**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

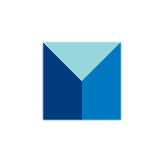
Hamza Gharbi

William Le Duin

Responsable. : Anik Soulière

Groupe 901 - 902

Collège de Maisonneuve

Avril 2025

**Table des matières**

[I – Introduction 3](#_Toc196708407)

[II – Réseau de neurones 3](#_Toc196708408)

[II – Fondements théoriques 5](#_Toc196708409)

[I - La descente du gradient | Rétropropagation 5](#_Toc196708410)

[IV – Méthodologie 12](#_Toc196708411)

[I – Jeu de données 12](#_Toc196708412)

[II – Réseau de neurones 12](#_Toc196708413)

[A – Erreurs notables sur les calculs matricielles 12](#_Toc196708414)

[III – Essais 12](#_Toc196708415)

[IV – Validation expérimentale 13](#_Toc196708416)

[V – Discussion 14](#_Toc196708417)

[Glossaire 15](#_Toc196708418)

[Bibliographie 17](#_Toc196708419)

[Annexes 19](#_Toc196708420)

**Résumé.** Ce rapport scientifique explique 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 même 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 testé expérimentalement.

**Abstract**. This scientific report explains 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 devenue un pilier de notre époque, rendant sa compréhension de plus en plus essentielle. Celle-ci est un développement technologique qui semble se développer à une vitesse fulgurante; 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 – Réseau de neurones

Une intelligence artificielle (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 artificiels 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. **Pour pouvoir comprendre le fonctionnement de l’intelligence artificielle et en recréer une, il faudra savoir comment fonctionne un réseau de neurone.**

Cette section explique de manière simplifiée le fonctionnement d’un réseau de neurones. L’IA reçoit l’image d’un chiffre entre 0 et 9 et doit deviner de quel chiffre il s'agit. Prenons l’exemple du chiffre 5. Comme illustré à la figure 1, chaque pixel de l’image possède une valeur entre 0 et 1 : plus cette valeur est élevée, plus le pixel est blanc et inversement. Ces valeurs sont associées aux neurones de la couche d’entrée. L’image ayant 784 pixels, la couche d’entrée comporte 784 neurones. Chaque neurone de la couche suivante calcule leur propre valeur à partir d’une combinaison linéaire des poids (la force des connexions) et des neurones de la couche précédente, suivi d’un biais (ajustement de l’ordonnée à l’origine de la combinaison linéaire). Le résultat passe ensuite par une fonction d’activation; dans notre cas, la fonction sigmoïde, qui a comme particularité de garder les valeurs entre 0 et 1 (voir figure 2).

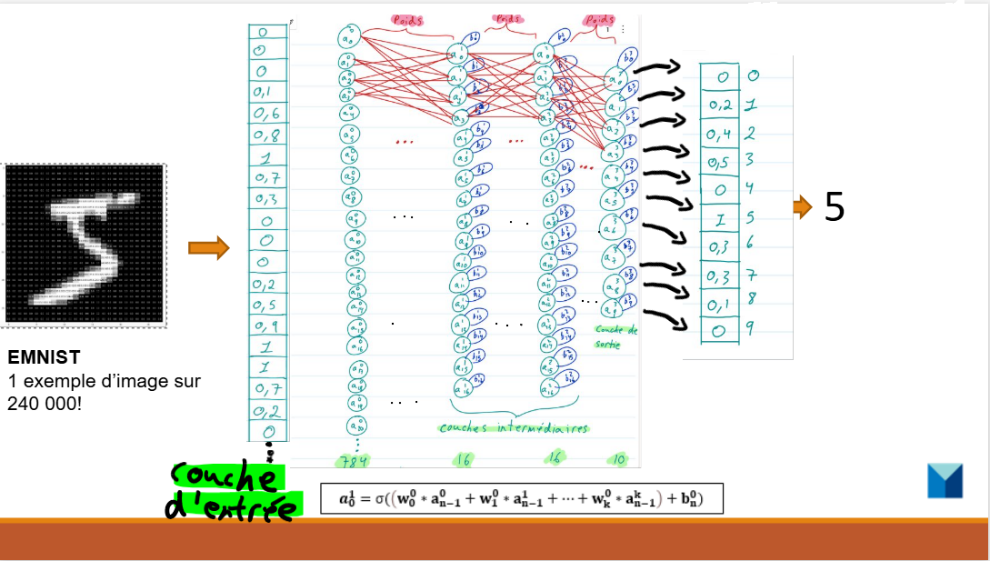


Figure 1 : Réseau de neurone de taille [784-16-16-10], dessiné par l’équipe d’ESP.

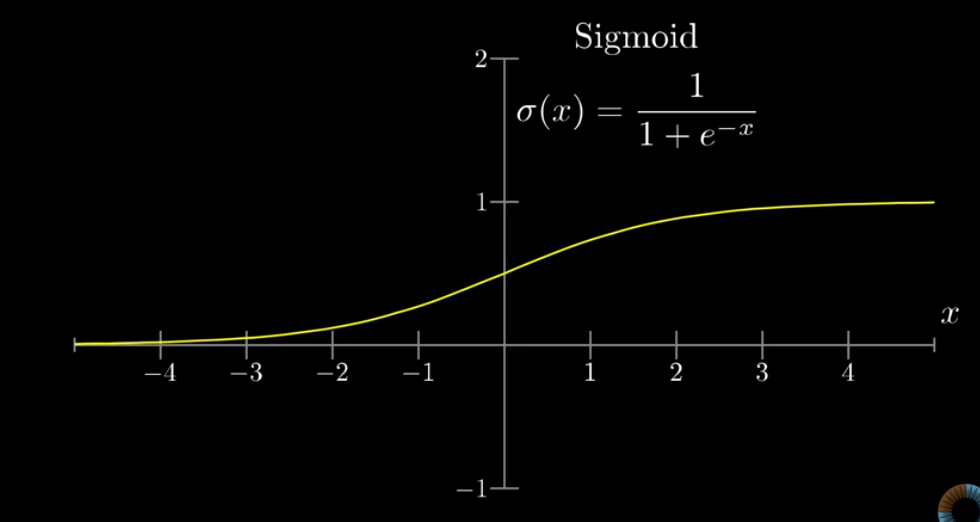


Figure 2 : Fonction sigmoïde [[1]](#_Bibliographie)

Ce processus se répète jusqu’à la couche de sortie, où la valeur de chaque neurone représente maintenant un score. L'IA choisit le chiffre correspondant au score le plus élevé. Dans notre exemple (figure 1), elle identifie correctement le chiffre 5. Les couches intermédiaires n’ont pas de rôle précis. Cependant, si l’IA se trompe, elles sont ajustées pour améliorer ses prédictions.

# II – Fondements théoriques

Cette partie va se concentrer sur les mathématiques en lien avec l’entraînement de l’intelligence artificielle. La propagation avant a déjà été résumé dans la section précédente (une combinaison linéaire de chaque poids et neurones de la couche précédente vers la suivante, suivi d’un biais, puis une fonction d’activation). Pour la calibration, on aura besoin de 2 éléments : La fonction de coût et la descente du gradient.

## I - La descente du gradient | Rétropropagation

La descente du gradient est une méthode d’optimisation 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. Dans notre projet, la fonction de coût part de , comme le montre les calculs de la figure 3.

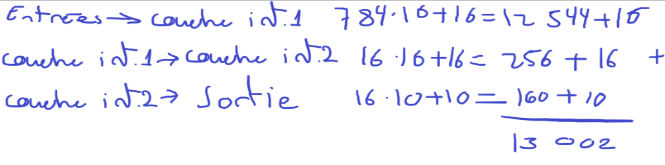


Figure 3: La fonction de coût prend en compte tous les poids et biais du réseau de neurone. Cette image montre le calcul arrivé à montrer la dimension de base que la fonction de coût calcule. Dessiné par l’équipe d’ESP

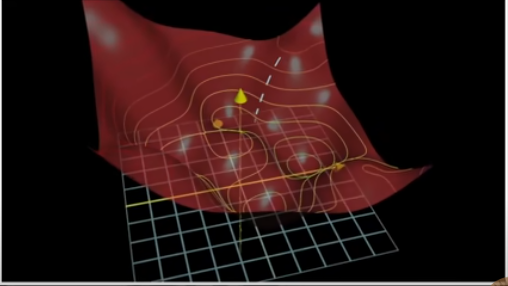


Figure 4 : Une surface en 3 dimensions qui représente la fonction de coût [[3]](#_Bibliographie)

Puisque la fonction de coût évolue dans un espace à plusieurs dimensions, on trouve un point critique en cherchant les points où toutes les dérivées partielles sont nulles, c’est-à-dire où le gradient est égal à zéro. Mais comment savoir si on a atteint un minimum, maximum ou point de selle ? En effet, un gradient nul peut indiquer les trois. 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. On applique le principe du gradient à la fonction de coût, qui est composée de la différence entre le vecteur des scores prédits et le vecteur des scores attendus, élevée au carré. Le but est de minimiser cette fonction de coût, afin de réduire l'erreur entre les prédictions et les vraies valeurs, et ainsi rendre notre IA plus performante. La figure 5 illustre la structure et le rôle de la fonction de coût dans ce processus.

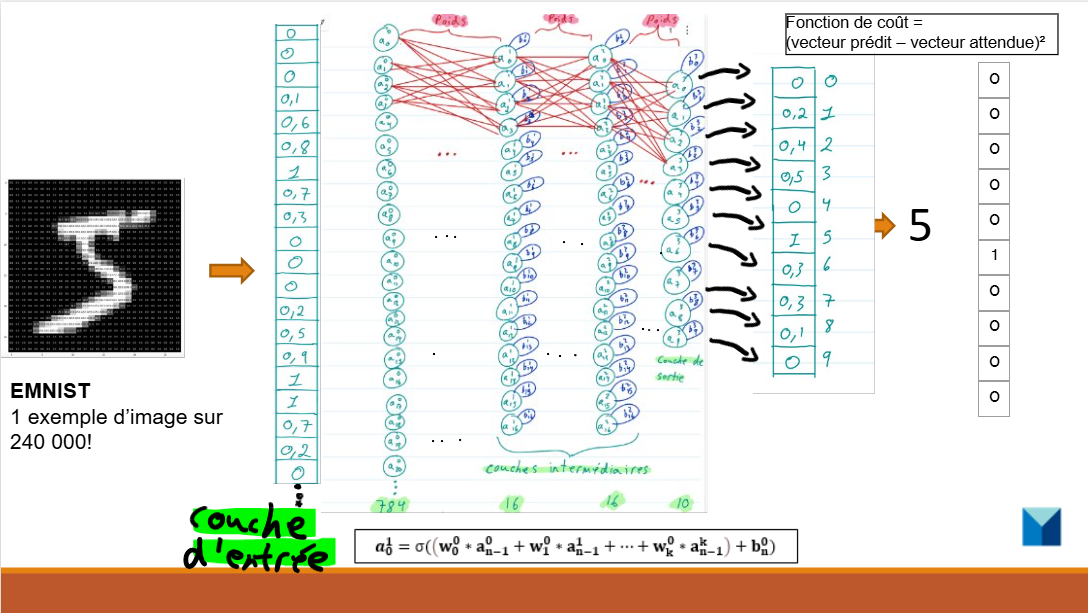


Figure 5: De quoi est composé la fonction de coût. Dessiné par l’équipe d’ESP

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) est le vecteur des scores prédits par l'IA, où désigne la dernière couche, et yréel est le vecteur des scores attendus. Encore une fois, la figure 5 illustre bien ceci. 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. Cependant, on ne peut pas calculer directement les dérivées partielles, car la formule de la fonction de coût n'exprime pas explicitement les poids ( ) et les biais () . Pour contourner ce problème, on utilise la méthode de la dérivation en chaîne. Voici comment cela s'applique :

Ici, est la fonction de coût, l'activation du neurone, un paramètre calculable et indique la dernière couche du réseau. Modifier influence directement l'activation du neurone, selon la formule suivante :

Où :

Les figures 6 et 7 illustrent l'impact du poids et du biais sur la fonction de coût.

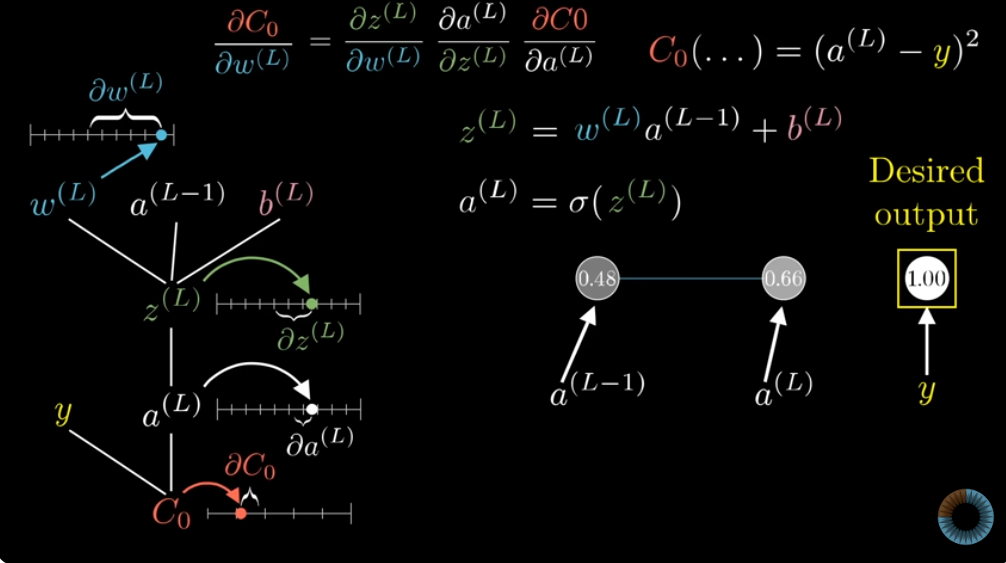


Figure 6 : Impact d'une légère variation des poids entre l’avant-dernière et la dernière couche sur la fonction de coût [[4]](#_Bibliographie)

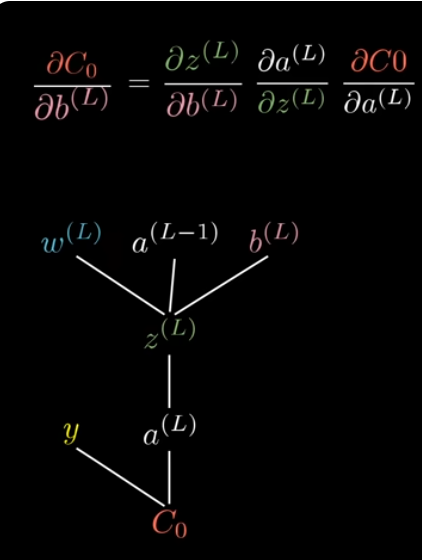


Figure 7 : Impact d'une légère variation des biais entre l’avant-dernière et la dernière couche sur la fonction de coût [[4]](#_Bibliographie)

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 par rapport . Dans ce calcul, et sont considérés comme des constantes :

Deuxièmement, il faut trouver la dérivée partielle d’ par rapport à . Voici les calculs :

On utilise la règle :

Ici, , donc :

Ainsi :

Mais on sait que :

Finalement :

Troisièmement, il faut trouver la dérivée partielle de la fonction de coût en fonction d’:

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 les poids dans les couches précédentes, on doit continuer la chaîne de la figure 6. Tout d’abord, il faut réorganiser la formule en partant de l’élément le plus proche de la fonction de coût vers l’élément le plus éloigné :

Mais vu qu’on cherche plus loin, donc , on doit éviter dans la figure 6 et changer vers en continuant la dérivation en chaine comme le montre la figure 8:

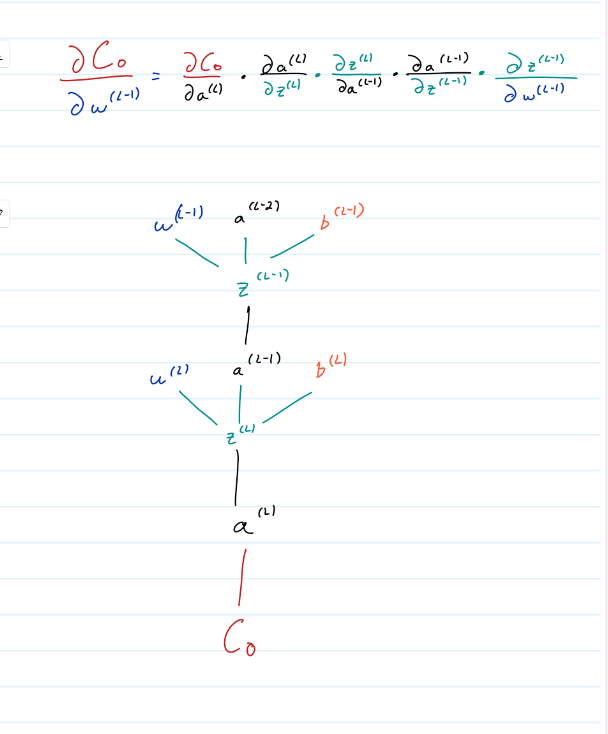


Figure 8 : Impact d'une légère variation des poids entre l’avant-dernière couche et celle qui la précède sur la fonction de coût. Dessiné par l’équipe d’ESP.

Ce qui va nous donner ceci :

où 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 chiffres manuscrits. Après, faudra montrer quel est la précision auquel il peut reconnaître ces chiffres.

## I – Jeu de données

Le jeu de données EMNIST (Extended Modified National Institute of Standards and Technology) contient une très grande quantité d’écritures faites à la main; des chiffres ou des lettres. Les chiffres seront les exemples utilisés et on aura accès à 2 sets; un de 70k images et un autre de 270k images. Avec NumPy, ces images seront mises dans un tableau (plusieurs listes contenant la valeur de chaque pixel, regroupé pour chaque image). Ce tableau sera récolté dans *main.py* et chaque liste de ce tableau servira à chaque itération du réseau neuronal.

## II – Réseau de neurones

Le réseau de neurone est géré par *neurone.py*. Ce module utilise NumPy pour contenir toutes les couches de neurones dans des listes ou des matrices, permettant d’exécuter des calculs matriciels, à la place de calculer dans une boucle pour chaque neurone, poids et biais (ce qui économise énormément de temps de calcul à l’ordinateur). Avec cette rapidité, cela va nous permettre d’expérimenter plus souvent avec les hyperparamètres et d’avoir un entraînement plus rapide (dans l’ordre de quelques minutes).

### A – Erreurs notables sur les calculs matricielles

Il y a quelques points à souligner sur les calculs matriciels qui sont exécutés dans ce module. NumPy a 2 types de valeurs : Des tableaux à n dimensions (np.ndarray) et des matrices (np.matrix). Le 2e est juste un tableau spécifiquement à 2 dimensions. Cette différence légère fut un grand trouble à régler dans notre code, car les multiplications matricielles ou les produits de Hadamard utilisait des fonctions différentes dépendamment de quel type de valeurs on utilisait.

De plus, vu que le calcul matriciel de la rétropropagation est inverse à la propagation avant, les matrices de poids doivent être transposée, pour que les dimensions des gradients de poids soient respectées. Ce petit problème m’a causé beaucoup de perte de temps à régler. Si vous voulez en savoir plus sur le pourquoi de cette transposition, il y a une source dans la bibliographie qui explique cela en détail [[13]](#_Bibliographie).

## III – Essais

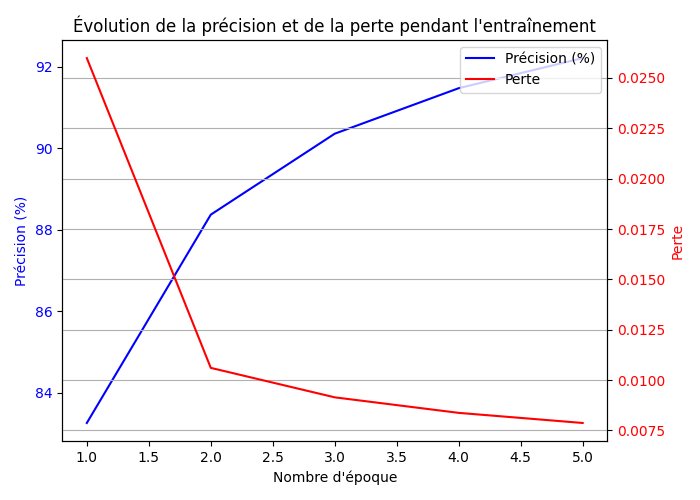
Après chaque itération du programme, les réponses de l’IA et de l’image sont sauvegardées, puis réutilisé par Matplotlib pour afficher la précision et la perte de l’IA. Le premier est un pourcentage de réussite de prédiction de l’image, tandis que le 2e montre l’écart entre l’IA et les valeurs attendue de l’image, comme la figure 5. Cela a permis de créer des graphiques tel que le graphique 1, qui va servir à prouver le fonctionnement de notre IA.

# IV – Validation expérimentale

Pour montrer que l’IA est capable de prédiction, il faut qu’il soit capable d’avoir une précision au-dessus de 10 %. Idéalement, faudrait avoir un pourcentage qui se trouve autour de 90 %. Pour cela, on peut utiliser des hyperparamètres pour optimiser les descentes de gradients. On va aussi utiliser le set à 270k images de EMNIST, pour plus d’exemples différents.

Parmi ces hyperparamètres, il y a la taille de lot, qui permet de définir le nombre d’images utilisé avant la rétropropagation. Dans notre cas, on a utilisé **une taille de lot de 10 images**. Il y a aussi l’époque, permettant d’augmenter artificiellement le nombre d’exemples d’images. Ayant augmenté cela a **5**, cela donne l’impression à notre IA d’avoir 5 \* 270k images, **soit 1.35 millions d’images manuscrites**, ce qui est énorme. Le taux d’apprentissage, qui est notre pas du gradient, est de 0.8, ce qui, après quelques tests expérimentaux, semblait idéal (un taux plus élevé bloquait le réseau à une précision plus basse, alors qu’un taux plus bas aurait pris trop de temps à atteindre la précision voulue). La perte du graphique 1 montre effectivement que le taux d’apprentissage était idéal, devenant un plateau plus tard (autour de la 4e – 5e époque).

Dans notre cas, on a réussi de produire un réseau avec une précision de 92 %, comme le montre le graphique 1.



Graphique 1 – La précision et la perte du réseau neuronal en fonction du nombre d’époque. Récolté à partir du code.

On a testé ces poids et biais obtenue avec le set à 70k images, pour voir si la précision restait la même et le résultat nous avaient surpris : La précision avait monté à 94 %, impressionnant.

# 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 de s’entraîner beaucoup plus rapidement que la fonction sigmoïde, ne demandant pas les calculs d’exponentiels de manière répétitive. 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. Le gradient devenait exponentiellement plus grand, augmentant les valeurs des poids et biais à des valeurs astronomiques (dans l’ordre de 1017). Cela rendait la prédiction presque impossible au réseau, donnant une couche de sortie dont tous les neurones avaient la même valeur. Cela nous avait contraint à tester la sigmoïde, qui avait fonctionné correctement.

Même avec ces hautes précisions, on pouvait atteindre plus haut avec un réseau de neurones à convolution. Il a comme avantage de changer les neurones de la couche d’entrée; au lieu d’utiliser les valeurs des pixels d’un image, on utilise le résultat de la convolution d’une image, permettant de changer l’image dépendamment de la matrice de convolution (la détection de contour étant ce qui serait de mieux pour la reconnaissance de chiffres). Ces nouvelles images sont ensuite passé dans un réseau de neurone de convolution. Ça aurait été l’idéal, mais on aurait eu besoin de comprendre un réseau de neurone comme celui de cette ESP. Peut-être que la prochaine équipe d’ESP sur l’IA prendra le relai et utilisera le réseau de neurones convolutifs (CNN). 😉

# Glossaire

- Combinaison linéaire / Linear combination : Une addition de variables {x1, x2, …, xn-1, xn} multiplié par leur constante respective {a1, a2, …, an-1, an}.

- Fonction d’activation / Activation function : Fonction mathématique appliqué à la fin de la combinaison linéaire des neurones et poids de la couche précédente et son addition à un biais. Il s’agit de la dernière étape de la propagation avant d’une couche. Il y a plusieurs fonctions d’activations, dont la fonction sigmoïde, ReLU et softmax.

- Fonction de coût / Cost function : La fonction de coût (ou de perte) est une mesure qui évalue l'écart entre les prédictions du modèle et les résultats réels. Elle quantifie l'erreur du modèle alors plus la valeur de la fonction de coût est faible, plus le modèle est performant. Il y a plusieurs formules de fonction de coût, dont L1 (Différence des valeurs, en absolue) et L2 (Différence des valeurs, au carré). Il y a aussi la perte logarithmique, étant beaucoup plus extrême entre 0 et 1.

- Gradient / Gradient : Un gradient est un vecteur qui indique la direction de la plus forte montée d'une fonction.

- Descente de gradient / Gradient Descent (GD) : Le GD est un algorithme d’optimisation de la fonction de coût qui met à jour les paramètres de poids et biais selon la fonction de coût. Le taux d’apprentissage est souvent réduit progressivement lorsque l’optimisation se rapproche d’un minimum local (ou global) de la fonction de coût. Il y a 3 types de descente du gradient; stochastique (SGD, changement à chaque exemple), mini-lot (changement à chaque lot) et par lots (changement à chaque époque).

- Jeu de données / Dataset : Le jeu de données représente l’entièreté des exemples qui peuvent être utilisé par l’intelligence artificielle. Ils peuvent représenter des images, des mots, des vidéos, des extraits de voix, etc.

- Lot / Batch : Un ensemble d’échantillons (d’exemples) de données utilisées pendant l’entraînement. Habituellement, il peut représenter un nombre plus petit que le jeu de donnée complet.

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

- 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 du gradient vers un minimum de la fonction de coût. 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. Les poids et biais sont très souvent les seuls paramètres du réseau.

- Itération / Iteration : Une itération est un passage complet des ensembles d’opérations ou d’étapes dans un algorithme. Pour l’entraînement, une itération effectue une propagation avant complète, suivi d’une rétropropagation grâce

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

- Couche d’entrée / Input layer : La couche d’entrée représente tous les neurones d’une couche représentant un exemple, qui sera utilisé à des fins d’entraînement ou de prédictions.

- Couche de sortie / Output layer : La couche de sortie représente tous les neurones qui serviront au programme de donner une réponse pour l’IA, comme celle de la prédiction; un neurone avec la valeur la plus élevée sera celle qui représente la réponse de l’IA, celle encodé dans ce neurone.

- Couche cachée / Hidden layer : 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.

# Bibliographie

1. **Sanderson, G. [3Blue1Brown].** (5 octobre 2017). *But what is a neural network? | Deep learning chapter 1* [Vidéo]. YouTube. <https://youtu.be/aircAruvnKk?si=iN6IWfq8URVrpy4a>

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

3. **Sanderson, G. [3Blue1Brown]**. (3 novembre 2017). *Backpropagation, intuitively | DL3* [Vidéo]. YouTube. <https://www.youtube.com/watch?v=Ilg3gGewQ5U>

4. **Sanderson, G.** **[3Blue1Brown].** (3 novembre 2017). *Backpropagation calculus | DL4* [Vidéo]. YouTube. <https://youtu.be/tIeHLnjs5U8?si=xe9QWa3HhaEfMaMD>

5. **Salesforce.** (s.d.). *What is the history of artificial intelligence (AI)?* Tableau. <https://www.tableau.com/data-insights/ai/history#history>

6. **Turing, A.** (1950). *Computing machinery and intelligence.* Oxford University Press.

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

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

*9.* ***GeeksforGeeks.*** *(s.d.). Sklearn different loss functions in SGD.* [*https://www.geeksforgeeks.org/sklearn-different-loss-functions-in-sgd/*](https://www.geeksforgeeks.org/sklearn-different-loss-functions-in-sgd/)

10. **Python Software Foundation.** (s.d.). *Python documentation.* <https://www.python.org/doc/>

11. **Matplotlib.** (s.d.). *Matplotlib documentation.* <https://matplotlib.org/stable/index.html>

12. **NumPy.** (s.d.). *NumPy documentation.* <https://numpy.org/doc/stable/>

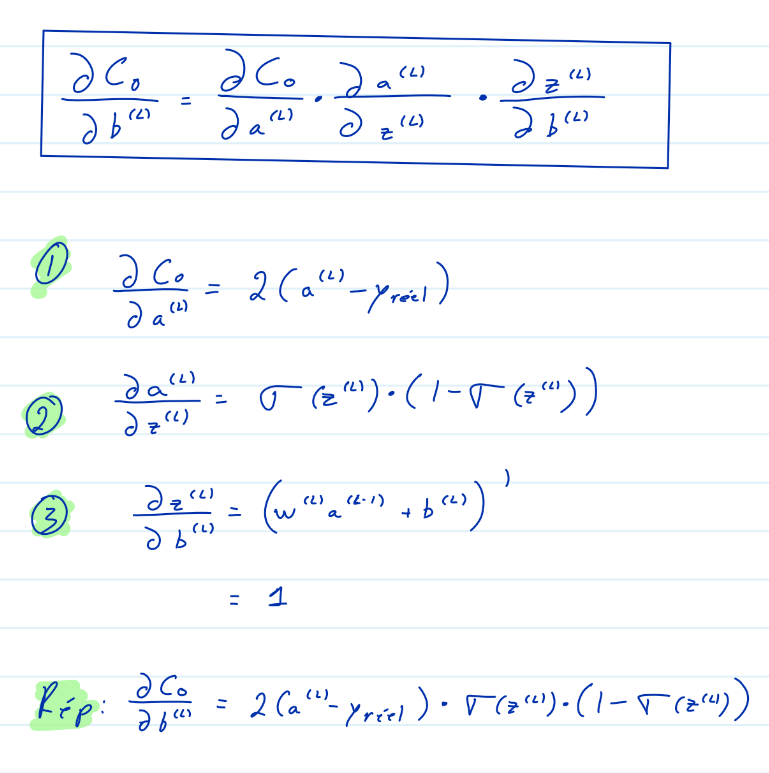
13. **Nielsen, M. A.** (s.d.). Chapter 2: How the backpropagation algorithm works. In *Neural networks and deep learning.* <http://neuralnetworksanddeeplearning.com/chap2.html>

14. **Soulière, A.** (1er février 2025). *Enrichissement IA partie1* [Vidéo]. Youtube. <https://www.youtube.com/watch?v=VZkYBUUiw5Q>

15. **Soulière, A.** (1er février 2025). *Enrichissement IA partie2* [Vidéo]. Youtube. <https://www.youtube.com/watch?v=erX3om_1Gh4>

Je reconnais avoir utilisé ChatGPT (<https://chat.openai.com/>) le 17 mars 2025 et le 27 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>