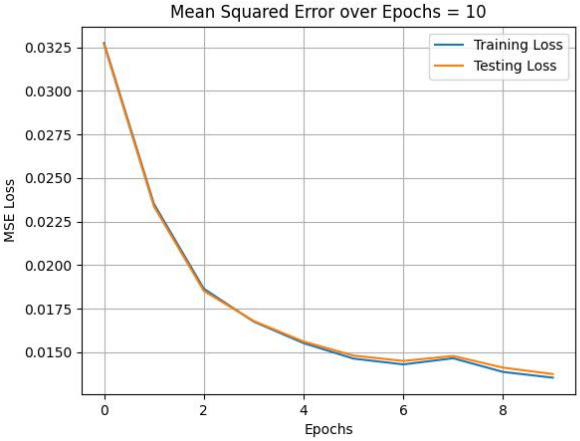


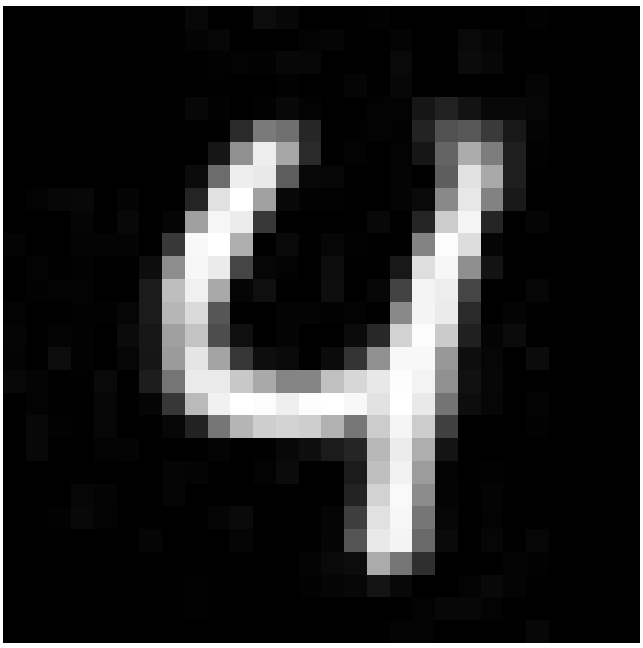


(a) Image d'un 4 reconstruit

Figure 8: Nombre d'*epochs* $e = 5$

Nous observons sur le graphe que la valeur de l'erreur commence doucement à se stabiliser. L'image est clairement reconstruite.





(a) Image d'un 4 reconstruit

Figure 9: Nombre d'*epochs* $e = 10$

La valeur de l'erreur a atteint un minimum, le gradient est quasiment nul. L'image est claire, il nous semble peu utile d'entrainer le réseau sur plus d'*epochs*. De plus, nous observons qu'après 5 *epochs* seulement, le résultat était déjà plus que correct. Cela peut se révéler intéressant dans le cas où nous travaillons avec un modèle très large où une *epoch* requiert une durée importante pour prendre fin.

4 Analyse

Dans cette section, nous considérons des aspects différents de l'*auto-encoder*; différents de l'optimisation des paramètres.

4.1 Hallucination de l'auto-encoder

Par contrainte de temps, nous n'avons pu expérimenter à ce sujet. L'hallucination d'un réseau neuronal est quelque chose de commun. Il est simple de s'imaginer ce qu'il se passerait.

Passer un vecteur encodé aléatoire à l'*auto-encoder* nous donnerait un résultat ressemblant à un chiffre même si l'image de base n'a rien à voir avec. Cela est dû à l'entraînement subi par le réseau. Il ne détecte pas que l'entrée est aberrante, il projette simplement ce vecteur dans l'espace des images plausibles apprises durant l'entraînement.

4.2 Optimisations possibles

Il est clair que notre réseau de neurones est relativement petit et peu complexe par rapport à ce qui se fait de nos jours. Le but étant ici de comprendre et d'apprendre, cela nous semble tout naturel. Cependant, il ne faut jamais négliger la réalité. Autrement dit, prenons tout de même en considération des modèles plus larges qui prennent plus de temps à s'entraînter. Dans ce cas ci, il est intéressant de considérer des optimisations pour améliorer les performances. Nous en abordons deux dans les sous-sections qui suivent.

4.2.1 Augmenter la profondeur

Utiliser un *auto-encoder* avec plusieurs couches, en augmentant sa profondeur offre plusieurs avantages [2]:

1. Peut réduire exponentiellement le coût de calcul pour représenter certaines fonctions
2. Peut réduire exponentiellement la quantité de données d'entraînement nécessaires pour apprendre certaines fonctions
3. Expérimentalement, les auto-encodeurs profonds offrent une meilleure compression que les auto-encodeurs peu profonds ou linéaires.

Naturellement, augmenter sa profondeur demande un espace en mémoire plus important. Le plus important est toujours de trouver le bon compromis.

4.2.2 Learning rate dynamique

Comme nous avons pu l'observer dans nos résultats, un learning rate trop lent permet en théorie d'obtenir de bons résultats, une bonne convergence mais nécessite beaucoup d'*epochs*. À l'opposé, un learning rate trop élevé peut diverger. L'idée est d'allier le meilleur des deux mondes. En d'autres termes, implémenter une learning rate qui, au début de l'entraînement est relativement élevé (sans tomber dans les excès bien sûr) permettant un début d'apprentissage plus rapide et peu stable. Au fur et à mesure de cet entraînement, diminuer le learning rate afin de stabiliser l'apprentissage et à terme, se rapprocher au mieux de la convergence.

5 Conclusion

Ce dernier projet d'Intelligence Artificielle conclut donc nos travaux sur le sujet. Lors de cette itération, nous avons abordé un sujet très à la mode à notre époque actuelle: les réseaux neuronaux et en particulier l'*auto-encoder*. Nous avons commencé par implémenter le programme en *Python* avant de pouvoir en extraire des données pertinentes et en produire des graphiques sur base desquels nous avons pu proposer une analyse espérons pertinente. Pour conclure ce thème, nous avons fini par aborder et analyser des aspects comme l'hallucination ainsi que certaines pistes d'optimisation.

[2]Source: https://en.wikipedia.org/wiki/Autoencoder#Advantages_of_depth