Projet 5 IA: Réseaux de neurones

Berthion Antoine - 566199

23 décembre 2024

1 Introduction

Dans ce rapport, nous analysons l'influence de divers **paramètres d'entraînement** sur des modèles de **réseaux de neurones**, et plus particulièrement ceux appartenant à la famille des **autoencodeurs**. Nous examinons également les **cas limites** ainsi que le **phénomène d'hallucination** souvent observé dans ces modèles.

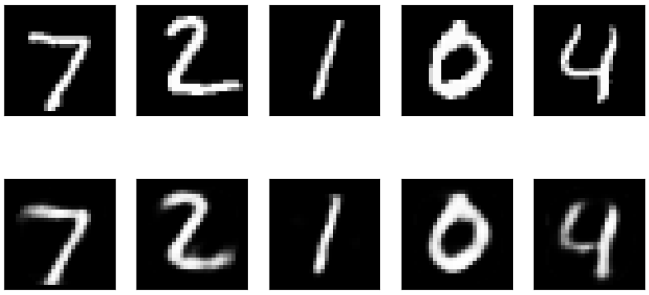

FIGURE 1 – Reconstruction des nombres après entraînement du modèle **mnist**

2 Expérimentations

Dans cette section, nous réalisons des expériences sur le paramétrage de l'entraînement de notre modèle. Nous commencerons par décrire le cadre expérimental dans la section 2.1, avant de présenter les différents résultats dans la section 2.2. Enfin, la section 3 sera dédiée à l'analyse de ces résultats.

2.1 Cadre Expérimental

Afin de mener correctement nos expériences, nous ferons varier un seul paramètre à la fois afin d'étudier les effets de ces variations sur les résultats de l'entraînement. Nous mesurerons les résultats, exprimés en termes de pourcentage de perte durant l'encodage et le décodage. Chaque expérience sera répétée cinq fois, et nous calculerons la moyenne des résultats pour assurer leur pertinence.

Nous ferons successivement varier la taille du vecteur compressé $\hat{x}$, la valeur du taux d'apprentissage μ, et enfin le nombre d'époques, noté e. Les résultats sont présentés ci-dessous, dans la section 2.2. Dès lors ou les paramètres ne varient pas, les valeurs sont fixées à :

— Dimension d'encodage de $\hat{x} = 32$
— Taux d'apprentissage $\mu = 0.01$
— Nombre d'epochs $e = 10$

2.2 Résultats

Dans cette sous-section, nous présenterons de manière succincte les résultats de nos expériences.

2.2.1 Influence du Learning-Rate

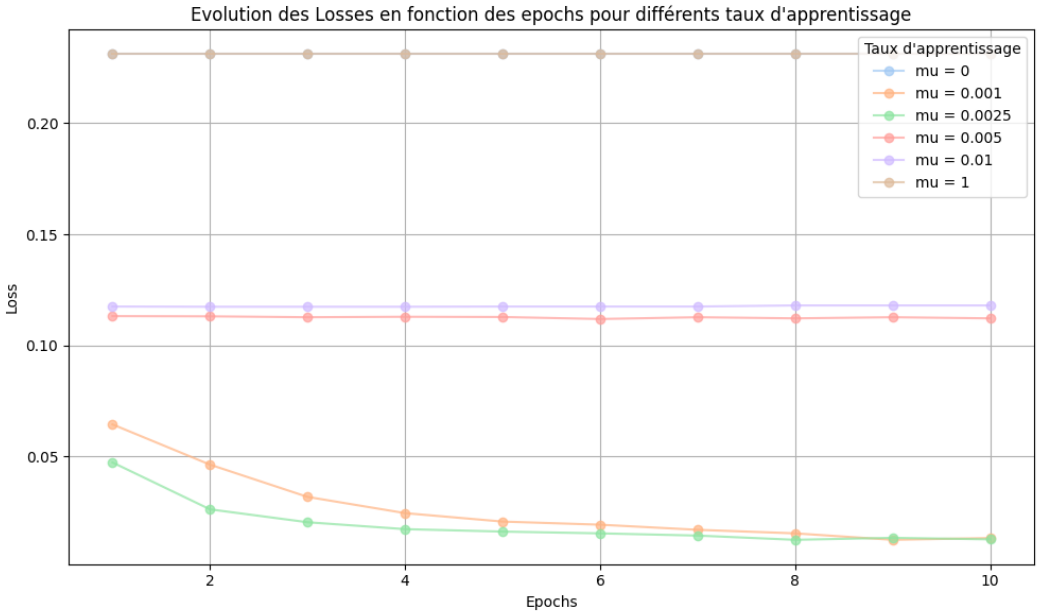

FIGURE 2 – Influence du *Learning-Rate* μ

2.2.2 Influence de la taille du vecteur $\hat{x}$

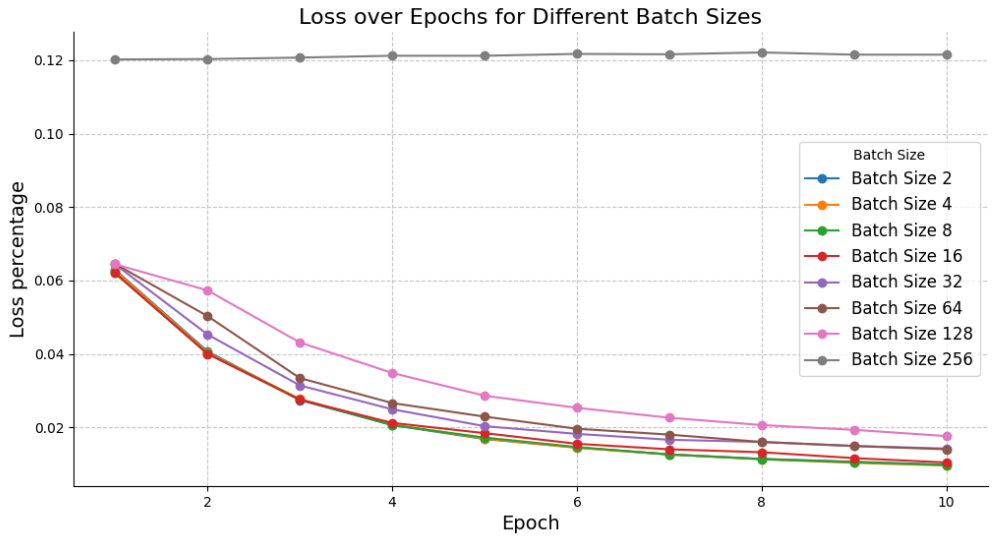

FIGURE 3 – Influence de la taille du vecteur $\hat{x}$

2.2.3 Influence du nombre d'Epochs

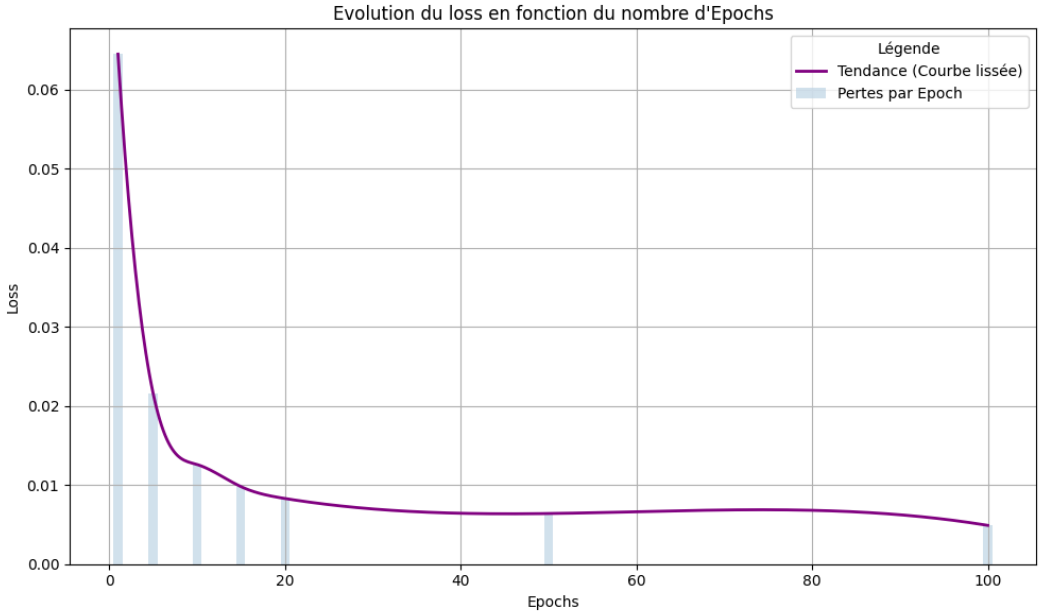

FIGURE 4 – Influence du nombre *d'Epochs e*

3 Discussion des résultats

Discutons à présent de l'impact de la variation des différents paramètres, testés expérimentalement dans la section 2.

3.1 Variation du learning rate.

Nous pouvons observer dans la figure 2 que les valeurs extrêmes, telles que 0 et 1, sont totalement inefficaces. Ces valeurs déterminent la vitesse à laquelle les poids (*weights*) du modèle sont ajustés lors de chaque itération. Une valeur de 0 empêche toute mise à jour des poids, rendant impossible tout apprentissage. De manière similaire, une valeur de 1 conduit à des mises à jour trop brutales, empêchant le modèle de converger vers une solution optimale. Les meilleures performances sont observées pour des valeurs comprises entre 0.001 et 0.003, ce qui est cohérent avec les bonnes pratiques d'entraînement des réseaux de neurones. En effet, avec un nombre modéré d'*epochs* (autour de 10), un faible learning rate permet des ajustements progressifs, limitant l'influence des erreurs individuelles. Cette approche évite que le modèle soit fortement perturbé par des fluctuations tardives dans les gradients, dues au bruit dans les données d'entraînement.

3.2 Variation du nombre d'*epochs*

La figure 4 montre que, pour des valeurs faibles, l'augmentation du nombre d'*epochs* améliore significativement la précision du modèle. Cependant, au-delà d'un certain seuil, l'amélioration devient marginale, voire négative, en raison du phénomène de *surapprentissage* (*overfitting*). Ce dernier se produit lorsque le modèle s'adapte trop aux données d'entraînement, perdant ainsi en capacité de généralisation sur des données non vues. Par ailleurs, nous notons que le temps d'entraînement croît de manière linéaire avec le nombre d'*epochs*, ce qui peut rapidement devenir un facteur limitant pour des valeurs élevées. Notre analyse expérimentale suggère qu'un compromis idéal se situe entre 10 et 20 *epochs* pour notre problème, offrant un bon équilibre entre précision et temps d'entraînement. Pour les cas extrêmes, une seule *epoch* entraîne des résultats médiocres, tandis qu'une augmentation à seulement deux *epochs* engendre une nette amélioration des performances.

3.3 Variation de la taille du vecteur encodé $\hat{x}$

En guise de dernière partie de cette section, discutons des résultats obtenus dans la figure 3, qui porte sur la taille d'encodage du vecteur $\hat{x}$. Les expériences montrent qu'une taille minimale d'encodage est favorable pour la performance du modèle. Cependant, nous observons que le temps d'entraînement augmente de manière exponentielle à mesure que la taille d'encodage diminue.
Plus précisément, les résultats obtenus pour des tailles de vecteur égales à 2, 4, et 8 sont très similaires en termes de perte, mais présentent des différences significatives en termes de temps d'entraînement. Une taille d'encodage de 8 semble offrir un compromis idéal pour notre modèle, combinant rapidité et performance. Cette taille permet d'obtenir des résultats proches de ceux des vecteurs plus petits, tout en évitant les temps de calcul excessifs associés à des encodages trop compacts.

4 Phénomène d'hallucination

Un aspect intéressant des autoencodeurs est leur comportement face à des vecteurs d'entrée aléatoires, jamais observés durant l'entraînement. Ce phénomène, appelé *hallucination*, correspond à la capacité du modèle à générer des sorties qui ressemblent aux données d'entraînement, bien que l'entrée ne soit pas issue de celles-ci.

En soumettant des vecteurs aléatoires, par exemple tirés d'une distribution uniforme ou normale, on observe que l'autoencodeur reconstruit des représentations plausibles dans l'espace des données originales. Cela s'explique par le fait que l'autoencodeur apprend une distribution réduite des données d'entraînement dans l'espace d'encodage. Ainsi, même des vecteurs encodés aléatoires proches de cette distribution génèrent des reconstructions qui reflètent les structures apprises.

Toutefois, si les vecteurs encodés sont trop éloignés de la distribution de l'espace d'encodage, les résultats deviennent incohérents ou chaotiques, mettant en évidence les limites de généralisation du modèle. Ce comportement peut être dangereux dans la mesure ou il pourrait induire les humains en erreur, donnant de très grandes garanties sur des données incohérentes présentant une structure coïncidant par hasard avec une structure apprise pendant l'entraînement du modèle.

5 Propositions d'améliorations

Pour améliorer les performances de l'autoencodeur, nous proposons deux modifications clés au réseau, en nous appuyant sur les principes d'implémentation décrits dans l'énoncé.

5.1 Ajout de régularisation pour éviter le surapprentissage

Le phénomène de surapprentissage (*overfitting*) peut limiter la capacité de généralisation du modèle sur des données non vues. Une approche efficace consiste à ajouter une régularisation L_2 (ou poids de décroissance). Cela implique de modifier les mises à jour des poids dans le *backward-forward* en y ajoutant un terme de pénalité proportionnel au carré des poids actuels. La mise à jour des poids pour une couche W_i deviendrait :

$$W_i = W_i - \mu \left(\Delta W_i + \lambda W_i \right)$$

où λ est le coefficient de régularisation, et ΔW_i représente le gradient calculé lors du *backward-forward*. Cette méthode limite les poids extrêmes, favorisant un modèle plus robuste. Une alternative intéressante pourrait être d'utiliser un dropout, qui consiste à désactiver aléatoirement des neurones durant l'entraînement pour éviter la dépendance excessive à certains chemins du réseau.

5.2 Augmentation de la capacité du modèle avec des couches supplémentaires

Une autre amélioration serait d'augmenter la capacité de représentation de l'autoencodeur en ajoutant des couches intermédiaires supplémentaires, aussi bien dans la phase d'encodage que dans celle de

décodage. Par exemple, en insérant une nouvelle couche intermédiaire x_{int} connectée par des matrices de poids W_{enc} et W_{dec}, les étapes du passage avant (*forward*) deviendraient :

$$\begin{aligned} x_{\text{enc}} &= \text{activation}(x \cdot W_1) \\ x_{\text{int}} &= \text{activation}(x_{\text{enc}} \cdot W_{\text{enc}}) \\ \hat{x} &= \text{activation}(x_{\text{int}} \cdot W_2) \\ x_{\text{dec\_int}} &= \text{activation}(\hat{x} \cdot W_{\text{dec}}) \\ y &= \text{activation}(x_{\text{dec\_int}} \cdot W_3) \end{aligned}$$

Cette approche pourrait permettre au réseau de capturer des structures plus complexes dans les données, à condition de maintenir une régularisation suffisante pour éviter un surapprentissage dû à l'augmentation du nombre de paramètres.

6 Utilisation des LLM

Dans cette courte partie, nous discuterons de l'utilisation des LLM dans le cadre du projet. Aucun LLM n'a été utilisé pour l'aspect implémentation ainsi que la compréhension du projet. Cependant, ce rapport à été corrigé du point de vue de la syntaxe et de la grammaire par DeepL ainsi qu'un modèle GPT. Notons tout de même qu'aucune des informations du rapport n'a été produite par une autre personne que l'auteur.

7 Conclusion

Ce rapport a permis d'explorer les performances des autoencodeurs en fonction de divers paramètres d'entraînement, ainsi que d'analyser des phénomènes spécifiques tels que l'hallucination. Nous avons expérimentalement identifié les configurations optimales, notamment pour le learning rate, le nombre d'epochs et la taille du vecteur d'encodage, tout en soulignant les limites imposées par le surapprentissage et la capacité de généralisation. Les propositions d'améliorations, telles que l'ajout de régularisation et l'augmentation de la complexité du réseau, ouvrent des perspectives intéressantes pour renforcer l'efficacité de ces modèles.