**Création d'une IA pour permettre à William de reconnaître les chiffres manuscrits, dignes d'un docteur, de Melissa dans les rapports de physique.**

Épreuve synthèse de programme

Zacky Charret

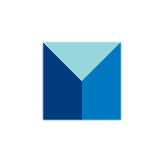
Hamza Gharbi

William Le Duin

Responsable. : Anik Soulière

Groupe 901 - 902

Collège de Maisonneuve

Mars 2025

**Table des matières**

[I – Introduction 3](#_Toc193148211)

[II – Description 4](#_Toc193148212)

[I - Réseau de neurones 4](#_Toc193148213)

[II - L'activation des neurones - Prédiction 5](#_Toc193148214)

[III – Fondements théoriques 6](#_Toc193148215)

[I - La rétropropagation et la descente du gradient 6](#_Toc193148216)

[IV – Méthodologie 10](#_Toc193148217)

[I – Jeu de données 11](#_Toc193148218)

[II – Réseau de neurones 11](#_Toc193148219)

[A – Propagation avant : 11](#_Toc193148220)

[B – Rétropropagation : 11](#_Toc193148221)

[III – Essais 11](#_Toc193148222)

[IV – Validation expérimentale 12](#_Toc193148223)

[V – Discussion 12](#_Toc193148224)

[Glossaire 13](#_Toc193148225)

[Bibliographie 15](#_Toc193148226)

[Annexes 16](#_Toc193148227)

**Résumé.** L'intelligence artificielle est devenue un pilier de notre époque, rendant sa compréhension de plus en plus essentielle. Ce rapport explique donc le fonctionnement des réseaux de neurones, cœur de l'IA, en développant un modèle capable de reconnaître des formes et des dessins avec Python. En s'appuyant sur la base de données EMNIST, composée d'images manuscrites, le réseau apprendra à identifier les chiffres (ou des lettres) grâce aux activations de ses neurones, ainsi qu'à l'ajustement de ses poids et biais. Le processus d'apprentissage, basé sur la descente de gradient et les produits matriciels, sera détaillé dans la partie mathématique du rapport, ainsi que vérifié expérimentalement.

**Abstract**. Artificial intelligence has become an essential part of our society, and understanding it is becoming increasingly important. Explaining how artificial intelligence works will involve understanding a neural network, the foundation of AI. The aim of this report is to explain how a neural network works, by creating one capable of recognizing shapes and figures in Python. While using a database such as EMNIST (a database known for its images of handwritings), the network will be able to predict, based on weights and biases, as well as recalibrate these weights and biases accordingly. That prediction and recalibration will be our mathematical focus, using gradient descent and dot product, and will be verified with our program.

# I – Introduction

L'intelligence artificielle est un développement technologique qui semble se développer à une vitesse fulgurante. Celle-ci devient peu à peu indispensable à notre société et engendre une révolution de notre histoire. Pourtant, il fut une époque où il était impensable d'imaginer qu'une machine pourrait réfléchir tel un humain. La naissance de l’intelligence artificielle remonte aux années 1950, avec le travail d'Alan Turing, mathématicien et cryptologue britannique, reconnu pour avoir décrypté les messages des nazis. En 1950, Turing publie *Computer Machinery and Intelligence*, un article qui pose la question fondamentale : *Les machines peuvent-elles penser ?* Ce questionnement marque le début de l’exploration de l’IA. En 1952, un scientifique crée un programme capable de jouer aux dames, l'un des premiers pas vers des systèmes intelligents. En 1957, Frank Rosenblatt invente le *perceptron multicouche*, qui est la base de notre modèle pour ce projet (un réseau de neurones artificielles). Puis, en 1958, John McCarthy conçoit LISP, le premier langage de programmation spécifiquement dédié à l’intelligence artificielle. En 1997, un moment clé se produit lorsqu’une IA, Deep Blue d'IBM, bat le champion du monde d’échecs Garry Kasparov. À partir de ce moment, les progrès de l'IA s’accélèrent, menant à des avancées comme ChatGPT en 2020, un modèle de langage révolutionnaire. Ces étapes soulignent l'évolution rapide de l'IA et sa capacité à transformer profondément divers secteurs.

# II – Description

Avant d’expliquer le fonctionnement d'une intelligence artificielle (IA), il est important de définir celle-ci. Une IA est la capacité d’un ordinateur à imiter au mieux les capacités de raisonnement humain. Parmi ces formes d’intelligence artificielles, les réseaux de neurones en font partie et sont couramment utilisé à des fins de recherches, tant sa ressemblance à un cerveau humain biologique donne le potentiel d’en être similaire. Ces réseaux de neurones peuvent donc effectuer des tâches similaires à des humains; cela peut aller de la simple classification entre 2 données différentes à la prédiction d'une suite de mots (ce qui est utilisé dans les robots conversationnels tels que ChatGPT), en passant par la reconnaissance visuelle, ce qui est notre exemple pour plus tard.

La question qui se pose maintenant est la suivante : **Comment fonctionne un réseau de neurones ? Comment parvient-il à simuler l’intelligence humaine ?**

## I - Réseau de neurones

Un réseau de neurones est composé de neurones artificiels. Ces neurones sont des variables qui possèdent une valeur qui dépend des données qu’on utilise. Plus la valeur est grande, plus elle est considérée importante. Ces neurones sont organisés en couches, comme le montre la figure 1.

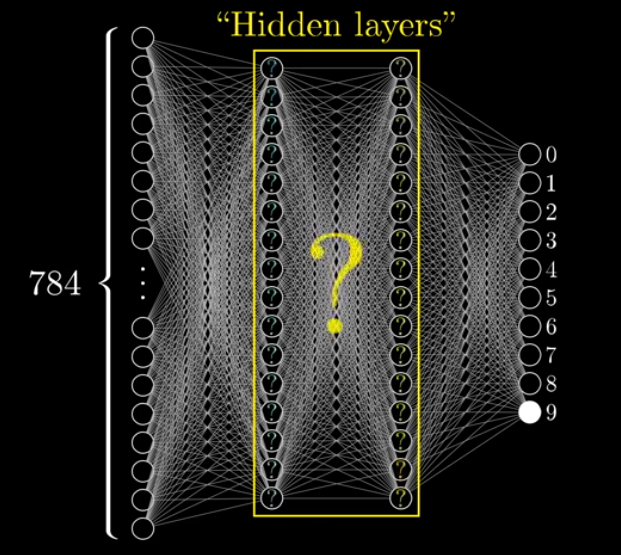


Figure 1: Neurones organisé en 4 couches différentes

Chaque couche joue un rôle précis et permet de réaliser une succession d’analyses. Par exemple, lors de la reconnaissance d’un chiffre, la première couche reçoit l’entrée (ex: une image d’un chiffre manuscrit). Ensuite, la deuxième couche détecte les formes élémentaires du chiffre, comme des courbes ou des lignes. Par la suite, la troisième couche associe ces formes à des caractéristiques spécifiques (par exemple, reconnaître qu’un cercle et une ligne droite correspondent au chiffre 9). Enfin, la dernière couche produit le résultat final, qui correspond à la réponse du modèle (exemple : "Ce chiffre est un 9"). La figure 2 illustre cette organisation des couches et le fonctionnement global du réseau de neurones.

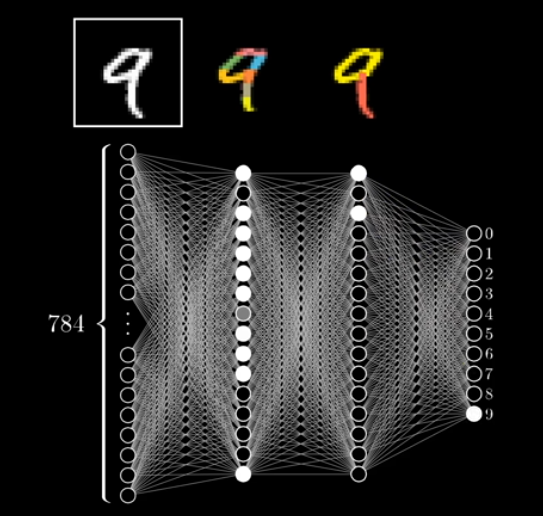


Figure 2 : Réseau de neurones qui essaye de reconnaître le chiffre 9

## II - L'activation des neurones - Prédiction

L’activation d’un neurone (c'est-à-dire la valeur qu’il prend entre 0 et 1 pour la fonction sigmoïde) ne se fait pas au hasard. Elle dépend de plusieurs éléments, notamment les poids, qui indiquent l’importance de chaque connexion entre neurones, le biais, qui est une valeur ajoutée pour ajuster l’activation et, finalement, la fonction sigmoïde, qui est une fonction mathématique transformant une valeur quelconque en un nombre compris entre 0 et 1, facilitant ainsi l’interprétation de l’activation. La figure 3 représente la fonction sigmoïde et son effet sur les valeurs d’entrée.

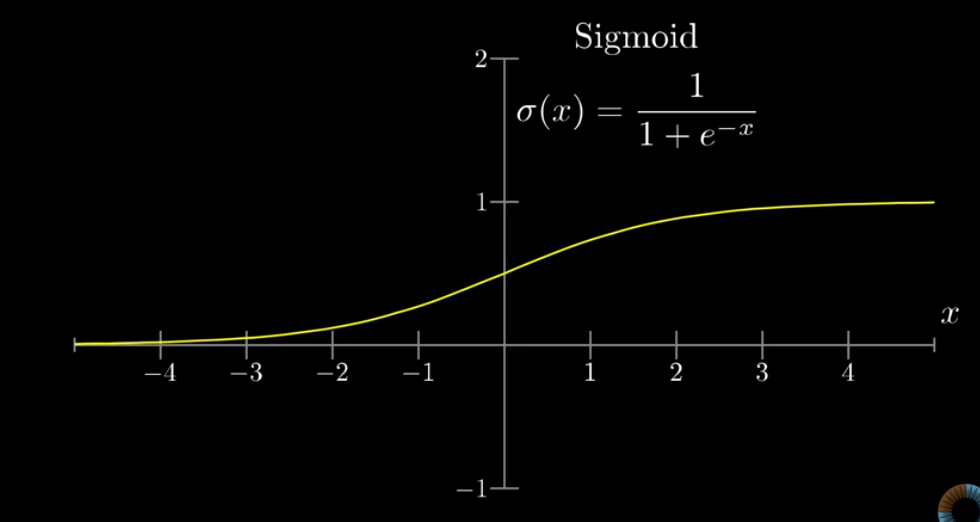


Figure 3 : Fonction sigmoïde

# III – Fondements théoriques

## I - La rétropropagation et la descente du gradient

La descente du gradient est une méthode qui permet d’ajuster les poids et les biais d’un modèle d’intelligence artificielle. La fonction de coût, qui mesure l’erreur du modèle, possède des minimums (coût faible) et des maximums (coût élevé), comme illustré dans la figure 4. L’objectif est d’atteindre un minimum de cette fonction afin d’obtenir un modèle optimal.

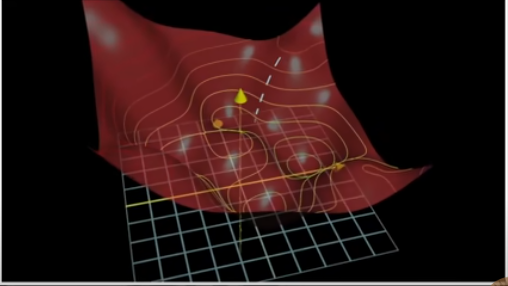


Figure 4 : Un plan en 3 dimensions qui représente la fonction de coût

En calcul différentiel, on trouve un minimum ou un maximum en cherchant les variables auquel la dérivée est égale à zéro. Ici, comme la fonction de coût dépend de plusieurs variables, elle forme un espace à plusieurs dimensions. On utilise donc le gradient plutôt que la simple dérivée : le minimum ou le maximum est atteint lorsque le gradient est nul. Mais comment savoir si l’on a atteint l’un ou l’autre ? En effet, un gradient nul peut indiquer les deux. Pour s’assurer d’atteindre un minimum, on suit le principe de la descente de gradient : au lieu de suivre le gradient (qui pointe vers la plus forte augmentation de la fonction de coût, donc vers un maximum), on prend la direction opposée, là où la fonction décroît le plus rapidement. Cependant, cette approche ne garantit pas d’atteindre le minimum global de la fonction de coût. Il est possible et beaucoup plus probable de se retrouver bloqué dans un minimum local. La seule manière de contourner ce problème d’atteindre un minimum local trop élevé, réduisant ainsi la précision en situation réelle, est d’entraîner le réseau de neurones plusieurs fois et d’ajuster les paramètres du gradient pour améliorer l’optimisation. La rétropropagation est un algorithme essentiel en intelligence artificielle. C’est grâce à lui que l’on peut ajuster le gradient dans la bonne direction. Pour l’implémenter dans notre code, il est d’abord nécessaire de connaître la formule de la fonction de coût, qui est :

Ici, a(L) représente la valeur prédite par l'IA, avec L indiquant le nombre de couche au totale, et yréel la valeur attendue. Par exemple, si l’image du chiffre 9 est présentée au réseau et qu’il prédit un 8, alors yréel = 9 et a(L) = 8. Une fois la fonction de coût définie, il s’agit d’analyser l’effet d’une variation minimale des poids et biais sur celle-ci. Pour cela, on calcule les dérivées partielles de la fonction de coût : d’abord par rapport aux poids, afin de mesurer leur impact, puis par rapport aux biais. Les formules sont les suivantes :

Ici, *C* est la fonction de coût, *a* l'activation du neurone et *z* un paramètre calculable. La fonction de coût ne dépend pas directement des poids *w* ni des biais *b*. Il est donc nécessaire d'appliquer la règle de dérivation en chaîne pour évaluer l'impact d'une variation de *w* ou *b*. En effet, modifier ces paramètres agit sur l’activation du neurone, déterminée par la formule suivante :

Où :

Dans cette formule, *a* représente l'activation d'un neurone, *b* le biais, *w* le poids, la fonction sigmoïde et L indique le nombre total de couches. Les figures 5 et 6 illustrent l'impact du poids et du biais sur la fonction de coût.

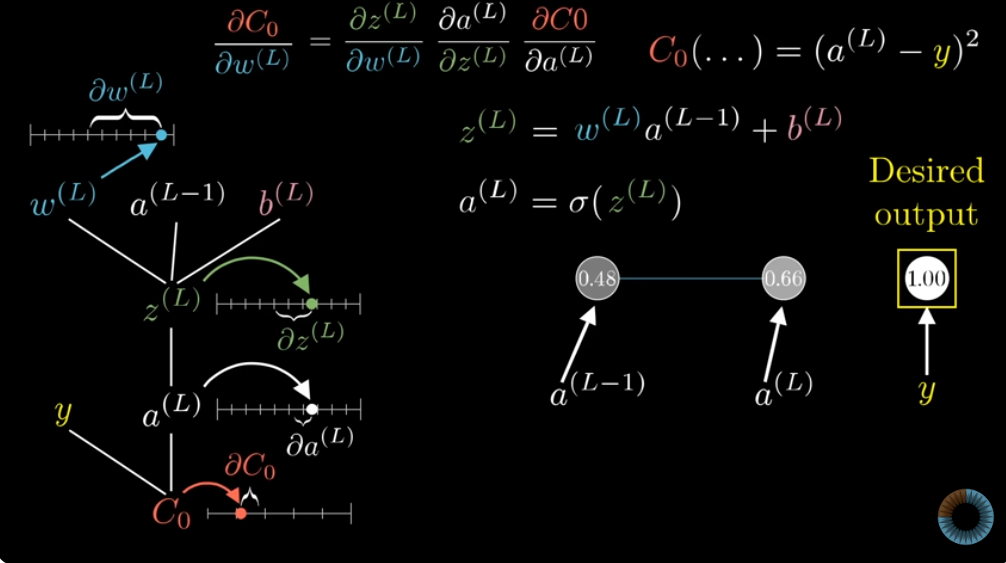


Figure 5: Effet d’une variation minime du poids sur la fonction de coût

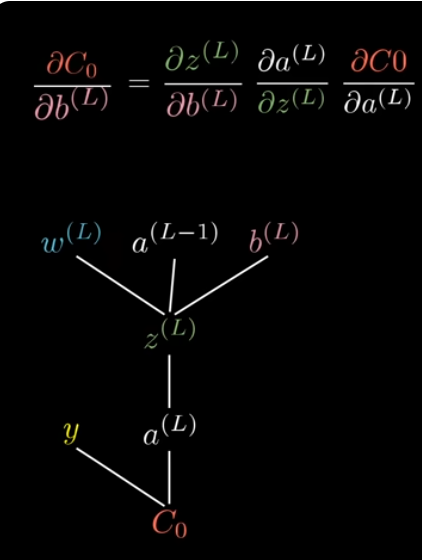


Figure 6: Effet d’une variation minime du biais sur la fonction de coût

D'ailleurs, voici la même formule du z sous forme matricielle qui est celle utilisé dans notre code:

Maintenant que les formules des dérivées partielles sont établies, il faut en déterminer les résultats. Cette section se concentre sur la dérivée partielle de la fonction de coût par rapport au poids. La dérivée par rapport au biais suit la même logique, seuls les résultats diffèrent; ces derniers et les calculs sont présentés dans l’annexe A. Pour commencer, on cherche la dérivée partielle de z(L) par rapport à w(L). Dans ce calcul, b(L) et a(L-1) sont considérés comme des constantes:

Deuxièmement, il faut trouver la dérivée partielle d’a(L) par rapport à z(L). Voici les calculs:

On utilise la règle:

Ici, f(z) = 1 + e-z, donc :

Ainsi:

Mais on sait que:

Finalement:

Troisièmement, il faut trouver la dérivée partielle de la fonction de coût en fonction d’a(L):

La dérivée de la fonction de coût par rapport au poids est donc égale à:

Ceci fonctionne **uniquement si le poids qu'on cherche se trouve entre la dernière couche et l'avant-dernière**. Pour ceux plus loin, on doit continuer la chaîne de la figure 6. Déjà de base, *faudra* réorganiser la formule pour que l'on va de l'élément le plus proche de la fonction du coût au plus loin :

À partir d’ici, on va remarquer que c’est similaire à cette fonction :

Mais vu qu’on cherche plus loin, donc *w*(L-1), on doit éviter *w*(L) dans la figure 6 et changer vers *a*(L-1)

Ce qui va nous donner ceci :

Où *a*(L-1) devient :

.

Cela suit le même modèle pour les couches suivantes.

# IV – Méthodologie

Dans cette partie du rapport, on va expérimenter les fondements théoriques d’un réseau de neurone. Pour cela, on va utiliser Python, accompagné de NumPy (pour les calculs matricielles) et Matplotlib (pour montrer les résultats de ces calculs). Le code est disponible dans l’annexe B. Notre objectif sera qu’il soit capable de reconnaître des formes; des chiffres en priorité, puis des lettres quand on peut s’assurer du fonctionnement du réseau envers ces chiffres.

## I – Jeu de données

Avant de commencer à faire des tests avec le réseau de neurone, il faut déjà savoir quoi utiliser. Dans notre cas, on va utiliser EMNIST, qui est un jeu de données contenant une très grande quantité d’écritures faites à la main; chiffres ou lettres, clair, ainsi qu’en nuances de gris. Avec cela, on va prioriser les chiffres (étant donné qu’il y a plus de ressources pour trouver où se trouve les problèmes, tel que 3Blue1Brown) et on va l’insérer dans le code en tant que variable (2D-Array de NumPy). Avec ces images dans le code (*main.py*), on est prêt à l’insérer dans le réseau.

## II – Réseau de neurones

Le réseau de neurones, d’une taille qu’on peut lui donner, se forme dans le fichier *neurone.py*. Ce fichier a pour objectif d’implémenter toutes les fonctions essentielles au réseau neuronal directement dans une classe qui pourra être utilisé dans d’autres fichiers; dans notre cas, *main.py*. On a utilisé NumPy pour former ces couches et contenir les poids et biais du réseau. Dans cette dernière, on a implémenté deux actions très importantes lorsqu’elle a accès aux images : La propagation avant et la rétropropagation.

### A – Propagation avant :

La propagation avant est l’action à l’intelligence artificielle d’activer, couche par couche, les activations des neurones à l’aide des poids, biais et de la fonction d’activation. Les calculs sont faits à partir de NumPy, comme toujours, sur *neurone.py*. Vu que ce module permettait de faire des calculs avec des matrices, elle nous a simplifié la tâche pour faire un seul gros calcul. Dans notre code, on fait donc deux calculs par couche; un pour la fonction z (voir Fondements théoriques) et un autre pour la fonction d’activation.

### B – Rétropropagation :

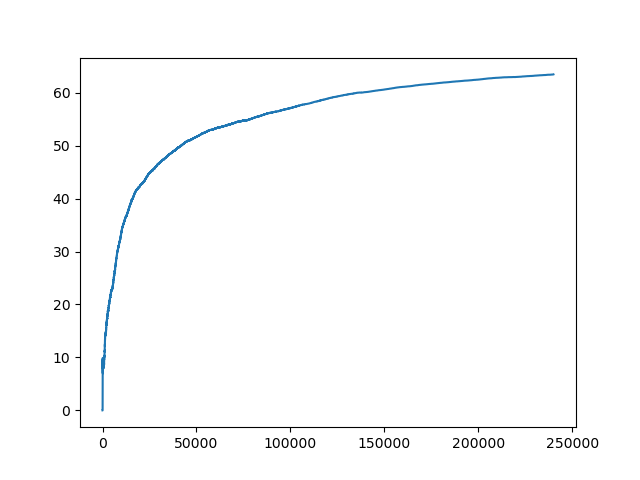
La rétropropagation est, comme mentionné dans les fondements théoriques, la possibilité de l’IA de modifier en soi ses poids et biais, pour que les futures propagations avant permettent de prédire correctement les images, grâce à la fonction de coût. Pour les matrices, on devait faire très attention aux dimensions et comment on fait la transposé de certaines matrices. Sinon, le calcul matriciel est quasiment identique au calcul présenté à la fin de la partie *Fondements théoriques*, ce qui va nous éviter de nous répéter. La transposition est nécessaire pour assurer une bonne dimension lors du produit matriciel, sans quoi il sera impossible de le calculer. Seule la multiplication avec les poids et les activations ont besoin d’être un produit matriciel; dans le cas contraire, on aura juste besoin d’un produit de Schur (une multiplication élément par élément entre deux matrices).

## III – Essais

Après que le réseau de neurone soit capable de faire ces deux actions, il doit être capable de faire cela dans une boucle, pour toutes les images. Après que l’entièreté des images d’entraînement est utilisée, il va nous afficher un pourcentage de précision globale, montrant la précision du programme. Cela va servir pour valider notre capacité de créer un réseau de neurones à partir d’un code de programmation et des fondements théoriques.

# IV – Validation expérimentale

Ici, on va tenter de voir si notre tentative d’une intelligence artificielle fonctionne. Un pourcentage conséquent (beaucoup plus grand que 10 %) sera assez pour conclure sur un fonctionnement. Cependant, si la précision de l’IA est autour de 10 %, cela va indiquer un problème à régler dans le futur.

Dans notre cas, on a réussi de produire un réseau avec une précision de 63.45 %, comme le montre le graphique 1.

Graphique 1 – La précision du réseau neuronal en fonction du nombre d’images données à l’IA

# V – Discussion

Lors de ce projet, notre équipe a rencontré de nombreux obstacles. Nous avons dû nous adapter et trouver de nouvelles solutions à chaque fois.

Par exemple, au début du projet, nous voulions utiliser la fonction ReLU au lieu de la fonction sigmoïde afin de créer notre IA. En effet, la fonction ReLU permet à l’IA d’apprendre beaucoup plus rapidement que la fonction sigmoïde. Toutefois, contrairement à cette dernière, elle ne normalise pas automatiquement les valeurs qu’elle produit, c’est-à-dire qu’elle ne les limite pas entre 0 et 1. Or, une activation supérieure à 1 ou inférieure à 0 n’aurait pas de sens.

Pour résoudre ce problème, nous avons tenté d’utiliser la fonction softmax afin de normaliser les valeurs issues de la ReLU. Cependant, nous avons rencontré plusieurs difficultés pour les faire fonctionner ensemble. Nous avons donc choisi de revenir à la fonction sigmoïde, qui est certes moins efficace, mais plus simple à manipuler.

De plus, un autre problème que nous avons rapidement résolu avec la sigmoïde, mais qui persistait avec la ReLU, est celui des *neurones morts.* Cela veut dire qu’après de l’entrainement, le poids donc l’importance d’une activation d’un seul neurone est priorisée par rapport aux autres qui deviennent nul. Toutefois, après avoir résolu ce problème avec la fonction sigmoïde, notre équipe a été capable d’atteindre une précision supérieure au seuil de 10 % (ce qui correspond à une simple supposition). Cela prouve que nos fondements théoriques étaient valables et qu’ils fonctionnent.

# Glossaire

-Lot / Batch: Un ensemble d’échantillons (d’exemples) de données utilisées pendant l’entraînement.

-Fonction d’activation / Activation function : Fonction mathématique, comme ReLU (Rectified Linear Unit) et la sigmoïde, appliquée à la somme pondérée des entrées d’un neurone dans un réseau de neurone.

-Descente de gradient stochastique / Stochastic Gradient Descent (SGD) : Le SGD est un algorithme d’optimisation de la fonction de perte qui met à jour les paramètres de poids et biais après chaque échantillon ou petit lot par itération. Le taux d’apprentissage est souvent réduit progressivement lorsque l’optimisation se rapproche d’un minimum local (ou global) de la fonction de perte.

-Taux d’apprentissage / Learning rate : Le taux d’apprentissage détermine l’ampleur des mises à jour des paramètres (poids et biais), c’est un hyperparamètre qui contrôle la vitesse (le pas) de descente vers un minimum de la fonction de perte. Il peut être fixe ou variable.

-Hyperparamètre / Hyperparameter : Un hyperparamètre est une variable que l’on ajuste avant l’entraînement qui contrôle l’apprentissage (ex : Taux d’apprentissage, le nombre de couches, etc.).

Paramètre / Parameter : Un paramètre est une variable qui est modifiée par la descente de gradient pendant l’entraînement (ex : poids et biais du modèle).

-Itération / Iteration : Une itération est un passage complet des ensembles d’opérations ou d’étapes dans un algorithme. Pour l’entraînement une itération est une mise à jour de tous les paramètres du modèle basée sur un lot de données.

-Data set / Jeu de données: Un jeu de données est un ensemble structuré de données utilisé pour entraîner le model. Avec notre algorithme, c’est une collection d’image représentant des chiffres écrits à la main entre 0 et 9 avec la réponse dans les informations du fichier.

-Précision / Accuracy : Nombre de prédiction de classification correctes divisé par le nombre total de prédiction

-Modèle de Deep Learning / Deep model: Réseau de neurones contenant plusieurs couches cachéesé

-Époque / Epoch : Cycle d’entrainement qui contient le nombre de lot. Une époche représente le nombre d’échantillons divisé par la taille de lot dans une itérations.

-Couche cachées / Hidden layers : Couche d'un réseau de neurones située entre la couche d'entrée (les caractéristiques) et la couche de sortie (la prédiction). Chaque couche cachée comprend un ou plusieurs neurones.

-Fonction de perte (Fonction de coût) / Loss function (Cost function): La fonction de perte (ou de coût) est une mesure qui évalue l'écart entre les prédictions du modèle et les résultats réels (ou cibles). Elle quantifie l'erreur du modèle alors plus la valeur de la fonction de perte est faible, plus le modèle est performant.

-Activation / Feedforward: L'activation est une fonction appliquée à la sortie d'un neurone dans un réseau de neurones, déterminant si ce neurone doit être activé ou non. Elle permet de moduler l'information qui traverse le réseau pour permettre au réseau de neurones de capturer des relations complexes dans les données. Les fonctions d'activation courantes incluent la sigmoïde et ReLU (Rectified Linear Unit).

-Softmax function / Fonction softmax: Fonction dont le but est de normaliser des valeurs, c’est-à-dire les coincer entre 0 et 1.

# Bibliographie

Salesforce. (2025, Mars 16). *What is the history of artificial intelligence (AI)?* Retrieved from Tableau: https://www.tableau.com/data-insights/ai/history#history

Sanderson, G. (2017, Octobre 5). *But what is a neural network? | Deep learning chapter 1*. Retrieved from Youtube: https://youtu.be/aircAruvnKk?si=iN6IWfq8URVrpy4a

Turing, A. (1950). *Computing Machinery and Intelligence.* Oxford: Oxford University Press.

3Blue1Brown. (2017, 5 octobre). *But what is a neural network ? | Deep learning chapter 1* [vidéo]. Youtube. <https://youtu.be/aircAruvnKk?si=uxIDpkcoXiNox8Ts>

3Blue1Brown. (2017, 3 novembre). *Backpropagation, step-by-step | DL4* [vidéo]. Youtube. <https://youtu.be/IHZwWFHWa-w?si=JVm15BatQPFDcOLt>

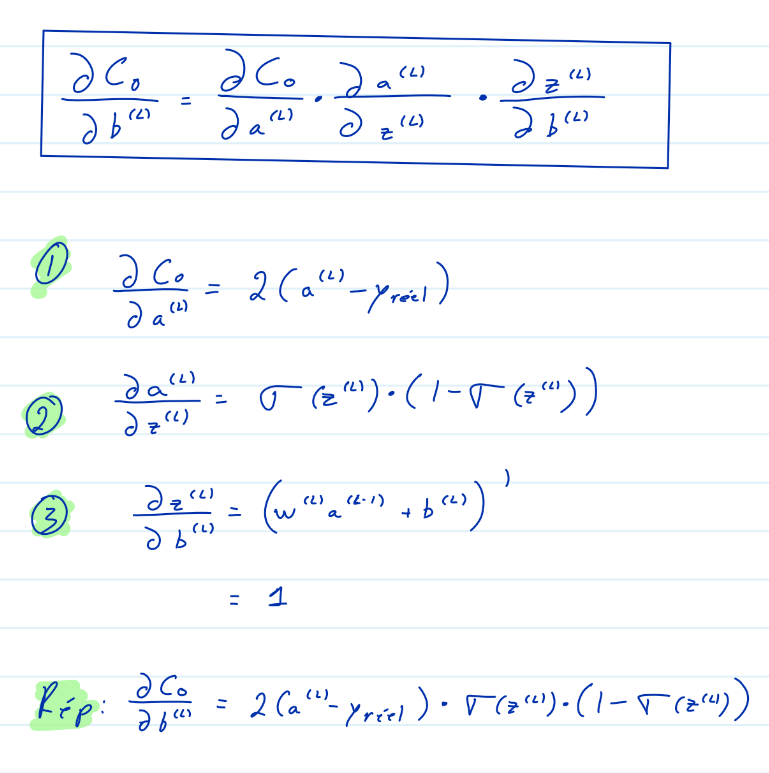
3Blue1Brown. (2017, 16 octobre). *Gradient descent, how neural networks learn | DL2* [vidéo]. Youtube.  
<https://youtu.be/IHZwWFHWa-w?si=JVm15BatQPFDcOLt.>

Oxford University Press. (s.d.). *Oxford Languages.* <https://languages.oup.com/>

*What is a neural network ?*. (s.d.). Elastic.co. <https://www.elastic.co/fr/what-is/neural-network>

Je reconnais avoir utilisé ChatGPT (<https://chat.openai.com/>) le 17 avril 2025 pour du soutien à la rédaction. Les résultats des invites ont été utilisés pour obtenir des suggestions d’amélioration du texte pour chaque section. J’ai choisi une formule que j’ai modifiée pour le personnaliser et le rendre conforme à la suite du texte. J’ai employé le modèle de requête suivant : « Rend ce passage plus court et plus clair stp ».

# Annexes

A – Calcul du gradient d’un biais de la fonction d’activation sigmoïde

B – Lien au code source du projet

Voici le lien du projet : <https://github.com/MisterZork/Maisonneuve_AI>