Génie Logiciel
Projet GL – *Réseau de neurones*

Contexte

Le but de ce projet est de séparer des populations distinctes de points. Nous prendrons comme exemple initial deux populations : les points rouges et les points bleus, chacune dessinant une spirale. Pour atteindre cet objectif, vous allez réaliser un réseau de neurones afin de lui faire "apprendre" à distinguer ces populations. Ce réseau de neurones devra ensuite être capable de dire à quelle population appartient un point quelconque.

1 Populations

Une population est un ensemble de points définis par leurs coordonnées x et y et leurs couleurs. Les différents ensembles de population doivent pouvoir être définis par l'utilisateur. Cependant, ce projet doit inclure par défaut un exemple comprenant deux populations de points (rouge et bleu) définissant des spirales, voir image ci-dessous :

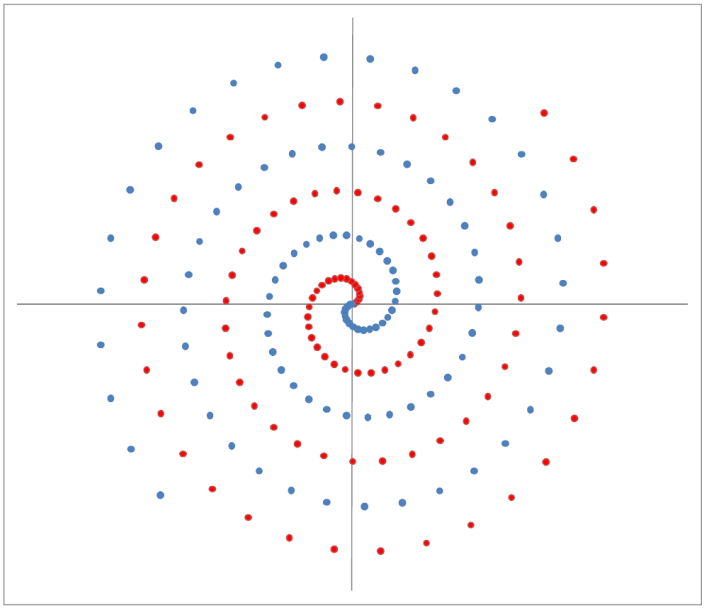

FIGURE 1 – Spirales utilisées comme jeu de données d'apprentissage initial.

Ces deux spirales sont les résultats de calculs de courbes paramétriques. Ainsi, la spirale bleue est obtenue en utilisant l'équation suivante :

$$\begin{cases} x = t.cos(t) \\ y = t.sin(t) \end{cases}$$

Quant à la spirale rouge, elle est l'opposée de la bleue et est définie de la manière suivante :

$$\begin{cases} x = -t.cos(t) \\ y = -t.sin(t) \end{cases}$$

Ces courbes évoluent en suivant les valeurs discrètes de t allant de 0 à $+\infty$ par pas de $0,2$. La figure 1 met en évidence la symétrie des spirales.

2 Réseau de neurones

Le neurone utilisé pour ce projet est un neurone informatique appelé *perceptron*. Il s'agit d'un opérateur mathématique simple qui, selon des valeurs en entrée et une fonction de transfert, calcul une valeur de sortie.

La figure 2 décrit un neurone formel. Les valeurs d'entrée sont nommées x_i pour i allant de 1 à n entrées et la sortie est nommée y. La transmission des valeurs en entrée à un neurone se fait par l'intermédiaire de synapses. Chaque synapse

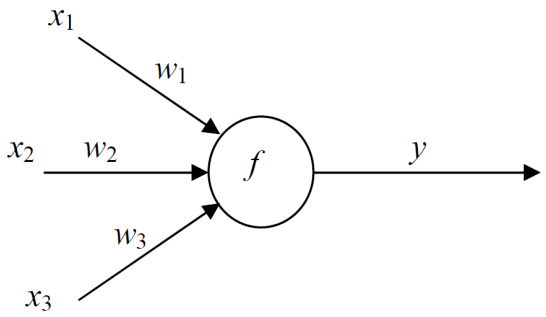

FIGURE 2 – Neurone de base avec ses entrées, leurs différents poids synaptiques, la fonction de transfert et la sortie obtenue.

est caractérisée par un poids synaptique appelé w_i. Enfin, f définit la fonction de transfert permettant à un neurone de calculer la valeur y en fonction des x_i et w_i. Ainsi, la valeur de sortie y est obtenue en utilisant l'équation suivante :

$$y = f(\sum_i x_i.w_i)$$

La fonction de transfert f à utiliser est la fonction tangente hyperbolique :

$$th(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Appliquée à la figure 2, le résultat est le suivant :

$$S = x_1.w_1 + x_2.w_2 + x_3.w_3$$

Et la sortie est :

$$y = f(S) = \frac{e^S - e^{-S}}{e^S + e^{-S}}$$

Enfin, cette valeur de sortie y peut être soit directement interprétée, soit utilisée comme valeur d'entrée pour un autre neurone.

3 Réseau de neurones multicouches

Un réseau de neurones multicouches consiste en une succession de couche de neurones reliées les unes à la suite des autres (Fig. 3). Les valeurs d'un problème sont transmises à la couche d'entrée, avec un neurone par valeur à traiter. Cette couche est une exception puisque qu'elle ne dispose pas de poids synaptiques sur ses entrées et ses sorties sont égales aux entrées. Cela signifie que si les valeurs $x_i = [x, y]$ lui sont transmises en entrée alors les valeurs de sortie obtenues sont $y_i = [x, y]$.

Après la couche d'entrée se trouvent plusieurs couches dites "cachées". Chacune de ces couches comporte un certain nombre de neurones variable. Déterminer le nombre de ces couches cachées et leurs tailles est délicat, des dimensions basses ne permettront pas nécessairement d'avoir un apprentissage correct du réseau. A contrario, un nombre trop élevé n'est pas forcément nécessaire. De plus, plus il y a de couches et de neurones plus les temps de calculs sont longs. Il vous faudra trouver de bonnes valeurs pour l'exemple des spirales, mais il faudra également offrir à l'utilisateur la possibilité de personnaliser son réseau.

Pour finir, en fin de réseau se trouve la couche de sortie. Elle contient autant de neurones que de valeurs de sortie. Pour l'exemple des spirales, elle contiendra deux neurones. Cependant, un utilisateur doit pouvoir définir le nombre de sortie de son réseau, en supposant que la possibilité de créer sa propre base d'apprentissage lui soit donnée. Les valeurs obtenues à la sortie de cette couche correspondent aux probabilités d'appartenance à une classe. Exemple avec les spirales, si un point $[x, y] = [3, 4]$ est envoyé en entrée du réseau et que la sortie obtenue est $[bleu, rouge] = [0.3, 0.6]$, alors ce point à une probabilité 20 fois supérieur d'appartenir à la couleur rouge que la couleur bleue.

Chaque neurone d'une couche prend uniquement en entrée toutes les sorties des neurones de la couche précédente. Tout ce réseau est couramment appelé "*layer fully connected*" et intervient généralement en dernière étape dans les algorithmes de *Deep Learning*.

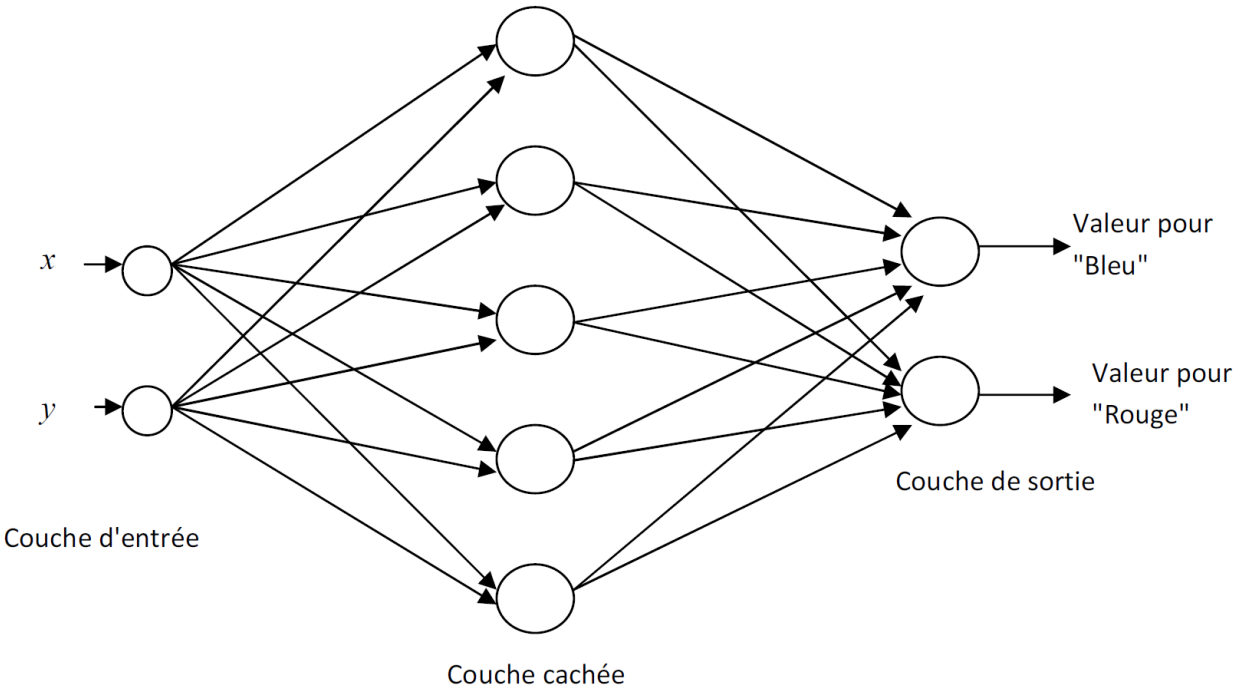

FIGURE 3 – Un réseau de neurones multicouches. Deux neurones dans la couche d'entrée, une couche cachée composée de cinq neurones et une couche de sortie avec deux neurones.

4 Propagation

Pour des valeurs $[x, y]$ données en entrée, la propagation du calcul se fait de manière séquentielle. Les valeurs passent par la couche d'entrée qui va envoyer les valeurs de sa sortie à tous les neurones de la première couche cachée. Cette première couche cachée va utiliser la fonction de transfert donnée précédemment pour calculer ses sorties et les envoyer à la prochaine couche. Ainsi de suite, jusqu'à arriver à la couche de sortie.

5 Algorithme d'apprentissage (Back-propagation)

La phase d'apprentissage consiste à indiquer au réseau l'erreur commise dans sa propagation et de le mettre à jour afin de corriger cela. Pour cela, il faudra modifier uniquement les valeurs des poids des différentes synapses du réseau. Ainsi, le réseau se trompera de moins en moins et convergera vers un état "stable" où il ne pourra plus progresser. Cela s'appelle de "l'apprentissage supervisé".

En se basant sur les spirales, à l'état initial, tous les poids du réseau sont initialisés avec des valeurs aléatoires (valeurs tirées sur un intervalle à définir). Puis une paire de valeur $[x, y]$, dont la réponse est connue (e.g. le point $[3, 1]$ est bleu donc la réponse attendue est $[1, 0]$), est propagée au réseau. À partir de l'erreur entre la réponse fournie et celle attendue, il faut corriger les poids synaptiques en utilisant l'algorithme de la **rétropropagation du gradient**. Les poids sont corrigés pour ces valeurs d'entrée, il faut alors recommencer pour de nouvelles valeurs.

6 Rétropropagation du gradient

Il existe de nombreuses démonstrations mathématiques de cet algorithme. Cependant, elles ne sont pas l'objectif de ce projet. C'est pourquoi, vous trouverez, ci-après, un pseudo-code d'une implémentation. Il s'agit d'une adaptation des éléments fournis à l'adresse suivante : `http://alp.developpez.com/tutoriels/intelligence-artificielle/reseaux-de-neurones/#LV`

```
Répéter
    Prendre un exemple (vecteur_x, vecteur_y) et calculer la sortie du
    réseau pour ce vecteur_x

    Pour tout neurone de sortie i // couche de sortie
        di = (1 - si * si)(yi - si)
    finPour

    Pour chaque couche de q - 1 à 1
        Pour chaque neurone i de la couche courante
            di = (1 - oi * oi) * Somme [pour k appartenant aux indices des neurones
                prenant en entrée la sortie du neurone i] de dk * w_ki
        finPour
    finPour

    Pour tout poids w_ij
        w_ij = w_ij + epsilon * di * x_ij // mise à jour des poids
    finPour
finRépéter
```

Les différents éléments de cet algorithme sont définis de la manière suivante :
- vecteur_x représente les entrées fournies au réseau.
- vecteur_y représente la sortie attendue.
- si représente la valeur calculée en sortie du neurone i de la couche de sortie.
- yi représente la valeur attendue en sortie du neurone i de la couche de sortie.
- oi représente la valeur calculée en sortie du neurone i d'une couche cachée.
- di représente un delta (différence) qui sera utilisée pour calculer la mise à jour effective des poids synaptiques du réseau.
- w_ij représente le poids synaptique entre les neurones i et j.
- x_ij représente la valeur transportée par la synapse entre les neurones i et j.

La valeur *epsilon* apparaissant dans la mise à jour des poids sert à pondérer (en réduisant) la considération de la correction d'erreur. Un grand *epsilon* permet une convergence rapide du réseau, mais vers un état final moins précis qu'avec un *epsilon* faible. Cependant un *epsilon* faible nécessitera un très grand nombre d'itérations avant de converger vers un état final stable. C'est pourquoi il faudra appliquer cet algorithme sur de nombreux exemples.

7 Phase d'apprentissage

Dans cette phase, le but est de faire évoluer les poids de manière à ce que le réseau soit capable de reconnaître et séparer les éléments (e.g. les spirales). Pour réaliser cela, il faut lui fournir des exemples de points tirés au hasard d'un jeu de donnée défini par l'utilisateur ou les spirales et appliquer l'algorithme de rétropropagation. L'apprentissage se termine lorsque le plus grand *di* atteint un seuil minimum (à définir). Cela signifie que les corrections apportées aux poids sont négligeables et qu'il n'est plus nécessaire de continuer d'apprendre. Il n'est pas possible de savoir combien d'exemples il faut présenter au réseau pour converger vers ce seuil. Il est même possible que le réseau n'apprenne plus (mise à jour des poids faibles) et que les réponses qu'il fournit ne soient pas forcément correctes ! (notion de minimum global).

8 Phase de généralisation

Une fois l'apprentissage fini, les poids sont donc figés et il est alors possible de voir le comportement du réseau pour une entrée quelconque. Il suffit de lui donner n'importe quel couple de valeurs $[x; y]$ et d'afficher le résultat. Ainsi, pour l'exemple des spirales, il est possible de voir le réseau les dessiner avec des gradients de couleurs. C'est l'intérêt de la généralisation, les données fournies au réseau sont des données sur lesquelles il n'a pas fait son apprentissage, qui lui sont étrangères !

9 Cahier des charges

En plus de tous les éléments cités précédemment, votre projet devra considérer les éléments suivant :
— Sauvegarder / charger des réseaux de neurones.
— Personnaliser un jeu de données, en choisissant le nombre de classe en sortie.
— Une interface graphique fonctionnelle et efficace.
— Enfin, beaucoup d'informations sont cachées dans les explications, lisez bien le sujet.

Pour ce qui est de la technique, le projet devra se faire en C en s'appuyant sur la bibliothèque SDL2 pour l'affichage, aucune autre bibliothèque ne sera acceptée (si ce n'est celle pour les tests). Votre projet devra contenir un `makefile` permettant de le compiler. Si vous souhaitez travailler sur Windows, alors uniquement une solution Visual Studio sera acceptée.