# SUDOKU SOLVER: SOUTENANCE 1

## Présentation de l'équipe + répartition des rôles:

Je m'appelle Charlotte et j'ai 19 ans. Contrairement au projet de l'année dernière, je n'ai pas vraiment de précédentes expériences de création d'application ou de programmation (sauf cours). Ce projet m'a tout l'air d'être plus technique en termes de code et de réalisation. En terminale, j'avais réalisé un article sur le Deep Learning qui utilise justement les réseaux de neurones sur WIx (création de site internet). Je suis donc très curieuse de pouvoir travailler sur un projet de Sudoku qui permet de créer un réseau de neurones.

Dans ce projet, j'ai réalisé la partie de résolution du sudoku pour débuter : la création du Solver. Pour le réaliser, il y a plusieurs fonctions que j'ai écrites. La première est la création du programme de résolution, puis la prise du fichier et enfin la création du fichier final.

Je m'appelle Kelyan, j'ai 18 ans. Je n'ai jamais réalisé de projets de ce genre et cela m'a l'air plutôt complexe, surtout que je n'ai pas beaucoup d'expérience dans le traitement d'image et que les algorithmes de résolution de problèmes ne sont pas mon fort.

Pour cette première soutenance, j'ai réalisé la partie traitement d'image, et donc un algorithme permettant de transformer une image en quelque chose qui sera exploitable par nos programmes et cela se fait en quatre étapes :

- Le chargement de l'image
- La suppression des couleurs
- La rotation de l'image par un angle donné par l'utilisateur
- et sauvegarde de l'image modifiée

Je m'appelle Evan, j'ai 18 ans. Je n'ai jamais réalisé de projets similaires mais j'ai déjà pu m'entraîner à la réalisation d'interface graphique notamment dans le cas du projet de première année ce qui sera utile pour la réalisation de ce programme.

Pour la première soutenance j'ai réalisé le découpage des images ainsi que les croquis pour l'interface graphique du programme final.

Je m'appelle Luke Goboyan, j'ai 19 ans. Bien que j'ai réalisé des tâches comme celles impliquées pour la construction de ce projet, je n'ai jamais travaillé sur un projet aussi complet en termes de contenu et fonctionnalités. Cela m'a donné l'occasion d'exercer mes connaissances acquises, ainsi que d'apprendre de nouvelles choses.

Pour la première soutenance, je me suis occupé de concevoir le réseau de neurones qui reconnaît une porte XOR.

# 1) Solver de Sudoku

## I-Solver

Pour commencer, j'ai regardé beaucoup de sites concernant la réalisation de programmes pouvant résoudre la grille de sudoku. La méthode qui ressort le plus pour réaliser ce programme est ce que l'on appelle le « BackTracking » également appelé en français « Retour sur trace ». Dans notre cas, il consiste à retourner en arrière si aucun chiffre n'est valide à cette case. Pour ce faire, le programme est récursif ce qui permet ce retour en arrière. On peut le comparer au système des arbres que nous apprenons où nous parcourons (généralement) vers le fils gauche puis on remonte au père pour atteindre le fils droit. Ce principe est presque le même que le « BackTracking » si le chiffre n'est pas valide on retourne à la case précédente (vide initialement) pour essayer un autre chiffre etc…

Une fois l'idée claire et précise de la fonction mère que je dois créer, j'ai réalisé 3 fonctions secondaires.

La première fonction permet de vérifier si le chiffre passé en paramètre est déjà sur la ligne. S'il est déjà présent, la fonction renvoie « 0 » équivalent à False et « 1 » équivalent à True.

La seconde fonction permet de vérifier si le chiffre passé en paramètre est déjà sur la colonne. De même, s'il est déjà présent la fonction renvoie « 0 » équivalent à False et « 1 » équivalent à True.

La troisième fonction permet de vérifier si le chiffre passé en paramètre est déjà sur le carré. De la même manière, s'il est déjà présent la fonction renvoie « 0 » équivalent à False et « 1 » équivalent à True.

A présent, la création de la fonction récursive principale « IsValid ». Elle a plusieurs tests à faire. Le premier est de vérifier si toute la grille a été parcouru si c'est le cas la fonction renvoie le tableau résolu. Ensuite, si la case n'est pas vide cela veut qu'aucun chiffre doit être ajouté alors on incrémente la position. Enfin, une boucle permet d'ajouter un chiffre. On appelle ensuite récursivement la fonction

IsValid dans un « if « qui vérifie que la prochaine position est valide également sinon la case retourne à zéro car aucun chiffre est possible. C'est ici que le retour en arrière se passe.

## II-Ouverture du fichier : open_my_file

Dans cette deuxième partie, j'ai appris à utiliser la lecture d'un fichier. On peut y trouver des ressemblances avec le python même si ce n'est pas exactement pareil. Tout d'abord pour la simple et bonne raison que le C utilise un pointeur de type FILE.

Pour cette fonction, la grande partie du travail est de comprendre comment les fonctions pour ouvrir un fichier fonctionne. Il a donc fallu encore de nombreuses recherches pour pouvoir l'écrire. De plus, j'ai ajouté une erreur qui permet de savoir si le fichier a eu un problème pour s'ouvrir. Finalement, j'ai utilisé la fonction « fgetc » qui prend notre variable du pointeur comme paramètre et renvoie un entier correspondant au caractère du code ASCII. J'ajoute par la suite le caractère à mon tableau d'entier qui représente la grille de sudoku.

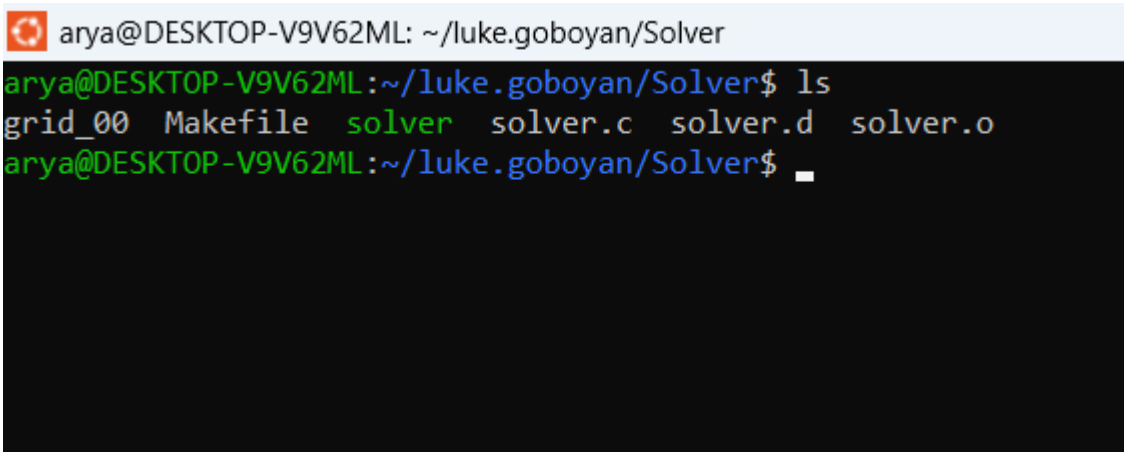
## III-Création du fichier final : my_result_file

Pour cette dernière fonction, j'ai également dû apprendre à utiliser les fonctions afin de créer et écrire dans un fichier. Une nouvelle fois, la partie la plus longue a été la recherche et la compréhension des fonctions et de choisir la meilleure solution.

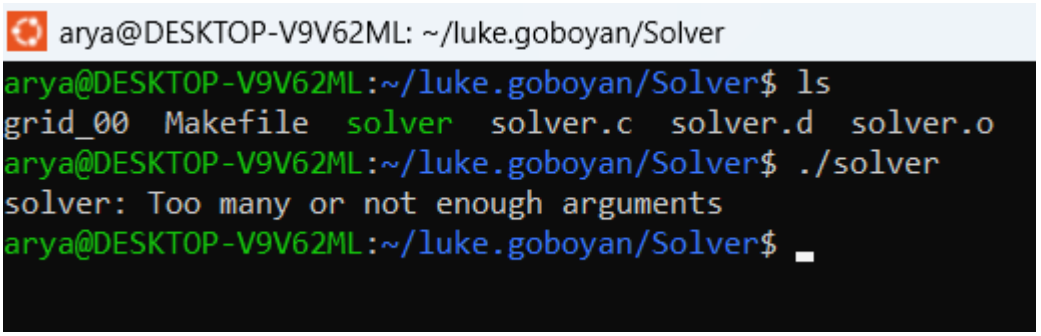
Dans premier temps, j'ai créé mon fichier avec le «.result » pour identifier la grille de sudoku résolu. Ensuite, je parcours mon tableau d'entier résolu, après avoir appelé IsValid, puis j'écris dans mon fichier le caractère correspondant grâce à la fonction « fputc ». Elle permet d'écrire le caractère souhaité tant qu'il est placé en tant que paramètre de cette fonction.

Voici le fonctionnement :

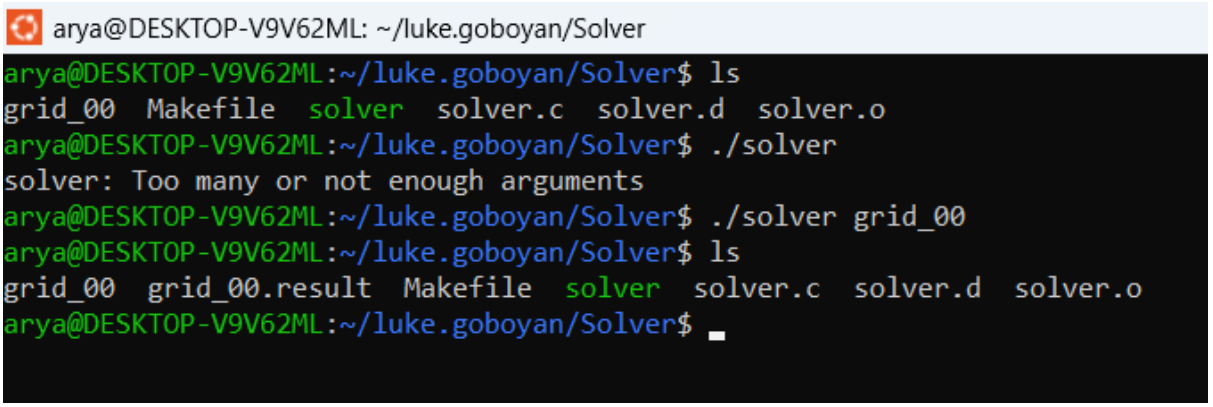
En tapant "make" on obtient le résultat ci dessus




S'il n'y a pas le nom du fichier il y a une erreur :




Sinon on obtient le fichier ".result"




Enfin le fichier ".result" contient la grille résolu

```
arya@DESKTOP-V9V62ML:~/luke.goboyan/Solver$ ls
grid_00  grid_00.result  Makefile  solver  solver.c  solver.d  solver.o
arya@DESKTOP-V9V62ML:~/luke.goboyan/Solver$ cat grid_00.result
127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952
arya@DESKTOP-V9V62ML:~/luke.goboyan/Solver$
```

## 2) Traitement de l'image

### I-Chargement de l'image :

Pour manipuler l'image, j'utilise la librairie SDL2 de C qui me permet notamment de charger et créer des images ainsi que de les manipuler directement au niveau des pixels.

La première étape du programme consiste a charger la bonne image.Le programme peut retrouver cette image simplement en suivant le chemin indiqué par l'utilisateur.

Une fois l'image trouvée il faut la sauvegarder sous forme de SDL_Surface, un type de donnée de SDL qui me permet de la manipuler à souhait pour cela, il suffit d'utiliser la fonction IMG_Load().

### II-Suppression des couleurs :

Il faut ensuite convertir l'image en noir et blanc,pour cela il faut:

- parcourir les pixels de l'image
- récupérer les valeurs R ,G et B du pixel ainsi que son format
- calculer une valeur moyenne en fonction de R G et B
- appliquer cette valeurs en tant que nouvelle valeur RGB du pixel

On peut récupérer les pixels d'une surface avec Surface -> pixels. On aura alors un tableau de pixels.

### III-Rotation de l’image :

Parfois, il se peut que l’image soit penchée, l’utilisateur peut donc choisir de la tourner d’un angle qu’il aura choisi, pour cela il faut:

- Calculer l’inverse de la matrice de rotation (trouvée avec le cos et le sin de l’angle)
- Créer une Surface destination.
- Pour chaque pixels de la surface destination trouver son pixel correspondant sur l’image source (grâce à l’inverse de la matrice de rotation)
- appliquer la valeur du pixel trouvé au pixel.
- sauvegarder la surface destination.

On a au préalable rempli la surface destination avec la couleur blanche.

### III-Sauvegarde de l’image modifiée:

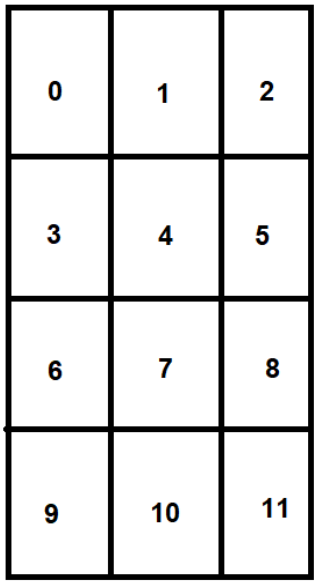
Une fois que l’on a notre SDL_Surface qui contient toute les modifications, il faut la sauvegarder, pour cela j’ai choisi (arbitrairement) le format png Pour cela j'utilise la fonction IMG_Save en faisant attention a bien spécifier la surface que je veux sauvegarder et à choisir un nom se terminant en .png

## 3) Interface graphique

### I-Découpage d’images:

Il y a 2 cas pour le découpage d’images. Soit le découpage est régulier et connu (de longueur et de hauteur fixe avec chacun des découpages collé aux autres), soit le découpage est décomposé en plusieurs rectangles donnés en entrée.

Dans le premier cas, il est nécessaire de créer les rectangles de découpage manuellement. Ainsi, on divise la longueur et la largeur de l’image initiale par la longueur et la largeur donnés pour obtenir le nombre de case dans les deux directions. Dans notre cas, nous utilisons un système d’index pour obtenir la case désirée. On parcourt le tableau créé par la détermination des rectangles en fonction de celui-ci. Le découpage de l’image ressemble donc au schéma suivant :

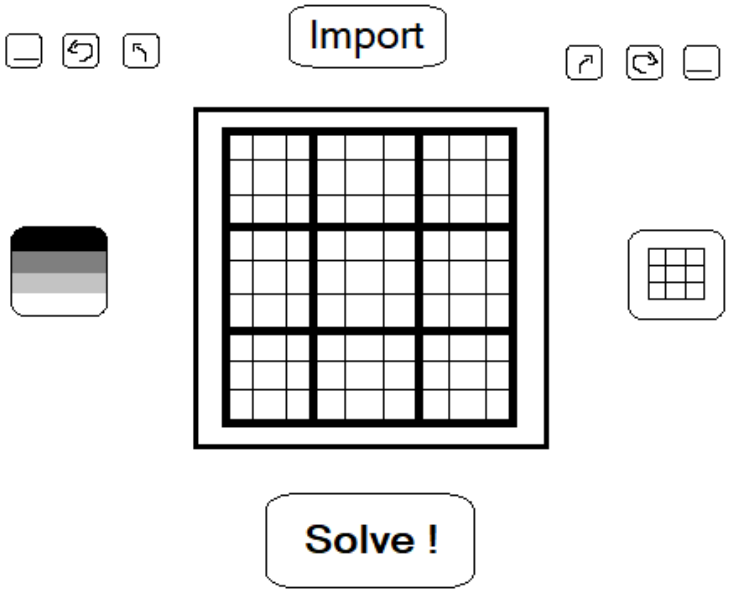



Dans le second cas, il suffit de créer une nouvelle image de destination aux dimensions du rectangle donné et de copier la partie correspondante sur l'image initiale.

## II-Croquis d'interface graphique:

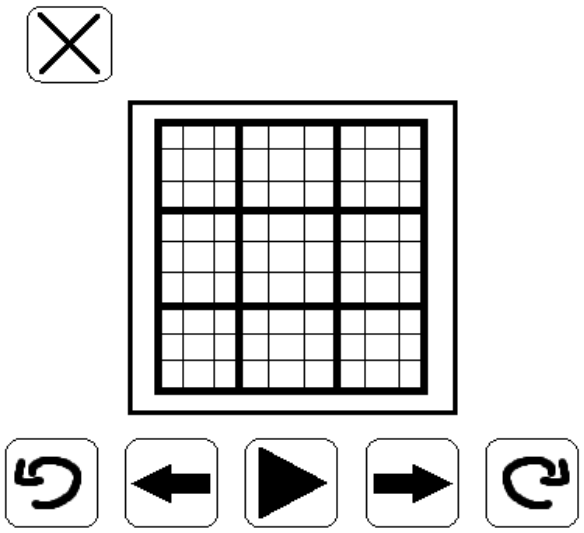
L'objectif de l'interface graphique est de rendre l'utilisation du programme plus simple et intuitive. Ainsi, la majorité des fonctions permettant de modifier l'image se trouveront sur la même page (de façon similaire aux logiciels de visualisation d'images).

Voici ce à quoi le l'interface graphique devrait ressembler (cette version peut être changées suivant les contraintes ou ajouts dans le projet):



En haut de l'écran se trouve le bouton permettant d'importer des images. C'est le premier bouton sur lequel cliquera l'utilisateur, il est donc mis en évidence. On retrouve tout autour de l'image de boutons permettant d'appliquer des modifications à l'image. Dans les deux coins supérieurs se trouvent les boutons de rotation, pour appliquer une rotation légère, moyenne où en précisant directement le degré. En cliquant à droite de l'image, il est possible d'alterner entre l'affichage de l'image importée et la grille générée par le programme. De même, à gauche, l'affichage des couleurs peut être sélectionné ou non. Enfin, le dernier bouton mis en évidence permet de résoudre le sudoku.

Lorsque l'utilisateur clique sur le bouton de résolution, la fenêtre change pour afficher les options de visualisation. La page ressemble donc désormais à ceci :



Il est possible d'aller directement à la fin de la résolution ou au début, d'avancer étape par étape dans l'évolution de l'algorithme ou de laisser l'animation se faire. Enfin, l'utilisateur peut quitter cette fenêtre à tout moment en cliquant sur la croix pour retourner à l'écran principal.

## 4) Réseau de neurone

### I-Structure d'un réseau de neurone:

En développement, un neurone est considéré comme un objet algorithmique qui prend un certain nombre de valeurs, par example n nombres numérotés $x_1$, $x_2$,... à $x_n$ , et renvoie un résultat unique. Il existe deux types de neurones: les perceptrons et les sigmoides.

Soit un neurone qui prend une liste de valeurs [$x_1$, $x_2$,...,$x_n$ ], et z résultat d'une équation:

$$z = (\sum_{k=1}^{n} x_k * w_k) + b$$

w est le poids et une valeur unique à chaque x qui représente son importance dans l'équation. b représente le biais du neurone, et un seuil que z-b doit dépasser pour que z soit supérieur ou égal à 0, dans le cas contraire il est inférieur. Le perceptron ne renvoie que deux résultats: 1 si z >= 0,  0 si z < 0. Il est le plus simple des deux modèles dans son fonctionnement et renvoie des résultats clairs, mais il pose un problème: si l'on change une des valeurs de poids ou de biais dans une tentative d'obtenir un résultat final différent de peu, la nature du changement brusque des résultats de perceptrons rend la manière dont le changement final est affecté de manière imprévisible.

Voici le résultat d'un neurone sigmoïde:

$s = 1/(1+e^{-z})$

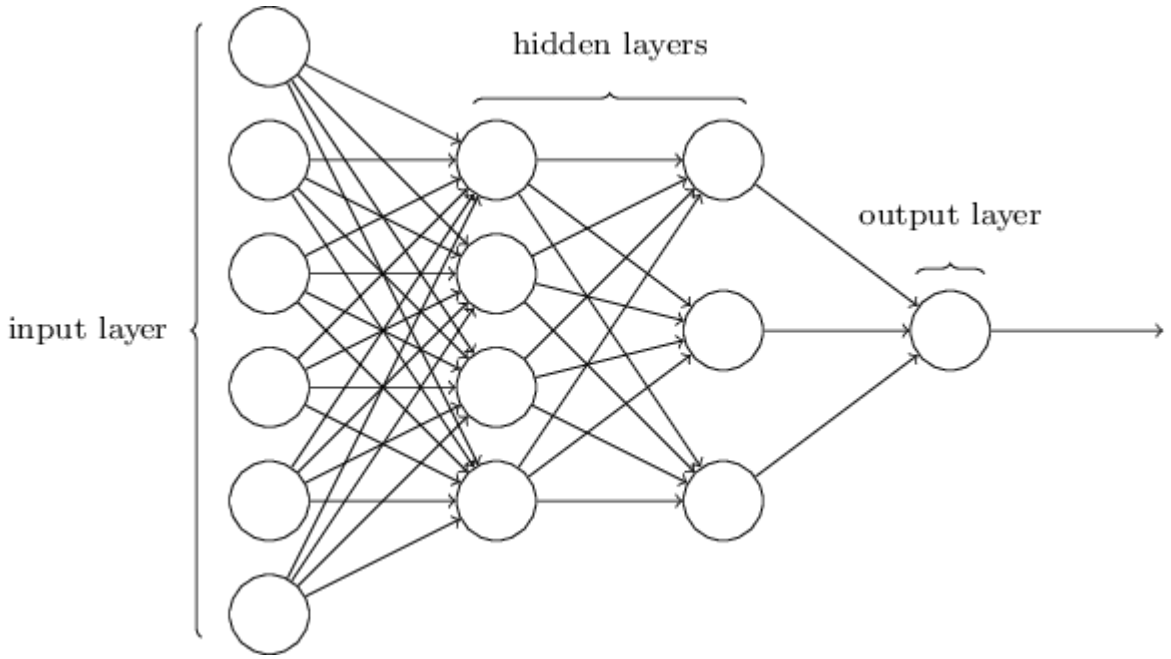
Sachant que z représente le même calcul, on remarque que le résultat n'est plus un 1 ou un 0, mais une fraction qui représente un nombre entre 0 et 1; de plus, si z = 0 alors s = ½ le milieu de l'intervalle, tandis que plus z s'éloigne de 0, plus il se rapproche de 1 si positif ou 0 si négatif. Si on regarde les limites:

$z \to +\infty \Rightarrow e^{-z} \to 0 \Rightarrow s \to 1$

$z \to -\infty \Rightarrow e^{-z} \to +\infty \Rightarrow s \to 0$

Ce fonctionnement permet à ce qu'un changement de valeur dans un réseau de neurones sigmoïdes entraîne un changement de même ampleur dans le résultat final.

Finalement, voici l'organisation d'un réseau de neurones complet:



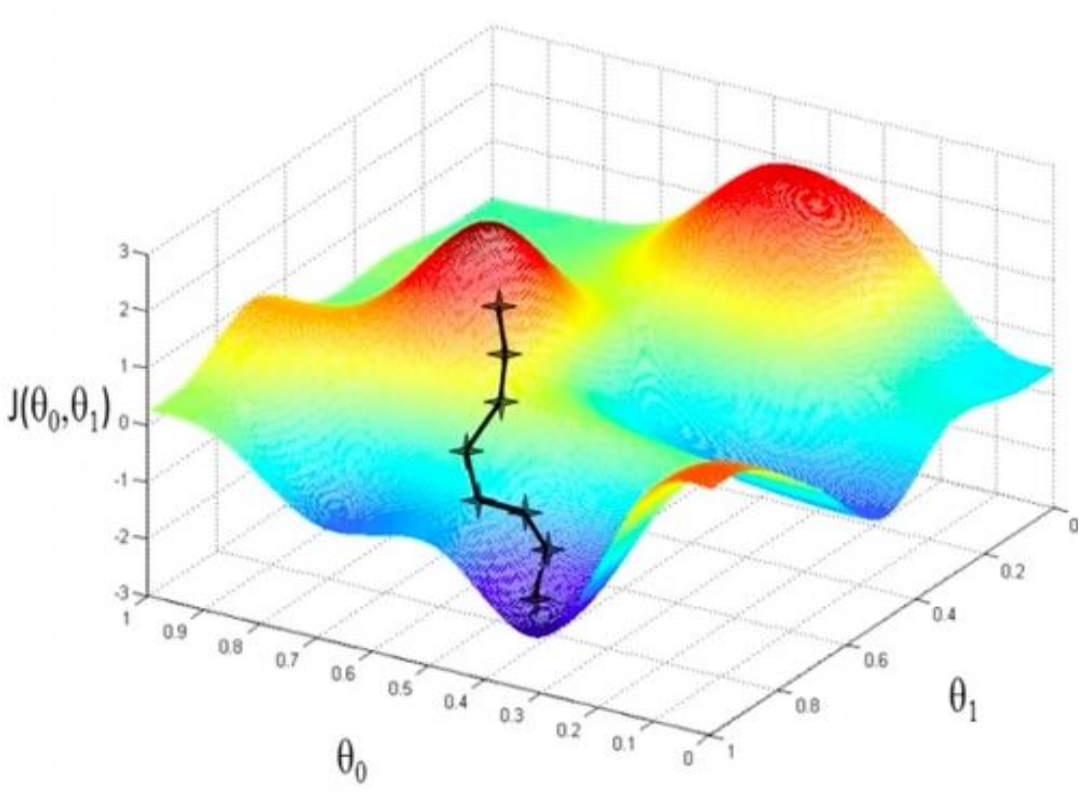

Les neurones sont répartis en plusieurs colonnes, dites "couches" se suivant de gauche à droite. La première couche, dite couche d'entrée, est composée de neurones trous prenant en argument les valeurs données manuellement au réseau.

Toutes les autres couches sauf la dernière sont dites “cachées”, et chacun des neurones prend en argument les résultats de tous les neurones de la couche précédente. Enfin, la couche finale renvoie le résultat attendu du réseau de neurones. Il faut noter que bien que les tailles et chaque couches et types de neurones utilisés sont décidés “manuellement”, les valeurs de poids et biais sont décidées par apprentissage automatique.

## II-Apprentissage automatique d’un réseau:

Afin d’adapter le réseau au modèle le plus correct possible pour de meilleurs résultats, il faut une série d’exemples composés d’un objet en entrée ainsi que du résultat attendu qu’il doit donner. Le processus d'apprentissage consiste à initier les valeurs des neurones de manière aléatoire, puis passer chaque exemple dans le réseau de neurones, comparer son résultat avec le résultat voulu et changer les valeurs de biais et poids afin de “corriger” le réseau.

Le programme utilise une méthode appelée la rétropropagation qui consiste à calculer une fonction de perte P, prenant en argument (w,b) et représente la différence entre résultat voulu et obtenu, puis cherche comment rapprocher le résultat de la fonction de 0. La notion de descente graduelle décrit le processus de calculer les résultats de P(w,b) pour tout w et b d’une rangée de valeurs disponibles, et les changer en celles qui font que P donne le résultat le plus petit; on continue le processus jusqu’à que les w et b actuels donnent le résultat minimum visible.



## III-Réseau de neurone de porte XOR:

Pour cette soutenance, il a fallu créer un réseau de neurones pour l'apprentissage de la porte logique XOR. Le réseau actuel est composé de deux couches de neurones: première couche de deux neurones, dernière avec un seul neurone. On utilise des matrices pour stocker les poids et les biais des différents neurones. On déclare une valeur **lr** (assigné 0,1 ici) qui détermine à quel point l'apprentissage affecte les valeurs du réseau, dont le fonctionnement sera vu après. Les exemples d'entraînement sont composés de l'ensemble d'entrées {{0,0},{1,0},{0,1},{1,1}} et l'ensemble de résultats attendus respectifs {0,1,1,0}. L'algorithme de rétropropagation utilisé est la *stochastic gradient descent (SGD)*, qui change les poids selon une pair (valeur d'entrée, valeur de retour attendue). Après avoir mis des valeurs aléatoires pour compléter le réseau, on démarre le processus d'entraînement que l'on fait récurer 10 000 fois avant de montrer les valeurs finales et finir le programme.

Tout d'abord il faut mélanger l'ordre des exemples à chaque récursion, car cela est requis pour que la *SGD* soit plus efficace. Pour chacun des exemples, on calcule le résultat donné par le réseau, et on l'affiche sur la console avec le résultat attendu pour observer l'évolution du réseau à chaque récursion. Après cela commence la rétropropagation, où pour le neurone final on calcule d'abord une valeur **errorOutput** comme la différence entre le résultat attendu et celui obtenu, puis on la multiplie avec la fonction dérivée de sigmoide, qui est x(1-x) avec x remplacé par le résultat obtenu, ce qui nous donne **deltaOutput**. La méthode est la même pour chacun des deux neurones de départ, excepté que **errorHidden** est égal à la somme de tous les poids du neurone final fois **deltaOutput** , et on obtient **deltaHidden** en multipliant **deltaOutput** par la dérivée sigmoïde, x égalant le résultat du noeud spécifique. Maintenant que l'on a **deltaOutput** et **deltaHidden**, représentés ici comme **delta** pour les neurones de leur couches respectives:

- on modifie les biais de base en leur additionnant **delta** * **lr**;

- on modifie chacun des poids en leur additionnant **input1 * delta** * **lr + input2 * delta** * **lr** (avec input1 et input2 les deux valeurs d'entrée pris en exemple).

Finalement, avant de fermer le programme, on affiche sur la console de commande tous les poids et biais finaux pour montrer l'efficacité du programme, et comparer les résultats.

## Etat d’avancement du projet et conclusion

Cette partie récapitule ce qui à été fait dans toutes les phases du projet, ainsi que les étapes qui restent à compléter.

En premier est l’algorithme de solver, qui est complètement terminé et opérationel pour les grilles en 9x9, puisque le programme est déjà capable de partir d’un fichier texte d’un sudoku incomplet et renvoyer un fichier .result du sudoku résolu.

Au niveau du traitement de texte, le programme peut déjà charger l’image, la rendre en noir et blanc et la tourner; le découpage est terminé à 50%.

Le GUI de l’interface graphique est 10% complet, tourner l’image manuellement n’est pas entièrement possible.

Le réseau de neurone identifiant la porte XOR est fait, il peut de plus sauvegarder ses valeurs de poids et biais dans un fichier consacré bien que le programme ne soit pas encore fait pour les charger à l’entraînement. Un jeu d’image composé de cinq exemplaire de tous les numéros de 1 à 9 a été sauvegardé dans le dossier Git.