BUT Sciences des Données

SAE 4-02 Reporting d'une analyse multivariée : Projet tutoré 2024

Exploration des réseaux de neurones

Modèle de reconnaissance de chiffre

Auteurs :
Mme Jehanne BALEYDIER
M. Mathieu GILBERT
M. Kenny BANGAKÉ
M. Cédric JESTIN

Encadrants :
Mme Maeva PARADIS
M. Clement JACQ

Version du 27 mars 2024

Résumé

Résumé : À travers l'implémentation d'un modèle permettant de détecter le chiffre présent sur une image, nous avons cherché à comprendre le fonctionnement d'un réseau de neurones. Dans un premier temps, nous en avons codé un par nous-mêmes sur python, puis en utilisant les librairies TensorFlow et Keras pour développer un modèle convolutif. Nous expliquons les aspects théoriques et mathématiques d'un réseau de neurones en nous appuyant sur notre exemple classique. Nous présentons également le modèle convolutif. Nous présentons nos divers tests, ainsi que quelques visualisations de l'apprentissage des réseaux de neurones artificiels que nous avons utilisés.

Table des matières

Introduction

Depuis peu, l'intelligence artificielle générative s'est ouverte au grand public, pour beaucoup, c'était une révolution, un nouveau pas vers une aire technologique encore plus développée, mais qu'y a-t-il vraiment derrière le concept d'IA ?

Notre objectif était de comprendre comment est construit un réseau de neurones, le principe à la base de l'intelligence artificielle, en codant un modèle de détection de chiffres sur un ensemble d'images provenant de la base de données MNIST (Mixed National Institute of Standards and Technology).

Avant d'expliquer le fonctionnement d'un réseau de neurones, un petit point historique s'impose pour comprendre d'où il provient et de quand il date.

On pourrait prétendre que l'intelligence artificielle est jeune et récente, mais c'est en réalité une fausse conception que l'on se fait. L'intelligence artificielle en termes de science a officiellement vu le jour en 1956 lors d'une université d'été organisée au Dartmouth College aux États-Unis. Cette nouvelle discipline suppose que toutes les fonctions cognitives peuvent être reproduites par un ordinateur.

Mais cela fait plus longtemps que les travaux sur ce sujet ont débuté. Alan Turing a décrit mathématiquement des réseaux de neurones qui s'auto-organisent, en 1948, dans l'article Intelligent Machinery. L'hypothèse ayant lancé ces recherches vient du philosophe Thomas Hobbes (1588-1679) qui dit que toute pensée résulte d'un calcul. Turing et Church ont, par la suite, avancé l'idée que la pensée pouvait être simplifiée par des machines en supposant que tout calcul puisse être fait par une machine.

En 1957, le premier réseau de neurones à couche est proposé par Franck Rosenblatt. En 1990, Yan Le Cun (France) développe la notion de réseau convolutif pour traiter des images qui permettra par la suite de faire des progrès significatifs dans l'IA et la reconnaissance d'images. Après des évolutions plus ou moins rapides, c'est en 2012 qu'Alex Krizhevsky, Ilya Sutskever et Geoffrey Hinton publient leurs résultats sur la classification d'images à l'aide d'un réseau de neurones convolutif grâce à la mise en place d'une nouvelle architecture du réseau de neurones convolutif : AlexNet.

Dans un premier temps, nous expliquerons ce qu'est un réseau de neurones, comment il est construit. Nous expliquerons ensuite la mise en place de l'apprentissage dans le réseau que nous avons créé. Enfin, nous développerons la mise en place d'un modèle utilisant un modèle utilisant le principe de convolution.

1 Réseau de neurone, application au cas de la détection d'image

L'intelligence artificielle est un sujet vaste, dans ce domaine, on retrouve un sous-ensemble : le machine learning, une technologie visant à tirer des enseignements des données et à s'améliorer avec l'expérience. Le Deep Learning encore un sous-ensemble du machine learning, qui consiste à imiter la façon dont les humains assimilent de nouvelles informations. Les réseaux de neurones font donc partie du deep learning.

1.1 Fondements

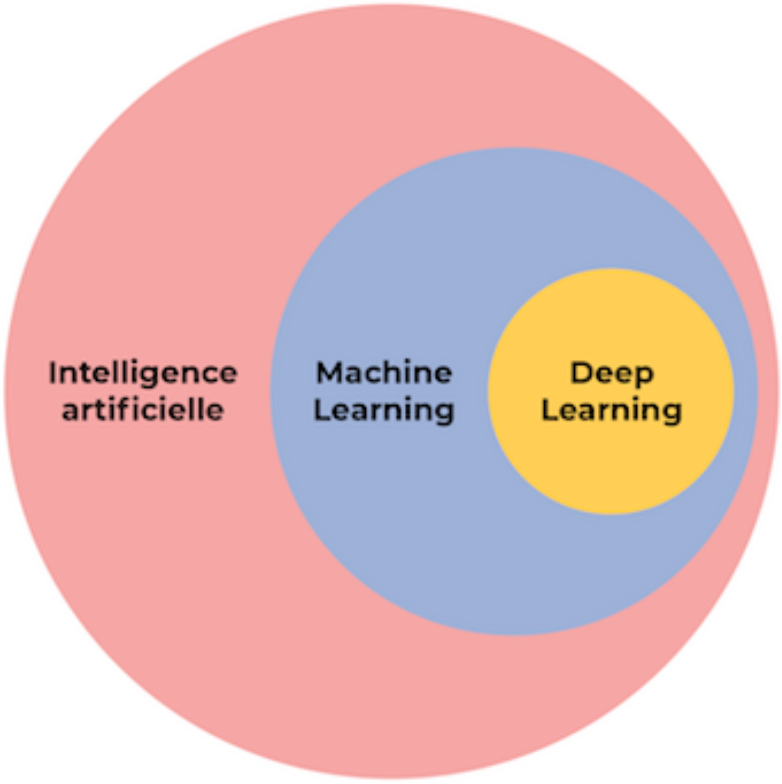

FIGURE 1 – Domaine de l'IA

Nous allons maintenant présenter un réseau de neurones simple.

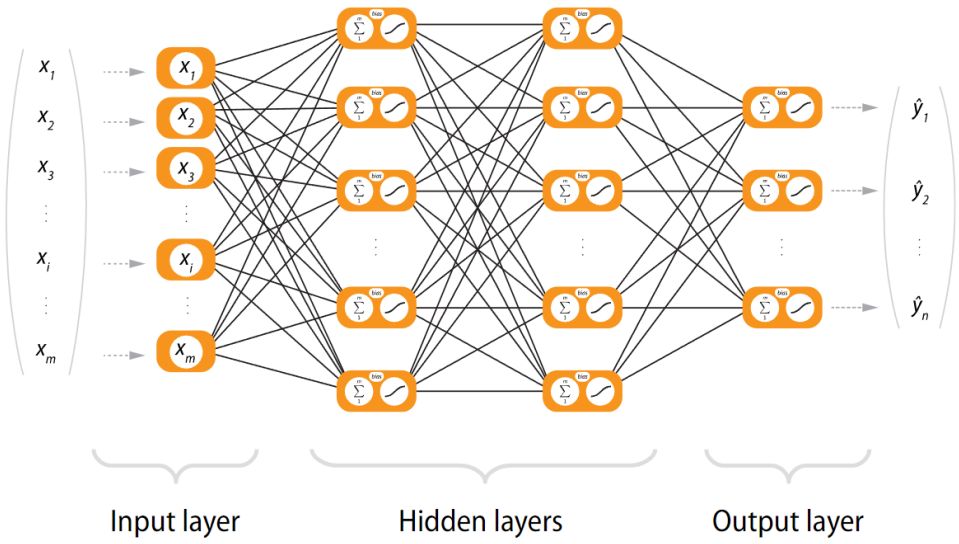

FIGURE 2 – Structure d'un réseau de neurone

Un réseau de neurone est comme son nom l'indique un ensemble de neurones. Chaque neurone est inspiré du fonctionnement d'un neurone humain. Il se résume en une fonction mathématique, une sorte de somme pondérée. Un réseau de neurones classique comprend : une couche d'entrée, des couches dites cachées et une couche de sortie.

La couche d'entrée est simplement une couche permettant de capter un ensemble d'informations brutes, qui sont ensuite transmises aux couches cachées. Dans notre cas, le réseau est dit complétement connecté puisque l'ensemble des neurones sont connectés entre eux. Les couches cachées sont composées de neurones disposant chacun d'un ensemble de poids w_i selon le nombre de signaux d'entrées x_i ainsi que d'un biais b. Chaque neurone dispose également d'une fonction d'activation (ex : sigmoïde).

Un neurone est simplement une fonction mathématique qui réalise un ensemble d'opérations sur les données comme résumé dans la page suivante.

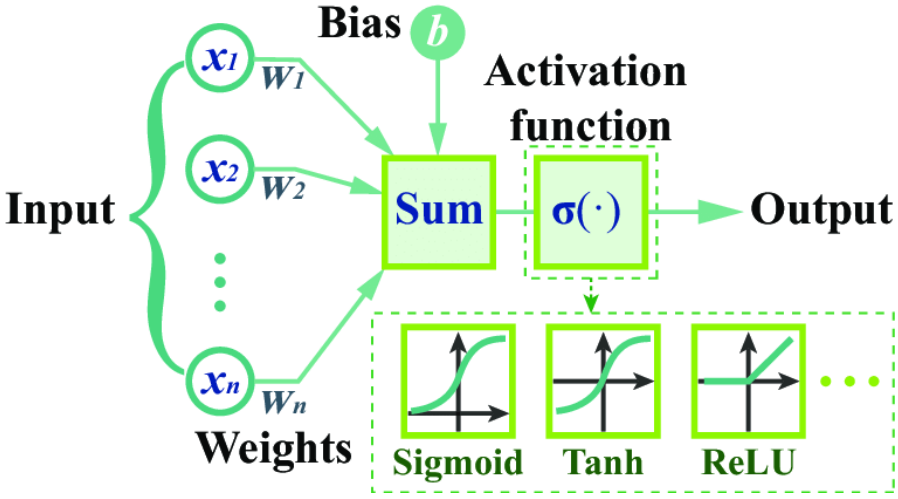

FIGURE 3 – Description neurone artificiel

La couche de sortie comprend également des poids et des biais pour chaque neurone ainsi qu'une fonction d'activation (pouvant être différente de celle des couches cachées). Le résultat de cette couche peut être un vecteur de probabilité indiquant la prédiction du modèle. Chaque réseau possède également une fonction dite de coût ou de perte qui peut être une fonction de perte quadratique, logistique ou autre.

Pour qu'un réseau soit fonctionnel dans un contexte donné, autrement dit pour qu'il puisse dans notre cas réaliser une prédiction sur le chiffre compris dans une image, il doit tout d'abord être entraîné. L'entraînement consiste à passer plusieurs fois dans le réseau de neurones pour modifier les poids et biais en fonction des erreurs.

Il existe différents types d'apprentissage, l'apprentissage supervisé, non supervisé et par renforcement. Dans notre cas, nous avons utilisé l'apprentissage supervisé. C'est-à-dire que le programme va apprendre à partir de données labellisées par l'être humain. À chaque nouvelle donnée, nous lui donnons la réponse attendue pour qu'il se corrige tout seul. Dans l'apprentissage non supervisé, la machine est capable de faire des regroupements, mais elle n'est pas capable de définir les différents libellés. Enfin, l'apprentissage par renforcement fonctionne avec un système de récompenses positives ou négatives.

1.2 Principes théoriques liées à la détection de chiffres

Pour expliquer la mise en place d'un réseau dans le contexte de la détection de chiffre, nous allons considérer un réseau de neurones composé de 100 entrées,

2 couches cachées et une couche de sortie composée de 10 neurones.

1.2.1 Prétraitements des images

Pour ce projet, nous avions à notre disposition des données du MNIST (Mixed National Institute of Standards and Technology). Ces données sont 1000 images de taille 10×10 (pixels), dont on connait le chiffre correspondant (100 images par chiffre).

Pour rendre ces images "lisibles" par notre réseau de neurone, nous avons parcouru chaque pixel pour le traduire en un nombre qui est normalisé (division par 255), afin d'avoir une valeur entre 0 et 1. Pour chaque image, on obtient au final un vecteur de 100 valeurs numériques ainsi qu'une étiquette associée (sous une forme de vecteur one-hot composé de 0 et de 1). Ces images sont ensuite divisées en deux groupes : un groupe d'entraînement (80 %) et un groupe de test (20 %).

1.2.2 Initialisation

La première étape est l'initialisation du réseau, c'est-à-dire définir les poids, les biais et les fonctions utilisées dans notre réseau. Au début, les vecteurs poids (W) et biais (B) sont définis de manière pseudo aléatoire (expliqué dans la partie **??**). Pour chaque couche, on définit une matrice de poids, le nombre de lignes correspond au nombre de neurones de la couche et le nombre de colonnes correspond au nombre d'entrées. Le biais est un vecteur composé d'une colonne et d'un nombre de lignes correspondant au nombre de neurones dans la couche.

Dans notre cas, en ce qui concerne les fonctions d'activation, pour essayer d'obtenir les meilleurs résultats possibles, nous avons utilisé la fonction ReLU (Rectified Linear Unit : unité linéaire rectifiée) pour les couches cachées, définie par :

$$\mathrm{ReLU}(x) = \max(0, x)$$

Et la fonction sigmoïde pour la couche de sortie, définie par :

$$f(x) = \frac{1}{1 + e^{-x}}$$

1.2.3 Propagation

Dans un premier temps, les images doivent passer dans le réseau de neurones. À partir des pixels normalisés en entrées (X), on calcule pour une couche :

$$Z = W.X + B$$

Ensuite, on applique la fonction d'activation à ce résultat, ce qui nous donne la sortie de la couche. On effectue cette étape à chaque couche en prenant en entrée la valeur de la sortie de la couche précédente.

1.2.4 Rétropropagation

Le principe de la rétropropagation est simple, l'objectif est de remonter le réseau de neurones en partant de la fin en comparant la sortie avec celle attendue et modifier les poids et biais en conséquence, et répéter le processus.

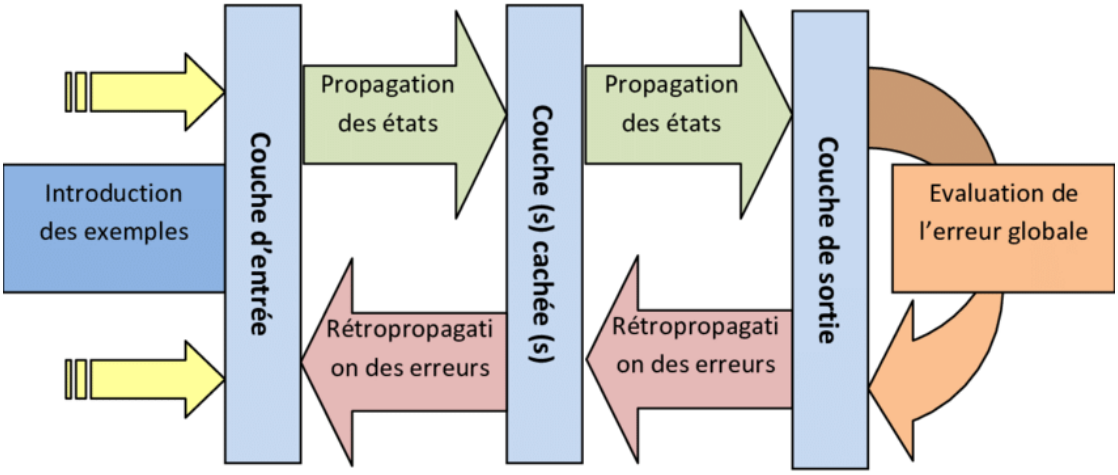

FIGURE 4 – L'Apprentissage du modèle.

Nous avons utilisé une fonction de coût quadratique, le MSE (Mean Squared Error) dont la formule est :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

y_i : sortie attendue (chiffre attendu)
$\hat{y}_i$: sortie du réseau (de la couche de sortie)

L'objectif est de minimiser la fonction de coût, pour cela, on utilise l'algorithme de la descente de gradient. Cet algorithme se base sur un taux d'apprentissage, le learning rate (α) qui doit être ni trop grand, au risque de ne pas trouver le minimum global, ni trop petit, sinon l'apprentissage va être trop long. Une image utilisée fréquemment pour illustrer cette descente de gradient est d'imaginer être en haut d'une montagne sans rien voir et vouloir attendre le fond de la vallée. Pour réussir, il faut progresser petit à petit jusqu'à attendre le bas de la vallée.

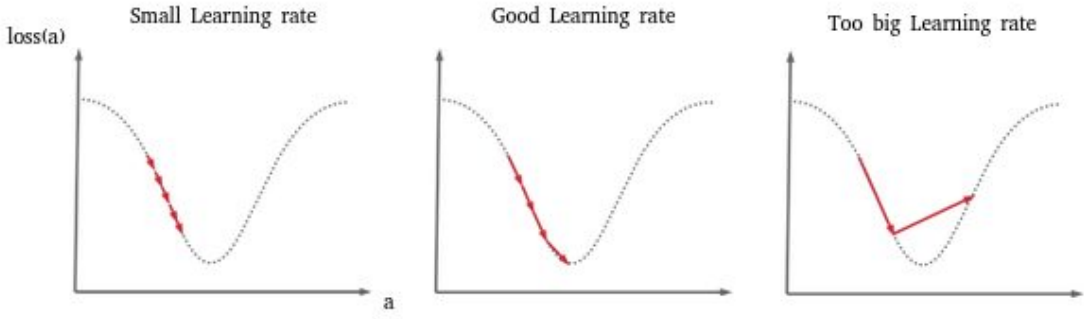

FIGURE 5 – La descente de gradient.

Pour effectuer la modification des poids et des biais, on utilise la dérivée partielle de la fonction de perte ou Loss ($\mathcal{L}$) en fonction du poids ou du biais :

$$W^{(k)} = W^{(k)} - \alpha \frac{\partial \mathcal{L}}{\partial W^{(k)}}$$

$$B^{(k)} = B^{(k)} - \alpha \frac{\partial \mathcal{L}}{\partial b^{(k)}}$$

On répète ce processus pendant un nombre de répétitions défini (epochs) ou jusqu'à obtenir un taux d'erreur suffisamment faible.

1.3 Résultats de l'apprentissage

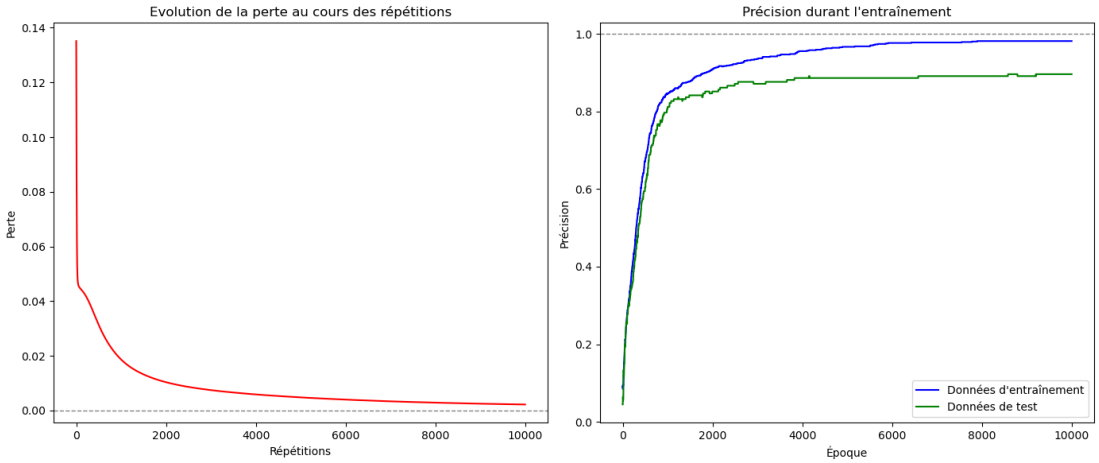

FIGURE 6 – Apprentissage du réseau (pertes et prédictions)

Comme on peut le voir sur le graphique de gauche, nous avons réussi à obtenir de bons résultats sans utiliser de librairie. Nos pertes arrivent très vite autour de 0 et ne font que converger de plus en plus au fil du temps. Nos précisions sur les chiffres se trouvent dans un intervalle de 85 à 90 %. Toutefois, dans le graphique de gauche, nous remarquons un phénomène. Plus on va dans les époques, plus les données de tests s'éloignent des données d'entraînement. Ce phénomène s'appelle le surapprentissage, c'est le fait de "trop" entraîner notre modèle, ainsi, celui-ci serait très performant et très précis pour les données d'entraînement, mais il ne s'adapterait pas à d'autres données de tests.

1.4 Stratégies d'optimisation

Pour l'initialisation des poids, nous avons utilisé la méthode He qui consiste à calculer des valeurs aléatoires qui suivent une distribution gaussienne de probabilité avec valeur moyenne nulle et écart-type égale à 2/n, où n est le nombre d'entrées de la couche. Ainsi les poids suivent une loi normale :

$$W \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n}}\right)$$

Nous avons utilisé cette initialisation parce qu'elle correspond à la plus optimale avec l'utilisation de la fonction ReLU, en effet, elle permet d'empêcher

les poids de devenir trop petits et donc inactifs. En maintenant une variance appropriée, l'initialisation He facilite la convergence de l'apprentissage dans les réseaux de neurones profonds.

Pour optimiser la précision de notre réseau de neurones et la complexité de notre algorithme, nous avons tenté de modifier plusieurs paramètres afin de les optimiser. Nous avons fait varier le taux d'apprentissage pour avoir le plus grand taux de précision possible. Faire varier le taux d'apprentissage est une pratique courante dans l'entraînement des réseaux de neurones. Le taux d'apprentissage contrôle la taille des pas que le modèle prend lors de la mise à jour des poids pendant l'entraînement. Il influence donc la rapidité avec laquelle le modèle converge vers une solution optimale.

L'objectif de cette variation du taux d'apprentissage est d'optimiser la vitesse de convergence du modèle tout en évitant les problèmes de divergence ou de convergence vers un minimum local peu optimal. Un taux d'apprentissage trop élevé peut entraîner une convergence rapide, mais il peut également provoquer des oscillations autour du minimum global ou conduire à une divergence du processus d'optimisation. En revanche, un taux d'apprentissage trop faible peut ralentir la convergence du modèle et le piéger dans des minima locaux.

En modifiant dynamiquement le taux d'apprentissage pendant l'entraînement, nous pouvons adapter la vitesse d'apprentissage du modèle en fonction de la progression de l'entraînement. Nous avons tenté plusieurs méthodes pour optimiser la variation du taux d'apprentissage. La première a été de le baisser linéairement à chaque époque suivant cette formule :

$$\text{learning rate} = 0.01 \times (0.99^{\text{époque}})$$

Notre taux d'apprentissage initial était de 0.01 et on le baisse suivant un taux de décroissance du taux d'apprentissage, qui est dans ce cas précis équivalent à 0.99. Le taux de décroissement évalue à quel point le taux d'apprentissage va baisser au fil des époques. Plus on va dans les époques, plus le taux de décroissance augmente et moins le taux d'apprentissage est élevé.

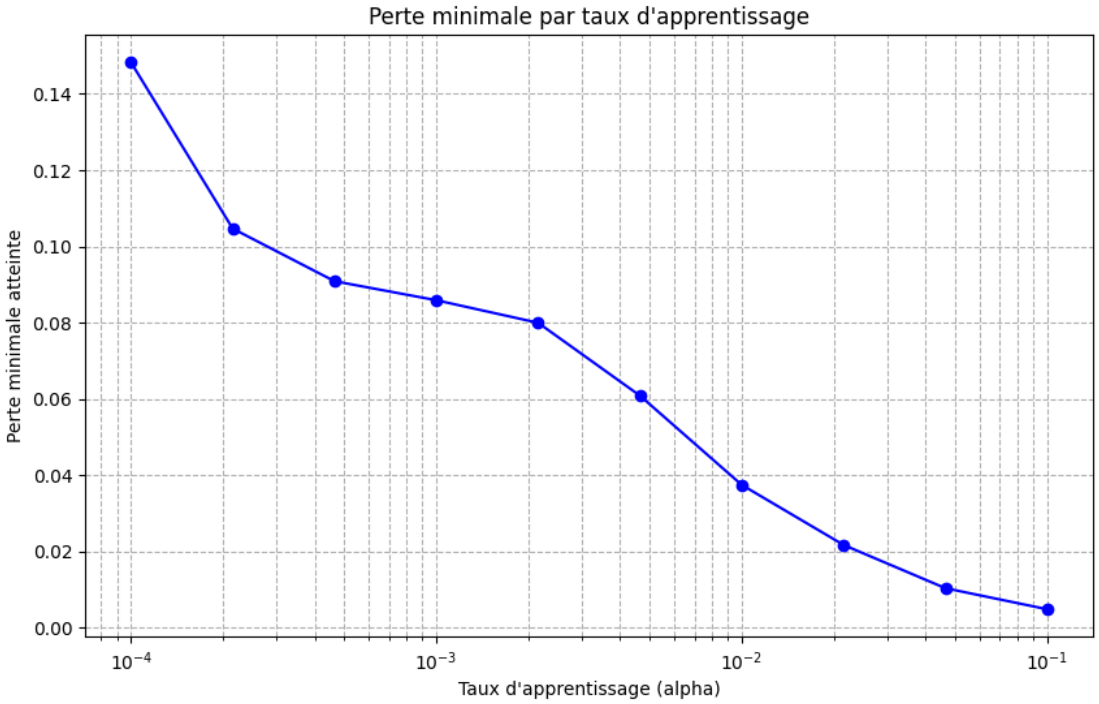

FIGURE 7 – Évolution du taux d'apprentissage

Nous pouvons voir par ce graphique que les minimums de perte obtenus sont plus faibles avec un taux d'apprentissage proche de 0,1. Notre meilleur taux d'apprentissage serait donc de 0,1.

Nous avons essayé par la suite différentes méthodes pour choisir et varier notre taux de décroissement. Voici un résumé de tous nos tests et de tous nos résultats selon différents paramètres :

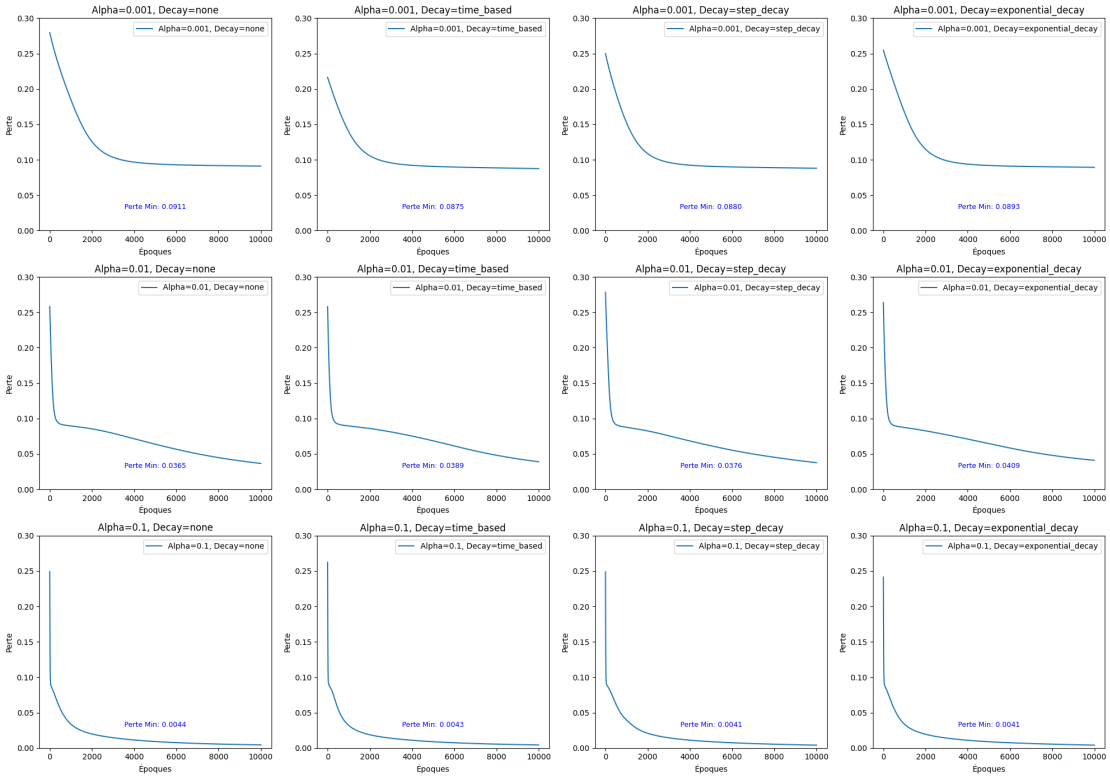

FIGURE 8 – Évolution du taux d'apprentissage

Au regard des graphiques, il n'y a pas l'air d'y avoir beaucoup d'impact des diverses méthodes d'ajustement du taux d'apprentissage. Cela pourrait être lié à la simplicité du modèle utilisé, avec peu de couches et de neurones.

2 Réseaux convolutifs (CNN)

2.1 La théorie des réseaux convolutifs

Les réseaux de convolution ou réseaux convolutifs (en anglais CNN : convolutive neural network), ont été développés notamment par Yann Le Cun dans les années 90 afin de détecter des écritures sur les chèques. Dans un réseau convolutif la différence avec le modèle classique est l'ajout de couches de neurones dits convolutifs après la couche d'entrée et avant les couches de neurones cachés. Ces couches convolutives comprennent pour chaque neurone : un poids et un biais, mais on ajoute à cela des traitements de convolution et de pooling. La convolution est une transformation linéaire que nous allons expliquer à l'aide de

la figure suivante :

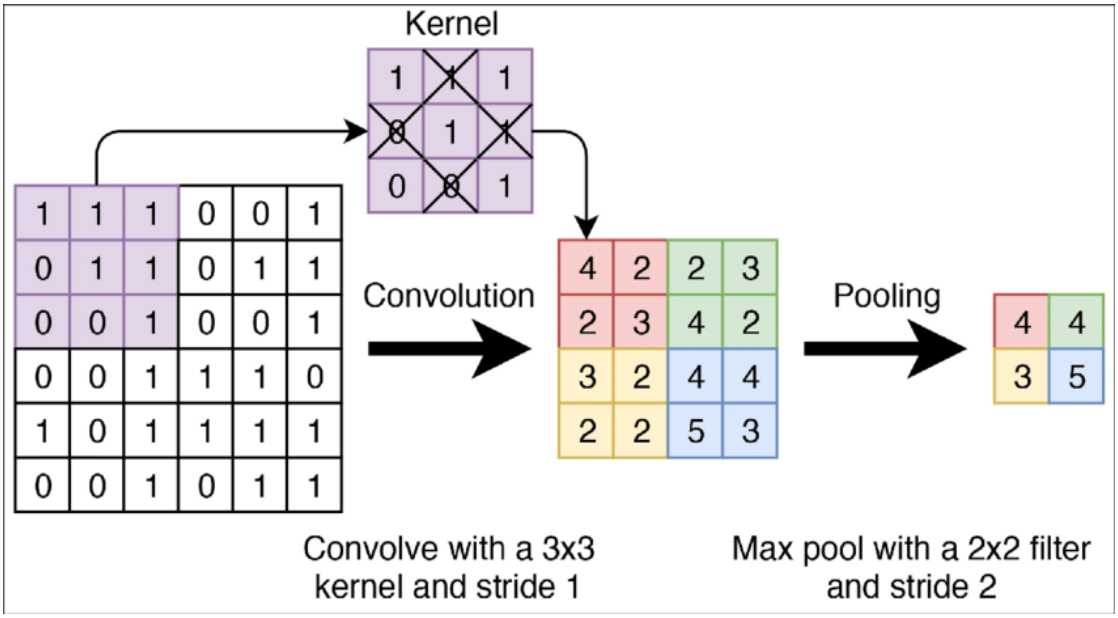

FIGURE 9 – Convolution et pooling

La convolution consiste à multiplier des sous-parties de la matrice de données par un kernel puis à sommer les résultats et répéter cela sur l'ensemble afin d'obtenir une matrice de résultat de convolution comprenant des informations modifiées. Cette opération permet selon les kernels ou filtres de convolutions utilisés de faire ressortir divers aspects de l'image originale. Les couches de convolutions comprennent également des poids (présent dans les kernel) et des biais ainsi qu'une fonction d'activation. Ensuite, on applique une mise en commun ou pooling dans laquelle l'image est synthétisée, résumée dans un format plus petit en prenant le maximum dans chaque zone de l'image selon une certaine taille de filtre de pooling (exemple : 2*2). Cette étape de mise en commun permet de faire ressortir les caractéristiques principales de l'image et également de diminuer par la suite le nombre d'entrées donc de paramètres à modifier au sein du réseau.

Cela permet de synthétiser l'image initiale afin de diminuer par la suite la complexité du traitement neuronal. Ces opérations sont répétées sur plusieurs phases de convolution puis de pooling et on obtient ainsi un ensemble d'images de faible dimensionnalité, mais en nombre plus élevé. Par la suite, ces données sont envoyées dans les couches cachées du réseau puis la couche de sortie

donne un résultat pour chaque image traitée via l'ensemble du réseau convolutif.

L'apprentissage des réseaux de neurones convolutifs est basé sur les mêmes principes que nous avons précisés précédemment, à savoir la minimisation d'une fonction de coût via l'algorithme de descente de gradient.

2.2 Description de notre réseau convolutif créé avec Tensorflow

Afin de simplifier la création d'un réseau convolutif (CNN : convolutive neural network), nous avons utilisé Tensorflow, l'une des bibliothèques python les plus populaires concernant le deep learning.

Dans notre cas, le réseau comprend trois couches de convolution de compositions différentes utilisant toutes comme fonction d'activation ReLU, puis trois couches cachées avec toujours la fonction d'activation ReLU puis une couche de sortie possédant quant à elle la fonction softmax.

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Cette fonction softmax est particulièrement adaptée dans notre cas puisque qu'elle renvoie un vecteur de probabilité. Concrètement pour chaque image entrant dans le réseau, nous avons en sortie un vecteur de probabilité (de somme 1), dont chaque élément est la probabilité que selon le réseau le chiffre soit 0, 1, ..., 9.

La fonction de coût utilisé ici est une fonction d'entropie croisée catégorique de formule :

$$L(y, \hat{y}) = -\log\left(\frac{e^{\hat{y}[y]}}{\sum_j e^{\hat{y}[j]}}\right)$$

La méthode d'apprentissage utilisée ici est particulière et sera précisée par la suite. Nous avons également utilisé diverses méthodes d'optimisations que nous évoquerons dans la prochaine partie.

2.3 Diverses Optimisations du réseau convolutif

2.3.1 Apprentissage

L'apprentissage pour notre réseau convolutif est bien différent de l'apprentissage simple par lot (complet), ici, nous avons réalisé un apprentissage par mini

lots (mini batch) sur des données qui ont été augmentées de manière aléatoire à partir d'images de référence. Ainsi, à chaque période d'entrainement, les données sont "augmentées" par des images aléatoires proches (ayant une étiquette similaire). Cela permet de limiter les calculs nécessaires à l'apprentissage de par la sélection de sous-lots parmi les données totales et également d'éviter le surapprentissage en ajoutant de l'aléa.

2.3.2 Description du réseau

Afin d'obtenir un taux de précision proche de 99%, nous avons construit un réseau relativement dense avec deux couches de convolution (elles sont elle-même composées de dizaines de couches de convolution/pooling), puis de trois couches cachées ayant respectivement : 100, 80, 40 neurones. L'ensemble des couches comporte une fonction d'activation ReLU qui est adaptée dans ce contexte pour amener une forme de non-linéarité dans le modèle. La couche de sortie comporte une fonction d'activation softmax, qui est indiquée dans notre cas, car elle renvoie un vecteur de probabilité (dont la somme vaut 1) qui indique ainsi la probabilité que l'image soit un chiffre entre 0 et 9.

2.3.3 Optimisation d'Adam

Cette optimisation proposée en 2015 par des chercheurs [[1]] dont le nom Adam provient de adaptive moment estimation. C'est une méthode d'amélioration stochastique basée sur une estimation de premier et de second moment à partir d'une moyenne mobile exponentielle. La méthode Adam combine deux méthodes précédemment proposées : Adagrad et RMSProp. C'est à ce jour l'une des méthodes les plus poussées d'optimisation de l'apprentissage des réseaux de neurones. Bien qu'elle ne donne pas toujours la meilleure convergence ou précision finale, elle permet très souvent un apprentissage plus rapide dans divers contextes. L'objectif de cette méthode est, pour faire simple, de contrôler, d'ajuster les variations des poids et biais du modèle d'une manière à favoriser une convergence plus rapide du modèle vers un optimal. Nous ne rentrerons pas ici dans les détails techniques et mathématiques, permettant de comprendre en profondeur la méthode Adam.
Afin d'améliorer la descente de gradient, ce procédé ajoute un ensemble d'hyperparamètres qui, ensemble, permettent d'ajuster le taux d'apprentissage d'une manière à favoriser un apprentissage rapide à l'aide d'estimations mathématiques. C'est une manière d'obtenir une convergence plus directe à partir de calcul supplémentaire autour du gradient.

2.3.4 Ajout d'un taux d'abandon

Lors de notre définition du modèle convolutif à l'aide de Tensorflow, nous avons ajouté dans les couches cachées un paramètre appelé drop out rate (fixé à 20%). Ce paramètre indique que de manière aléatoire, 20% des neurones sont désactivés, ce qui ajoute de l'aléa et limite le surapprentissage. Cette méthode permet au réseau de prédire des images plus complexes avec une meilleure précision et ainsi augmenter la qualité du modèle.

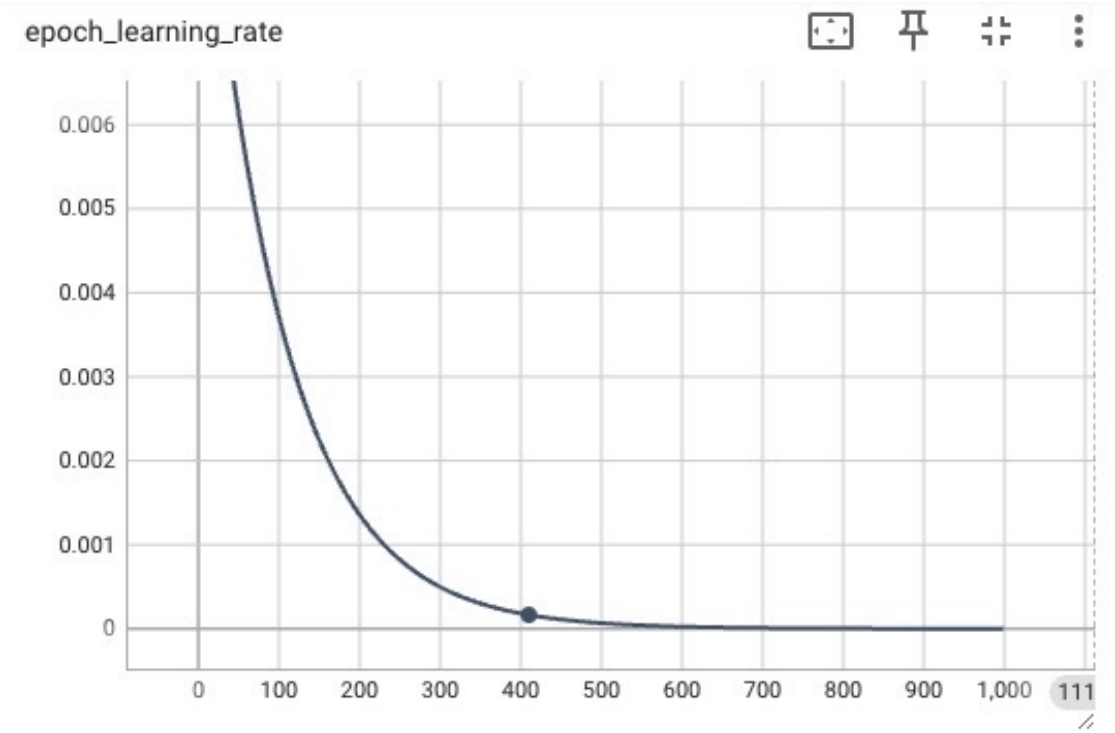

FIGURE 10 – Évolution du taux d'apprentissage

Ici, nous avons représenté l'évolution du taux d'apprentissage au fur et à mesure de l'apprentissage de notre réseau convolutif, on peut voir qu'il suit une forme de décroissance exponentielle.

2.4 Résultats du modèle convolutif

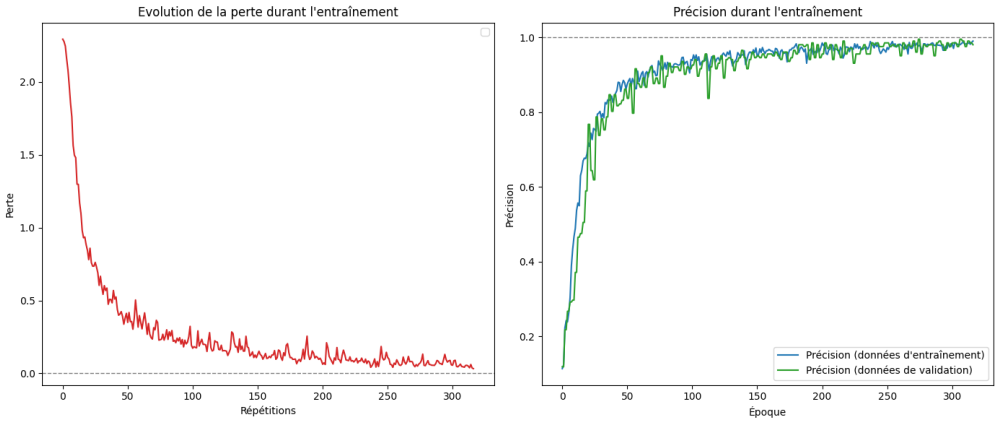

FIGURE 11 – Apprentissage du réseau convolutif

Pour évaluer la qualité de l'apprentissage du modèle, nous avons représenté l'évolution de la fonction de perte ainsi que la précision des prédictions du modèle. Les diverses méthodes d'optimisation employées semblent fonctionner et ont contribué à améliorer considérablement la performance du modèle. Parmis ces méthodes, l'utilisation de techniques de régularisation comme le dropout et l'utilisation de mini-lots a permis de réduire le surapprentissage, garantissant ainsi que le modèle reste généralisable à de nouvelles données inédites. L'adoption d'une stratégie d'apprentissage adaptatif, en ajustant le taux d'apprentissage au cours de l'entraînement, a également joué un rôle clé dans la stabilisation de la convergence du modèle et dans l'atteinte d'un plateau de performance élevé après environ 300 périodes d'apprentissage.

En outre, l'emploi d'un algorithme d'optimisation avancé, tel que Adam a facilité une descente de gradient plus efficace. Pour ce type de tâches de classification sur des images en noir et blanc simples, une fonction de perte comme l'entropie croisée a été particulièrement adaptée. Elle mesure efficacement la différence entre les distributions de probabilité prédites par le modèle et les distributions réelles des étiquettes de classe. Ces éléments, combinés à une architecture de réseau de neurones adéquatement choisie, ont permis d'atteindre une précision de prédiction élevée (99 %), tout en maintenant la capacité du modèle à généraliser à de nouvelles données non vues lors de l'entraînement. Ce succès démontre l'efficacité des stratégies d'optimisation et des choix de conception employés dans le développement de ce modèle d'apprentissage automatique.

Conclusion

Dans ce projet, nous avons étudié l'application des réseaux de neurones, à la fois simples et convolutifs, pour la reconnaissance d'images. Nous avons réussi à améliorer la performance du modèle en intégrant diverses méthodes d'optimisation, telles que le dropout, l'ajustement du taux d'apprentissage, et l'utilisation de l'algorithme d'optimisation Adam. Ces approches ont contribué à réduire le phénomène de surapprentissage, renforçant ainsi la capacité du modèle à généraliser à des données inédites.

L'importance de maîtriser les concepts théoriques des réseaux de neurones a été mis en évidence tout au long du projet. Pour les réseaux neuronaux simples, nous nous sommes concentrés sur des aspects fondamentaux tels que l'initialisation des poids, le déroulement des phases d'apprentissage forward et backward, et l'application de fonctions de perte pour le réglage des paramètres du modèle. Cette compréhension approfondie a facilité la création d'un premier modèle en Python, avant de passer à des techniques plus sophistiquées avec TensorFlow pour élaborer un modèle convolutif.

La préparation et le prétraitement des données se sont avérés être des étapes cruciales pour la réussite de notre projet. En normalisant les données et en les séparant en ensembles d'entraînement et de test, nous avons pu entraîner efficacement le modèle et évaluer sa capacité à généraliser sur de nouvelles images.

Ce travail constitue une base solide pour de futures explorations des capacités des réseaux de neurones dans le domaine de la classification d'images. Il ouvre des perspectives pour l'exploration de nouvelles architectures de réseaux, l'affinement des techniques d'optimisation, et l'extension de l'application de ces modèles à d'autres catégories de données visuelles. La poursuite de l'amélioration de la précision et de la généralisation du modèle demeure un objectif clé pour les recherches à venir, illustrant le potentiel significatif des réseaux de neurones dans l'avancement de l'intelligence artificielle.

Sources

— Cours réseaux de neurones : Master DAC : https://dac.lip6.fr/wp-content/uploads/2023/02/cours5.pdf Cours fidle CNRS
— Yann le Cun : les réseaux convolutifs
— Adam : A Method for Stochastic Optimization (Kingma, Diederik and Ba, Jimmy), In International Conference on Learning Representations (ICLR), 2015. https://arxiv.org/pdf/1412.6980v9.pdf
— *Intelligence artificielle vulgarisée*, le Machine Learning et le Deep Learning par la pratique, Aurélien Vannieuwenhuyze, éditions Eni.
— *L'intelligence artificielle en pratique avec Python*, Recherche, optimisation, apprentissage, Huges Bersini et Ken Hasselmann, éditions Eyrolles.
— Figure 4 : Commande de la machine asynchrone à double alimentation – apport des techniques de l'intelligence artificielle - Scientific Figure on ResearchGate. Available from : https://www.researchgate.net/figure/Apprentissage-des-reseaux-de-neurone-par-lalgorithme-de-retropropagation_fig34_324929383 [accessed 27 Mar, 2024]
— Figure 5 : https://datacorner.fr/sgd-learning-rate/

Références

[1] Kingma, Diederik and Ba, Jimmy, *Adam : A Method for Stochastic Optimization*, In International Conference on Learning Representations (ICLR), 2015. https://arxiv.org/pdf/1412.6980v9.pdf

[2] LeCun1989BackpropagationAT, title=Backpropagation Applied to Handwritten Zip Code Recognition, author=Yann LeCun and Bernhard E. Boser and John S. Denker and Donnie Henderson and Richard E. Howard and Wayne E. Hubbard and Lawrence D. Jackel, journal=Neural Computation, year=1989, volume=1, pages=541-551, url=https ://api.semanticscholar.org/CorpusID :41312633