1 Réseau de neurones

Le réseau de neurones est la partie du projet permettant d'identifier les chiffres écrits dans les cases envoyées par le programme de traitement d'images.
Après cela le résolveur utilisera ces informations pour visualiser la grille de sudoku et la résoudre.

Un réseau de neurones, c'est un système utilisé dans l'intelligence artificielle qui a pour but d'apprendre une fonction (ici celle de la reconnaissance des caractères du sudoku) à l'aide d'exemples. Dans un premier temps, nous allons exécuter une phase d'apprentissage visant à calibrer le réseau de neurones à l'aide d'exemples dont nous connaissons le résultat voulu (ici des blocs d'images où l'on connait déjà les chiffres représentés). Une marge d'erreur servira à calibrer le réseau. Après un certain nombre d'exemples, notre réseau aura des bonnes réponses dans la majorité des cas. La sortie sera la probabilité qu'une telle image représente tel chiffre. Le chiffre retenu sera celui avec la plus grande probabilité.

1.1 La conception

1.1.1 Le neurone

Ceci est le but du réseau de neurones, mais de quoi est-il composé ?
Les composants principaux d'un réseau de neurones sont ,comme son nom l'indique, ces neurones, plus précisément, j'ai utilisé des neurones fonctionnant avec la fonction sigmoïde.
Un neurone va fonctionner comme ceci :

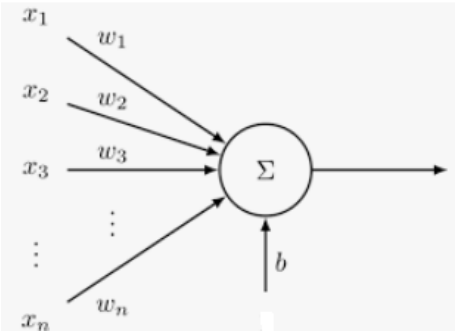

FIGURE 1 – Neurones sigmoïdes

Le neurone va recevoir plusieurs entrées, chacunes accompagnées d'un poids, puis il va calculer la somme des produits des entrées avec leurs poids respectifs et va y soustraire son biais.

$$\sum_j w_j x_j - b$$

FIGURE 2 – Somme calculé dans un neurone

Après avoir obtenu cette somme, le neurone va l'utiliser dans la fonction sigmoide :

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

FIGURE 3 – Fonction sigmoïde

où z est la somme calculée par le neurone et qui sera son output.

1.1.2 le réseau

Le réseau est composé de plusieurs couches contenant un ou plusieurs neurones. En premier il y aura une couche d'entrées où seront mises, comme son nom l'indique, les entrées pour lesquelles on veut entrainer la fonction. Les neurones de cette couche n'utilisent pas la fonction sigmoïde, ne calculent pas de somme et ne renvoient que les entrées.
Il y a ensuite une ou plusieurs couches cachées servant à transformer les entrées grâce au poids et biais mentionnés plus tôt afin d'avoir une sortie à la dernière couche. Enfin il y aura la couche de sortie qui, comme son nom l'indique, retournera un ou plusieurs résultats. Les entrées de tout neurone de toutes couches (sauf la première) sont les sorties des neurones de la couche précédente.
Voici un exemple de réseaux servant à apprendre la fonction XOR :

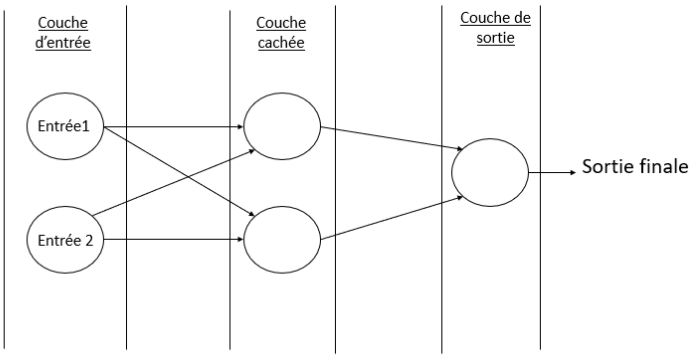

FIGURE 4 – Réseau de la fonction XOR

Aussi un réseau a un taux d'apprentissage qui lui servira lors de son entrainement.
Avant son entrainement, tout les poids et biais des neurones seront générés aléatoirement.

1.2 Entrainement

Maintenant que nous vous avons présenté la structure d'un réseau de neurones, nous allons vous présenter comment on l'entraine.

Pour cela j'ai utilisé l'algorithme de descente de gradient stochastique (ou SGD) qui utilise une paire d'entrées et de sortie attendue (le résultat que l'on est censé avoir avec la fonction qu'on entraine) afin de modifier les poids des neurones du réseau et d'obtenir des résultats plus proches de la sortie attendue.

1.2.1 Première partie : choix des entrées

C'est pour cela que la première chose que l'on a faite c'est fournir au réseau une base de données composée d'une liste d'entrée et d'une liste de sortie attendue.
Ainsi pour que nous puissions affiné les résultats de notre réseau,nous mettons en place plusieurs périodes d'entrainement (sous la forme d'une boucle for allant jusqu'a l'entier n pour faire n périodes)dans lesquelles on prendra une paire d'entrée et de sortie dans la base de données. Afin que cette paire soit choisie aléatoirement et que toutes les paires soient utilisées nous utilisons la fonction shuffle qui mélange une liste d'index donnée ,les index seront utilisés pour choisir les entrées et sorties dans la base de données.

1.2.2 Deuxième partie : Avancement vers l'avant et rétropropagation

Ensuite nous faisons avancer les valeurs des inputs choisis dans le réseau, jusqu'a atteindre la couche de sortie. Une fois la couche de sortie atteinte nous calculons les changements à faire dans les poids de la ou les couches cachées et de la couche de sortie en utilisant la fonction de l'erreur quadratique moyenne qui calcule la moyenne de la différence quadratique entre la sortie obtenue par le modèle et celle attendue. Ainsi, dans le cadre des neurones de la couche de sortie, leur erreur sera la différence entre leur sortie et la sortie attendue et le changement dans ces poids sera égal à la multiplication entre cette erreur et le résultat de la dérivée de la fonction sigmoïde avec pour paramètre la somme calculée dans le neurone. On fait la même chose avec la ou les couches cachées sauf que cette fois, pour chaque neurone leur erreur sera égale à la somme des erreurs des neurones de sorties(avec les bons poids appliqués à ceux-ci). Le changement en poids dans ces neurones est calculé de la même manière que dans la couche de sortie (l'erreur multipliée par la dérivée de la fonction sigmoïde avec la somme du neurone comme paramètre).
Ensuite nous appliquons ces changements à tous les poids des neurones de sortie, pour ça nous ajoutons à leur biais la multiplication du changement de poids et du taux d'apprentissage et nous ajoutons à chacun de ses poids la multiplication entre le taux d'apprentissage, l'entrée associée au poids et le changement en poids. Puis nous faisons de même avec la ou les couches cachées.

1.2.3 Troisième partie : sauvegarde des poids

Après avoir fini toutes les périodes d'entrainements, il faut sauvegarder les poids et biais modifiés afin de pouvoir les récupérer et recréer le réseau sans avoir besoin de le réentrainer.

Cette partie n'est pas terminée, mais l'idée est de marquer tous les poids des neurones dans un fichier texte en utilisant une boucle, les poids des différentes couches cachées et de la couche de sortie étant dans des fichiers différents.

1.3 Application au XOR

Pour montrer l'efficacité du SGD, je l'ai appliqué au réseau de la fonction XOR montré plus tôt.

Pour rappel voici la table de vérité du XOR :

x	0	0	1	1
y	0	1	0	1
z	0	1	1	0

Table 1 : la fonction XOR.

Après 100000 périodes d'entrainements nous avons ces résultats :

x	0	0	1	1
y	0	1	0	1
z	0.013011	0.988874	0.988875	0.011457

Comme vous pouvez le voir, le SGD a permit au réseau de la fonction XOR d'avoir des résultats extrêmements proches de ceux voulus.

1.4 Application à l'OCR

Pour appliquer cela à l'OCR nous allons cette fois faire le réseau de la fonction qui doit identifier le nombre inscrit dans les cases données par le traitement d'image. Le réseau aura les mêmes 3 couches que celui du XOR (une d'entrée, une cachée et une de sortie) mais cette fois ce réseau aura 784 neurones d'entrées, car les cases données seront de format 28x28 (784 pixels), 100 neurones dans la couche cachée et 10 dans la couche de sorties, car il peut y'avoir 10 sorties différentes (les chiffres de 0 à 9, 0 représentant une case vide).

1.5 Objectifs

Pour la prochaine soutenance, je prévois de pouvoir sauvegarder les poids et les biais du réseau que j'aurais entrainé, de lire la base de données MNIST afin d'entrainer le réseau pour l'ocr et possiblement d'améliorer l'apprentissage du réseau en utilisant la fonction d'entropie croisée.