

OCR : Rapport de projet

Alexis Cognet

Alexandre Posta
Quentin Gillet

Flavien Geoffray

November 2022

Table des matières

1	Introduction	3
2	Solver	3
2.1	Manipulation du Sudoku	3
2.2	Gestion des erreurs	3
2.3	Résolveur	4
2.4	Sauvegarde du résultat	4
3	Prétraitement	4
3.1	Sauvegarde de l'image	4
3.2	Rotation Manuelle	5
3.3	Filtre Grayscale	5
3.4	Filtre des Contrastes	5
3.5	Réduction de Bruit (Filtre Median)	6
3.6	Binarisation (Filtre à seuil Adaptatif)	7
3.7	Filtre d'inversion	8
3.8	Filtre Sobel	9
3.9	Resultat Final	9
4	Ségmentation	10
4.1	Transformation d'Hough	10
4.2	Réduction des Lignes	11
4.3	Reconnaissance des Cases	11
4.4	Découpage et Chargement des Cases	12
5	Réseau de neurones	12
5.1	Structure en couches	13
5.2	Foward Propagation	13
5.3	Back Propagation	14
5.4	Entrainement du réseau	14
5.5	Sauvegarde des poids et des biais	14
5.6	Datasets	14
6	Ressentis et Organisation	15
6.1	Resumé des Avancements	15
6.2	Difficultés	15
6.3	Points Positifs	15

6.4 Conclusion	16
--------------------------	----

1 Introduction

À mis parcours de ce projet, ce rapport presente tout les avancements techniques que nous avons réalisé. Ce groupe a été formé assez tôt dans l'année, grâce à cela, nous avons réussi à commencer de travailler.

Tableau de répartition				
Matière	Alexandre	Alexis	Quentin	Flavien
Solver				X
Prétraitement	X		X	
Ségmentation	X	X	X	
IA		X		X

2 Solver

2.1 Manipulation du Sudoku

Tout d'abord, pour arriver à résoudre une grille de sudoku à l'aide d'un programme, il a fallu trouver un moyen de représenter le fichier d'entrée, qui contiens le sudoku à résoudre et qui est rédigé dans un format spécial, en une donnée manipulable facilement par la machine.

Pour ce faire, le moyen le plus simple était de transformer le fichier, contenant neuf lignes de neuf caractères, en un tableau à une dimension, de longueur 81.

2.2 Gestion des erreurs

Pour que le résolveur de sudoku soit complet et fonctionne correctement, il était important de mettre en place une gestion d'erreur, qui teste si le fichier d'entrée est correct et correspond à un format de sudoku soluble. Une fois que nous avons le tableau à une dimension contenant les

81 caractères du sudoku, vérifier les erreurs n'est pas compliquer. L'algorithme teste si les éléments du tableau sont bien des chiffres ou des « cases vides ». Si ce n'est pas le cas, une erreur est renvoyée.

Un teste des lignes, des colonnes et des « box » est aussi réalisé pour vérifier que le sudoku est bien correct.

2.3 Résolveur

L'algorithme de résolution est assez simple, une fois que la gestion des erreurs a été faite. L'algorithme utilise le « Backtracking » qui est une méthode dite « brute force » qui teste toute les possibilités cases par cases en récursion. Il retourne en arrière s'il rencontre une erreur, pour tester une autre valeur et ainsi trouver la solution.

2.4 Sauvegarde du résultat

Une fois le tableau complété avec les bonnes valeurs. Le résultat est sauvegardé sous la même forme que le fichier d'entrée. 9 lignes de 9 caractères. Cette fois les 9 caractères sont des chiffres.

3 Prétraitement

Le prétraitement de l'image va permettre deux choses. Premièrement, le découpage de l'image pourra être effectué dans la mesure où la grille du sudoku sera facilement détectable. Ensuite, il va permettre à l'intelligence artificielle de différencier chaque chiffre de la grille.

3.1 Sauvegarde de l'image

Pour sauvegarder l'image dans un fichier, nous avons utilisé le format bitmap très répandu pour les fichiers d'images pixelisés. Cette sauvegarde nous a été très utile pour le débogage de nos programmes de prétraitement puisque qu'il permet d'afficher à quoi ressemble notre sudoku après chaque étape qu'on lui applique.

3.2 Rotation Manuelle

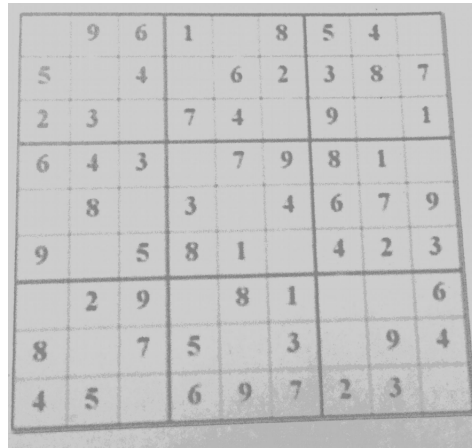
La rotation manuelle de l'image a été assez réalisée grâce à une fonction incluse dans la bibliothèque SDL. L'angle utilisé sera donné en degrés.

3.3 Filtre Grayscale

Pour débiter le traitement de l'image, nous avons premièrement appliqué le filtre grayscale. Pour chaque pixel de l'image la fonction utilisée est :

$$f(P) = 0.3 * Pr + 0.59 * Pg + 0.11 * Pb$$

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



	9	6	1		8	5	4	
5		4		6	2	3	8	7
2	3		7	4		9		1
6	4	3		7	9	8	1	
	8		3		4	6	7	9
9		5	8	1		4	2	3
	2	9		8	1			6
8		7	5		3		9	4
4	5		6	9	7	2	3	

FIGURE 1 – Image 1 et 6 après application du filtre grayscale

3.4 Filtre des Contrastes

Nous avons ensuite décidé d'utiliser le filtre de contraste. Ce filtre permet tout simplement de réduire la différence entre les parties les plus contrastées et les moins contrastées de l'image.

La formule, qui est répétée 10 fois pour chaque pixel de l'image, est la suivante :

$$\forall P \in [i * (255/10), (i + 1) * (255/10)] \quad f(P) = (i + 1) * (255/10)$$

Avec i : index de 0 à 9

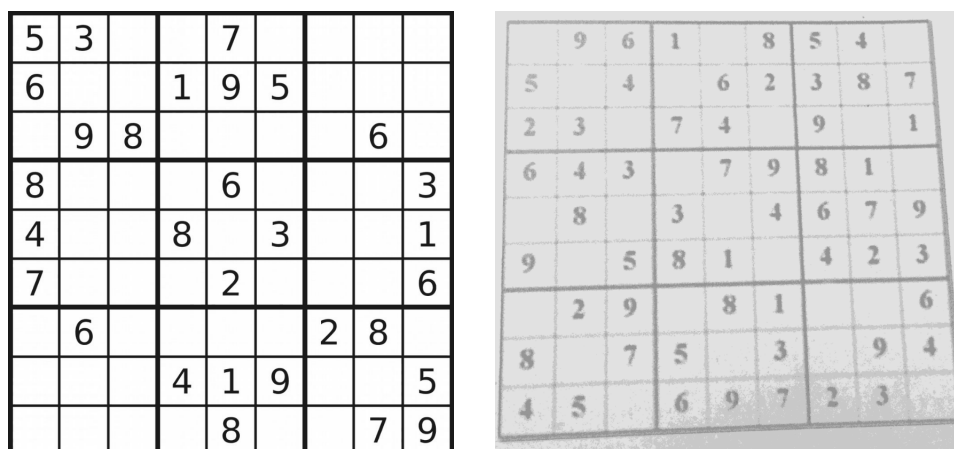


FIGURE 2 – Image 1 et 6 après application du filtre de contraste

3.5 Réduction de Bruit (Filtre Median)

Nous avons réduit le bruit de l'image grâce au filtre médian. Ce filtre permet d'éliminer les contours et les imperfections des formes présente sur l'image. Le protocole est le suivant pour chaque pixel de l'image :

- Créer une matrice 3x3 avec les pixels adjacents -
- Créer une liste triée à partir des valeurs de cette matrice -
- Définir la nouvelle valeur du pixel comme la médiane de la liste -

Exemple :

$$\begin{pmatrix} 90 & 49 & 45 \\ 90 & 68 & 68 \\ 84 & 78 & 58 \end{pmatrix}$$

↓

$$[45, 49, 58, 68, 68, 78, 84, 90, 90]$$

↓

$$f(P) = 68$$

Le majeur problème de ce programme est que les pixels situés sur les extrémités ne possèdent pas exactement 8 pixels adjacents, la matrice n'est donc pas complète.

Pour palier à cela, nous avons décidé de ne pas prendre en compte les pixels aux extrémités supposant que la marge d'erreur sur l'image finale sera minime.

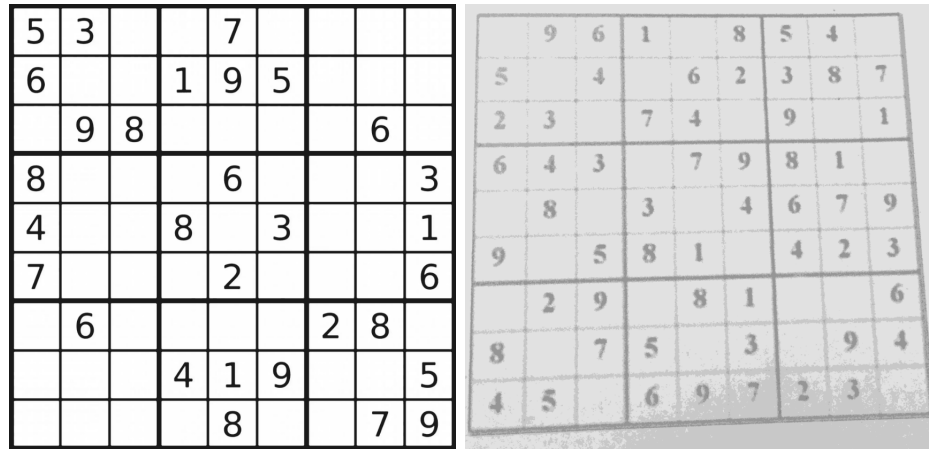


FIGURE 3 – Image 1 et 6 après application du filtre médian

3.6 Binarisation (Filtre à seuil Adaptatif)

Pour obtenir une image traitable, il faut que les chiffres et la grille du sudoku soit facilement repérable. Pour ce faire, il faudrait dans l'idéal que les parties intéressantes de l'image soient blanches et le reste noir. La binarisation de l'image intervient donc ici. Pour chaque pixel, on compare son intensité à une variable, si l'intensité est supérieure, on met la valeur du pixel à 255, sinon 0.

Cette variable a été préalablement calculée à l'aide de la méthode d'Otsu. Cette méthode va calculer la variance minimale entre les pixels de l'image supposés dans deux plans différents (premier plan/arrière-plan). Elle détermine ainsi la valeur de la variable avec la plus grande efficacité de binarisation selon une image donnée.

La formule se présente ainsi :

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

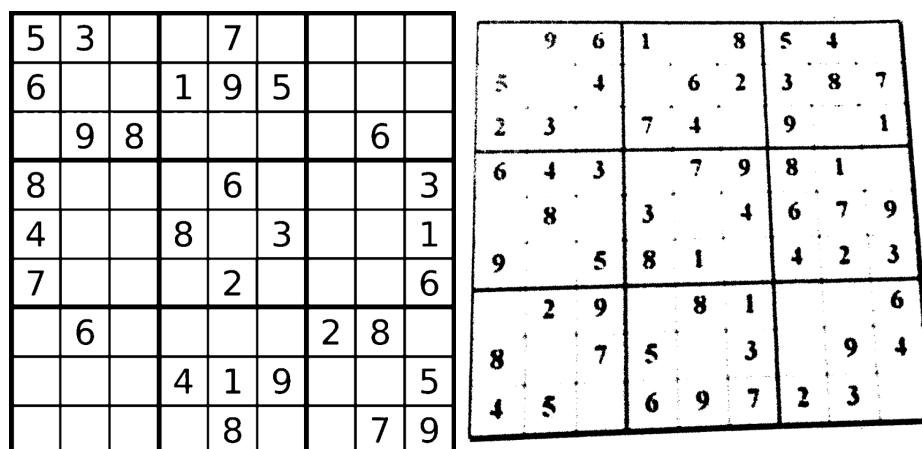


FIGURE 4 – Image 1 et 6 après application de la binarisation

3.7 Filtre d'inversion

Pour détecter les zones importantes, il est plus efficace de traiter ces zones quand elles sont blanches plutôt que noir. Comme on peut le remarquer ici, ce n'est pas le cas. On prend alors l'inverse de l'image grâce à cette formule :

$$f(P) = 255 - P$$

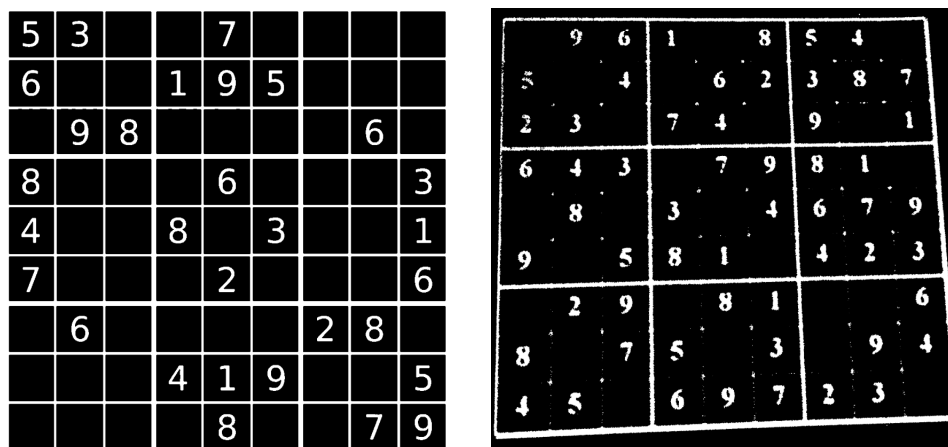


FIGURE 5 – Image 1 et 6 après application du filtre d'inversion

3.8 Filtre Sobel

Ce dernier filtre mis en place permet de détecter les contours des éléments de l'image. Il utilise une convolution entre des matrices et les axes x, y pour finalement calculer le gradient du pixel :

$$f(P) = \sqrt{G_x^2 + G_y^2}$$

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * P \quad \text{et} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * P$$

Bien sûr, les pixels avec une valeur assez élevée sont mis à 255, les autres à 0.

3.9 Resultat Final

Voici finalement l'image après le traitement en entier :

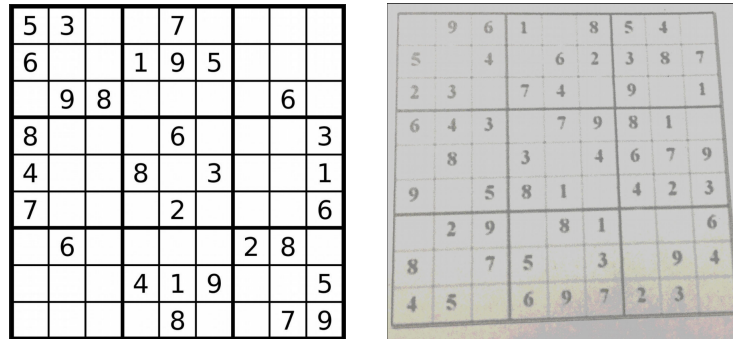


FIGURE 6 – Image 1 et 6 AVANT application de l'ensemble des prétraitements

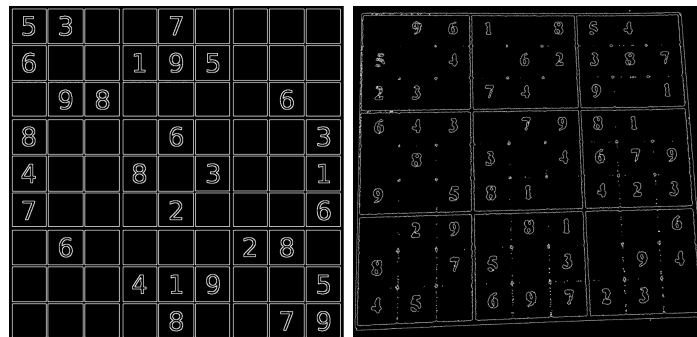


FIGURE 7 – Image 1 et 6 APRES application de l'ensemble des prétraitements

4 Ségmentation

Après avoir fini le traitement d'image, il ne reste plus qu'à récupérer chaque case de la grille pour les donner à l'intelligence artificielle.

4.1 Transformation d'Hough

C'est une technique de reconnaissance de formes qui est utilisée dans le traitement d'image.

C'est la technique que nous avons utilisée pour permettre à notre programme de faire de la reconnaissance de lignes. C'est une étape indispensable pour résoudre un sudoku, car si nous n'avons pas les cases de ce sudoku, il devient impossible de le résoudre.

Donc le principe de la transformée de Hough est de passer d'une représentation scalaire à une représentation polaire. Cette méthode va permettre de pouvoir détecter des lignes même si elles sont incomplètes ou partiellement obstruées.

Pour que l'algorithme puisse être efficace il nous faut un moyen de garder seulement le contour des éléments sinon le nombre de lignes serait énorme et l'algorithme très lent pour des résultats difficilement exploitable. On a donc utilisé pour cela, le filtre de Sobel. Une fois que nous avons des bords bien définis. Nous parcourons l'image et à chaque pixel d'un bord (pixel blanc), on calcule « rho » pour chaque « theta » allant de 90 degrés à -90 degrés.

$$Rho = x * \cos(theta) + y * \sin(theta)$$

Après, nous allons incrémenter un accumulateur de « rho » « theta » de 1. Nous avons donc un accumulateur qui contient le nombre de fois qu'un pixel a été parcouru en coordonnée polaire. Ce qui va nous permettre à l'aide d'un seuil défini de récupérer les coordonnées d'une ligne qui nous retransformons en coordonnée scalaire. Nous obtenons donc un x et y de fin et de début qui va nous permettre de tracer des lignes sur notre image en fonction des lignes de l'image. Ces lignes nous permettent ensuite de faire le découpage de cette même image.

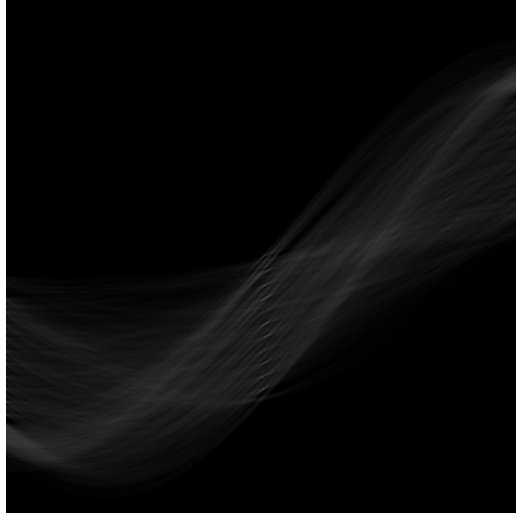


FIGURE 8 – Accumulateur en coordonnée polaire

4.2 Réduction des Lignes

Pour que la suite de notre programme soit la plus efficace et la plus précise, il faut réduire toutes les lignes superflues de l'image. Pour cela, nous avons créé un algorithme qui, pour chaque ligne adjacente, créer une ligne résultante moyenne :

$$f(L1, L2)_{x1} = (L1_{x1} + L2_{x1})/2$$

$$f(L1, L2)_{x2} = (L1_{y1} + L2_{y1})/2$$

$$f(L1, L2)_{y1} = (L1_{x2} + L2_{x2})/2$$

$$f(L1, L2)_{y2} = (L1_{y2} + L2_{y2})/2$$

4.3 Reconnaissance des Cases

Après avoir réduit considérablement le nombre de lignes, il faut maintenant détecter chaque case du sudoku. Pour ce faire, nous avons utilisé une fonction récursive déterminant tout les carrés que forment les lignes. Nous avons en même temps trier les carrés avec les meilleures dimensions.

Dans un premier temps, on prend une ligne au hasard. Puis on détecte toutes les intersections qu'elle a avec les autres lignes de l'image :

$$(x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4) \neq 0 \Rightarrow \exists intersection$$

On calcule ainsi le (x,y) de l'intersection des deux lignes.

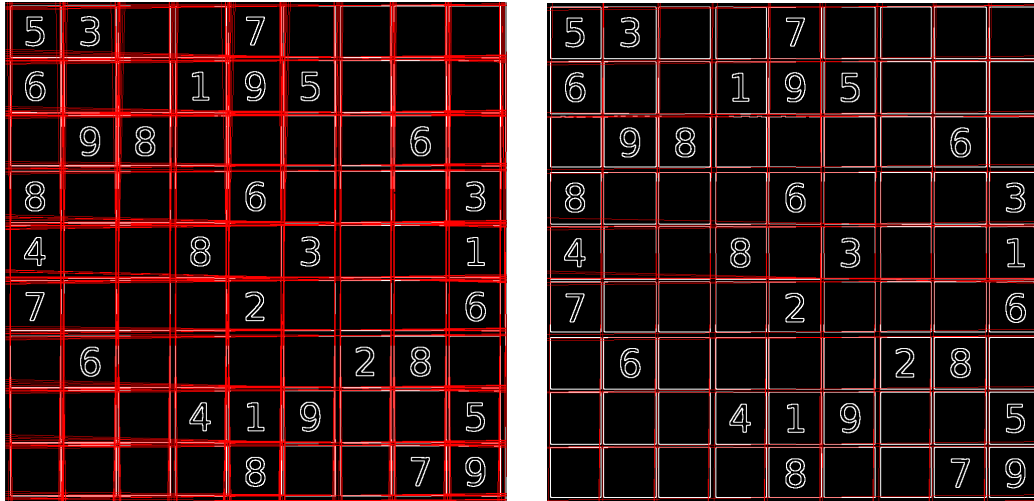


FIGURE 9 – AVANT et APRES l'application de l'algorithme de réduction de ligne

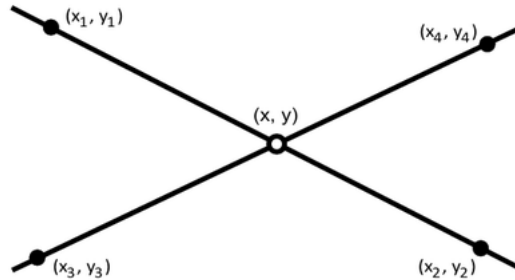


FIGURE 10 – intersection de deux droites

On relance ensuite le programme dans chaque intersection de la ligne. On relance ce programme 4 fois pour obtenir les 4 coins du carré. Les carrés sont testés avant d'être retenus. De plus, puisque que ce ne sont pas des carrés parfait, une marge d'erreur est prise en compte sur les valeurs des côté et des angles.

4.4 Découpage et Chargement des Cases

Finalement, après avoir récupérer les quatres coins de chaque carré de la grille dans un tableau, il ne nous reste plus qu'à enregistrer les valeurs des pixels dans un tableau traitable par l'intelligence artificielle.

5 Réseau de neurones

Inspirés par le cerveau humain, les réseaux de neurones artificiels constituent un sous ensemble de l'apprentissage machine. Ils sont au cœur des algorithmes de deep learning.

5.1 Structure en couches

Les réseaux de neurones artificiels sont constitués de différentes couches de nœud (ou neurone artificiel), contenant une couche en entrée, une ou plusieurs couches cachées et une couche en sortie.

Chaque nœud, ou neurone artificiel, se connecte à un autre et possède un poids et un biais seuil. Dans notre cas nous avons plusieurs réseaux de neurones à implémenter, un qui puisse résoudre le XOR et un qui puisse reconnaître un chiffre depuis une image.

Le principe est le même dans les deux cas, seule le nombre de neurones par couches change.

Pour le cas du XOR, le réseau comprend 2 neurones en entrées (0 ou 1 pour les deux), une couche cachée de 2 neurones, et une couche de 1 neurones en sorties (0 ou 1).

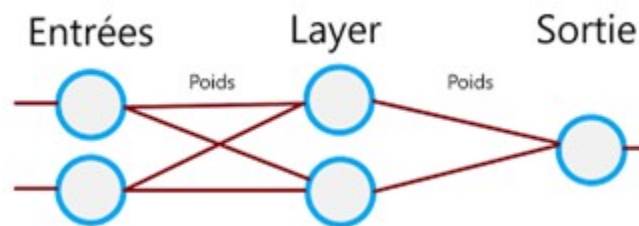


FIGURE 11 – Exemple de réseau de neurones

Pour le cas de la reconnaissance de chiffres, le nombre de nœuds dans la couche d'entrée correspondra au nombre de pixels d'une image standardisé (28*28 pixels), et la couche de sortie aura 9 nœuds (pour les 9 chiffres).

5.2 Forward Propagation

Pour déterminer une valeur de sortie en fonction des valeurs dans les neurones d'entrées, il a fallu implémenter l'algorithme de Forward propagation. Cet algorithme permet de trouver la valeur des nœuds de la prochaine couche en fonction de la couche précédente ainsi que des poids et des biais de la couche de sortie.

La Forward propagation pour le XOR détermine les valeurs sur la couche cachée avec la couche d'entrée puis de la couche de sortie avec la couche cachée. Elle utilise aussi les poids et les biais de chaque nœud.

Pour déterminer ces valeurs, l'algorithme utilise aussi une fonction d'activation : pour notre réseau, nous utilisons la fonction sigmoïde :

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

5.3 Back Propagation

La Forward Propagation renvoie une valeur en sortie qui ne correspondent pas à la véritable valeur engendrée par les entrées. Par exemple, si pour le XOR, nous avons en entrée 0 et 1, nous devrions avoir 1 en sortie. Dès lors pour approcher cette valeur il faut modifier les poids et les biais utiliser dans la Forward Propagation. C'est ce que fait la Back Propagation avec la dérivée de la fonction sigmoïde.

5.4 Entraînement du réseau

Pour entraîner le modèle il faut enchaîner plusieurs fois la Forward Propagation et la Back Propagation. Pour la première étape, les poids et les biais sont générés aléatoirement. Ensuite pour les étapes d'après les poids et les biais sont ajuster pour s'approcher de la véritable valeur de sortie.

Plus le réseau est entraîné, plus le résultat en sortie est précis.

5.5 Sauvegarde des poids et des biais

Une fois que le modèle a été entraîné, les poids et les biais de chaque neurone permettent de trouver un résultat précis. Il faut donc sauvegarder ces poids et ces biais pour que la machine soit capable de trouver le bon résultat en fonction des entrées.

Dans notre cas, après avoir entraîné le réseau, un dossier dans lequel un fichier par couche sauvegarde les poids et les biais de la couche.

5.6 Datasets

Pour entraîner nos différents réseaux de neurones, il est nécessaire d'avoir des datasets, c'est dataset permettent à l'IA d'être performante lorsqu'on lui présente une image ou une valeur en entrée et ainsi reconnaître la bonne sortie.

Pour le XOR, le data set est assez simple, puisque les valeurs en entrées sont (0 1), (1 0), (1 1) ou (0 0).

Pour le reseau de la reconnaissance de chiffre, nous avons decider d'utiliser la base de données MNIST qui est une base de données de chiffres très souvent utilisée dans les reseaux de neurones artificiels. Nous complèterons cette base de données avec des chiffres découpés dans des images de sudoku.

6 Ressentis et Organisation

6.1 Résumé des Avancements

Pour résumer l'avancé que nous put atteindre en ces 2 mois de travail sur ocr, nous pensons avoir remplis les demandes concernant la première soutenance. Nous avons fini le Solver, le prétraitement et la détection des cases de la grille. Nous avons aussi créé le XOR qui nous permet d'avoir les bases de l'intelligence artificielle. Néanmoins, certaines optimisations vont sûrement être nécessaires pour pouvoir atteindre le meilleur taux de réussite possible.

6.2 Difficultés

Les difficultés rencontrées jusqu'à présent sur ce projet ont été principalement liées à deux sujets.

La première, bien entendu, étant le code. Nous sommes confrontés à un tout nouveau langage de programmation qui nous était, pour la majeure partie, inconnu. La difficulté était ici principalement à cause de l'introduction des pointers qui sont réelle découverte pour nous.

Ensuite, la deuxième difficulté a été l'organisation générale, étant intrinsèquement quelque chose que tout le monde redoute. Il nous a fallu faire des efforts pour arriver à un espace de travail convenant à tout le monde.

6.3 Points Positifs

Malgré toutes les difficultés citées précédemment, nous trouvons que ce début de projet c'est très bien passé en général. Les difficultés rencontrées sur le code nous ont permis de progresser significativement. Et même si

nous n'avions pas l'habitude de travailler ensemble chacun a réussi à s'adapter au mieux pour réaliser ce projet.

6.4 Conclusion

Pour conclure, le bilan de cette première soutenance est plutôt positif, chaque membre de l'équipe a pu apporter ses compétences pour avancer et mettre en oeuvre les étapes clés. Le planning et les différentes parties à réaliser ont été respectées pour cette première soutenance. Bien que nous ayons rencontré quelques difficultés, le projet avance bien et nous sommes fiers du travail accompli. Il nous reste maintenant à poursuivre le travail, notamment au niveau de la partie réseau de neurone, puis créer une interface graphique pour regrouper chaque partie du projet.