Table des matières

Table des figures

Acronymes

Remerciements

1 Introduction

Ces dernières années l'apprentissage automatique (Machine Learning) à pris une place très importante dans le monde de l'informatique et nous avons pu assister au prouesses accomplies par des intelligences artificielles tels que la victoire d'AlphaGo face à Lee Sedol au jeu de Go en 2016. Cet événement n'est pas sans rappeler la defaite de Kasparov contre Deep Blue en 1997 qui avait defrayé le chronique.
Mais comment en sommes nous arrivés là, et quels sont les techniques qui se cachent derrière ces impressionnant résultats ?
Il y a plusieurs manière de faire du machine learning ; une des plus populaire, celle qui est utilisée pour ce travail, est le réseau de neurones artificiels. Cette technologie existe depuis bien longtemps, mais ce sont les récent progrès en terme de puissance de calcul qui ont amorcé sa montée comme technologie phare du domaine.

La méthode classique pour entrainer des réseaux de neurones est la rétropropagation du gradient (back-propagation). Cette technique constiste à comparer la sorties du réseau à la sortie attendu pour une entrée donnée afin de corriger les poids du réseau.[5] 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 connait 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.

Une de ces techniques, qui a été utilisée pour l'apprentissage d'AlphaGo, est le l'apprentissage par renforcement (Reinforcement learning). Cette méthode consiste à juger les décisions prises par un agent autonome en lui donnant une recompense positive ou négative afin que l'agent, au fur et à mesure des expériences trouve une strategie optimale. Cette façon de faire peut être comparé au comportement d'un enfant qui apprends à faire du vélo. Quand il tombe il va avoir mal, c'est une récompense négative. Il va donc corriger son comportement de façon à ne plus tomber. A l'inverse, quand il arrive a aller loin il va être fier et donc favoriser ce comportement, c'est une récompense positive.

Cependant il existe une méthode encore plus généraliste pour l'apprentissage : La neuroévolution. Cette méthode se base sur les algorithmes génétiques dont le fonctionnement est inspiré de la séléction naturelle qui a guidé l'évolution de la vie sur terre. En effet à partir d'un ensemble d'organismes que nous allons évaluer à leur capacité effectuer une tâche donnée, ce que nous appelons le *fitness* de l'organisme. Nous allons séléctionner les meilleurs d'entre eux afin de les faire se "reproduire" pour créer la génération suivante qui sera à son tour évaluée et on recommence ce processus autant de fois que nécessaire.

Le but de ce travail de bachelor est d'étudier et de comparer le comportement deux algorithmes de neuroévolution en analysant la faculté de chacuns à apprendre à jouer à des jeux vidéo.

2 Réseau de Neurones Artificiels

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.[2] 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).[3]

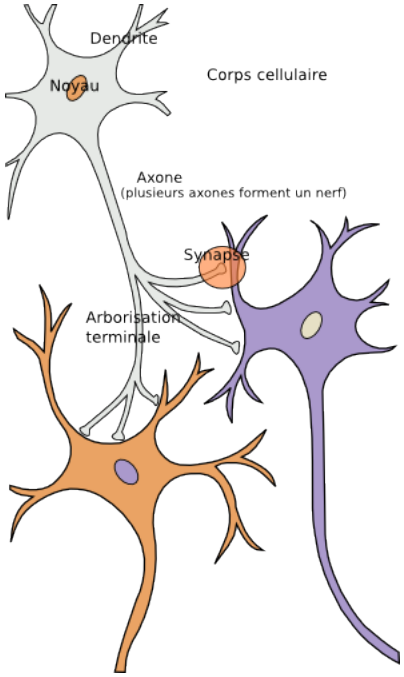

FIGURE 1 – Représentation des neurones dans le cerveau humain [4]

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 est le vecteur des signaux des voisins la couche précedente
W est le vecteur des poids entre le perceptrons et ses voisins de la couche précédente

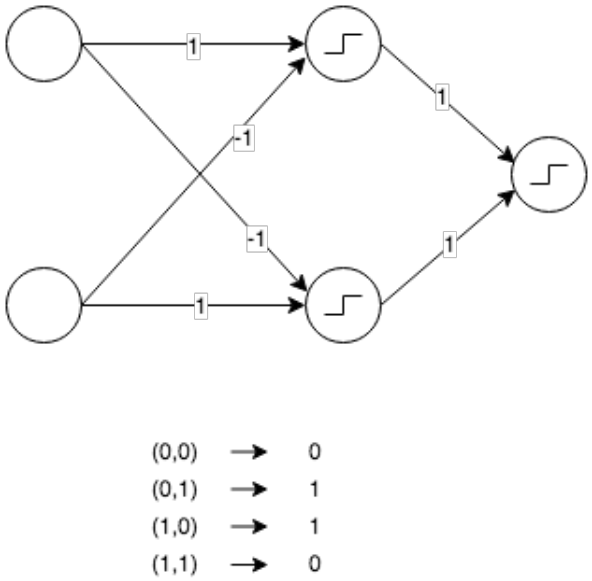

FIGURE 2 – Porte logique "ou exclusif" (XOR) simulée par un réseau de neurones

3 Algorithmes Génétique

La neuroévolution est une technique qui constiste à faire évoluer des réseaux de neurones artificiels afin qu'ils arrivent à effectuer une tâche, dans notre cas, jouer à un jeu vidéo simple.[1]

3.1 Génome & Phénome

3.2 Population initiale

La base d'un algorithme génétique est la population initiale, celle ci doit être générée aléatoirement afin de représenter un vaste spectre de possibilitées. Nous ne retrouverons dans cette popultation aléatoire aucuns individu capable d'accomplir parfaitement la tâche demandée, mais certains aurons des comportements qui les aiderons a d'obtenir un fitness plus élevé que leurs voisins, ceux-ci vont donc passer leurs caracteristiques à la génération suivante.

3.3 Selection

3.4 Crossover

3.5 Mutation

3.6 Elitisme

4 Evolution du perceptron multicouche

Cette section détaille le fonctionnement de l'algorithme génétique mis en place pour faire évoluer les perceptrons multicouche à topologie fixe, celui-ci à été créé en s'inspirant des concepts connu de ce domaine.

4.1 Population initiale

4.2 Génome & Phénome

Le génome est l'ensemble des gènes d'un individu, celui-ci content la totalité du code génétique constituant l'individu.

Pour cet algorithme le génome est simplement la liste des poids constituant le réseau de neurones. Ainsi, étant donne que la topologie du réseau est fixe, on peut le reproduire à l'identique en assignant les bons poids.

Le phénome dans le cas de cet algorithme sera une réseau de neurones à topologie fixe et entièrement connecté (Chaque neurones d'une couche donnée est connecté a chaque neurones de la couche suivante).

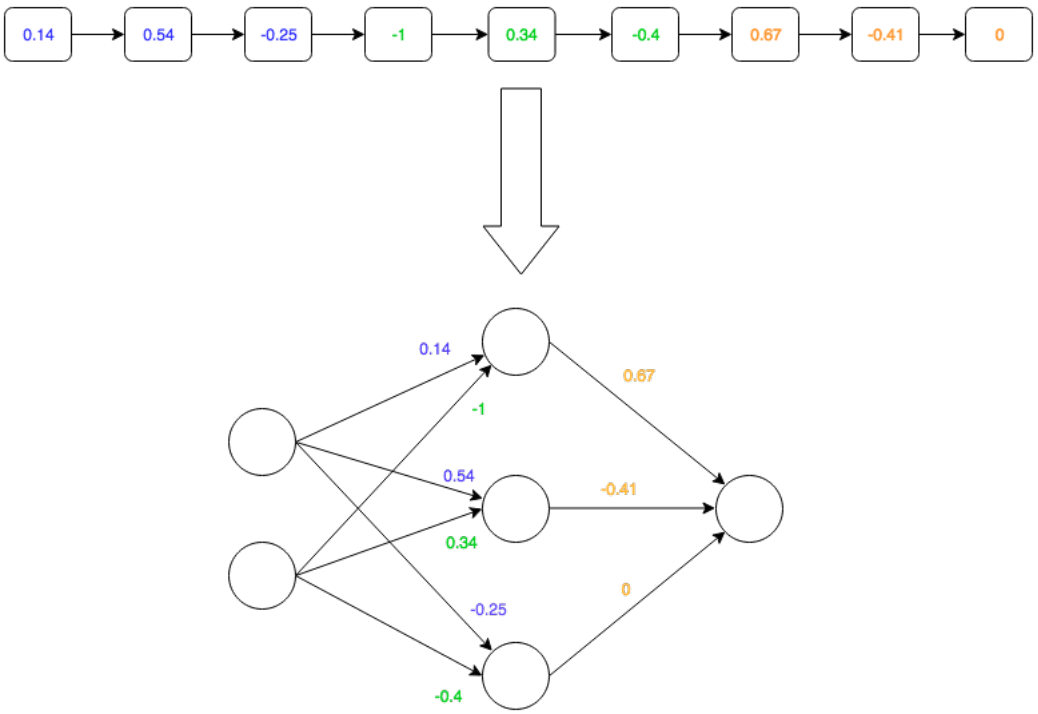

FIGURE 3 – Passage du génome au phénome

4.3 Crossover

Un crossover (où enjambement en français) est une opération génétique qui croise les gènes de deux parents afin de créer le gène de l'enfant, ce dernier va donc hériter de certaines caractèristiques de l'un où l'autre parent. Un exemple visible de cette opération chez les humains est que l'on reconnait des traits (couleur de peaux, yeux, cheveux) des parents chez les enfants.

Le crossover fonctionne d'après un principe très simple :

- On choisis un point de croisement
- On assigne chez l'enfant tout les gènes qui précèdent ce point depuis le premier parent
- On assigne chez l'enfant tout les gènes qui suivent ce point depuis le second parent

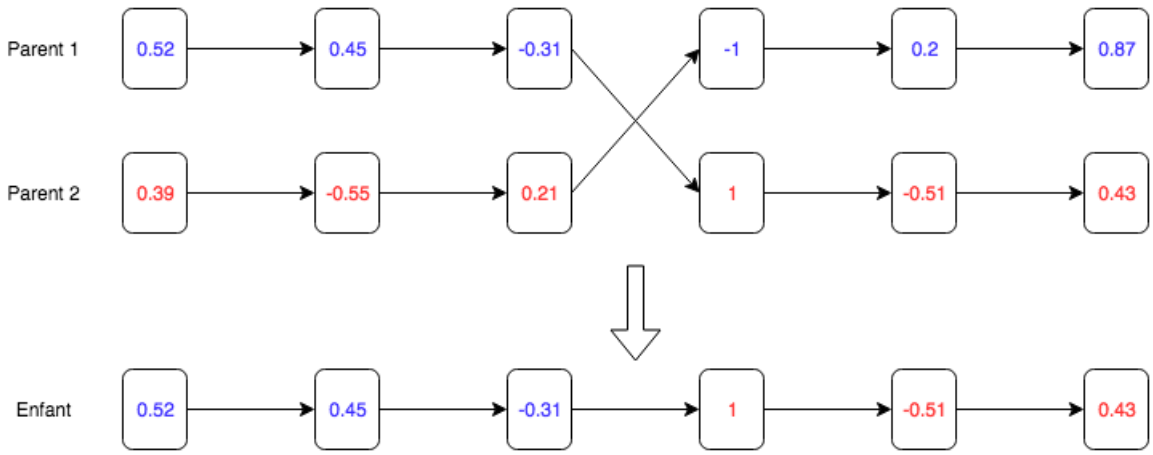

FIGURE 4 – Fonctionnement d'un crossover

Ainsi, les traits premettant une meilleure survie vont rapidement se répendre dans la population car les individus ayant obtenu un meilleur fitness serons plus souvent séléctionnés pour faire une crossover avec un autre individu.

4.4 Mutation

Les mutations sont des événements aléatoires qui vont altérer le génome de façon à créer une innovation, qui va par exemple se traduire par un comportement différent du génome. Dans certains cas les mutations seront benefiques (p.ex La capacité à respirer hors de l'eau), dans d'autre la mutation causera un comportement désavantageux (p.ex Malformation des membres).

Dans le cas de cet algorithme, étant donne que la topologie est fixée, la mutation sera simplement une variation aléatoire d'un poids dans le réseau de neurones. Ainsi chaque enfant de chaque génération aura 10% de chance que l'un de ses gènes subisse une mutation.

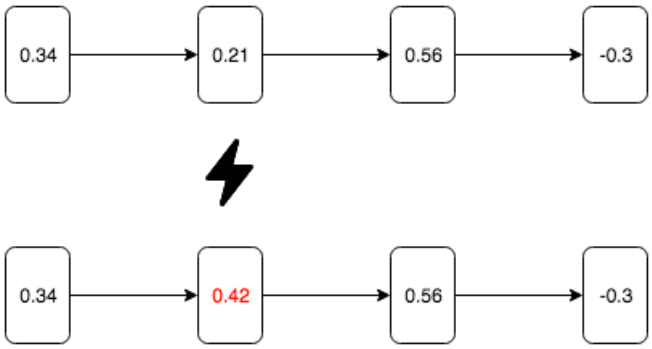

FIGURE 5 – Exemple de mutation dans un génome

4.5 Elitisme

L'Elitisme est un concept qui va permettre la survie des meilleurs individus d'une génération vers la génération suivante sans que leur code génétique soit modifié.

Pour cet algorithme, seul le champion de la génération est préservé sans altération de son code génétique.

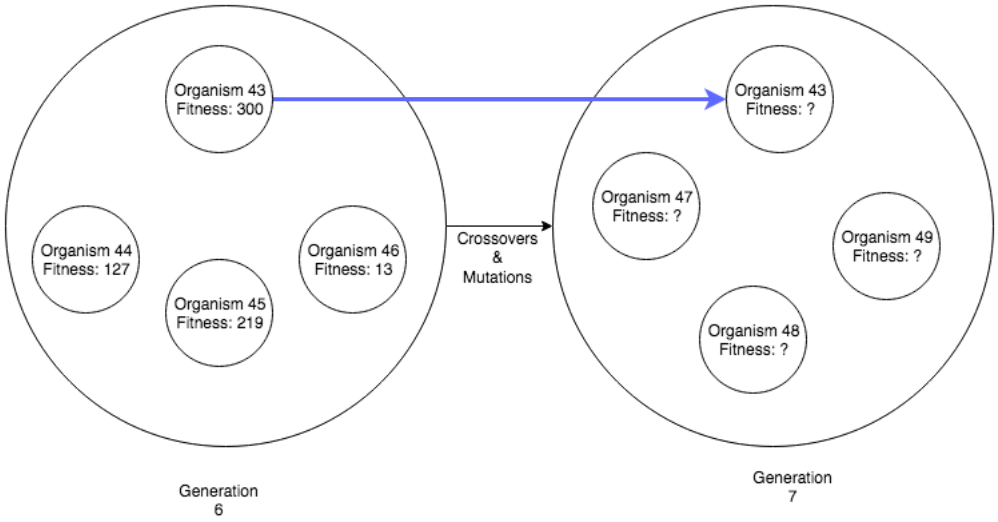

FIGURE 6 – Elitisme d'une génération vers la suivante

5 NEAT

L'autre algorithme génétique mis en place est le NEAT, créé par Ken Stanley à Université du Texas à Austin. Cet algorithme détaille comment obtenir des réseaux de neurones dont la topologie change au fil des évolutions.
Il apporte trois techniques essentielles à son fonctionnement : suivre l'évolution des gènes avec un historique pour permettre les crossover parmis les différentes topologies, séparer les organismes en espèces afin de préserver l'innovation et faire évoluer les topologies de manière incrémentales en partant de structures simples afin d'obtenir des résultats minimaux.[6]

5.1 Génome & Phénome

Dans le cas de NEAT, le génome est plus complexe, en effet il doit représenter tout les neurones et toutes les connexions. Il a donc fallu trouver une manière d'encoder qui soit suffisement souple pour permette de représenter tout les réseaux possibles.

5.2 Population initiale

La population initiale doit être aussi simple que possible, cependant il n'y a pas de règles dictant la manière dont elle doit être faite. Pour cet algorithme, la population initiale est donc un ensemble de perceptron multicouche avec seulement une couche d'entrée et une couche de sortie dans lequel chaque entrée est reliée à chaque sorties.

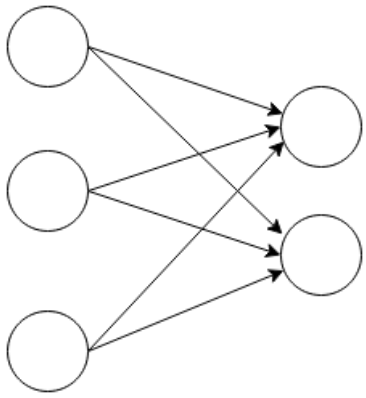

FIGURE 7 – Exemple de population initiale dans NEAT

Ce choix à été fait pour conserver un équilibre entre une population simple (pas de couche intérmédiaire) mais qui soit rapidement capable de s'ajuster car les connexions sont déjà disponnibles, on ne pert donc pas de générations uniquement pour créer les connexions.

5.3 Crossover

Le crossover dans le cadre du NEAT pose le problème des conventions concurrentes (competing conventions). En effet il possible que deux réseaux produisent le même résultat mais de manière différente, le risque est que lorsque nous croisons les deux réseau nous perdions une partie de l'information qui les rends efficace.

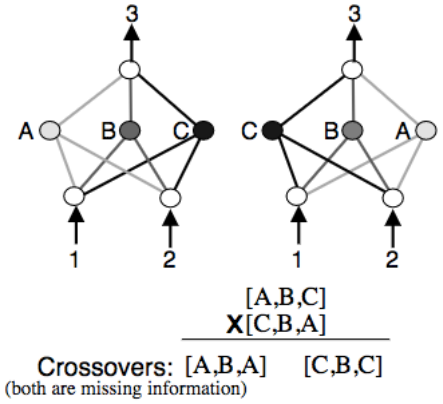

FIGURE 8 – Problème des conventions concurrentes, les deux réseaux atteignent le même résultat mais leurs possibles enfants en n'aurons que 2/3 de l'information [7]

Afin de pallier à ce problème, nous suivons l'apparitions d'innovations,

5.4 Mutation

5.5 Elitisme

6 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.

6.1 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.

6.2 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 :

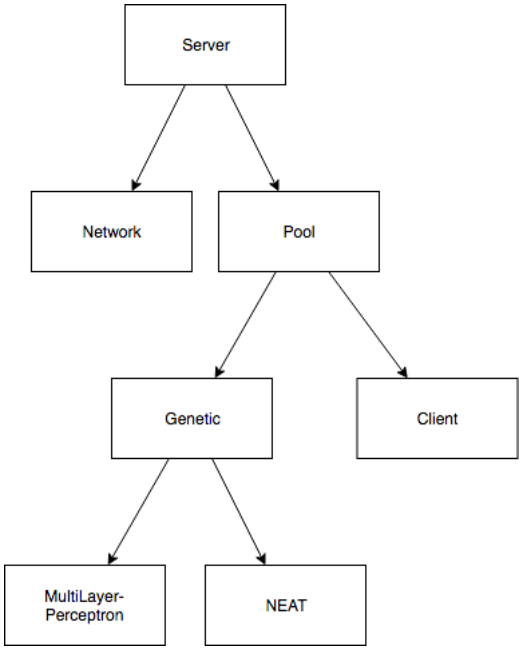

FIGURE 9 – Architecture des modules du serveur

6.3 Client

Le client va devoir effectuer la simulation demandée par le serveur, avec le bon réseau de neurones

7 Jeux

7.1 Asteroid

7.1.1 Principe

7.1.2 Entrées

7.1.3 Sorties

7.1.4 Résultats

Références

[1] Wikipedia contributors. Neuroevolution — Wikipedia, the free encyclopedia, 2018. [Online; accessed 22-April-2018].

[2] Wikipedia contributors. Artificial neural network — Wikipedia, the free encyclopedia, 2018. [Online; accessed 30-June-2018].

[3] Stephen Williams and Pankaj Sah. Brain functions, 2018. [Online; accessed 21-April-2018].

[4] Fernandes. Exosquelette et bionique, le système nerveux, 2013. [Online; accessed 30-June-2018].

[5] Raúl Rojas. Neural networks, 1996.

[6] Wikipedia contributors. Neuroevolution of augmenting topologies — Wikipedia, the free encyclopedia, 2018. [Online; accessed 30-June-2018].

[7] Kenneth Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *The MIT Press*, page 6, 2002.