

Projet Final Machine Learning

Jeanne SENTENAC, Alexis EMANUELLI

Table des matières

1	Introduction	2
2	Réseau linéaire	2
3	Réseau non-linéaire	5
4	Encapsulage et Multi-classe	9
4.1	Combien de couches pour classifier n classes ?	10
4.2	Combien de neurones par couches ?	12
4.3	Analyse du learning rate avec fonction d'activation Softmax	15
4.4	Analyse du learning rate avec fonction d'activation LogSoftmax	18
5	Conclusion	20
6	Pistes d'améliorations	20
7	Code	21

1 Introduction

L'apprentissage automatique, et plus particulièrement les réseaux de neurones, sont devenus des outils essentiels pour résoudre des problèmes de classification et de prédiction. Dans ce contexte, la sélection et l'optimisation des architectures de réseaux de neurones sont des tâches cruciales pour obtenir des performances de modélisation optimales.

Dans cette étude, nous nous sommes intéressés à deux aspects de l'apprentissage automatique : les réseaux linéaires et les réseaux non linéaires. Tout d'abord, nous avons réalisé l'implémentation d'un réseau linéaire et évalué son impact sur un ensemble de données linéairement séparable. Ensuite, nous avons abordé le problème du XOR, un exemple classique de problème non linéaire, en utilisant un réseau de neurones à une couche cachée.

L'objectif de cette étude était de comprendre l'influence des différents paramètres, tels que le learning rate et le nombre d'itérations, sur les performances de ces réseaux. Pour cela, nous avons réalisé plusieurs expériences et générées des visualisations telles que des heatmaps, des courbes de perte et d'exactitude, ainsi que des frontières de décision.

Ensuite, nous nous sommes intéressés à l'encapsulation et à la classification multi-classe, deux concepts essentiels en apprentissage automatique. Nous avons exploré la manière dont les modèles d'apprentissage automatique. Nous avons utilisé un jeu de données contenant des chiffres de 0 à 9 pour étudier l'impact du nombre de couches et des fonctions d'activation sur la classification multi-classe. Pour cela nous avons utilisé le dataset MNIST.

Cette étude vise ainsi à approfondir notre compréhension des réseaux de neurones, de leur capacité à modéliser des relations linéaires et non linéaires, ainsi que des stratégies d'encapsulation et de classification multi-classe. Les résultats obtenus nous permettront d'identifier les paramètres et les architectures les plus appropriés pour différents types de problèmes et d'améliorer ainsi nos capacités en matière de modélisation et de prédiction.

2 Réseau linéaire

Dans cette section consacrée au réseau linéaire, nous allons explorer en détail l'implémentation de ce type de réseau et ses performances sur un ensemble de données linéairement séparable. Nous commencerons par étudier l'impact du learning rate et du nombre d'itérations sur le modèle, en analysant comment ces paramètres influencent sa précision. Ensuite, nous présenterons une heatmap illustrant visuellement cette relation, suivie des résultats moyens de 30 modèles distincts pour tenir compte de la variabilité liée à l'initialisation des paramètres. Nous avons réalisé l'implémentation d'un réseau linéaire et effectué des tests sur un ensemble de données linéairement séparable. Dans le cadre de cette expérience, nous avons étudié conjointement l'impact du learning rate et du nombre d'itérations sur ce modèle et sur cet ensemble de données.

Afin de visualiser l'effet de ces deux paramètres sur la précision du modèle, nous avons créé une heatmap qui représente l'exactitude du modèle en fonction de ces deux paramètres. Cette heatmap est présentée dans la Figure 1 ci-dessous.

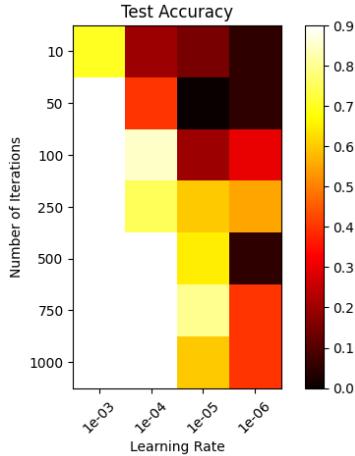


FIGURE 1 – Heatmap de l’exactitude du modèle en fonction du learning rate et du nombre d’itérations pour le seed 42

Il convient de noter que les résultats peuvent varier en fonction de l’initialisation des paramètres du modèle. Pour pallier à cette variabilité, nous avons réalisé une seconde série de tests. Cette fois-ci, nous avons instancié le modèle 30 fois de manière distincte en utilisant des ensembles de validation et de test différents à chaque fois. Les résultats obtenus ont ensuite été moyennés.

La Figure 2 présente les résultats moyens de ces 30 modèles instantiés séparément, en mettant en évidence l’effet du learning rate et du nombre d’itérations sur la précision du modèle.

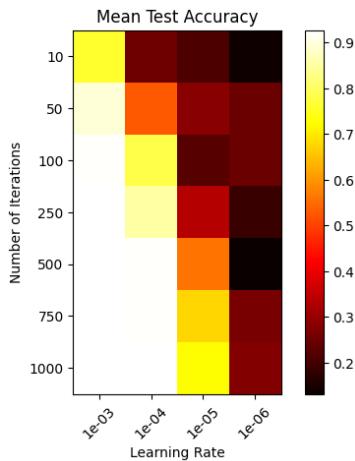


FIGURE 2 – Résultats moyens de 30 modèles instanciés séparément.

Ces deux visualisations nous permettent d’analyser de manière approfondie l’impact du learning rate et du nombre d’itérations sur la performance du modèle linéaire.

Le modèle obtient une accuracy moyenne de 1 pour plusieurs gammes de paramètres, pour choisir nous avons donc pris le nombre d’itérations minimales possible, et le learning rate en conséquence. Ce qui a fini par nous donner pour ce problème 250 itérations et un learning rate de 10-3.

Après avoir déterminé les hyperparamètres optimaux, nous avons entraîné le réseau linéaire sur l’ensemble de données linéairement séparable. Pour évaluer la performance du modèle pendant l’entraînement,

nous avons tracé les courbes de la perte (loss) et de l'exactitude (accuracy) en fonction des époques. De plus, nous avons généré un graphique de la frontière de décision du modèle avec les données.

La Figure 3 présente la courbe de la perte par époque (loss per epoch) du modèle en entraînement

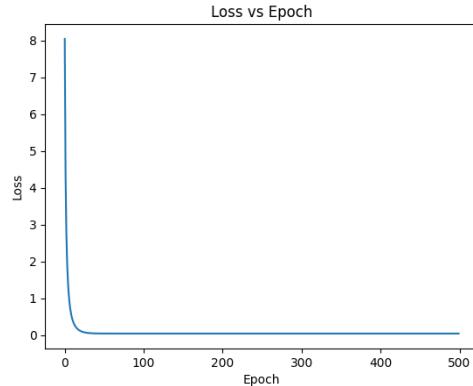


FIGURE 3 – Courbe de la perte par époque (loss per epoch).

La Figure 4 illustre la courbe de l'exactitude par époque (accuracy per epoch) du modèle en entraînement.

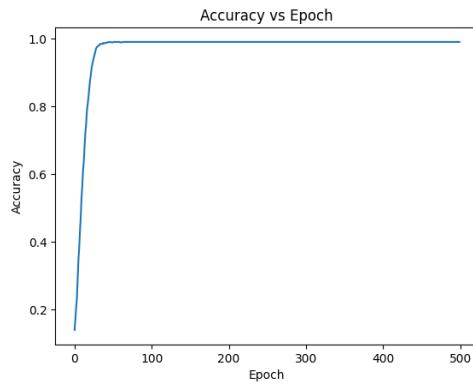


FIGURE 4 – Courbe de l'exactitude par époque (accuracy per epoch).

On s'est ensuite intéressés à des données légèrement bruitée, et l'algorithme s'est avéré converger tout de même. La frontière de décision en entraînement est représentée figure 5 et celle en test Figure 6

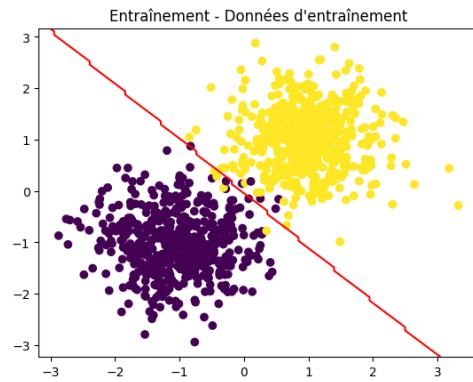


FIGURE 5 – Frontière de décision du modèle avec les données d'entraînement.

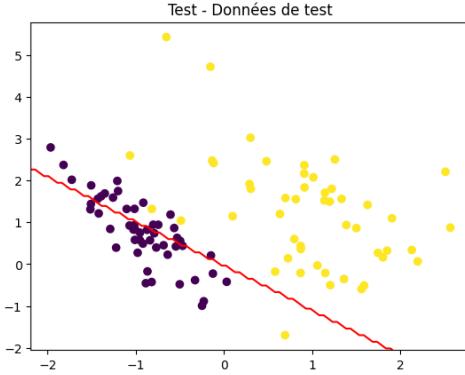


FIGURE 6 – Frontière de décision du modèle avec les données de test.

Dans la Figure 5, nous pouvons observer que la frontière de décision du modèle réussit à séparer efficacement les deux classes de l'ensemble de test. Les points des classes positives et négatives sont correctement classés de part et d'autre de la frontière, ce qui indique que le modèle a appris avec succès à distinguer les motifs caractéristiques des deux classes.

Il est important de noter que la performance du modèle sur l'ensemble de test peut varier en fonction de la complexité des données et de la qualité de l'apprentissage. Il est recommandé de toujours évaluer le modèle sur des données indépendantes pour avoir une estimation plus fiable de sa capacité à généraliser.

La Figure 6 (Figure 6) fournit donc une représentation visuelle de la capacité du modèle à classer les données d'un ensemble de test, ce qui permet d'analyser et d'évaluer ses performances de manière plus complète.

Ces 4 graphiques nous donnent une vue d'ensemble de la performance du modèle linéaire après avoir sélectionné les hyperparamètres optimaux. Ils permettent de mieux comprendre l'apprentissage du modèle, la convergence de la perte et l'adaptation de la frontière de décision aux données.

3 Réseau non-linéaire

Le problème du XOR est un exemple classique de problème non linéaire qui ne peut pas être résolu par un réseau de neurones linéaire. Pour résoudre ce problème, nous allons utiliser un réseau de neurones à une couche cachée.

Notre réseau de neurones aura une couche cachée avec 30 neurones et des fonctions d'activation spécifiques. Nous utiliserons la fonction d'activation tangente hyperbolique (\tanh) dans la couche cachée et la fonction d'activation sigmoïde en sortie.

Pour trouver les paramètres optimaux du réseau de neurones, nous avons effectué une analyse en utilisant différentes combinaisons de learning rates et de nombre d'epochs. Les résultats ont été enregistrés sous forme d'une heatmap montrant l'accuracy moyenne sur 30 essais pour chaque combinaison de paramètres. Dans un premier temps un nombre arbitraire de 30 neurones dans la couche cachée a été choisi pour effectuer les simulations.

La Figure 7 présente la heatmap obtenue. Nous pouvons observer que certains paramètres offrent de meilleurs résultats en termes d'accuracy moyenne. Ces paramètres seront utilisés pour sélectionner les

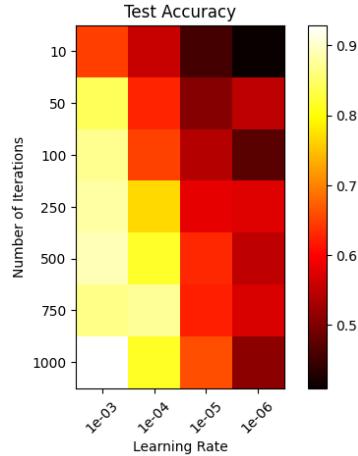


FIGURE 7 – Heatmap de l’accuracy moyenne en fonction du learning rate et du nombre d’epochs

paramètres optimaux du réseau. En l’occurrence, on retiendra un learning rate de 1×10^{-3} et 1000 itérations.

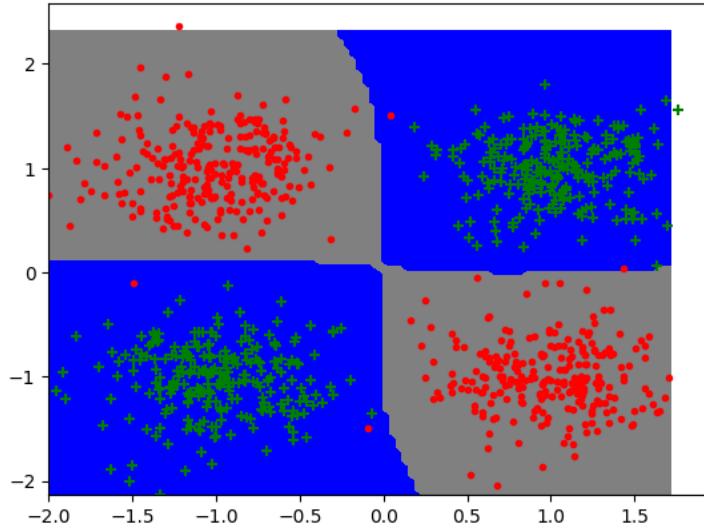


FIGURE 8 – Frontière de décision du problème XOR avec une précision de 0.997 en test

La Figure 8 présente la frontière de décision obtenue pour le problème du XOR en utilisant les paramètres optimaux que nous avons trouvés. Le réseau de neurones a été entraîné avec un learning rate de 1×10^{-3} sur 1000 itérations, avec 30 neurones cachés.

Cette frontière de décision a atteint une précision de 0.997 lors des tests, ce qui indique une très bonne capacité du réseau de neurones à séparer les exemples positifs des exemples négatifs du problème du XOR.

Nous sommes ensuite revenu sur notre choix initial arbitraire du nombre de neurones dans la couche cachée,

La Figure 9 présente l’accuracy moyenne en test du réseau pour différentes configurations de neurones dans la couche cachée. Les barres d’erreurs relatives sont également indiquées, ce qui permet de visualiser la variabilité des résultats.

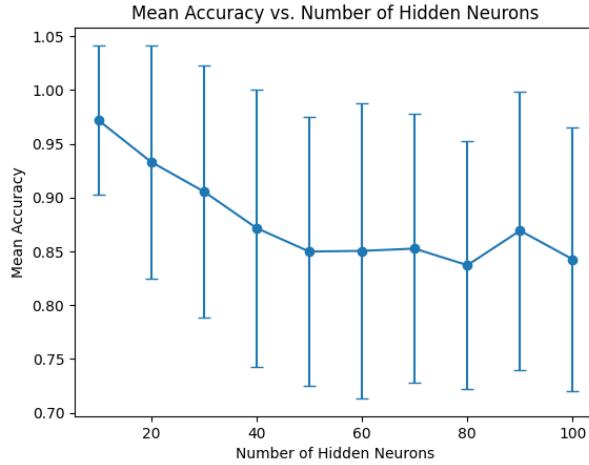


FIGURE 9 – Accuracy moyenne en test avec les barres d'erreurs relatives en fonction du nombre de neurones dans la couche cachée

Nous avons utilisé 50 seeds différents pour obtenir une estimation robuste de la performance du réseau. Chaque seed représente une répartition aléatoire du jeu d'entraînement et de test, ce qui permet de prendre en compte différentes variations des données. (on s'approche donc d'une cross validation en quelque sorte)

Nous pouvons observer dans la Figure 9 que l'accuracy en test est plus élevée pour un nombre de neurones égal à 10. Cependant, ces résultats nous ont encouragés à effectuer d'autres simulations avec moins de neurones pour évaluer l'effet de la diminution du nombre de neurones sur les performances du modèle.

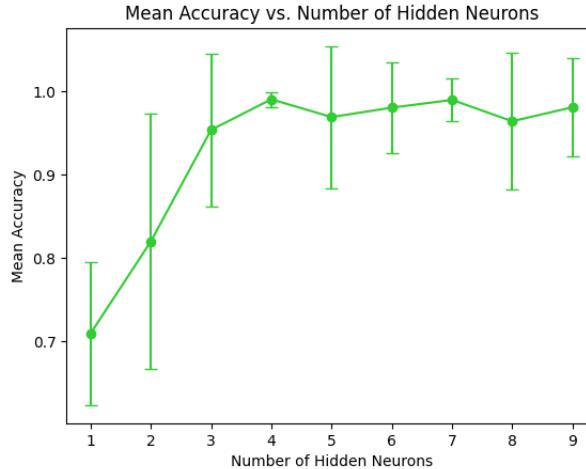


FIGURE 10 – Accuracy du modèle en test pour un nombre de neurones dans la couche cachée variant de 1 à 10

Voici le texte avec les corrections apportées :

En réduisant le nombre de neurones dans la couche cachée, nous avons constaté que l'accuracy du modèle varie. La Figure 10 présente les résultats de ces simulations, où nous avons fait varier le nombre

de neurones dans la couche cachée de 1 à 10. Nous pouvons observer que l'accuracy la plus élevée est atteinte pour 4 neurones, et c'est également celle présentant la variabilité la plus faible, ce qui en fait un choix idéal pour la suite.

La variabilité des résultats en test varie également en fonction du nombre de neurones dans la couche cachée, suggérant ainsi des architectures intrinsèquement plus "stables" et adéquates que d'autres dans ce cas précis.

Il est également important de noter que pour un seul neurone dans la couche cachée, les résultats sont en moyenne plus bas et beaucoup plus variables. Cela est dû au fait que le réseau de neurones à 1 couche cachée est équivalent à un Perceptron standard, à la seule différence des fonctions d'activation non linéaires. Ce réseau n'est donc pas en mesure de séparer linéairement le problème du XOR, ce qui explique les résultats moyens de 0.7, puisque l'accuracy maximale théorique obtenable avec une droite pour ce problème est de 0.75.

Pour compléter cette analyse, nous avons à nouveau simulé la heatmap, mais cette fois avec un modèle à 4 neurones cachés. La Figure 11 présente cette nouvelle heatmap obtenue.

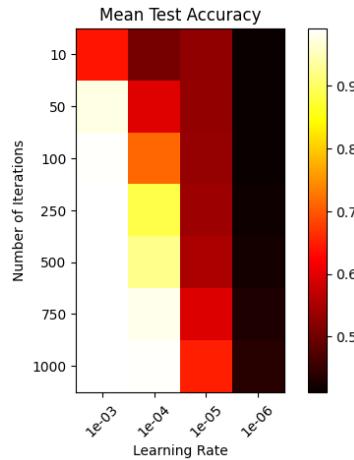


FIGURE 11 – Heatmap de l'accuracy moyenne pour un modèle à 4 neurones cachés

En observant la Figure 11, nous pouvons visualiser l'accuracy moyenne obtenue pour différentes combinaisons de learning rate et de nombre d'epochs en utilisant un modèle à 4 neurones cachés. Nous remarquons que toutes les accuracées sont améliorées par rapport au modèle à 30 neurones cachés ! Cette heatmap nous permet d'identifier les combinaisons de paramètres qui conduisent à de bonnes performances du modèle.

Nous pouvons donc garder le même learning rate que précédemment, mais notons que l'algorithme a besoin de beaucoup moins d'itérations pour obtenir une accuracy supérieure à 0.9 en test.

Pour mieux comprendre l'impact de cette configuration sur la frontière de décision du modèle, nous avons également tracé la frontière de décision correspondante pour un apprentissage avec 4 neurones, un learning rate de 1e-3 et seulement 250 itérations. La Figure 12 présente cette frontière de décision.

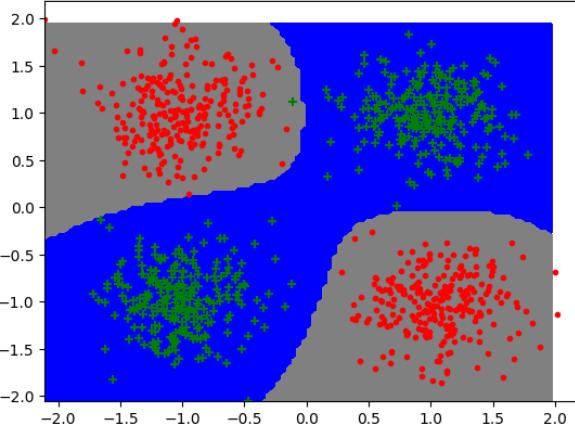


FIGURE 12 – Frontière de décision pour un apprentissage avec 4 neurones dans la couche cachée, pour un learning rate de 0.001 et 250 itérations. L'accuracy obtenue est de 0.999

En analysant la Figure 12, nous pouvons observer que la frontière de décision obtenue avec 4 neurones dans la couche cachée permet de bien séparer les exemples positifs des exemples négatifs du problème étudié. Cette configuration semble donc être un bon compromis entre performance et complexité du modèle, même si la frontière de décision présentée est plus "originale" que celle de la Figure 8 dans le sens où elle s'apparente moins à un échiquier. Cela peut être dû au fait que nos données ne sont pas très bruitées et qu'il est donc possible dans ce cas de tirer profit de la légère marge les séparant pour maximiser notre accuracy.

En conclusion, le problème du XOR nécessite un réseau de neurones non linéaire pour être résolu efficacement. En utilisant une combinaison optimale de paramètres, tels qu'un nombre approprié de neurones dans la couche cachée, un learning rate adéquat et un nombre suffisant d'itérations, nous pouvons obtenir des performances élevées pour la classification du XOR, qui restent très abordables computationnellement ! Cela montre qu'augmenter le nombre de neurones n'améliore pas toujours le problème, là où un choix judicieux d'hyperparamètres le peux.

4 Encapsulation et Multi-classe

L'encapsulation et les classes multiples sont deux concepts essentiels en apprentissage automatique qui permettent de résoudre des problèmes de classification complexes. L'encapsulation fait référence à la capacité d'un modèle d'apprentissage automatique à regrouper des fonctionnalités et des données connexes dans une seule entité, appelée classe. Une classe peut représenter un groupe d'observations partageant des caractéristiques similaires ou appartenant à la même catégorie.

En utilisant des techniques d'encapsulation et de classes multiples, les modèles d'apprentissage automatique peuvent apprendre à identifier et à prédire avec précision les différentes catégories présentes dans les données. Ces concepts jouent un rôle crucial dans de nombreux domaines de l'apprentissage automatique, tels que la vision par ordinateur, le traitement du langage naturel et la bioinformatique, où la classification précise des données est essentielle pour la prise de décision et la résolution de problèmes

complexes. Pour les analyses qui suivent, nous avons utilisé les données générées par `sklearn.load_digits` : des matrices 8x8 (64 pixels) de chiffres de 0 à n, avec n représentant le nombre de classes.

4.1 Combien de couches pour classifier n classes ?

Nous avons exploré l'optimisation d'un réseau de neurones dans le cas multi-classe. Notre objectif principal était de déterminer le nombre optimal de couches nécessaires dans un réseau de neurones pour classifier n classes. Nous avons également comparé les performances des fonctions d'activation LogSoftmax et Softmax dans ce contexte. Nous avons mené plusieurs expériences en utilisant différents nombres de classes à classifier. Nous avons d'abord testé notre modèle pour la classification de 10 classes en utilisant les fonctions d'activation LogSoftmax et Softmax. Ensuite, nous avons poursuivi nos tests en utilisant uniquement la fonction d'activation LogSoftmax pour les classifications à 2, 4, 6 et 8 classes. Pour chaque réseau de neurones à x couches, nous avons fixé le nombre de neurones à 64 pour la couche d'entrée, 32 pour les couches cachées et n pour la couche de sortie, avec n correspondant au nombre de classes à classifier. Pour la classification des 10 classes, nous avons observé, en regardant la loss en test et en train, que le réseau de neurones idéal utilisant la fonction d'activation LogSoftmax comportait 2 couches cachées (Figure 13 (b)). En revanche, pour la fonction d'activation Softmax, nous avons obtenu de meilleurs résultats en utilisant 3 couches cachées (Figure 13 (a)).

Pour les classifications à 2, 4, 6 et 8 classes (Figure 14), en utilisant la fonction d'activation LogSoftmax, nos résultats ont montré que le nombre optimal de couches cachées se situait entre 1 et 2. Ces résultats confirment que le choix du nombre de couches cachées dépend fortement du nombre de classes à classifier ainsi que de la fonction d'activation utilisée.

Il convient de noter que ces résultats sont spécifiques à notre ensemble de données et à notre architecture de réseau de neurones choisie. D'autres ensembles de données et architectures de réseaux peuvent présenter des comportements différents.

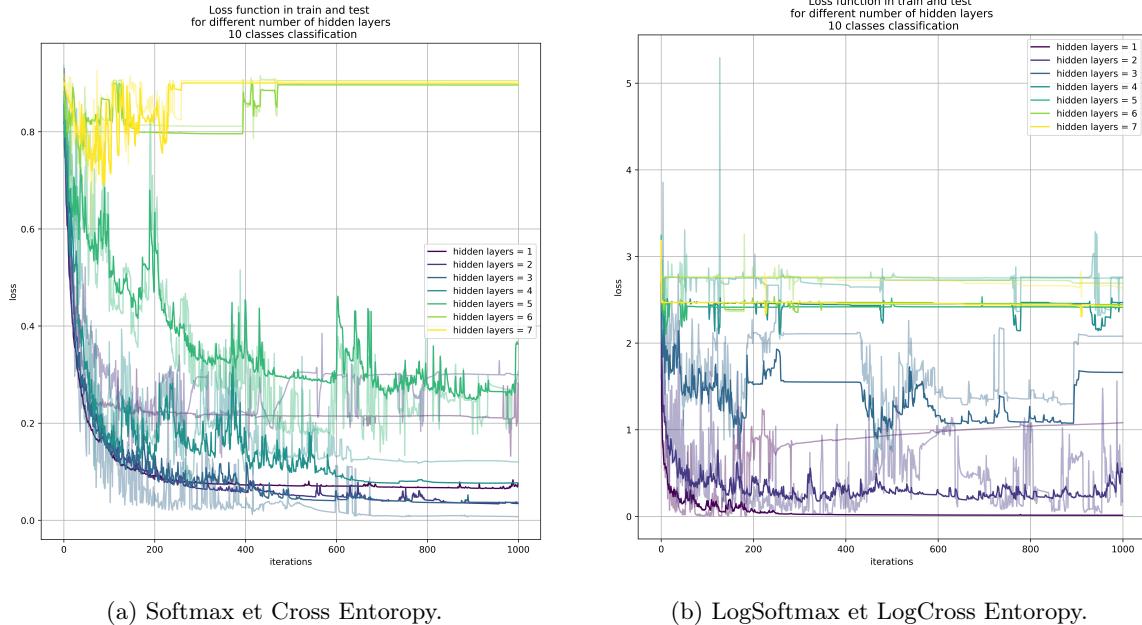
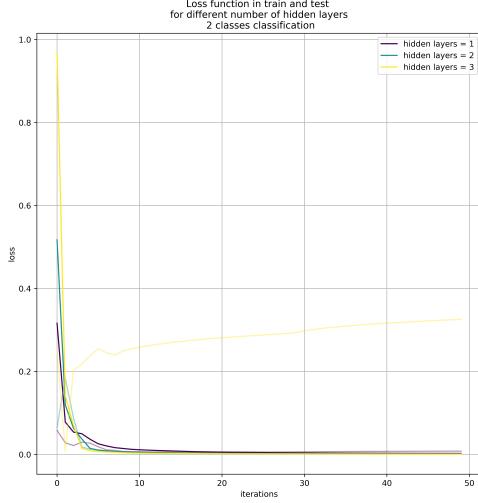
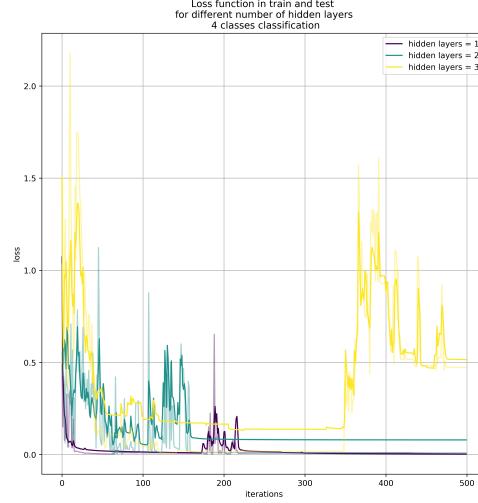


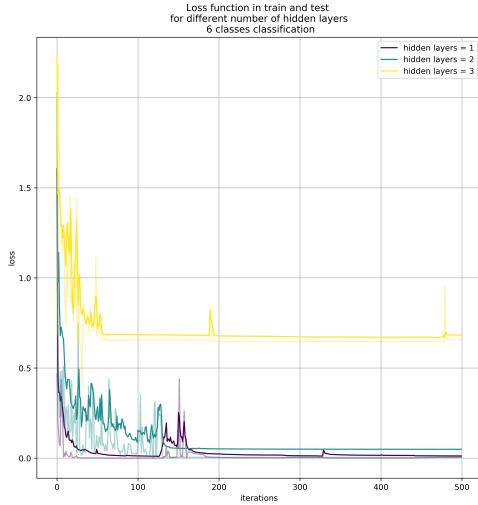
FIGURE 13 – Loss en train et en test pour les fonctions d’activation LogSoftMax et Softmax et des loss LogCrossEntropy et CrossEntropy. La couleur la plus foncée représente la loss en train tandis que la couleur plus claire représente la loss en test. Le nombre d’itérations totales est de 1000. Le réseau classifie 10 classes. Nombre d’inputs : 64, nombre de neurones pour chaque couche cachée : 32, nombre d’outputs : 10. La légende spécifie le nombre de couches cachées utilisées pour chaque couleur différente.



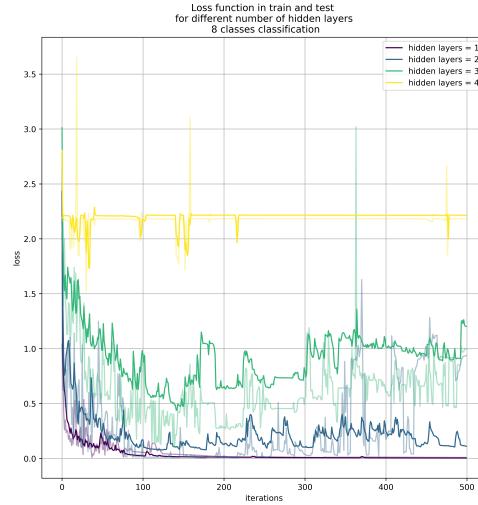
(a) Classification de 2 chiffres, 0 et 1



(b) Classification de 4 chiffres de 0 à 3



(c) Classification de 6 chiffres de 0 à 5.



(d) Classification de 8 chiffres de 0 à 7

FIGURE 14 – Loss en train et en test pour la fonction d’activation LogSoftMax et la loss LogCrossEntropy. La couleur la plus foncée représente la loss en train tandis que la couleur plus claire représente la loss en test. Le nombre d’itérations totales est de 50 pour la figure (a) et de 500 pour les autres. Le réseau classifie n classes, spécifié pour chaque image. Nombre d’inputs : 64, nombre de neurones pour chaque couche cachée : 32, nombre d’outputs : n. La légende spécifie le nombre de couches cachées utilisées pour chaque couleur différente.

4.2 Combien de neurones par couches ?

Dans cette deuxième partie de notre rapport, nous allons utiliser les informations de la section précédente pour effectuer des analyses sur le nombre optimal de neurones par couche. En nous basant sur un réseau à 2 couches cachées pour la fonction d’activation LogSoftmax et à 3 couches pour la fonction d’activation Softmax, nous allons examiner la perte (loss) de plusieurs réseaux de neurones en faisant varier le nombre de neurones par couche.

Pour le réseau utilisant la fonction d’activation Softmax, nous allons considérer les combinaisons

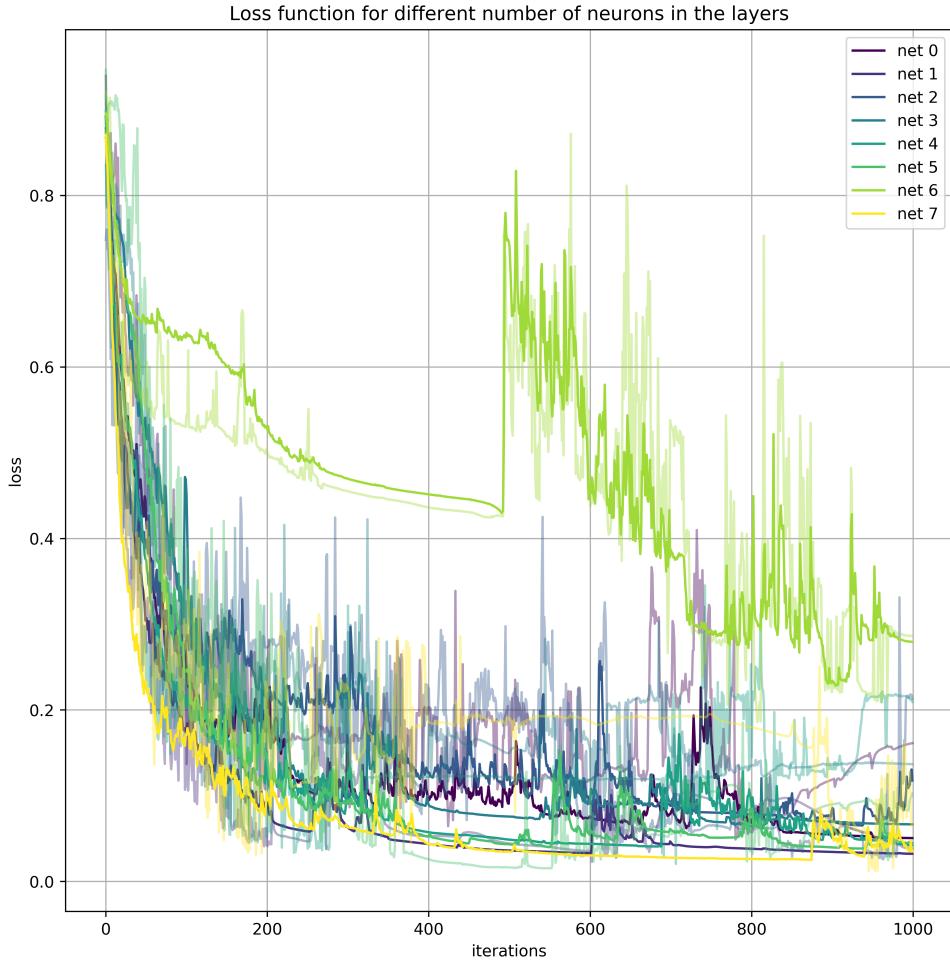
suivantes de neurones par couche : 32 ou 52 neurones pour la première couche cachée, 16 ou 30 neurones pour la deuxième couche cachée, et 8 ou 14 neurones pour la troisième couche cachée. Cela nous donne un total de 8 réseaux différents à étudier. Comme plus de 10 losses différentes auraient saturé l'image, car on aurait pas pu distinguer laquelle était la meilleure, c'est pourquoi nous avons choisi un maximum de 2 numéros différents par couche, pour avoir un total de 8 losses différentes.

En revanche, pour le réseau utilisant la fonction d'activation LogSoftmax, nous allons prendre 32 ou 52 neurones pour la première couche cachée, et 16 ou 30 neurones pour la deuxième couche cachée. Nous examinerons toutes les combinaisons possibles de ces chiffres, soit un total de 4 réseaux différents.

Lors de nos expérimentations avec les réseaux utilisant la fonction d'activation Softmax, nous avons constaté que celui qui fonctionne le moins bien, voire pas du tout, est celui qui présente le plus grand écart de nombre de neurones entre les couches. Par exemple, pour ce réseau, nous avons 52 neurones dans la première couche cachée, 30 neurones dans la deuxième couche cachée et seulement 8 neurones dans la troisième couche. Le réseau qui fonctionne le mieux en train est celui qui a le nombre maximum de neurones pour chaque couche (net 7, Figure 15). Cette observation est cohérente, car plus nous avons de neurones par couche, plus l'algorithme est précis en train. Cependant, ce n'est pas le cas en test. En effet, il semblerait que net 7 ait un problème de sur-apprentissage, ce qui est prévisible à cause de son grand nombre de neurones par couche. Le réseau le plus performant en termes de cohérence entre la loss en train et la loss en test est net 5 (aussi net 1 est pas mal), qui a 52 neurones dans la première couche cachée, 16 dans la deuxième et 14 dans la troisième. Ceci suggère que dans ce cas spécifique, il n'est pas nécessaire d'avoir beaucoup de neurones dans la deuxième couche cachée, mais qu'en revanche, dans la troisième couche cachée, plus de 8 neurones sont nécessaires.

En ce qui concerne les réseaux utilisant la fonction d'activation LogSoftmax, tous les réseaux fonctionnent de manière similaire, ce qui rend difficile la distinction (Figure 16). Cependant, en analysant les résultats des tests, nous constatons que, de manière similaire au réseau utilisant Softmax, celui qui fonctionne le mieux en train est celui avec le nombre maximum de neurones, tandis que celui qui fonctionne le moins bien est celui qui présente le plus grand écart entre les couches.

En résumé, nos analyses montrent que le choix du nombre de neurones par couche dans un réseau de neurones dépend de la fonction d'activation utilisée et du nombre de classes à classifier. Dans le cas spécifique de notre étude, nous avons constaté que l'utilisation du nombre maximal de neurones par couche conduit généralement à de meilleures performances, à condition que cet choix ne soit pas excessivement différent entre les couches. Cependant, il est important de noter que ces conclusions sont basées sur notre ensemble de données spécifique et notre configuration de réseau de neurones. D'autres ensembles de données et architectures de réseaux pourraient présenter des comportements différents, et il est nécessaire de les étudier pour obtenir des résultats généralisables.

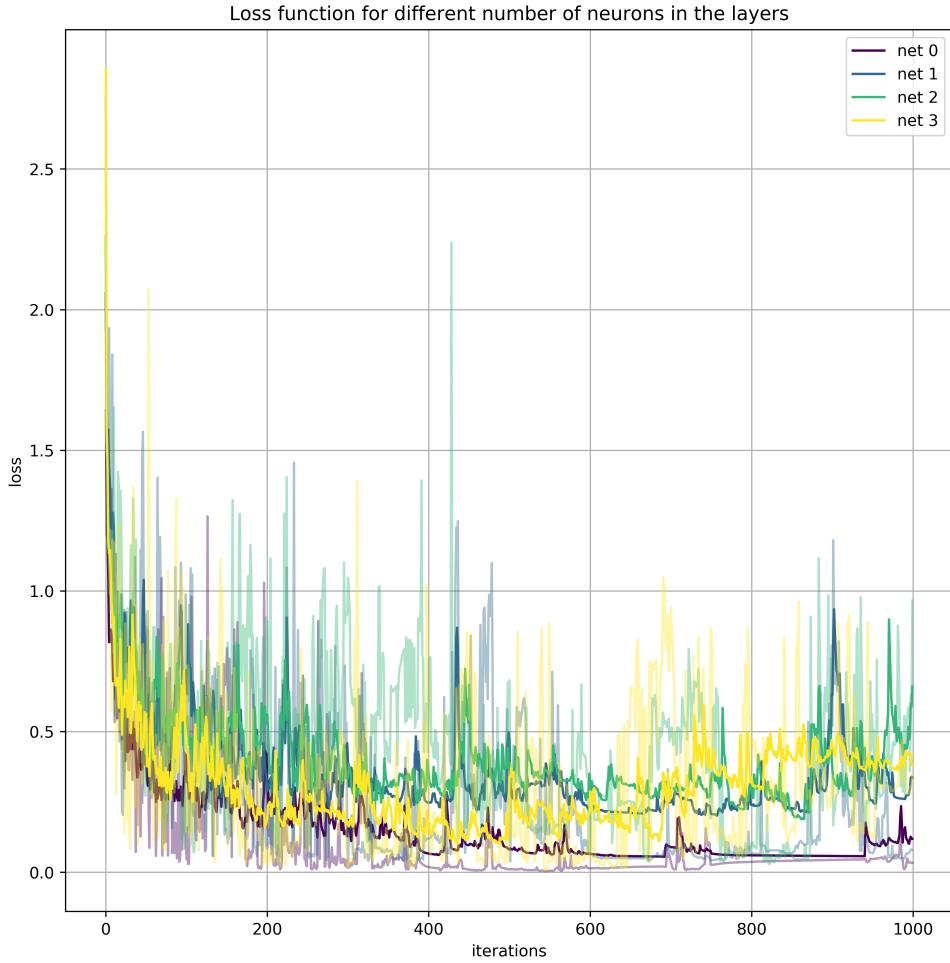


(a) Loss en train et test Softmax activation.

net	nb_neurons_input	nb_neurons_layer1	nb_neurons_layer2	nb_neurons_layer3	nb_neurons_output
0	0	64	32	16	8
1	1	64	32	16	14
2	2	64	32	30	8
3	3	64	32	30	14
4	4	64	52	16	8
5	5	64	52	16	14
6	6	64	52	30	8
7	7	64	52	30	14

(b) Table, combinaisons du nombre de neurones

FIGURE 15 – Loss en train et en test pour les fonctions d’activation SoftMax et pour une loss CrossEntropy. La couleur la plus foncée représente la loss en train tandis que la couleur plus claire représente la loss en test. Le nombre d’itérations totales est de 1000. Le réseau classifie 10 classes. On utilise un réseau à trois couches cachées et on fait varier le nombre de neurones pour ces trois couches. Le nombre de neurones par couche utilisé est décrit dans la Table (b). Nombre d’inputs : 64, nombre d’outputs : 10. La légende spécifie que (a) indique à quel réseau appartient la courbe de la loss, et l’architecture du réseau est ensuite décrite dans (b).



(a) Loss en train et test LogSoftmax activation.

net	nb_neurons_input	nb_neurons_layer1	nb_neurons_layer2	nb_neurons_output
0	0	64	32	16
1	1	64	32	30
2	2	64	52	16
3	3	64	52	30

(b) Table, combinaisons du nombre de neurones

FIGURE 16 – Loss en train et en test pour les fonctions d’activation LogSoftMax et pour une loss logCrossEntropy. La couleur la plus foncée représente la loss en train tandis que la couleur plus claire représente la loss en test. Le nombre d’itérations totales est de 1000. Le réseau classifie 10 classes. On utilise un réseau à deux couches cachées et on fait varier le nombre de neurones pour ces deux couches. Le nombre de neurones par couche utilisé est décrit dans la Table (b). Nombre d’inputs : 64, nombre d’outputs : 10. La légende spécifie que (a) indique à quel réseau appartient la courbe de la loss, et l’architecture du réseau est ensuite décrite dans (b).

4.3 Analyse du learning rate avec fonction d’activation Softmax

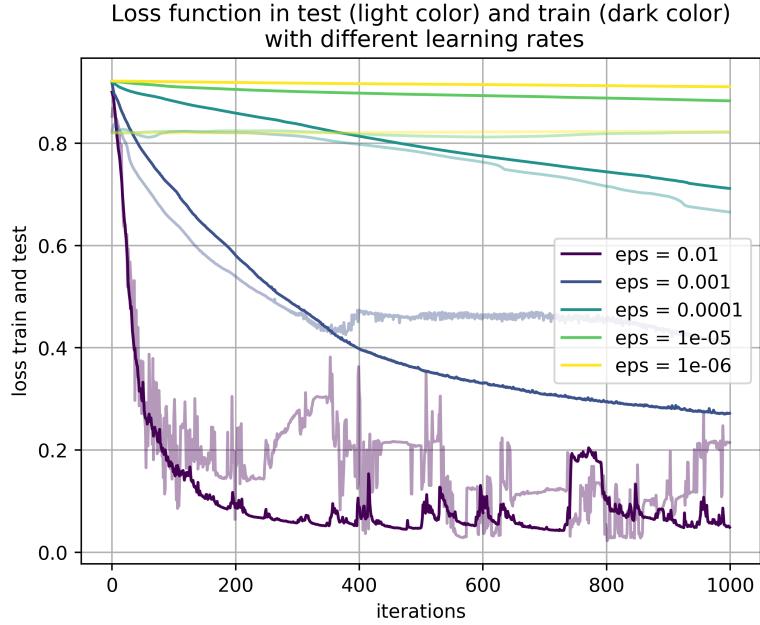
En utilisant les deux parties précédentes, nous allons maintenant analyser l’influence du taux d’apprentissage (learning rate) sur la perte (loss) des réseaux utilisant Softmax et LogSoftmax. Dans cette

partie, nous allons examiner la perte en fonction du taux d'apprentissage pour le réseau utilisant Softmax.

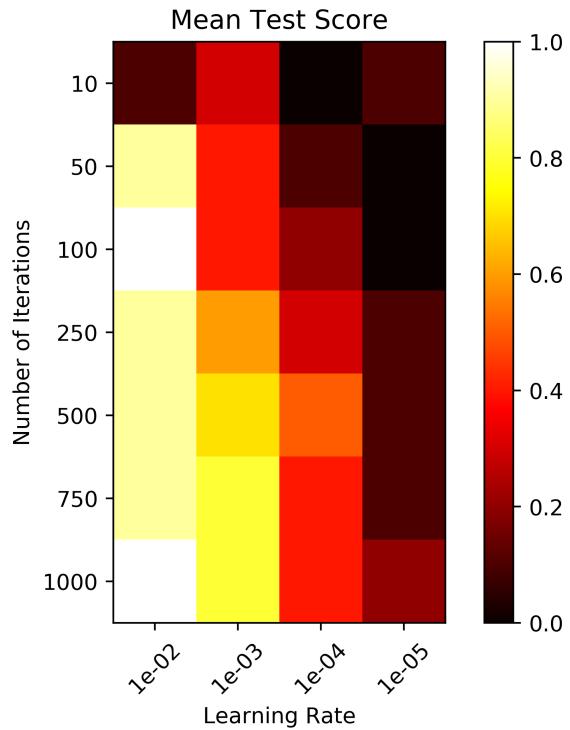
Pour ce faire, nous avons utilisé le réseau avec le nombre optimal de neurones par couche, tel qu'identifié précédemment. Dans la première figure, nous avons tracé la perte en test et en entraînement pour différents taux d'apprentissage : $\{1e - 2, 1e - 3, 1e - 4, 1e - 5, 1e - 6\}$, sur 1000 itérations. À chaque itération, le modèle calcule la perte pour chaque lot de données (batch) et fait la moyenne des pertes de tous les lots disponibles.

Nous observons dans cette figure que le meilleur taux d'apprentissage sur 1000 itérations est de 1e-2. Cela signifie que ce taux d'apprentissage conduit à une convergence plus rapide et à une perte minimale pour notre réseau utilisant Softmax.

Pour obtenir une analyse plus précise, nous avons également tracé le score en test en faisant la moyenne des performances du réseau, pour chaque taux d'apprentissage, pour 100 essais, chacun de 1000 itérations. Ces résultats indiquent donc que, pour notre réseau utilisant Softmax, un taux d'apprentissage de 1e-2 semble être le plus approprié en termes de convergence et de minimisation de la perte.



(a) Loss en train et test Softmax activation.



(b) Mean score en test et test Softmax activation.

FIGURE 17 – (a) Loss en train et en test pour les fonctions d’activation SoftMax et pour une loss CrossEntropy. La couleur la plus foncée représente la loss en train tandis que la couleur plus claire représente la loss en test. Le nombre d’itérations totales est de 1000. Le réseau classe 10 classes. On utilise un réseau à trois couches cachées avec le nombre optimal de neurones par couche. Chaque couleur différente correspond à un learning rate différent. (b) Matrice des scores moyens en test pour des learning rates différents. Chaque réseau a effectué 1000 itérations 100 fois (à chaque fois en repartant de 0), et chaque carré représente la moyenne du score en test sur ces 100 itérations.

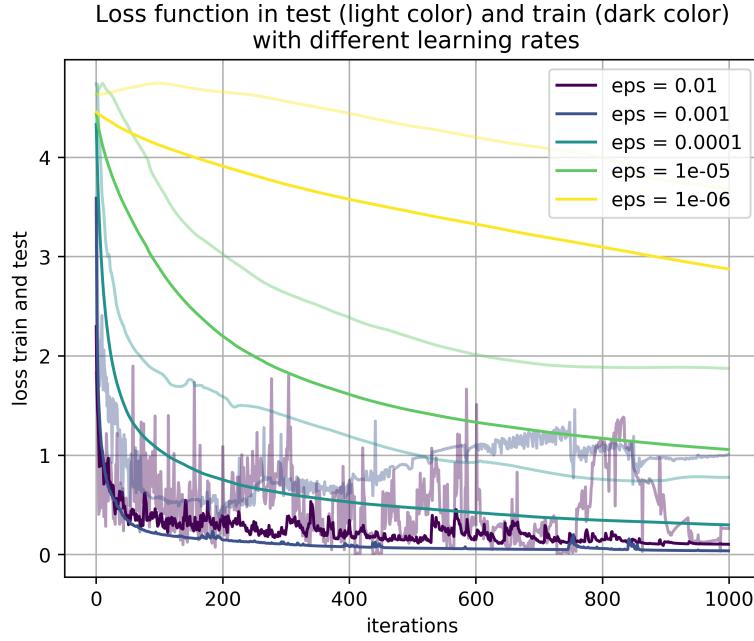
4.4 Analyse du learning rate avec fonction d'activation LogSoftmax

Dans cette partie, nous allons examiner la perte en fonction du taux d'apprentissage pour le réseau utilisant LogSoftmax.

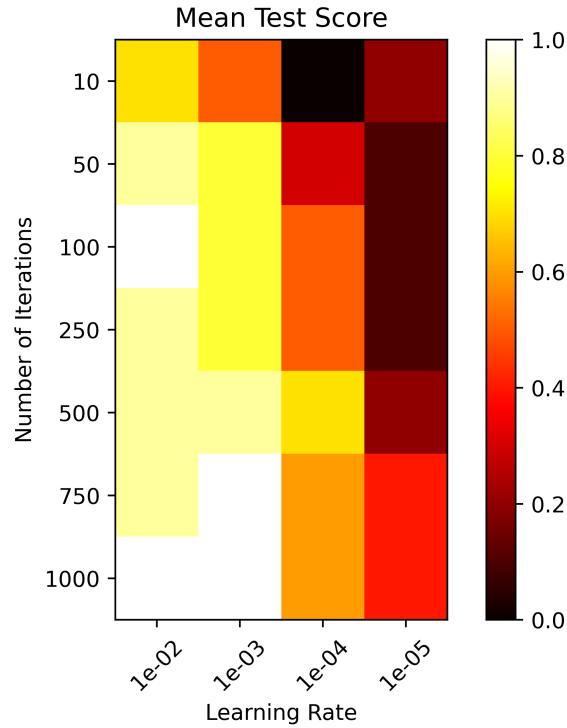
Comme avant, nous avons utilisé le réseau avec le nombre optimal de neurones par couche, tel qu'identifié précédemment. Dans la première figure, nous avons tracé la perte en test et en entraînement pour différents taux d'apprentissage : 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, sur 1000 itérations. À chaque itération, le modèle calcule la perte pour chaque lot de données (batch) et fait la moyenne des pertes de tous les lots disponibles.

En observant cette figure, nous pouvons constater que le meilleur taux d'apprentissage sur 1000 itérations pour le réseau utilisant LogSoftmax est le même que celui du réseau utilisant Softmax. Les résultats suggèrent que le taux d'apprentissage de 1e-2 est optimal pour notre réseau utilisant LogSoftmax, conduisant à une convergence plus rapide et à une perte minimale.

Pour une analyse plus précise, nous avons également tracé le score en test en moyennant les performances du réseau, pour chaque taux d'apprentissage, sur 100 essais chacun de 1000 itérations. Cette matrice de performances confirme (comme en Softmax) que un learning rate de 1e-2 est le plus approprié pour notre réseau utilisant LogSoftmax en termes de minimisation de la perte.



(a) Loss en train et test LogSoftmax activation.



(b) Mean score en test et test LogSoftmax activation.

FIGURE 18 – (a) Loss en train et en test pour les fonctions d’activation LogSoftMax et pour une loss LogCrossEntropy. La couleur la plus foncée représente la loss en train tandis que la couleur plus claire représente la loss en test. Le nombre d’itérations totales est de 1000. Le réseau classifie 10 classes. On utilise un réseau à deux couches cachées avec le nombre optimal de neurones par couche. Chaque couleur différente correspond à un learning rate différent. (b) Matrice des scores moyens en test pour des learning rates différents. Chaque réseau a effectué 1000 itérations 100 fois (à chaque fois en repartant de 0), et chaque carré représente la moyenne du score en test sur ces 100 itérations.

5 Conclusion

En conclusion, nous avons réalisé une analyse approfondie de l'impact des hyperparamètres et de l'architecture des réseaux neuronaux sur la performance des modèles. Nous avons étudié un réseau linéaire et un réseau non-linéaire, en analysant l'effet du learning rate, du nombre d'itérations et du nombre de neurones dans la couche cachée.

Pour le réseau linéaire, nous avons observé que des hyperparamètres optimaux conduisaient à une précision maximale du modèle. Nous avons également étudié la convergence de la perte et de l'exactitude pendant l'entraînement, ainsi que la frontière de décision du modèle. Ces analyses nous ont permis de mieux comprendre l'apprentissage du modèle linéaire.

Dans le cas du réseau non-linéaire, nous avons résolu le problème du XOR en utilisant un réseau de neurones à une couche cachée. Nous avons identifié les paramètres optimaux du réseau en utilisant une heatmap pour déterminer le meilleur couple (iterations, learning rate) pour un réseau de neurones à une couche cachée avec 30 neurones. Nous avons ensuite étudié l'effet du nombre de neurones dans la couche cachée sur les performances du modèle, pour le couple optimal d'hyperparamètres précédemment déterminé. Enfin, nous avons calculé une nouvelle fois la heatmap, dont les résultats présentés étaient nettement améliorés, tout en gardant le même learning rate optimal mais en réduisant le nombre d'itérations minimales nécessaires à l'obtention de très bons résultats. Obtenant ainsi d'excellentes performances très économies computationnellement.

Enfin, nous avons abordé les concepts d'encapsulation et de classification multi-classe, en explorant l'optimisation d'un réseau de neurones pour la classification de plusieurs classes. Nous avons déterminé le nombre optimal de couches nécessaires pour classifier n classes et comparé les performances des fonctions d'activation LogSoftmax et Softmax.

Dans l'ensemble, cette analyse approfondie nous a permis de mieux comprendre l'impact des différents hyperparamètres et architectures sur la performance des réseaux neuronaux. Ces connaissances peuvent être utilisées pour améliorer la conception et l'optimisation des modèles d'apprentissage automatique dans divers domaines d'application.

6 Pistes d'améliorations

Voici quelques pistes d'amélioration pour notre rapport et nos algorithmes que nous n'avons malheureusement pas eu le temps de traiter.

- Explorer différentes méthodes d'initialisation : Au lieu d'une initialisation aléatoire selon la loi normale des paramètres, il peut être intéressant d'explorer d'autres méthodes d'initialisation telles que l'initialisation de Xavier. Cette méthode vise à maintenir une variance constante des activations et des gradients à travers les couches du réseau, ce qui peut être bénéfique pour les réseaux de grande taille ou profonds.
- Utiliser un taux d'apprentissage décroissant : Dans de nombreux cas, il est avantageux de réduire progressivement le taux d'apprentissage au fur et à mesure de l'entraînement. Cela permet de raffiner les poids du modèle et de se rapprocher du minimum global de la fonction de perte. Des schémas d'ajustement du taux d'apprentissage, tels que la décroissance linéaire ou exponentielle, peuvent

être utilisés pour atteindre cet objectif. Des algorithmes d'optimisation adaptatifs comme Adam ajustent automatiquement le taux d'apprentissage en fonction de l'historique des gradients.

7 Code

Tout notre code est disponible à https://github.com/AEmanuelli/ML_Final_project/tree/main