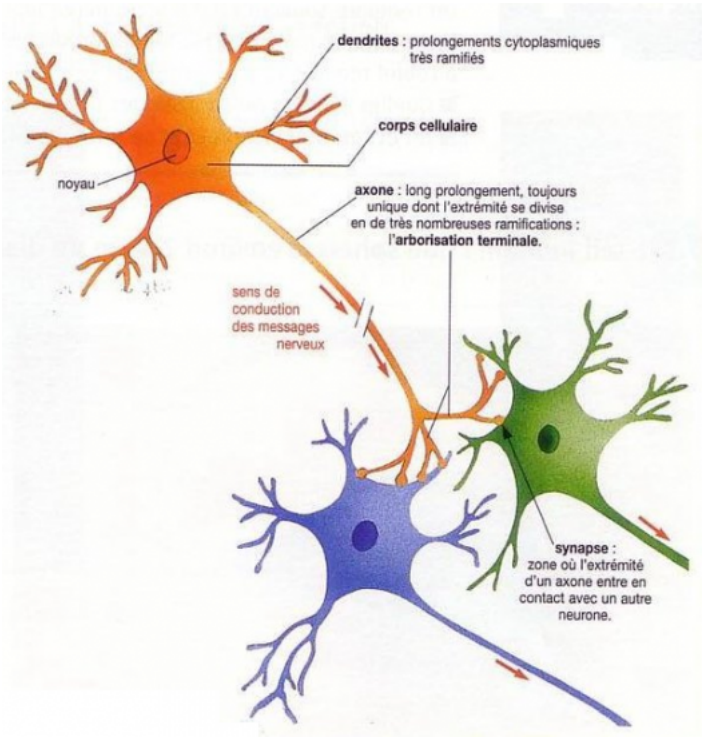
Table des matières

1 Introduction

Le but de ce travail de bachelor est d'étudier et de comparer le comportement de différentes techniques de Machine Learning basés sur la neuroévolution en analysant la faculté des algorithmes à apprendre à jouer à des jeux-vidéos.

La neuroévolution est une technique qui constiste à faire évoluer des réseaux de neurones artificiels afin qu'ils arrivent à effectuer une tâche.

Un réseau de neurones est un modèle, inspiré du fonctionnement du cerveau humain, qui va consister en un ensembles de neurones artificiels (aussi appelés perceptrons), disposés en couches qui vont communiquer en propageant une information. En effet les neurones de notre cerveau vont collecter les signaux en provenance de leurs dendrites, puis si les signaux sont assez fort envoyer une impulsion le long de leur axone vers les neurones suivant qui vont faire de même. A noter que les connexions entre deux neurones peuvent être plus ou moins fortes (le signal va donc se propager avec une intensité variable).

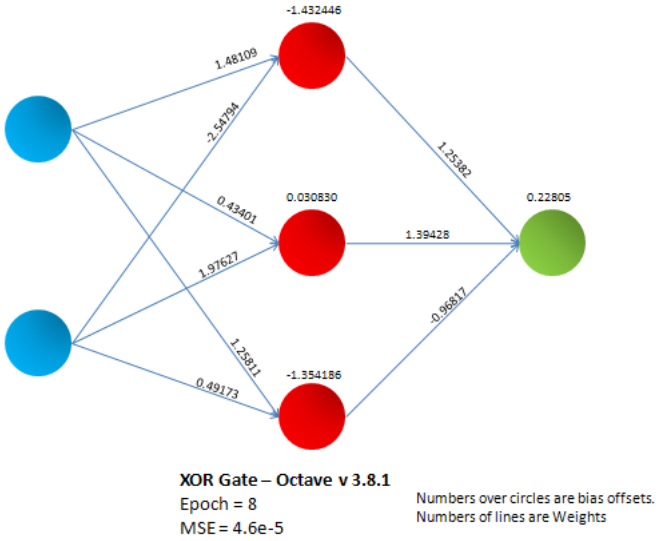


Les perceptrons quant à eux vont imiter (de manière simplifiée) ce comportement, chaque perceptron va prendre le signal envoyée par chacuns de ses voisins de la couche précédente, puis multiplier cette valeur par le poids le connexion et finalement faire le somme de toutes les valeurs pondérées et passer cette somme dans une fonction (dites fonction d'activation) qui va placer cette somme dans un interval (entre 0 et 1 par exemple) et renvoyer le resultat de cette fonction à tous ses voisins de la couche suivante qui vont faire de même.

D'un point de vue mathématique le comportement d'un perceptron peut être écrit comme :

$$o = f(\sum_{i=0}^{n} S_i * W_i) \tag{1}$$

Où
o est le signal qui va sortir du perceptron
f est la fonction d'activation
n est le nombre de voisins de la couche précédente
S le vecteur des signaux des voisins la couche précedente
W le vecteur des poids entre le perceptrons et ses voisins de la couche précédente



La méthode classique pour entrainer des réseaux de neurones est la rétropropagation du gradient (backpropagation). Cette technique constiste à comparer la sorties du réseaux à la sortie attendu pour une entrée donnée. L'avantage de cette technique est qu'elle va converger très rapidement vers le résultat attendu (à condition que les hyperparamètres du réseau soit adaptés au problème). Le désavantage est qu'il faut avoir à sa disposition un grand nombres d'entrées dont on connais la nature afin de pouvoir les comparer aux résultats du réseau. Par exemple il existe une base de données de chiffres écrit à la main avec leur valeur réelle sur `http://yann.lecun.com/exdb/mnist/` (70'000 entrées).

Cependant générer une telle base de données est un travail énorme, ainsi ces dernières années nous avons assisté à l'émergence de nouvelles technique ne nécessitant pas d'exemples

2 Architecture Réseau

Afin d'optimiser la vitesse de calcul, un système distribué à été mis en place. Celui-ci repose sur une architecture client-serveur classique où le client effectue les simulations et le serveur gère les génomes et

s'occupe de distruber de manière équitable le travail entre tous les clients.

2.1 Protcole

Le protocole à été établi à l'aide du langage protobuf qui permet d'avoir une définition claire ne dépendant pas des langages dans lesquels le protcole est ensuite implémenté.
Voici le fichier qui défini le protocole :

Protocole

```
syntax = "proto2";
option java_package = "me.pv.mg.protobuf";

enum MGMessages {
    MG_JOIN = 1;
    MG_JOIN_RESPONSE = 2;
    MG_COMPUTE_REQUEST = 3;
    MG_COMPUTE_RESPONSE = 4;
    MG_COMPUTE_RESULT = 5;
    MG_END = 6;
}

enum MGNetworkType {
    MG_MULTILAYER_PERCEPTRON = 1;
    MG_NEAT = 2;
}

message MGJoin {
    optional string pretty_name = 1;
    optional bool spectator = 2;
}

message MGJoinResponse {
    required bool accepted = 1;
    optional string reason = 2;
}

message MGComputeInfo {
    required string game = 1;
    required .MGNetworkType net_type = 3;
    required string net_metadata = 4;
}

message MGComputeRequest {
    required .MGComputeInfo compute_info = 1;
    required string genome = 2;
}
```

```
message MGComputeResponse {
    required bool can_do = 1;
}

message MGComputeResult {
    required float fitness = 1;
    optional uint32 time = 2;
}

message MGEnd {
    optional string message = 1;
}
```

Le protocole définit donc 6 types de messages :

- MG_JOIN, Envoyé par le client, c'est une demande à rejoindre le groupe de calcul. En paramètres sont donnés le nom du client et si il veut rejoindre en tant que specateur ou non (fonctionnalité expliquée dans la partie client).
- MG_JOIN_RESPONSE, Envoyé par le serveur, confirme ou infirme l'ajout au groupe du client. Il n'existe pour l'instant aucune raison pour le rejet d'un client.
- MG_COMPUTE_REQUEST, Envoyé par le serveur, demande au client de calculer le fitness d'un génome donné sur un jeu donné.
- MG_COMPUTE_RESPONSE, Envoyé par le client, indique au serveur si oui ou non le client est en mesure d'effectuer la simulation.
- MG_COMPUTE_RESULT, Envoyé par le client, donne le fitness obtenu par le génome donné dans le message MG_COMPUTE_REQUEST sur le jeu donné dans ce même message et le temps (ms) pris par le client pour effectuer la simulation.
- MG_END, Envoyé par n'importe quel entité, indique un désir de terminer la connexion.

2.2 Serveur

Comme écrit ci-dessus, le serveur à la lourde tâche de gérer tous les génomes et leurs fitness afin de mettre en oeuvre les algorithmes génétiques qui vont permettre l'évolution. En plus de cela il va également devoir servir les pages web servant à la gestion et à la surveillance du processus évolutif.
Afin de pouvoir effectuer toutes ces tâches, le serveur est constitué de modules node.js qui vont chacun gérer une partie de ce qui est lui est demandé.
L'architecture se présente ainsi :

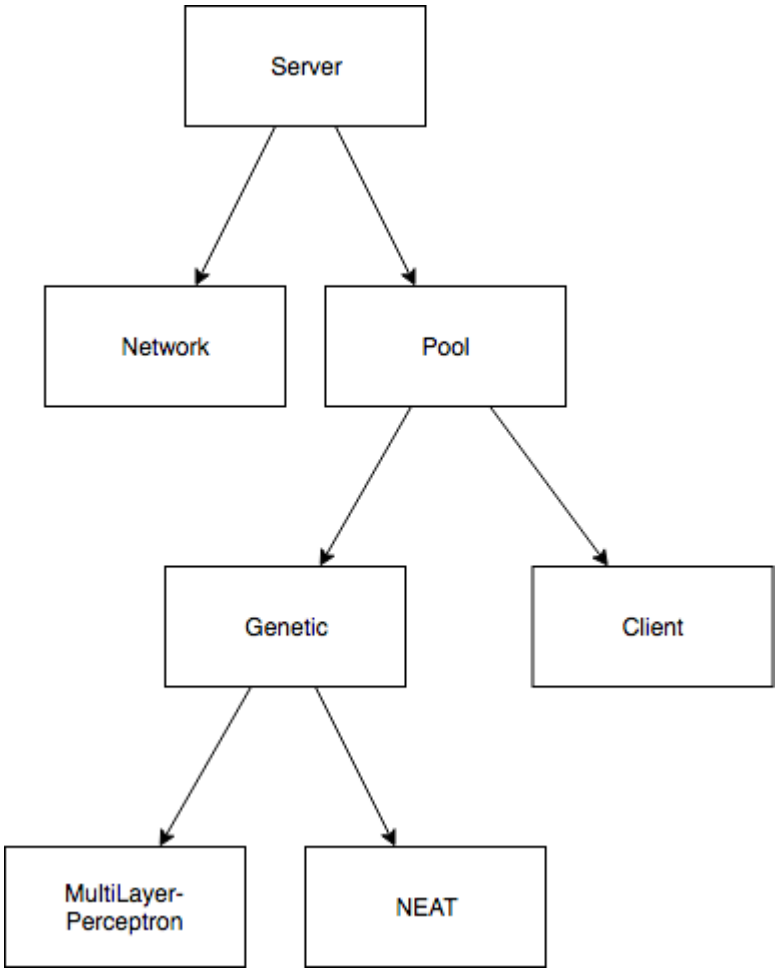


FIGURE 1 – Architecture des modules du serveur