Evolutionnary Neural Network Tools for Life Emergence and More

F. Furfaro

2021

Abstract

L'idée de vie artificielle remonte à 1940 avec le concept d'auto-réplication, au même moment où les concepts sur l'intelligence artificielle ont été créés. Nous proposons un modèle à définition incomplète, mais utile pour étudier la fonctionnalisation des structures cérébrales au cours de l'évolution. Nous combinons l'apprentissage par renforcement et la modification aléatoire des connexions intercouches pour résoudre un problème « fonctionnalisable ». Pour tester notre modèle, le jeu du chat et de la souris est un exemple de choix pour étudier l'évolution du comportement simplifié « proie/prédateur », tout en étant suffisamment complet pour une approche évolutive. Ces outils ne prétendent pas reproduire la vie sur ordinateur, mais peuvent donner des idées, pour l'optimisation des structures de réseaux de neurones et pour avancer vers la création de vie artificielle. En effet, lors du processus d'apprentissage via l'optimisation de la descente de gradient par l'évolution des connexions du réseau, nous observons par l'apparition de différentes structures, des stratégies convergentes pour notre problème. Ces structures sont conservées pendant un grand nombre de cycles alors que même des réseaux « concurrents » sont toujours ajoutés à chaque cycle de reproduction. En plus, on observe que ce type d'algorithme peut s'avérer utile pour des problèmes de classification, comme nous le proposons avec la base de données MNIST.

Introduction

La vie est apparue il y a 3,8 milliards d'année et n'a pas cessé d'évoluer et de coloniser l'environnement [Huxley [34]]. Elle a commencé de maniere unicellulaire sans noyau [Buick [15] and Mojzsis, Arrhenius, et al. [54]], à un large panel de forme avec l'apparition du noyau [Hedges, Blair, et al. [29] and Knoll, Walter, et al. [40]] et de la multicellularité [Bonner [11]]. En parallele, il y eu l'apparition du potentiel d'action, permettant la communication entre cellules [Matthews [49]], suivi de l'explosion de la diversité des systemes nerveux [Anctil [4]] et avec plus recement les systeme centraux [Child [17]]. Pour autant, il est assez difficiele de definir ce qui est vraiment vivant et cette question est toujours débatu. L'une des definition les plus simple est celle donnée par la NASA lors des programme de recherche de vie extra-terrestre : Un systeme chimique auto-entretenu capable d'évolution darwinienne [*NASA - Life's Working Definition* [58]]. L'evolution darwinienne est un principe fondamental dans l'etude des systeme complexe disposant d'heredité, de variation et de selection. On ne le retrouve pas uniquement en biologie, il se retrouve aussi dans l'evolution des courants musicaux, artistique, philosophique, technologique, societal et culturel [Chitwood [18] and Rogers and Ehrlich [65]]. Une autres définition de la "Vie" existe, et elle est plus lié à l'une des transformation thermodynamique fondamentale : L'entropie. La vie serait : Une structure dissipative capable d'auto-catalyse, d'homéostatie et d'apprentissage [Bartlett and Wong [6]]. Ce qui diminu localement l'entropie en favorisant l'auto-organisation. Meme si tout probleme n'est pas définissable [James [35]], ces deux definitions ce restreigne à un systeme "chimique". Est ce qu'un robot dotée d'une intelligence, d'emotion, voire de conscience, serait vivant avec ces définition ? Aujourd'hui, il existe un grand nombre de simulation de vie sur ordinateurs, comme par exemple Polyworld [Yaeger [82]], et sont, avec la construction de cellules minimaliste [Fredens, Wang, et al. [24]], d'une grande importance pour comprendre l'emergence de la vie.

AUTOMATE : Dans notre cas, nous allons essayer de reproduire un systeme vivant à l'intersection de ses deux definitions. Un systeme auto-entretenu capable d'apprentissage evolutif. Qu'on restreindra dans un premiers lieux à "un systeme capable d'apprentissage évolutif" pour en developper les outils futurs. Pour cela on a creer un envirommment de jeu simple : le jeu du chat et la souris, alternant ainsi deux comportements : L'attaque d'un prédateur et la fuite d'une proie. On se base sur deux outils : les automates cellulaires et les reseaux de neurones évolutifs. Les automates sont a l'origine des objets mathématique permettant de resoudre des probleme de decidabilité mathématique [Kemeny [38]]. On en retrouva ensuite des utilisation experimentale comme le plus

connu “jeu de la vie” [Wainwright [79]], qui à partir de 2 regles simple, faire emerger une forte complexité, voir meme reproduire une machine de Turing par l'existence de porte logiques [Berlekamp, Conway, et al. [8]]. Dans notre cas, l'état d'une cellule au temps t+1 est fonction de l'état au temps t d'un nombre fini de cellules appelé son « voisinage ». Le probleme est donc une grille ou l'on a des case “automates” qui se deplace dans la grille, elle a un adversaire qui va chercher soit à “attraper” l'automate, soit la “fuir”. La decision de deplacement de l'automate est controler/réguler par un reseau de neurone evolutif par principe d'apprentissage par renfocement. Les propriété de l'automate changeront ainsi à chaque génération, aussi bien parametre de “vue” de l'automate que les “actions” ou la structure du réseau.

STATISTIQUE : Les reseaux neuronaux ont ete construit à partir du modele de neurone biologique, le neurone formel [McCulloch and Pitts [50]], ou l'on constate qu'une neurone est une fonction “affine” avec une fonction d'activation. Aujourd'hui il s'est rapproché des methodes statistique non inductive pour resoudre des probleme descriptif comme la régression, la classification et le partitionnement. Les principaux outils d'analyse de grosse donnée en statistiques sont, pour ne citer que les plus connu : La PCA, analyse de composante principale, qui permet de projeter l'ensemble des points sur les regression multilinéaire des variable corrélé entre elle, ce qui reduit la dimension [F.R.S [25]]. Les SVM, séparateurs à vaste marge, generalisant les classificateur linéaire par des vecteurs supports, et formalisant la régularisation statistique et les concept de surapprentissage [Vapnik [76]]. Enfin, le K-mean qui divise un ensemble de points par minimisation d'un certain nombre de graine barycentrique [Bock [10]]. Tout ces outils encore utilisé aujourd'hui, sont peu à peu remplacé par des reseau de neurone vers 2000s apres le succes de l'apprentissage profond lors des compétition de classification d'objets sur des images [Hinton [31]] et l'augmentation de la puissance de calcul des processeurs [Moore [55]] .

APPRENTISAGE : Le question de l'apprentissage a été étudié au cours des année 1950, ou il a pu emerger la regles de Hebbs [Morris [56]] qui modifier simplemnt la valeurs des coefficient synaptique d'un reseau simple couche. Il eue un declin en fin 1970 par l'impossibilité de résoudre des probleme non linéaire XOR ou de connexité [Minsky and Papert [52]]. Mais ce probleme fut resolu par les perceptron multicouche, mais cette fois la modification des poids était plus compliqué. C'est le systeme de retropropagtion du gradient [Rumelhart, Hinton, et al. [67]] qu'on retrouve aussi dans le cerveau [Stuart, Spruston, et al. [73]] et l'algorithme du gradient, par minimisation itérative de l'erreurs, qui permit de resoudre ce probleme. En parallele, inspirer par les neurone moteurs de la vue [Hubel and Wiesel [33]], on devellopa les reseau de neurone convolutif qui reduit fortement le calcul des poids en entrée du reseau, car n'est pas entiereement connecté [Ciresan, Meier, et al. [20]]. Le souci de ce genre de reseau est qu'il faut un grand nombre de donnée pour adapté les poids et de passage à l'echelle avec un cerveau est de l'ordre de 10^{10} pour le modele GPT-3 [Brown, Mann, et al. [14]]. Pour resoudre un probleme de transition d'états stochastique, dit de decision markovien, il y eu l'apparition du Q-learning convergent par l'apprentissage des récompense [Watkins and Dayan [80]]. Ces reseau, à l'image des reseau génératif, ont encore était amélioré par des reseau adversariaux “*actor-critic*” et permette d'accelerer la convergence des poids [Konda and Tsitsiklis [42]].

FONCTIONALISATION : Néamoins, ces reseaux optimise l'apprentissage du reseau mais ne cherche pas à maximiser la structure des couches du reseau. Seule les reseau de neurone recurant ont été le plus traité car il y avait des probleme de “vanishing gradient” [Hochreiter and Schmidhuber [32] and Vaswani, Shazeer, et al. [77]], en particulier pour le traitement du texte. Une autre catégorie de reseau est apparu dans la fin des années 1990, les reseau NEAT, ou les poids et les connexion sont “appris” par un processus de selection [Stanley and Miikkulainen [71]] et les reseaux de Kononen, ou des stimuli de même nature excitent une partition d'une grille de neurones pour reorganiser la structure [Kohonen [41]]. Hors le cerveau n'a pas des poids qui ont été selectionné par l'evolution, mais ce qui est le cas la structure, dans la limite de la plasticité cerebrale des neurones à piques/impulsion [Maass [45]]. L'idée est donc de s'impirer des donnée en neuroscience recente, ou il est montré que le cerveau est pré-cablé/fonctionnalisé pour voir certain danger ou encore marcher à la naissance [Rakison and Derringer [63]]. De meme, il n'y a pas eu besoin d'avoir un temps d'apprentissage long pour optimisé une fonction, hormi les cas où il y a besoin de mémoire et où le sommeil est necessaire [Rasch and Born [64]]. A cette image, on a donc devellopé un algorithme où les couches interconnecté de maniere aléatoire de façon à fonctionnaliser le probleme “chat-souris” aussi bien dans la structure que dans l'apprentissage par renforcement.

Méthodes

L'ensemble du projet est disponible sur le lien `github.com/fabienfrfr`. Le projet a été codé intégralement en Python de facon à utiliser efficacement les bibliothèques logiciel : PyTorch pour les reseaux de neurone, Keras-Tensorflow pour la base de donnée MNIST, Numpy et Scipy pour les calculs scientifique, Pandas pour l'analyse et le stockage des données, Networkx pour la representation et l'analyse des Graphes et enfin Matplotlib pour la visualisation et l'animation des données. On distingue 6 fichiers d'experience, 2 fichiers d'analyse de donnée (DATA_ANALYSIS et EXTRA_FUNCTION) et un fichiers de debuggage modulaire. Dans les fichiers "experience" on retrouve au plus bas niveau GRAPH_EAT encapsulant GRAPH_GEN, generant la liste des connections pour chaque couche de neurone, ainsi que pRNN, générant la structure du reseau en fonction de la liste d'adjacence. À plus haut niveau, on retrouve AGENT qui contient les propriété d'entrée/sortie de l'agent et son algorithme d'apprentissage et de mémorisation, ainsi que TAG-ENV contenant cette fois les regles du jeu "chat-souris". Enfin on a le MAIN qui va lister l'ensemble des agents et environnement associé pour lancer les experiences d'evolution. On retrouve aussi un fichier LOG qui permet de stocker au format *csv* les experience.

1 Reseau de neurone évolutif

Le perceptron est un algorithme d'apprentissage fonctionnant comme une fonction de seuillage et decrivant le neurone formel. Une fonction de seuil est un classifieurs linéaire qui a pour entrée un vecteurs à valeurs réelle (virgule flottante) et une valeurs f(z) en sortie, qui peut etre suivant la fonction d'activation : binaire (Heaviside), reel (Sigmoid) ou entre les deux (ReLu) [Godin, Degrave, et al. [28]]. Le perceptrons se represente par le produit scalaire entre le vecteurs d'entrée et le vecteurs poids des neurones, soit l'application multilinéaire suivante :

$$o = f(z) = \begin{cases} 1 & si \quad \sum_{i=1}^{n} w_i x_i > \theta \\ 0 & sinon \end{cases}$$

Ce qui s'ecrit dans la plupart des bibliothèques logiciels comme la sequence d'un neurone à **"n"** entrée, suivie d'une fonction d'activation.

Code PyTorch :

```
Layers = nn.Sequential(nn.Linear(N, 1), nn.ReLU())
x = Layers(x)
```

Le perceptrons est un classifieurs linéaire, il converge uniquement si l'ensemble des donnée d'entrée sont linéairement séparable (droite). Lorsqu'on place à la suite plusieurs sequence de neurones, on obtient un classificateurs à composition linéaire, tel que :

$$y = \bigcirc_n w_i.x_n$$

Cette operation revient à une suite de composition de fonction, ou à chaque couche, les donnée d'entrée non linéaire se déforme à chaque étapes par transformation linéaire. C'est l'equivalent de l'astuce du noyau qui remplace un produit scalaire par une fonction d'estimation avec ou sans dimensition supplémentaire pour obtenir un probleme linéaire [AIZERMAN [2]]. L'ensemble des petites déformation transforme les données d'entrée pour etre linéairement séparable en sortie de reseau. Ce type d'opération à la suite revient numériquement à la construction d'un graphe computationnel.

Code PyTorch :

```
Layers = nn.ModuleList([nn.Linear(1, 1) for i in range(10)])

for i, l in enumerate(self.Layers):
            x = self.Layers[i](x) # == l(x)
```

Mettre à jours les poids **"w"** correspond à l'apprentissage du perceptrons, la *regles delta* (Widrow-Hoff) est la plus adapté pour les probleme de descente de gradient [Schmidhuber [69]] et consiste à comparer linéairement la valeurs obtenu par la sortie du reseau **"z"** et la valeurs attendu par le reseau de neurones **"d"**. Il existe d'autre methodes plus sophistiqué ou limitant les probleme d'annulation tout aussi simple (Least Mean Square), mais ici présenté, la regle delta se presente sous la forme :

$$w_i(t+1) = w_i(t) + (d-z)x_i$$

Lorsqu'on a une suite de neurone, on appelle la variation de cette relation sur l'ensemble du reseau, la descente de gradient, car on calcul le différentiel entre t+1 et t. Cette derniere est dite stochastique lorsqu'on introduit un terme de variaton aléatoire et permet de trouver un minimum grobal plus efficassement. Le calcul de l'erreur global est assez complexe si le reseau de neurone est grand. C'est pour cela qu'il existe des methode ou l'on remonte l'erreur de sortie vers l'entrée pour trouver l'ensemble des minimums locaux. On parle alors de retropropagation du gradient, qui revient à chaque iteraton de calculer le gradient de chaque couche entre t et t+1 par le produit tensoriel symmetrique de la couche prededante. Pour le cas d'un seul neurone par couche, le produit tensoriel, n'est autre qu'une multiplication provenant de la dérivée des fonctions composée σ, la fonction d'activation, avec δ, les biais pour chaque couche. Dans un cas plus général, on a :

$$\begin{cases} w \leftarrow w - \eta \left[\frac{\partial R(w)}{\partial w} + \frac{\partial L(w_i)}{\partial w} \right] & Gradient \\ \delta^{x,l} \leftarrow ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l}) & BackProp \end{cases}$$

Dans la plupart des bibliothèques logiciels, on precise le type de calcul de perte (loss ou criterion) avant la boucle d'apprentissage, ainsi que l'algorithme de calcul de la descente de gradiant (optimizer). Les algorithme les plus utilisé sont SGD "Stochastic Gradiant Descent" et "Adam", plutot utilisé dans les problemes de regression et de classification [Ruder [66]].

Code PyTorch :

```
LOSS = torch.nn.MSELoss(reduction='sum')
optimizer = optim.SGD(Layers.parameters())
### Loop
LOSS = LOSS(x_pred, x)
optimizer.zero_grad()
LOSS.backward()
optimizer.step()
```

Un reseau de neurone correspond à une interconnexion entre couche de neurone contenant **"n"** perceptrons. Une unique couche de **"n"** perceptron corresponds au reseau le plus simple et permet de resoudre des probleme linéairement séparable de type hyperplan. Comme toute les entrée **"x"** sont interconnecté au **"n"** neurone, les vecteurs poids **"w"** forme une matrice **"m*n"** avec **"m"** le nombre d'entrée au réseau. La sortie s'exprime dans ce cas :

$$\vec{y} = \begin{bmatrix} w_{0,0} & & \\ & & \\ & & w_{m,n} \end{bmatrix} . \begin{bmatrix} x_0 & \dots & x_m \end{bmatrix} + \begin{bmatrix} b_0 \\ \\ b_n \end{bmatrix}$$
$$\vec{s} = f(\vec{y})$$

Pour que l'algorithme mette à jours efficassement les poids du réseau, on utilise un certain lot de donnée d'apprentissage iterativement, qu'on appele "batch". Pour ce batch, soit il contient l'ensemble des données de l'experience, soit il est réparti en "mini-batch" stochastique pour eviter la sur-utilisation de la mémoire et augmente la vitesse de convergence de la vallé de stabilité. La taille du lot ne doit pas etre trop petit non plus car augmente le bruit [Kiwiel [39]]. Dans ce cas, le vecteurs d'entrée devient une matrice **"m*b"**, et la matrice des poids devient un tenseurs d'ordre 3 **"n*m*b"**. La plupart des bibliothèques logiciels s'adapte automatiquement à la taille du batch, et n'a pas besoin d'etre spécifié.

Code PyTorch :

```
Layers = nn.Sequential(nn.Linear(N, M), nn.ReLU())
x = Layers(x)
```

Pour autant, les reseaux linéaire ne sont pas les seules à etres employé et ne sont pas forcement les plus adapté pour les problemes d'images [Ciresan, Meier, et al. [19]]. En effet, une image est une donnée 2D dont les données spatiale sont localement lié, mais les informations ne sont pas interconnecté, c'est pour cela qu'il existe les reseaux de convolution. Ces reseau ont l'avantage d'etre interconnecté mais par fenetre "spatiale" ce qui reduit fortement le nombre de calculs pour l'actualisation des poids. Elle est composé de **"N"** perceptron, mais ne represente plus comme un produit tensoriel, mais comme un produit de convolution où les entrée sont moyenné pondérément par le glissement **"C"** d'une fenetre de taille **"k"** sur les poids **"w"**. La sortie s'exprime de la sorte :

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k)$$

Dans le cas le plus simple, on peut considerer qu'il y a qu'une seule entrée par neurone, pour cela, le nombre d'entrée est egal au nombre de sortie du canal de la couche. Ensuite, la taille de la fenetre est de **"1"** et le groupement de connexion doit etre egal au nombre de neurones, qui est un nombre divisible de l'entrée et de la sortie par lui meme. Ce type parametrage est equivalent à **N** neurones indépendants à 1 seules connexion chacune. Contrairement au réseau linéaire, la taille du batch doit etre specifié lorsqu'on donne les données d'entrée à la couche de convolution.

Code PyTorch :

```
Layers = nn.Sequential(nn.Conv1d(I, I, 1, groups=I), nn.ReLU())
x = Layers(x.view(BATCH_SIZE, I ,1)).view((BATCH_SIZE, I))
```

1.1 Fonctionnalisation

S'inspirer du cerveaux "modèle" pour construire des algorithmes neuronaux à toujours été present dans le developpement de nouvelles methodes en *Intelligence Artificielle*. Contrairement à l'optimisation de la retropropagation [Liu, De Mello, et al. [44]], nous nous sommes inpiré de la specialisation fonctionnele du cerveau pour optimiser la structure par fonctionnalisation. En effet, le moyen le plus efficace de réaliser des fonctions différentes c'est de confier la réalisation de chaque fonction à un outil particulier et spécialisé [Barkow, Cosmides, et al. [5] and Tooby and Cosmides [75]]. C'est pour cette raison que la sélection naturelle (survie, reproduction) a organisé notre corps en différents organes qui sont chacun spécialisés, mais aussi dans le domaine de la cognition [Confer, Easton, et al. [21]]. La specialisation fonctionnele du cerveau est observé empiriquement par plusieurs troubles neuropsychologique, comme par exemple avec le syndrome de Capgras, ou une defaillance d'une region entraine la perte de reconnaissance des visage [Thomas Anterion, Convers, et al. [74]].

Dans notre cas, nous avons simplifié le plus possible un cerveau, où l'on a en entrée des neurones de "vision", suivi de couches sequentielle "fonctionnelle" et en sortie, des neurones "moteurs". On distingues des neurones stables, les neurones d'entrée et de sortie qui ne changeront pas pendant le processus évolutifs. Les neurones de "visions" sont à l'image des cones de la rétines, c'est à dire des neurones connecté à des batonnée de vue unique, qu'on represente dans notre cas comme une couches de convolution "simple sparse" : une seule entrée par neurones. Les neurones "moteurs" quand à eux sont analogue au motoneurones qui active la contraction musculaire et donc le mouvement des organismes [Purves, Augustine, et al. [62]], on les represente dans notre cas comme une couche linéaire de neurones où les sortie seront lié au action du reseau de neurone complet.

La vrai particularité du projet est que les couches intermediaire "fonctionnalisé" ne sont pas contruite manuellement, mais adapté par un processus evolutif de selection. Ainsi, l'objectif est de faire emerger un structure intermediaire fonctionnalisé à notre probleme, ici le "jeu du chat et la souris". Les connections intermediaires sont initialement aléatoire avec au minimum une connection entre la couche "n+1" et "n", pour avoir toujours un lien entre l'entrée et la sortie et ne pas avoir de probleme lors du calculs de poids des neurones. Au cours de l'apprentissage, on s'attend à ce que le reseau soit de plus en plus structuré.

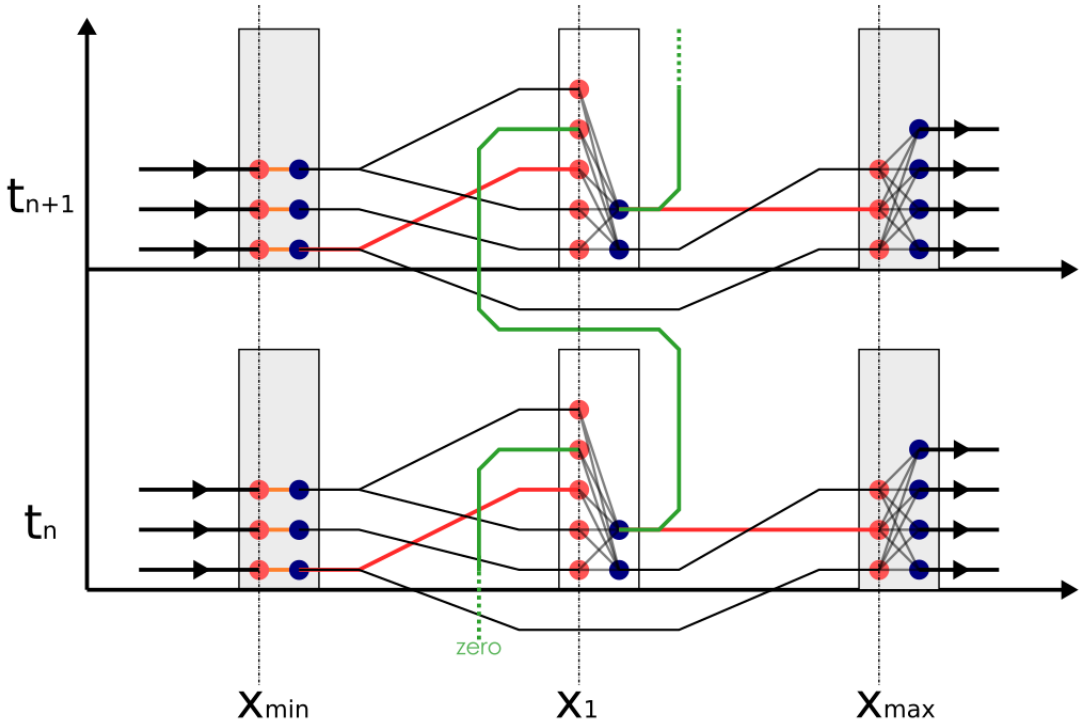

Figure 1: Reseau monocouche à connection aléatoire; Le meme reseau est representé 2 fois à deux intervalle de temps à la suite, les nombres entrée/sorties sont pour l'exemple. De bas en haut : La commucation entre deux temps. De gauche à droite : L'entrée vers la sortie.

Comme les connections entre l'entrée d'une couche X_n et la sortie d'une couche en X_{n+m}, avec $m \geqslant 0$, n'est pas possible pour le calcul de la retropropagation à un temps donnée, on se trouve avec la connection $X_{n+m,t-1} \rightarrow X_{n,t}$. Ce type de connection laisse apparaitre une forme de reseau recurrent pour le calcul du gradient, mais comme nous voulons eviter les structure artificielle comme LSTM qui empeche le *vanishing gradient* des reseau recurrent [Hochreiter and Schmidhuber [32]], nous avons consideré cette entrée à "t-1" comme une ***entrée virtuelle***. Cette entrée virtuelle, n'est pas la solution la plus elegante, mais est la plus flexible dans le cas ou les connections sont completement desordonné et reduit le temps de calculs car la taille de l'entrainement reste linéaire. Une couche connecté à une entrée virtuelle d'une couche elle meme connecté à une entrée virtuelle, permet de stocker une information à "t-2", la taille du stockage possible est proportionnel à la taille du reseau. L'entrée virtuelle consiste au détachement de la sortie du graphes computationnelle à l'algorithme du gradient. On obtient un réseau de "*neurones à propagation avant*" que l'on qualifiera dans le programme de ***pseudo-récurrents*** à cause des entrée virtuelle.

1.2 Construction du graphe

La topologie du reseau presenté precedement peut etre vu comme un graphe orienté acyclique, en effet, les connection entre l'entrée X_n et la sortie X_{n+m} sont vue comme des entrée virtuelle indépendante de la rétropropagation. Les connection intercouche peuvent etre négligé pour la construction initiale du graphe, car elles sont completement connecté (voir code PyTorch *nn.linear*) . Pour les connections entre les couches, on se retrouve face à plusieurs contraintes :

- Quel est le nombre minimale de connection pour que le reseau soit complement connecté ?
- Combien de connection sont attribué par couche ?
- Comment attribuer les connection par couche de neurone ?
- Comment stocker ces informations pour reconstruire le graphe ?

Empiriquement, le nombre total de perceptron dans la couche intermédiaire "N_h" est initialement défini comme la racine entiere du nombre d'agent par génération d'entrainement et ne peut dépasser 32.

Propriétés "Unicité des entrée": Les liens/connections entre les différentes couches de neurones depend des entrée de celle ci. On peut se connecter plusieurs fois à une sortie, mais "une" entrée n'a qu'une "seule et unique" entrée.

À partir de cette propriétés, si on veut que toute les entrée soit non vide, on peut definir le nombre minimal de connection pour notre reseau comme la somme du nombre de perceptron de la couche intermédiaire "N_h" et du nombre de perceptron en entrée "N_i". Ainsi, pour que le graphe soit complet, la relations du nombre minimal de connection "C_{min}" est :

$$C_{min} = N_h + N_i$$

Le nombre de connection total "C_{tot}" est ensuite attribué aléatoirement dans l'interval $[C_{min}, 2C_{min}]$. Puis le nombre de couche intermediaire du reseau "N_L" est attribué aléatoirement dans l'interval $[1, C_{tot}]$. Lorsque que "C_{tot}" et "N_L" sont défini, il reste à définir le nombre de connection et de perceptron que va avoir chaque couche de neurone.

Propriétés "Connection limité" : Il ne peut y avoir moins de "une et unique" connection par couche, mais elle ne peut dépasser l'ecart entre "C_{tot}" et "N_L".

Propriétés "Perceptron limité" : Il ne peut y avoir moins d'"un seule et unique" perceptron par couche, mais elle ne peut dépasser l'ecart entre "N_h" et "N_L".

À partir de ces deux propriétés, on peut definir par récurrance l'attribution aléatoire du nombre de connection et perceptron par couche intermediaire de neurone. Le schéma de l'evolution de la densité de probabilité uniforme à discrétiser est le suivant :

$$f_{C_n}(x) = \begin{cases} \frac{1}{(C_{tot} - C_{n-1} - N_L - n) - 1} & pour\, 1 \leq x \leq (C_{tot} - C_{n-1} - N_L - n) \\ 0 & sinon \end{cases}$$

$$f_{N_n}(x) = \begin{cases} \frac{1}{(N_h - N_{n-1} - N_L - n) - 1} & pour\, 1 \leq x \leq (N_h - N_{n-1} - N_L - n) \\ 0 & sinon \end{cases}$$

Une fois l'attribution du nombre de connection réalisé par couche, il est necessaire de positionner ces couches spatialement pour le calcul de la rétropropagation. Dans notre cas, nous avons limité le nombre de couche à 32, avec une seule et unique couche possible par position spatiale discrete. Ainsi la position des couches intermediaire est atribué aléatoirement et sans remplacement dans l'interval $[1; 31]$, où la position ***zero*** est reservé à l'entrée et ***32*** à la sortie. Les positions spatiales des couches de neurone n'influence pas le calcul de la rétropropagation, ainsi, les connection entre couche sont relative à l'ordre positionnel. On stocke ensuite l'ensemble des connection possible dans une liste. Mais une fois le positionnement des couches réalisé, comment connecter les noeud de facon à ce qu'il y ait un lien entre l'entrée et la sortie et qu'il n'y ait pas que des entrée virtuelle ?

Pour cette question, nous avons considéré 3 cas où l'attribution des connections suit des loi uniforme discrétisé avec des densité de probabilité différente. On distingue les 3 cas suivants :

1. La premiere connection : on se connecte à l'une des sortie de la couche la plus proche derriere, si c'est la premiere, on se connecte à l'une des sortie des neurones d'entree. En effet, il est necessaire qu'il y est au moins un chemin qui mene d'entrée vers la sortie pour la retropropagation, ce cas garantis cette condition. La probabilité est directement défini par $P(X_k) = \begin{cases} 1 & X_k = X_{k+1} - 1 \\ 0 & sinon \end{cases}$ pour ce cas.

2. Tant que toute les connections n'ont pas été attribué : On privilégie les connections vers l'arriere à 2/3 des probabilité. De cette facon, on limite l'excess d'entrée virtuelle sans que cela ne soit impossible. Cette regle à été défini empiriquement et ne semble pas avoir d'effet sur le processus de selection sur le long termes (non calculé). La probabilité est calculé à partir de la discretisation de la densité de probabilité discrétisé $f_X = \begin{cases} \frac{2}{3} & pour\, a \leq x \leq b \\ \frac{1}{3} & pour\, b \leq x \leq c \\ 0 & sinon \end{cases}$ dans ce cas.

3. Lorsque toute les connection avec les neurones de sortie ont été au moins attribué une fois, alors il n'y a plus de contrainte spatiale au choix des neurones de sortie : La probabilité suit une loi uniforme sur l'ensemble de l'intervalle de definition spatiale.

Au cours de ce processus, les connections des couches au sortie sont listées dans l'ordre d'attribution par couche, ce qui nous donne une liste d'adjacence par couche. Chacune des listes ont par définition la taille du nombre de connection par couche. En representant l'ensemble des listes dans une matrice d'adjacence, on remarque que la diagonale sépare les connections en amonts des connection en aval. La matrice à par définition la structure d'un graph orienté.

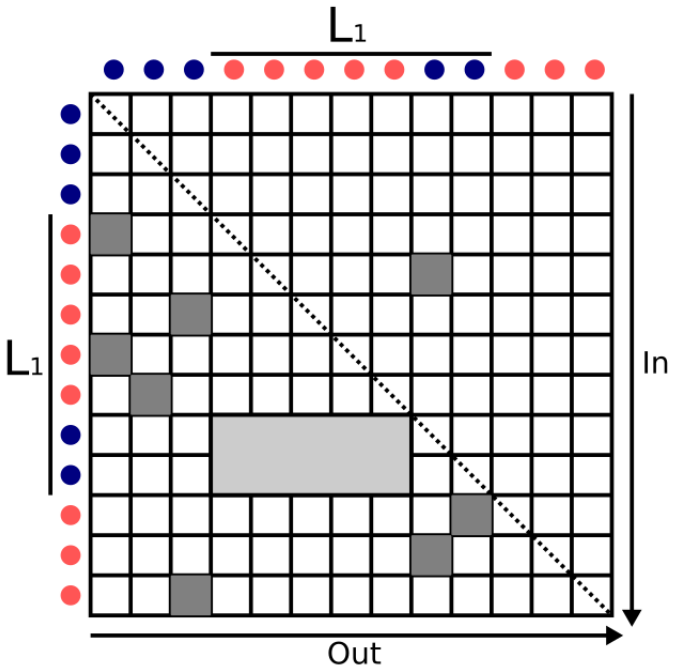

Figure 2: Matrice adjacente des liens du reseau de neurone précedent. En rouge, les noeud de connection (entrée d'une couche) et en bleu, les noeuds de perceptron (sortie de couche). Les case grisé correspondent à une connection de poids constants (1 par défauts).

Cette matrice, n'est pas utiliser dans le processus de restructuration du reseau. La matrice etant creuse, on peut se limiter à un ensemble de liste adjacente par couche de neurone. La representation en liste simplifie la restructuration du reseau et optimise la taille d'utilisation de la mémoire. Cette matrice sera toutefois necessaire pour l'analyse des donnée de graphe, on la reconstruiera à posteriori. Les connection intercouches representé en gris clair sur la [Fig 2.] ne sont pas stocké en mémoire car ils sont définit par des opérations tensoriel déjà implémenté.

2 Methodes d'apprentissage

Les methodes d'apprentissages standards ont besoin d'un grand nombre de donnée pour que l'ajustement des poids permettent à un modele d'etre prédictif [Malinowski, Zurada, et al. [46]]. Avec l'ere du Big Data, cette approche fut concluante, meme si elle risque de favoriser le surapprentissage d'un reseau en classifiant trop avec les données d'entrainement [Al-Jarrah, Yoo, et al. [3]]. Pour autant, dans le vivant et d'apres les théories de l'apprentissage, il n'y a pas besoin d'un tres grand nombre d'experience pour qu'un cerveau apprenne efficassement un probleme [Mowrer [57]] ou soit conditionné [Pavlov [60]], et l'on peut apprendre de plusieurs facons, soit par la répétition, soit par le jeu [E. Stengel]. Dans certain cas, le cerveau est meme pré-cablé chez certaine espece pour repondre à certaine fonction, on peut citer comme exemple la marche des enfants à la naissance [Modrell and Tadi [53]] ou des mammifere juvénile "proie" [Garwicz, Christensson, et al. [26]]. C'est le principe de fonctionnalisation cérébrale qu'on essaye ici de reconstituer en selectionnant des reseau "optimum" à la descente de gradient d'un probleme de decision. A partir du graphe vu precedemant, on peut reconstruire les connections entre les différentes couches du *réseau de neurones à propagation avant*. L'algorithme "forward" de la plupart des bibliothèques logiciel se retrouve restructuré, mais est compatible avec la structure des graphes computationnels classiques.

Algorithm 1 Forward construction by Adjency List; **X** correspond au Tenseur d'entrée de taille I*Batch. **NET** correspond au information des neurones (index, position, liste adjacente). **Trace** correspond à la mise en mémoire des sorties des couches modulés des liste de neurones **Layers** (PyTorch : *nn.ModuleList*). **h** correspond à la copie détaché de la liste des Tenseurs Trace à t-1.

```
Batch size, Input size ← dim(X)
BATCH_ ← np.arange(len(x))
Trace[-1] ← Layers[-1](X.view(Batch size, Input size,1)).view(Batch size, Input size)
for i, network index ordered by position do :
    tensor ← []
    for j,k in NET[i, -1] :
        if j == 0 :
            tensor += [Trace[-1][BATCH_,None,k]]
        else :
            if (NET[i, 3] >= NET[i, 3]) :
              tensor += [h[j][BATCH_,None,k]]
            else :
              tensor += [Trace[j][BATCH_,None,k]]
    tensor_in ← torch.cat(tensor, dim=1)
    Trace[i] ← Layers[i](tensor_in)
    Trace[-2] ← Layers[-2](tensor_in)
for t in range(len(Trace)):
    h[t][BATCH_] ← Trace[t][BATCH_].detach()
return Trace[i], Trace[-2]
```

Une fois le reseau completement cablé, celui ci est equivalent à un *réseau à propagation avant* avec plusieurs entrée intermédiaire. Ces entrée intermédiaire corresponde comme on l'a vu dans la partie precedante au *entrée virtuelle*. L'objectif qui suit est d'optimiser le calcul de la fonction de cout $J(a, b)$ avec $J(a, b) = \frac{1}{2m} \sum_{i=1}^{m} (f - y)$. Dans notre cas, on distingue deux etapes dans l'optimisation du calcul la fonction de perte, d'abords à "***temps courts***" qui correspond à un apprentissage classique de descente de gradient à chaque cycle de reproduction, et à "***temps long***" où l'on va selectionner à chaque étapes les reseau qui auront le mieux reussi pendant de l'entrainement à temps court. La premiere étapes correspond à une methodes classique en apprentissage par renforcement, le Q-learning, la seconde, plus exploratoire, correspond à l'ajout de mutation dans le graphe du reseau qui va changer les connections entre les différents noeuds.

2.1 Temps courts : Q-Learning

A temps courts on cherche un algorithme de descente de gradient adapté à notre probleme. La plupart du temps, on donne un jeu de donnée d'entrainement adapté à un probleme, comme par exemple *MNIST* pour la reconnaissance des chiffres au format image [Deng [22]], ou encore, *ImageNet* pour la detection et classification d'image d'objet. Dans notre cas, on a essaie de reproduire un systeme "vivant" caractérisé par un automate cellulaire [Voir partie suivante]. Ainsi, le système suit une succession d'états et d'action distincts dans le temps et ceci en fonction de probabilités de transitions. L'evolution du systeme correspond à un processus de decision markovien, representé par une chaine de markov. Une chaine de markov est une suite de variable aléatoire (X_n) dans l'espace probabilisé (E, B, P), où pour chaque **n**, sachant X_n, X_{n+1} indépendant de X_k, on a la probabilité de transition (Hypothèse de Markov) [Markov [47]] :

$$P(X_{n+1} = i_{n+1} | X_1 = i_1, \ldots, X_n = i_n) = P(X_{n+1} = i_{n+1} | X_n = i_n)$$

Dans notre cas, on associe à chaque transition, des action et des récompense associé à l'agent de facon à le guider dans le temps. On peut représenter cela suivant le couple de matrices (T, R), où $\boldsymbol{T}$ correspond à la matrice de

transition et $\boldsymbol{R}$, la matrice des récompense. La complexité est que l'on ne connait pas la probabilité de transition. Le but dans un processus décisionnel markovien est de trouver une bonne « politique » pour le décideur : une fonction π qui spécifie l'action $\pi(s)$ que le décideur choisira lorsqu'il sera dans l'état **s**. Une politique décrit les choix des actions à jouer par l'agent dans chaque état. L'agent choisit une politique à l'aide de la fonction de récompense R. Lorsqu'une politique et un critère sont déterminés, deux fonctions centrales peuvent être définies : **V**, la fonction valeurs des etats qui représente le gain engrangé par l'agent s'il démarre à l'état s et **Q**, la fonction de valeur des états-actions qui représente le gain engrangé par l'agent s'il démarre à l'état s et commence par effectuer l'action a. Les expressions de V et Q, sont ainsi déterminer par les relations de récurrence :

$$V_{k+1}(s) = (1-\alpha)V_k(s) + \alpha[r + \gamma V_k(s')]$$
$$Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha[r + \gamma \max_{a'} Q_k(s',a')]$$

Cette derniere correspond à l'*équation de Bellman* [Bellman [7]] et c'est elle qui est utilisé pour calculer la fonction de coût du reseau de neurone lorsqu'on realise une action. La premiere equation ne peut etre utilisé seul car elle nous donne l'efficassité de l'agent pour une action donnée, mais peut utiliser en cas d'utilisation de reseau adversariaux [Konda and Tsitsiklis [42]]. Dans notre cas, on realise initalement une suite d'evenement avec une matrice de transition aléatoire (reseau non entrainé), puis on calcul de la fonction de perte entre la prédiction des valeurs Q obtenu au cours de l'experience avant l'action et le resultat de l'equation de Bellman Q apres l'action d'un batch d'entrainement donnée, puis on repete N_{cycle} fois cette etapes avant le prochain cycle de reproduction.

Code PyTorch :

```
old_state, action, new_state, reward, DONE = MEMORY
actor = MODEL(old_state)
pred_q_values_batch = torch.sum(actor.gather(1, action),dim=1).detach()
pred_q_values_next = MODEL(new_state)
target_q_values_batch = reward+(1DONE)*GAMMA*torch.max(pred_q_values_next, 1)[0]
MODEL.zero_grad()
loss = criterion(pred_q_values_batch,target_q_values_batch)
```

L'action "acteur" correspond à la sortie du reseau, et correspond à la probabilité d'action optimale. Normalement, à chaque pas de temps, on choisi la valeurs maximale en sortie du reseau et l'on attribu une probabilité d'action aléatoire, c'est le **dilemne exploration-exploitation**. Ce dilemne permet au reseau de neurone d'explorer des parametre et de ne pas se stabiliser dans une valée non optimale. Mais dans notre cas, nous avons choisi d'attribuer une probabilité d'action à chaque "***pas***" de l'entrainement à partir de la sortie du reseau, qui est equivalent à un dilemne d'exploration-exploitation.

Code Python :

```
DILEMNA = np.squeeze(action_probs.detach().numpy())
p_norm = DILEMNA/DILEMNA.sum()
next_action = np.random.choice(self.IO[1], p=p_norm)
```

2.2 Temps longs : Structure neuronale

A partir des différents modele de réseau entrainée à temps courts, l'objectif qui suit est de selectionner ceux qui on eu le meilleurs score d'entrainement, puis de modifier legerement la structure du graph neuronal. Le calcul des scores est relatif à l'environement d'entrainement et est tres utiliser dans les modele evolutif. L'attribution des points sera plus detaillé dans la partie 3. Le principe est le suivant : Lors du premier entrainement à temps court, on a N_r reseau en parallele et distinct, chacun vont suivre le processus d'entrainement vue dans la sous-partie précedante. Ensuite, les N_b reseau ayant le meilleurs score sont selectionné, N_b reseau garde la meme topologie pour le le cycle suivant, N_m reseau par N_b meilleurs reseau hérite d'une mutation alétoire. Enfin N_c nouveau reseau aléatoire sont introduit dans le processus d'entrainement suivant, ceux ci peuvent hériter de parametre optimisé des reseau precedant, mais pas de la structure du reseau, c'est les reseau "compétiteur". Les nombre d'agent sont definit comme $N_r = (N_b + 1)^2$, de facon à ce que N_r est une racine entiere, $N_m = N_b$, tel que le nombre total de mutation soit $N_{mtot} = N_b^2$ et enfin $N_c = N_b + 1$, de cette facon, on a bien $N_r = N_b + N_{mtot} + N_c$. Dans notre modèle, on distingue 5 types de mutations sur le graphe neuronal :

1. Ajouter une connection : Ne change pas le nombre de couche neural, mais ajoute une connection à l'une d'entre elle. Comme on n'ajoute pas de neurone, la liste des connections possible ne change pas et la nouveau noeud va etre connecté aléatoirement à une neurone pre-existant.

2. Ajouter un neurone à l'une des couches : Comme on ajoute une connection de sortie possible, on renouvelle la liste des connection. Mais vue que le reseau doit etre complet, on rajoute également une connection dans l'une des couches qui va se connecter exclusiement à cette derniere connection de sortie.

3. Ajouter une couche 1*1 : L'ajout d'une couche n'est composé que d'une connection d'entrée et d'une seule et unique sortie, ce qu'y est equivalent à un seul neurone. La position de la couche doit etre différent de celle precedante et compris entre $[1, 31]$. L'entrée de cette nouvelle couche se connecte uniquement vers l'arriere (pas forcement ? juste position différente ?), par contre, il n'est pas necessaire que la sortie soit connecté vers l'avant, le reseau etant complet, seul lui meme est interdit. Si ce neurone se connecte vers l'arriere, il devient equivalent à un générateur d'entrée virtuelle. Comme on ajoute une connection de sortie, la liste de connection et une nouvelle connection est ajouté tout comme l'ajout d'un neurone. **(necessite correction)**

4. Enlever une connection doublon : Enlever une connection quelquonque n'a pas été envisagé dans notre cas, car ils changerait radicalement la structure du reseau. En effet, elle supprimerai possiblement plusieurs connection, ce qui rendrait fortement le reseau imcomplet. Par contre, comme le reseau peut etre connecté plusieurs fois à la meme sortie, il est possible d'enlever une connection doublon. Cela ne changerait pas la liste des connections de sortie possible. **(necessite correction)**

5. Enlever un neurone : Cette opération contient deux contrainte, on ne peut pas supprimer un neurone qui correspond au premier lien d'une couche donnée et que le décallage des connections soit possible. Corrolaire des définitions : on ne peut pas supprimer une couche qui ne contient qu'un seul et unique neurone et de meme pour la connection d'entrée. Tout comme l'ajout de neurone, la liste des connections de sortie possible est mise à jours. **(necessite correction)**

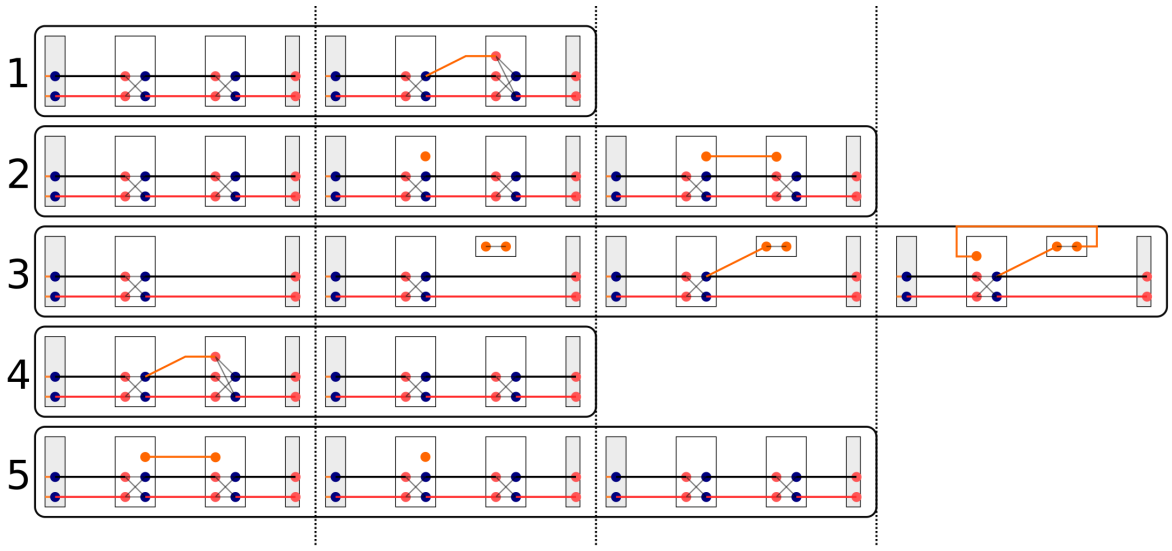

Figure 3: Algorithme visuel des mutations. Respectivement les 5 types de mutations citer en paragraphes. De gauche à droite, l'etats initial vers l'etat final. Les propriétés du graphe sont à titre indicatives, cette representation tres simplifié ne reflete pas l'optimum d'une fonction. Les opérations $1 \leftrightarrow 4$ et $2 \leftrightarrow 5$ sont symetriques, l'operation 3, n'a pas de symetrique stable simple.

Dans notre modele, on conserve au minimum un neurone par couche et toujours le premier, et on ne supprime pas les couches de neurone. Ce choix plus simple permet de conserver la symetrie des opérations $1 \leftrightarrow 4$ et $2 \leftrightarrow 5$ [voir Fig 3], mais aussi de maintenir une structure vestigiale du reseau. En effet, au cours de l'evolution certaine structure sont maintenu, mais ne sont pas les choix optimums por remplir certaine fonction. Par exemple, le chemin

qu'emprunte les vaiseau sanguin entre la tete et le coeur chez les mammifere emprunte un chemin non optimal, pourtant il est maintenu car serait trop couteux evolutivement pour changer de trajectoire [Smith, Burian, et al. [70]]. La vallée de stabilité evolutive lié à ce choix criticable est plus abordé en discussion.

3 Regles du jeu : TAG-GAME

Dans le cadre des processus de decision markovien, les donnée sont généré au cours de l'entrainement, c'est l'**environement** d'entrainement. Cet environement contient des regles et s'apparente à un jeu, la biblioteques la plus utilisé en python est *OpenGym* et sa structure à servi d'exemple pour realiser l'environement de notre probleme. Dans notre cas, on souhaite reproduire certaine stratégie de survie que l'on observe dans le vivant, en particulier, les stratégies de prédations et de fuites d'une proie. On a choisi pour cela le **jeu du chat et de la souris**, ou *Tag game* en anglais, en effet, on alterne entre proie lorsqu'on est souris, et prédateurs lorsqu'on est chat, ce qui pousse le modele à s'adapter à deux configurations différente. Le chats et la souris ont des "couleurs" différentes suivant les états chat/souris/adversaire/agent. Les étapes d'une partie sont les suivantes :

1. Initialement, l'agent est une proie "la souris". Il doit eviter de se faire attraper par le prédateur, "le chat". Le prédateur n'est pas un agent "intelligent", celui-ci calcul uniquement le déplacement optimal qui minimise la distance euclidienne entre le chat et la souris pour l'etat suivant. ***Chat = 1, Souris = 2***.

2. Lorsque l'agent se fait "attraper" par le prédateur, il en devient lui meme un et les role s'inverse. L'agent doit maintenant attraper la proie, la nouvelle souris. La souris cette fois n'est pas un agent "intelligent", celui ci calcul uniquement le déplacement optimal qui maximise la distance euclidienne entre le chat et la souris pour l'etat suivant. ***Chat = 3, Souris = 4***. La nouvelle position des agents/adversaire est calculer comme la collision entre les deux pour ne pas rester bloquer sur la meme position au temps t+1. La collision est définit comme la symmetrie axiale des deux vecteurs de deplacement à l'instant t, soit les deux équations **(à implementer)** :

   - $\begin{cases} Thales \\ Pythagore \end{cases}$

3. On revient à l'etapes 1 si l'agent attrape la souris, la partie s'arrete apres N_g pas de temps.

On a dans ce cas, un agent qui interagit avec un environement de jeu. L'agent contient les informations de **vue** et d'**action** qui va transmettre à l'environment. L'environement contient les information de la position de l'agent et son adversaire, ainsi que les regles de jeu et comptage de points. L'environement peut etre vue comme une grille à limite périodique (CLP) de dimension $N \times M$ où les agents et adversaire sont des automates cellulaires avec des fonctionnement différents. L'agent dispose de 9 entrée "états", 3 sortie "action" et un reseau de neurone entre les deux, où les propriété positionnel entrée/sortie sur la grille sont généré aléatoirement en debut d'experience, puis maintenu à la descendance. Ainsi, l'agent lit 9 case sur une grille 5×5 centré sur la position de l'agent, soit 36% de son environement local, et l'agent se déplace d'une seul case parmit 3 mouvement d'une grille 3×3 centré sur la position de l'agent, soit 33,3% de son environement local. Par contre, l'adversaire voit uniquement la position de l'agent et de lui-meme, mais n'est qu'une fonction mathématique se deplacant suivant 4 trajectoires, haut $(0, 1)$, bas $(0, -1)$, gauche $(-1, 0)$ et droite $(1, 0)$, minimisant ou maximisant la distance entre elle et l'agent suivant son état "chat" ou "souris". La informations positionnel sont discretisé (grille), le calcul du déplacement de l'adversaire revient à :

$$\begin{cases} d_2 = \sqrt{(v_{adv} - v_{agent})^2} \\ m_{vt} = \begin{cases} m_{vt} - min(d_2) & if\ IT \\ m_{vt} + max(d_2) & else \end{cases} \end{cases}$$

Enfin, l'environement attribue des points à chaque étapes du jeu, l'ensemble des points donne le score de l'agent. L'attribution des points à l'agent est un parametre important en apprentissage par renforcement, un desequilibre des points peut rendre soit un agent tres "agressif" ou encore à l'opposé "passif" [Peter Auer 2002]. Pour cela, les points ont été reflechi en fonction de la taille de grille de jeu, pour une grille de jeu 16×16, la moyenne des déplacement pour atteindre le centre quelque soit la position est d'environs 8 déplacement. Dans cette configuration, pour un calcul equilibré, les comptages des points sont les suivants :

- -1 par “pas” de temps où l’agent est un prédateur “chat”
- +10 si l’agent attrape la proie “souris” et devient la proie à son tour.
- +1 par “pas” de temps où l’agent est une proie “souris”.
- -10 si l’agent est attrapé par le prédateur “chat” et devient le prédateur à son tour.

Corrolaire : L’agent restant le plus longtemps sous l’état proie (souris) et le moins longtemps sous l’état prédateur (chats) à le plus grand score de jeu.

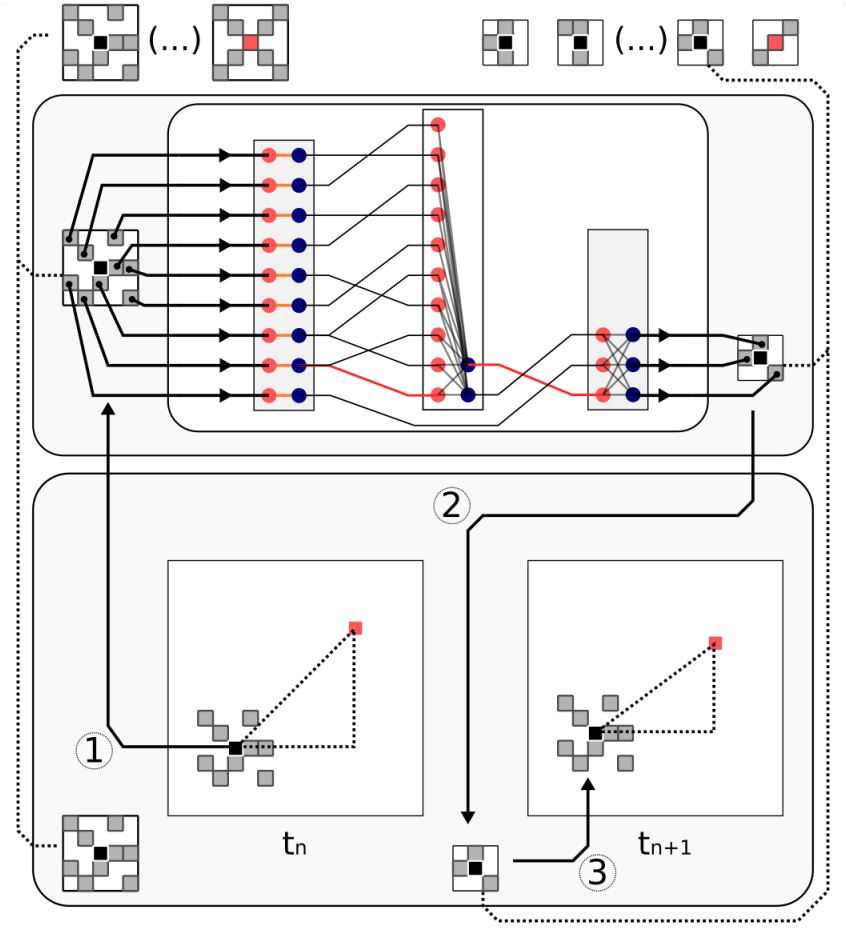

Figure 4: Les interactions entre un agent et un environement de jeu. En haut, la génération alétoire des positions relative possible des entree et sortie. Suivi des interactions entre l’agent en haut et l’environement de jeu en bas. 1 : envoi de l’environement des informations d’entrée (pixel) à l’agent; 2 : envoi de l’information de sortie de l’agent (mouvement) à l’environement; 3 : mise à jour de la carte de jeu, l’adversaire effectue son déplacement à ce moment.

Les case des positions relative d’entrée et de sortie sont défini aléatoirement, mais comme le pourcentage des case utilisé sur la grille est au alentour de 35%, les combinaisons possibles sont de l’ordre factoriel. Pour la sortie, on a 3 positions parmi un total de 9 cases, ce qui donne $\binom{9}{3} = 84$ et comme le probleme est symetrique au 4 mouvement de l’adversaire (exemple : $(1,2,3) \iff (7,8,9)$), le nombre de combinaison est de l’ordre du nombre du nombre d’agent par génération. Par contre, pour l’entrée, on a 9 positions parmi un total de 25 cases, ce qui donne $\binom{25}{9} = 2042975$ ce qui pose un probleme de convergence de la vision optimal. Mais, comme les quadrant d’observation sont equivalent, on peut considerer uniquement les 9 cases par quadrant, ce qui nous donne

par ratio de 36% $\begin{pmatrix} 9 \\ 3 \end{pmatrix} = 84$, cela est presque un ordre au dessus du nombre de génération et ne pose moins probleme de stabilité. Les cases ayant obtenu les meilleurs scores sont classé dans l'ordre et l'on attribu la somme des points par case sur les grilles d'observation et d'action. La normalisation de la grille donne la probabilité de choisir l'une des cases pour les etapes suivantes des "challenger" et sera plus abordé dans la partie résultats. Pour les action, on observe quelque cas typique : 3-cyclique (Exemple : $(1, 6, 8) \Rightarrow (\nwarrow, \downarrow, \rightarrow)$), 4-cyclique (Exemple : $(1, 3, 8) \Rightarrow (\nwarrow, \downarrow, \downarrow, \nearrow)$), semi-2-cyclique (Exemple : $(2, 6, 8)$), asymétrique (Exemple : $(1, 2, 3)$) et statique (Exemple : $(1, 5, 8)$). Les positions des entrée et des sorties sont donc aussi selectionné dans le processus evolutif des agents.

Résultats

Pour générer l'ensemble des données experiementale, il est necessaire d'avoir plusieurs agents par génération pour la convergence de la densité de probabilité des positions d'entre/sortie et plusieurs génération pour verifier s'il y a une convergence des structures neuronales. Pour que le nombre d'agent par génération soit de l'ordre du nombre de combinaison d'entree/sortie, nous avons choisi 25 agents par cycle de reproduction. 25 est le carré de 5, ce qui compatible avec l'algorithme de mutation de la structure neuronale. Ce qui donne, 4 meilleurs agents par cycle, ceux ci sont maintenu au prochain cycle et chacun recoit 4 mutation, ce qui fait 20 agents ayant des propriétés analgogue à la génération precedante, les 5 restants sont des nouveaux agents avec des structures neuronale aléatoire, mais avec informations d'entrée/sortie hérité de la densité de probabilité des evement precedant. Les grilles de jeu sont de 16×16, ce qui permet à l'agent de pas voir tout le plan de jeu, mais ne pas que l'adversaire soit trop loin lorsque l'agent est un prédateur. L'avantage de cette configuration est que l'on peut representer l'ensemble des environement à un temps donnée par une grille $(5*16) \times (5*16)$, ce qui facilité la visualisation et l'interpretation des resultats. Il est possible aussi de representer cette grille par décomposition de deux nombre premier si le nombre d'agents par cycle n'est pas le carré d'un nombre. Cette methodes correpond à la verification de la parité puis recurrance des nombre impaire jusqu'a la racine entiere supérieur du module du nombre d'agent par génération : $n\% \begin{cases} 2 \\ impair(\sqrt{n}) \end{cases}$. Cette derniere n'a pas été utilisé ici, mais est inclu dans le programme.

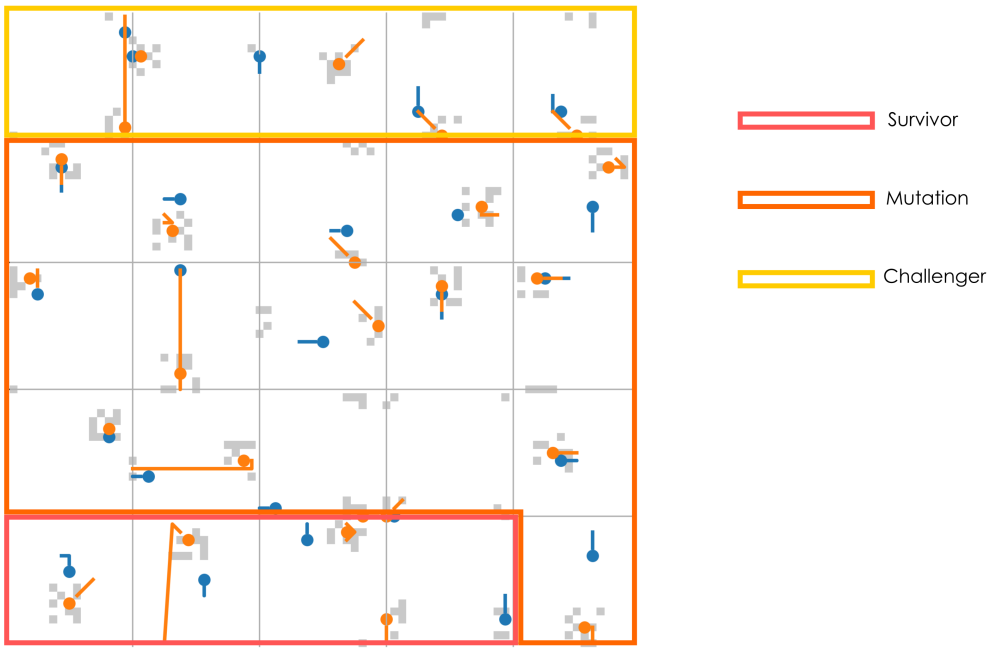

Figure 5: Representation d'une donnée experimentale à un instant t. En encadré, les différentes catégories d'agent (survivant, mutation et compétiteur). En points bleu, les adversaires et en orange, l'agent intelligent. Les cases grisé corresponde à la vue de l'agent à un instant t.

Le nombre de génération n'est pas clairement défini, mais peut, tout comme le nombre d'agent par génération, etre justifié par la loi des grands nombres, si l'on considère que la loi de proabilité à une esperance. On va donc verifier, pour 25 agents par génération, combien, il faudrait pour que la densité moyenne des entrée et des sortie converge. Soit $N_a = 25$, le nombre de d'agent par génération, $N_c = 84$ le nombre de configuration de combinaison maximal entre entrée et sortie et N_g le nombre de génération que l'on cherche. Soit X_n, la variable aléatoire indépendante nous donnant une combinaison à chaque tirage, la limite à l'infini de l'inégalité de markov devient :

$$\lim_{n\to+\infty} P\left(\left|\frac{1}{N_g}X - p\right| \leq \varepsilon\right) = 0$$

De cette facon, la moyenne est un estimateur de l'esperance. On trouve pour un ε de 1% et d'un calcul par itération directe, une valeurs de 10 générations pour que la moyenne converge. Cette approximation n'est pas valable pour la convergence du reseau de neurone, ainsi, on considere que les données experiementale seront exploitable qu'a partir de $N_g > 10$.

4 Caractérisation de la convergence de la densité positionnelle des entrées/sorties

L'entrée de l'agent est définit comme la combinaison de 9 cases parmis 25, sa sortie est défini comme la combinaison de 3 cases parmis 9 [Section §3]. On cherche à verifier l'existance ou non des configurations "vue" et "déplacement" optimales au probleme du jeu du chat et la souris. Dans notre cas, à chaque cycle de reproduction, on ordonne les scores pour chacun des agents, et on additionne les scores obtenu par case de la grille de l'agent sur la grille complete. Comme le probleme est symetrique suivant les 4 directions, on considere les 4 rotation de $k\pi/2$ dans le comptage des points. On réalise cela par le produit matriciel des positions centré avec la matrice de rotation $\begin{pmatrix} x_0 + x_c & y_0 + y_c \\ \vdots & \vdots \\ x_n + x_c & y_n + y_c \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$. La normalisation de la grille nous donne la proportion des meilleurs scores obtenus des évenement aléatoire. Si l'on considère que la distribution suit une loi avec esperance, alors le

théoreme de transfert nous permet d'exprimer l'espérance d'une fonction d'une variable aléatoire X_n en fonction d'une intégrale convergente. Numériquement, il correspond au methode de *Monte-Carlo* [Metropolis and Ulam [51]], décrit comme :

$$G = E[g(X)] = \int g(x) f_X(x) dx$$

A partir de ce calcul, on obtient la densité de probabilité de succes lorsqu'on génère une nouvelle grille d'entrée "vue" et de sortie "déplacement". Par la loi des grands nombres, on a vue que cette densité convergent au moins à partir de 10 cycle de reproduction. On peut ainsi representer ces deux densités sur des grilles à deux dimensions.

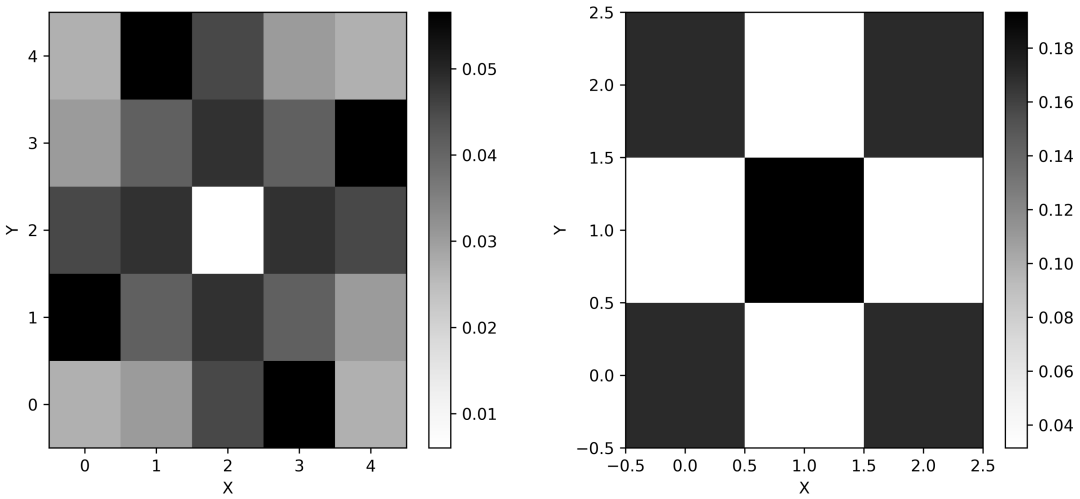

Figure 6: Mesure de la densité positionnel de succes des entree/sorties. A gauche, l'entree correspondant à la vue de l'agent et à droite la sortie, correspondant au mouvement relatif de l'agent. En nuance de gris le niveau de probabilité d'avoir plus de succes avec cette case.

On obtient pour l'entrée et la sortie deux distributions spatiale différente [voir Figure 6]. Pour l'entrée on observe 4 positions dominantes et 1 position centrale faible. Pour la sortie, on observe 4 positions en diagonal forte. En entree, l'ensemble des positions $(0,1)$, $(1,4)$, $(3,0)$ et $(4,3)$ à la probabilité la plus forte et est symétrique à l'ensemble $(1,0)$, $(0,3)$, $(4,1)$ et $(3,4)$. Cet ensemble d'entrée correspond au position au quasi-extremité de la grille 5×5, ce qui pourrait permettre d'anticiper le mouvement de l'adversaire. Ensuite, les positions d'entree $(1,2)$, $(2,3)$, $(2,1)$ et $(3,2)$ corresponde au case qui entoure le centre de la grille, celle-ci permettent de "voir" l'adversaire lorsque celui ci est au plus proche en terme de mouvement. De plus, ces dernieres prolonge les cases en diagonale precedante. En sortie, on l'ensemble des position $(0,0)$, $(0,2)$, $(2,0)$ et $(2,2)$ ont par somme, la probabilité la plus forte. Cette ensemble de sortie correspond au position en diagonale. On remarque que les positions qui entoure le centre $(0,1)$, $(1,0)$, $(1,2)$ et $(2,1)$ ont les probabilités les plus faibles. Ces deux observations sur la sortie impliques que le mouvements est privilégié en diagonale et semi-2-cycliques. Ces cases pourrait etre privilégier car l'adversaire n'a que 4 deplacement $(\uparrow, \downarrow, \leftarrow, \rightarrow)$ non diagonaux, ce qui donne un avantage en cas de déplacement diagonal. Par contre, l'absence de symetrie complete des mouvements de l'agents ne semble pas etre un désavantage pour l'obtention d'un meilleurs score.

On remarque que les positions optimales en entrée pourrait etre corrélé avec les mouvements optimaux de sortie. En effet, les mouvements diagonaux necessite de savoir si un adversaire est positionné à sa diagonale avant que l'adversaire est fait le mouvement à $t+1$. Par exemple, si l'agent peut faire les mouvements $(1,6,9) \Longleftrightarrow (\nwarrow, \emptyset, \searrow)$, si l'adversaire est en position relative $(3,0)$, alors en $t+1$, l'adversaire sera en $(3,1)$, si l'agent fait le mouvement $(\searrow)$, dans ce cas, il sera à la meme position que l'adversaire ce qui lui fera perdre des points, alors qu'il sera à la position a plus eloigné si il fait le mouvement $(\nwarrow)$. Pour vérifier la correlation entre l'entrée et la sortie, on peut regarder la convergence des familles de densité des entrées et sorties. Pour cela, on mesure la densité projeté suivant une seule dimension spatiale entre $t = 0$ et $t = N_g$, ainsi que sa variation entre chaque génération.

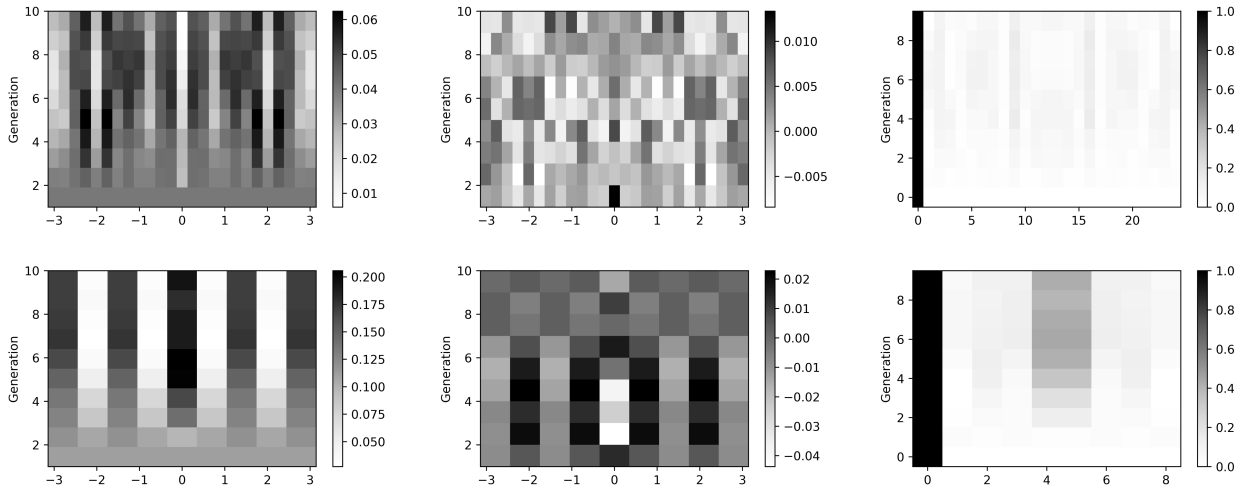

Figure 7: Mesure de la densité positionnel projeté de succes des entree/sorties en fonction du temps. À gauche, l'entrée et à droite la sortie. En haut, la densité de probabilité avec en nuance de gris le niveau de probabilité d'avoir plus de succes avec cette case. Au milieu, la variation de probabilité avec en nuance de gris le niveau de variation. En bas, les frequences harmoniques au cours du temps.

On obtient l'evolution de la distribution spatiale des entrée et des sorties [Voir figure 7]. On observe que pour l'entrée, certaine position se stabilise à partir de la 4eme génération, mais qu'il peut y avoir une alternance des positions en quasi-limite avec des positions plus centrale. Par contre, pour la sortie, les positions sont plus stable et se stabilise completement à partir de la 5eme générations, la variation de la distribution devient quasi-homogène et semble tendre à etre constant. La variation de l'entrée est bien plus importante, on n'atteind pas d'homogénéité, meme apres 10 cycles, ce qui semble montrer qu'il y a des positions dominante, mais celle-ci oscilleront tout le temps entre plusieurs position : la suite d'entrée est semble etre semi-alternée convergente. La convergence de la suite semi-alternée pour chaque position d'entrée u_n est verifié par le rayon de convergence R_n du critere de Leibnitz :

$$u_n = (-1)^n \epsilon_n \quad \Longleftrightarrow \quad R_n \leq |a_{n+1}|$$

Comme la densité de sortie semble etre un invariant, et que l'entrée oscille entre quelques position d'instabilité, on peut considerer qu'il n'influencera pas trop fortement le processus de selection de reseau une fois stabilisé. De meme l'ecart frequenciel des positions defini par la transformée de fourier "$f_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} jk}$" se stabilise (pas sur). Ainsi les challenger qui hériterons de la densité de probabilité d'I/O ne devrait pas influencer leur victoire par leurs propriétés d'entrée/sortie, mais de la nouvelle structure aléatoire. Cette hypothèse forte, sera vérifié dans les parties suivantes et plus discutté en fin de partie.

5 Caractérisation de la connectivité du reseau de neurone

Le reseau de neurone est défini comme un graphe orienté acyclique [1.2]. Comme celui-ci est généré aléatoirement, il est necessaire d'explorer l'etudes de certain parametre de graphes pour caractériser s'il existe ou non des propriétés dominantes au cours de la selection.

Contrairement à la mesure de la densité, au cours de la selection, les agents ayants les meilleurs scores sont selectionnées, mais, les nouvelles structures sont defini aléatoirement comme à l'instant initial. Les agents selectionnée subissent 5 mutations aléatoire, on cherche à mesurer l'effet de la mutation sur la structure du reseau et la selection. Pour cela, on peut representer l'ensemble des matrice adjacente ayant eu le meilleurs resultats au cours des générations. Le graphes etant dirigée acyclique, il n'y a pas de probleme de *clique* à prendre en compte.

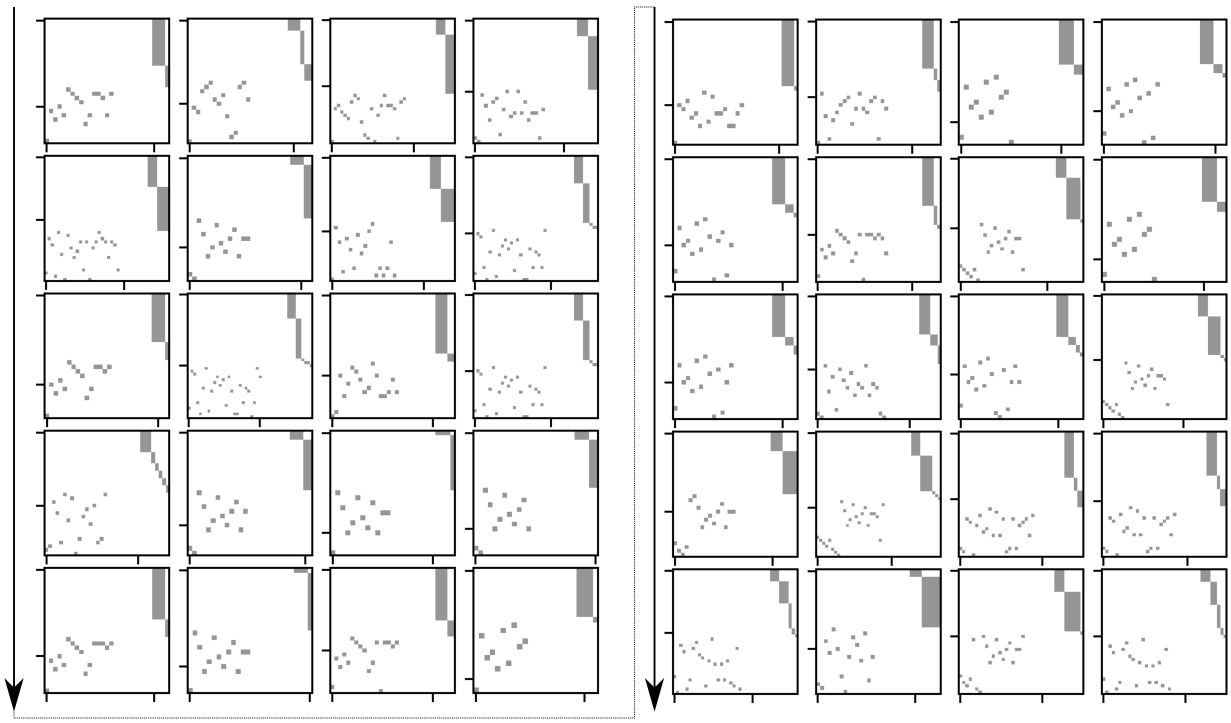

Figure 8: Evolution des reseaux des 4 meilleurs agents par générations. A gauche, de la premiere génération à la 5eme et à droite, de la 5eme à la derniere génération.

On obtient ainsi un ensemble de grille de matrice adjancte de taille $N_b \times N_g$, avec N_g, le nombre de génération et N_b, le nombre de meilleurs agents par génération. On n'obsere pas de structure tres organisé dominante au dixieme cycle, le reseau ne semble pas convergé vers une structure d'equilibre. Pour autant, la matrice adjacente n'est pas suffisante pour avoir les informations des graphes, il est necessaire de verifier certain parametre si l'on veut caractériser le reseau. Pour cela, on va comparer les **indicateurs de centralité**, tel que les *"degrées", "proximité/excentricité",* "intermédiarité" et "*centralité propre"* entre les 4 meilleurs reseaux et les 4 pires reseaux au cours du temps.

Le degré (ou valence) d'un sommet d'un graphe est le nombre de liens (arêtes ou arcs) reliant ce sommet, avec les boucles comptées deux fois. Il est défini pour un graphes orienté comme (...), tel que $deg(s) = d^+(s) + d^-(s)$. L'intermédiarité, *betweenness* en anglais, est une mesure de centralité d'un sommet d'un graphe. Elle est égale au nombre de fois que ce sommet est sur le chemin le plus court entre deux autres nœuds quelconques du graphe $g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$ (...) . La proximité, *closeness* en anglais, d'un nœud x est défini comme la somme des distances à tous les autres nœuds, et la proximité est définie par Bavelas comme l'inverse de l'éloignement $C(x) = \frac{1}{\sum_y d(y,x)}$. Enfin la centralité propre, ou *eigencentrality* en anglais, est une mesure de l'influence d'un noeud dans un réseau et est défini par l'equation du vecteur propre de l'ensemble des voisin $x_v = \frac{1}{\lambda} \sum_t x_t$.

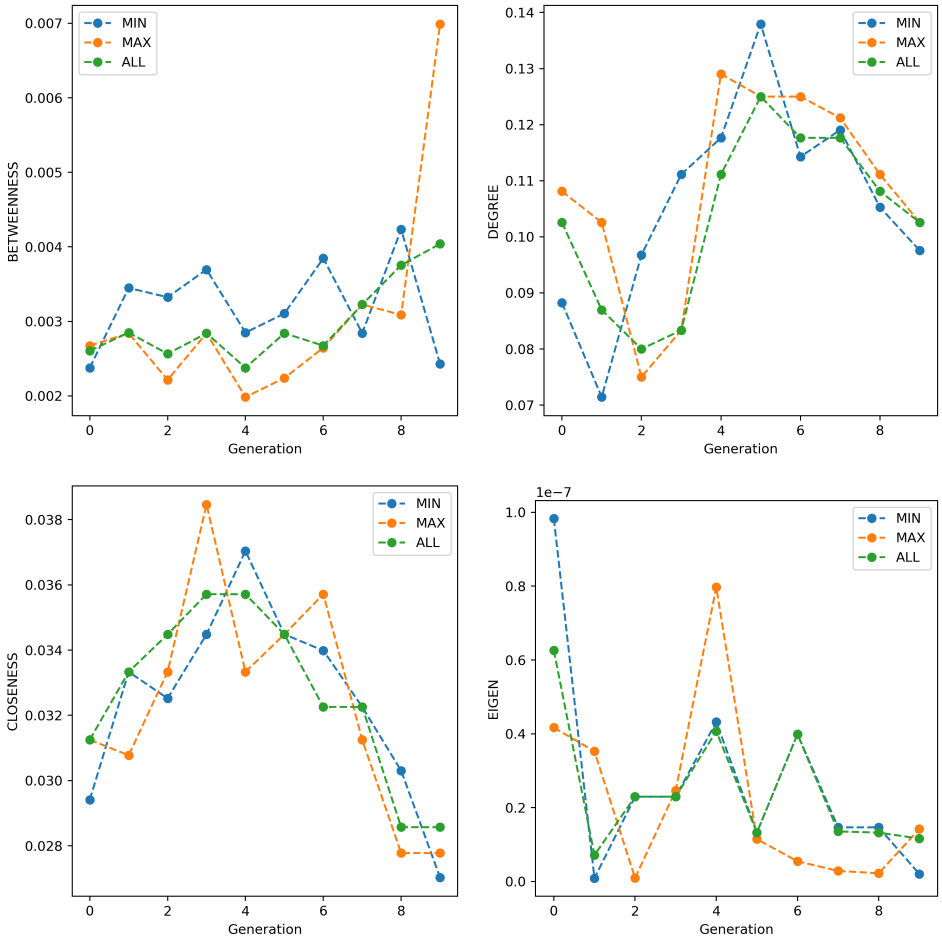

Figure 9: Comparaison des indicateurs de centralité entre les meilleurs et les pire agents. Pour chaque graphe, en abscisse le temps, en ordonnée le parametre “degree”, “betweeness”, “closeness” et “eigen”. Attention, les points sont tres proches, les effets de courbures sont accentué.

Ces différent indicateurs nous donnes l’evolution de la centralité des graphes au cours des génération. Elles sont des mesures censées capturer la notion d’importance dans un graphe, en identifiant les sommets les plus significatifs indépendament des algorithme d’apprentissage. On obtient ainsi 4 courbes temporelles. On remarque que les courbes de centralité propre et de degrée n’ont pas d’impact significatif. Par contre on observe 2 tendances différentes suivant l’indicateurs de proximité et de d’intermédiarité : L’intermédiarité augmente, mais la proximité diminu pour les agents ayant obtenu le plus grands score. Ces deux indicateurs sont plus lié à la structure des chemins qu’au flux de reseau.

Ces deux observations implique que les reseaux les plus optimaux au dernier instant sont plus efficasse si le chemins entre l’entrée et la sortie est le plus longs possible, mais avec plus de proximité entre couches. Pour autant, les données sont tres comprimé, ces indicateurs ne sont probablement pas tres correlé avec l’efficassité du reseau sur le long termes.

6 Caractérisation de la convergence évolutive

Les deux observations precedante nous ont montré qu’il y a une correlation entre la convergence des entrée et des sorties, mais qu’il n’y a pas de convergence significative de la centralité du reseau de neurone. Néanmoins, il est possible que la structure du reseau de neurone se conserve et est un impact sur l’efficacité de resolution du probleme du jeu du chat et la souris. Pour cela, il est necessaire de suivre l’arbre des relation de parenté entre les agents entre

génération, c'est la phylogénése. Chacun des nœuds de l'arbre représente l'ancêtre commun de ses descendants, soit une "fratrie" de taille N_b. On associe à chaque noeud la valeur du score obtenu, puis on mesure la distance qui sépare le premiere génération mere à la derniere génération pour chaque branche du reseau.

Cette approche consiste à resoudre un probleme de plus court chemin d'un graphe orienté acyclique. Tel qu'on minimise $\sum_{i,j \in A} w(i,j)x_{ij}$, avec $w(i,j)$, le cout de l'arc (i,j). L'approche permet de quantifier le nombre de génération où une famille d'agent arrive à etre conservé au cours du temps et à partir de combien de génération, on observe un agents qui est le plus efficace quelque soit l'ajout de nouveau challenger dans le processus evolutif.

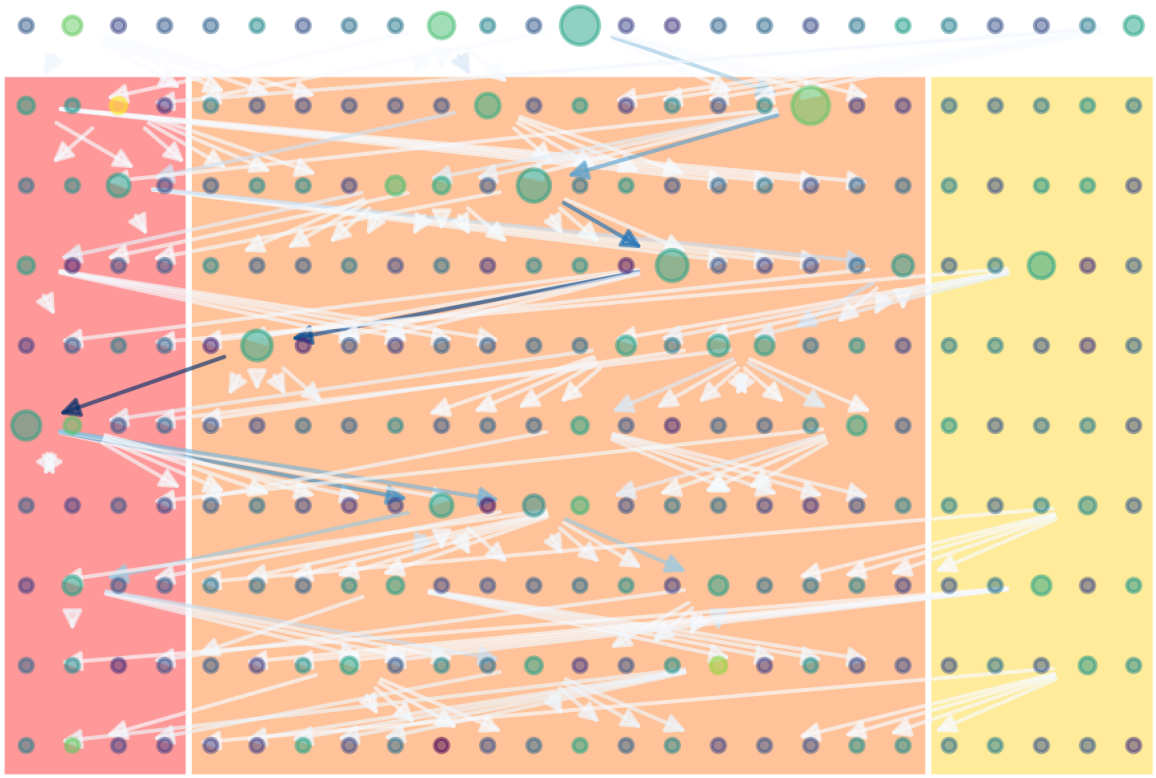

Figure 10: Arbres phylogenetique des agents en fonction de leurs score. De haut en bas, les generation successive. Les couleurs des noeud corresponde au score de l'agent à la fin de l'experience, leurs taille est proportionnele à leur nombre de descendance. En rouge le meilleurs scores. Les encadrés corresponde au catégorie d'agents : survivant, mutant et compétiteurs. (améliorer la representation graphique)

En sommant l'ensemble des plus courts chemin, on obtient un reseau de flot, où chaque arête correspond à celle qui a mener au plus de descendance. On observe qu'un seul agent de la premiere génération a reussi à se maintenir jusqu'a la derniere génération, celui ci a meme été le meilleurs de la génération 5eme génération et deuxieme de la génération suivante, ce qui l'a fait maintenir deux fois sans mutations. Un agent "challenger" de la 4eme génération a reussi a se distinguer et se maintenir jusqu'a la derniere génération. Seul deux agents de la 4eme générations se sont maintenu jusqu'a la derniere, mais ceux-ci domine à 50% la population d'agent à la derniere. Meme apres la stabilisation des densité positionnel d'entrée et de sortie, aucun agent challenger n'arrive à se maintenir plus d'une génération. On remarque que l'agent ayant eu le meilleurs score de l'ensemble des parties est apparu à la 2eme générations, mais celui-ci n'a pas tenu plus d'une génération ensuite.

On a donc 2 populations à la génération finale, pourtant, on aimerait savoir par quel moyen, ces agents ont reussi à ce maintenir autant. Est-ce qu'il y a une stategies mis en place par les agents pour rester le plus longtemps proie, et le moins longtemps prédateurs ? Est ce qu'il y a une strategie qui converge au fil des génération ? La convergence évolutive est le résultat de mécanismes évolutifs ayant conduit des espèces, soumises aux mêmes

contraintes environnementales, ici le jeu du chat et la souris, à adopter indépendamment plusieurs traits semblables. Dans notre cas, pour deux reseau de neurone different, on aurait 2 stratégie similaire. Pour cela, on va mesurer pour les deux groupes agents de la derniere génération ayant le plus grands ancetre, les mouvement recurant des agents par la dispersion des trajectoire et la périodicité des actions.

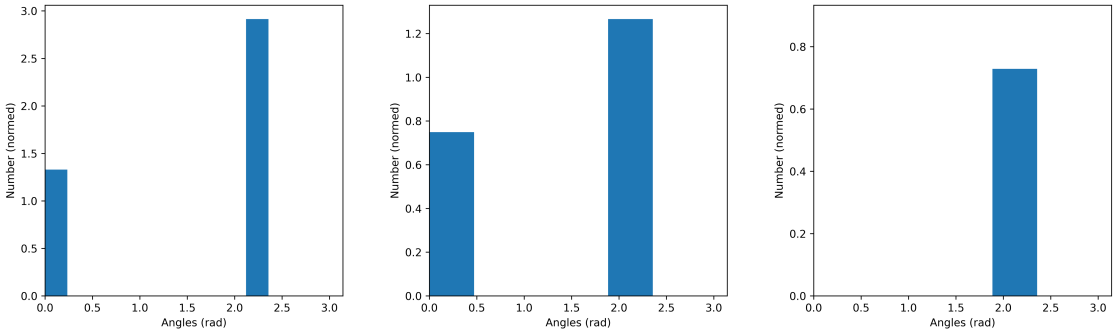

Figure 11: Convergence des strategie en fonction des embranchement les plus long : mesure de la dispersion moyenne, position moyenne et schéma recurrance (mouvement repeter de l'agent). De gauche à droite, de celui qui a le plus d'ancetre à celui qui en a le moins.

On mesure l'angle des vecteurs de deplacement par l'arc tangente arctan($\frac{x}{y}$). Ces mesure montre qu'il y a convergence des mouvements des meilleurs agents, où l'on observe que les agents oscillant entre mouvement statique (30%) et mouvement en diagonal (70%) est favorisé au cours de l'evolution. Néamoins, nous sommes en conditions de limites périodique ou l'adversaire ne se déplace que sur les quatres directions ($\uparrow\downarrow\leftarrow\rightarrow$), ce qui donne un nette avantage à n'importe qu'elle reseau ne se déplaceant qu'en diagonale. D'autant plus que la condition limites periodique $mod(v, N)$ permet d'envoyer l'agent à l'opposé de l'adversaire sans que lui puisse le suivre car ne peut franchir la limite.

Ces dernieres observations montre bien qu'il y a une stratégie dominante, mais qu'un reseau quelquonque suffit pour resoudre se probleme, il est necessaire de modifier les regles du jeu pour avoir un probleme plus optimal :

- Déplacement "collision" apres changement d'états (évite stabilisation).
- Pénaliser l'agent si répétion du meme mouvement N fois (à définir) .
- Pénaliser "legerement" l'agent s'il franchit la limite périodique de la grille. (probleme : l'agent doit voir limite : si on pose un valeurs au bord, c'est une condition limite semi-periodique)

Discussion

L'etudes de l'apparition de la vie et de la resolution des problemes d'apprentissage font partie de deux domaines tres différents en science. Pourtant, ces deux domaine ont souvent été impliqué l'un dans l'autre, l'un dans l'analyse des données expériementales de probleme de regression, classification et partitionnement (correlation de gene, transcriptome, etc, [Abeywardena [1]]), l'autre dans l'inspiration pour creer des modeles d'apprentissage lié au reseau de neurone. Ici on a vue un probleme qui lie les deux, comment les reseau de neurones peuvent se structurer au cours de l'evolution ? Cette problematique permettrait de modeliser l'apparition de vie intelligente, tout en creant de nouveaux outils pour l'apprentissage non supervisé.

CONTRAINTE GÉOMÉTRIQUE : Néamoins cette études peut comporter plusieurs contrainte limitante. La premieres est directement lié à la géométrie des problemes d'automates cellulaire dans des grilles 2D. Lorsque les bord du haut sont relié à ceux du bas, et ceux de gauche, sont relier à ceux de droite, on obtient des conditions limite périodique de type géométrique torique. Cette géométrie simple à mettre en oevre comporte plusieurs probleme comme (...) et dont on observe les limites de modelisation en biologie moélculaire [Cheatham, Miller, et al. [16]]. Une autre facons, serait soit d'avoir des d'autre lient entre les limites, mais cela changerait completement la symetrie

du probleme, tel que d'avoir une grille avec des limites décallés comme un pavage, ou encore d'avoir un rayon de limite, de facons à ce que le probleme corresponde à un cercle, plutot qu'a un carré.

VALLÉE DE STABILITÉ LIMITÉ : L'autre contrainte est lié au modele de construction du reseau : il ne peut y avoir de suppression de couche de neurone. Ce choix est motivé par le fait de ne pas destructurer completement la topologie du reseau à chaque suppression de couche. Mais en faisant cela, on se peut se retrouver avec des couches avantageuse à une génération donnée, mais inutile voir desavantageuse à une génération futur. A cette génération, il est plus probable que l'agent soit plus efficasse qu'un agent généré aléatoirement, on se retouve avec un probleme connu en biologie evolutive : l'apparition de structure vestigiale [Fowler, Roush, et al. [23] and Smith, Burian, et al. [70]]. Ces structures ne sont pas les plus optimale pour atteindre le potentiel du probleme "physique" la plus bas, mais leurs suppression necessiterai trop de mutation, ce qui ferait sortir l'agent d'un niveau obtenu par le reseau de neurone.

COMPORTEMENT DE TRICHE : Malgrés cela, on a obtenu une stratégie convergente au cours du processus evolutif et cela pour deux reseau de neurone différents. Néamoins le faite que le probleme soit de condition limite periodique et que l'adversaire ne peut pas se déplacer en diagonale, on se retrouve avec un artefact de trajectoire. En effet, l'agent est favorisé s'il fuit la grille suivant une trajectoire diagonale. Ce critere est donc à prendre en considération pour ne pas surinterpreter les resultats. Cette observation de "triche" du reseau de neurone dans le cadre d'apprentissage par renforcement n'est pas quelque chose d'inconnu. En effet, plusieurs modeles, comme le NEAT peuvent générer des structures qui exploite les failles d'un jeu, comme le cas ou l'agent tourne en rond entre l'arrivé et le départ dans un jeu de course, ou encore l'araignée qui marche à l'envers pour optimiser sa marche [Brockman, Cheung, et al. [13]]. Dans le cas positif pour la detection de parametre non pensée, on parle alors de raccourci [Geirhos, Jacobsen, et al. [27]]

VERS UNE SYSTÉMISATION : L'optimisation de la structure du reseau est un probleme qui pourrait etre plus envisagé dans l'avenir des reseaux de neurones [Leijnen and Veen [43]]. Comme on l'a vu, il existe déjà des modèle qui optimise la structure comme NEAT, mais il en exite d'autre. Pour ne citer que lui, il y a le modele GNN, *Graph Neural Network*, celui-ci est par contre reservé à l'analyse des graphes. Le modele GNN va construire un sous-graphe resultant de la convolution de plusieurs graphes à étudier, ce sous graphes correspond au reseau de neurone le plus optimisé pour donner les informations d'un graphes de grande taille [Zhou, Cui, et al. [83]]. Dans notre cas, nous avons utilisé un processus evolutif qui va changer les connections entre couches uniquement, pour n'importe qu'elle type d'information d'entree, meme les graphes par vectorisation de Louvain [Bhowmick, Meneni, et al. [9]]. Néamoins, il est possible de rendre notre modele modulaire, en effet, il peut-y avoir un niveau supplementaire de connection si celui ci contient plus de fonction, ou que les entrée change au cours du temps, comme par exemple le cas de l'etude l'evolution de la morphologie [Joachimczak, Suzuki, et al. [36]]. On aurait plusieurs fonction defini evolutivement via notre modele, ceux ci serait lié à des sortie d'action, mais aussi quelque connection entre module neuronaux, elle meme générer par un processus evolutif similaire. L'ensemble formerai ce qu'on pourrait appeller un système.

VERS DES MODELS COMPLETS : Notre modele repond à la question : Comment constuire un reseau de neurone où aussi bien la structure que le processus d'apprentissage pour que soit fonctionnalisé ? Cela reste dans le cadre d'un systeme capable d'apprentissage évolutif, mais la pour le completer, il faudrait que le systeme s'auto-entretienne, et pour cela, il faudrait ajouter la notion de nourriture/nutriment. De plus, nos agent sont indépendant les uns des autres, ce qui ne reproduit pas ce qu'il se passerait dans un écosysteme réel. On pourra ainsi avoir une mesure de la valeurs selective, ou ***fitness*** en anglais [Orr [59]]. Pour completer l'approche, on faudrait creer un environment ou l'ensemble des agents y serait contenu et où leurs survie et reproduction serait inclu dans les regles du jeu. On peut penser le cas, ou il y a des agents et de la nouriture distribuer aléatoirement avec des "taille/ressousce" différente. Lorsqu'un agent prend de la nouriture, il gagne en taille jusqu'a sa limite, et s'il double en quantité de ressource, il va engendrer une descendance avec une mutation. Si l'agent rencontre un autre agent plus grand que lui, celui ci peut lui faire perdre des ressources jusqu'a sa disparition et réciproquement. L'environment contiendrait un certain nombre limité d'agent, ou lorsque se nombre diminu, il y a une certaine probabilité de génération spontanée d'agent contenant certaine propriété optimal des agents precedant, et si celui ci dépasse, un des agents avec le moins de succes evolutif meurt aléatoirement, on qualifierai cela d'accident. Le nombre de ressource dans l'environement apparaitrait par génération spontanée et serait plus probable que la génération spontanée d'agent. Ce type de modele semble favoriser la compétition du plus fort, il serait interessant de voir si l'on observe des comportements de coopération émerger. Aussi, on pourrait ajouter de la complexité au modele en rajoutant des regles connu en science sociale, comme la dynamique de ségregation lié à l'erreur fondamental d'attribution [Schelling [68] and Jones and Harris [37]], les mouvement collectif et l'exploration [Vicsek, Czirok,

et al. [78]], le dilemme du prisonnier lié à la coopération [Press and Dyson [61] and Stewart and Plotkin [72]], les contrainte géographique et de langues [Wei [81] and Marshall [48]], les attachements entre parent-enfant [Bretherton 1992], ou encore, l'héritage culturel [Bourdieu [12]]. Ces outils pourrait etre interessant pour des simulations de vie et pour mieux comprendre l'emergence des structure sociale [Henaff [30]].

VERS UNE GÉNÉRALISATION : Pour aller plus loin, notre modele de structuration neuronal peut etre utiliser dans le cadre d'un probleme plus pratique et plus simple : la fonctionnalisation de la classification. En effet, il est possible d'utiliser une version β implémenté et modulaire de notre modele dans le cas d'un probleme d'analyse de données. Les nombres de reseaux aléatoire par processus d'apprentissage sont appellé des graines, à l'image des methodes de partitionnelent. Celui ci s'utilise de cette facons en Python :

```
import functional_fillet as ff
# init
model = ff.model(BATCH_SIZE, NB_GEN, NB_SEEDER)
# training
model.fit(DATA, LABEL)
# predict
result = model.predict(DATA_NEW)
```

L'avantage de cela est qu'on peut aussi comparer l'efficacité de notre reseau evolutif "structuré" avec un reseau de neurone classique predefini. Un base de donnée simple et qui ne dispose pas d'un nombre elevé d'etiquette est le jeux de donnée MNIST, adapté pour la reconnaissance de chiffre sur une image. Il sera possible ensuite d'analyser l'effet de différent parametre comme la taille du "batch" et la taille du nombre de graine par génération. Il sera aussi possible d'etudier la vitesse d'apprentissage d'un reseau : Est ce qu'un reseau tres adapté à un probleme apprend vite au debut, mais atteint un plafond moins optimale qu'un reseau moins adapté au debut, mais à la suite devient plus performant ? Cette question de comparasion entre le cycle d'hérédité et cycle d'entrainement est un probleme ouvert et pourra etre abordé dans le cadre d'un futur projet.

References

[1] V. Abeywardena. "An application of principal component analysis in genetics". In: *Journal of Genetics* 61.1 (July 1, 1972), pp. 27–51.

[2] M. A. AIZERMAN. "Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning". In: *Automation and Remote Control* 25 (1964), pp. 821–837.

[3] O. Y. Al-Jarrah, P. D. Yoo, et al. "Efficient Machine Learning for Big Data: A Review". In: *Big Data Research.* Big Data, Analytics, and High-Performance Computing 2.3 (Sept. 1, 2015), pp. 87–93.

[4] M. Anctil. *Dawn of the neuron: the early struggles to trace the origin of nervous systems.* OCLC: 909956847. Montreal ; Kingston ; London ; Chicago: McGill-Queen's University Press, 2015. 1 p.

[5] J. Barkow, L. Cosmides, et al. "The Psychological Foundations of Culture". In: *The Adapted Mind: Evolutionary Psychology and the Generation of Culture,* (Jan. 1, 1992).

[6] S. Bartlett and M. L. Wong. "Defining Lyfe in the Universe: From Three Privileged Functions to Four Pillars". In: *Life* 10.4 (Apr. 2020). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 42.

[7] R. Bellman. *THE THEORY OF DYNAMIC PROGRAMMING.* Section: Technical Reports. RAND CORP SANTA MONICA CA, July 30, 1954.

[8] E. R. Berlekamp, J. H. Conway, et al. *Winning Ways for Your Mathematical Plays.* 2nd ed. New York: A K Peters/CRC Press, Mar. 1, 2003. 212 pp.

[9] A. K. Bhowmick, K. Meneni, et al. "LouvainNE: Hierarchical Louvain Method for High Quality and Scalable Network Embedding *". In: *the 13th ACM International WSDM Conference.* Houston, United States, 2020.

[10] H.-H. Bock. "Clustering Methods: A History of k-Means Algorithms". In: *Selected Contributions in Data Analysis and Classification.* Ed. by P. Brito, G. Cucumel, et al. Studies in Classification, Data Analysis, and Knowledge Organization. Berlin, Heidelberg: Springer, 2007, pp. 161–172.

[11] J. T. Bonner. "The origins of multicellularity". In: *Integrative Biology: Issues, News, and Reviews* 1.1 (1998), pp. 27–36.

[12] P. Bourdieu. *Distinction : a social critique of the judgement of taste*. In collab. with Internet Archive. Cambridge, MA. : Harvard University Press, 1984. 640 pp.

[13] G. Brockman, V. Cheung, et al. "OpenAI Gym". In: *arXiv:1606.01540 [cs]* (June 5, 2016).

[14] T. B. Brown, B. Mann, et al. "Language Models are Few-Shot Learners". In: *arXiv:2005.14165 [cs]* (July 22, 2020).

[15] R. Buick. "When did oxygenic photosynthesis evolve?" In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 363.1504 (Aug. 27, 2008). Publisher: Royal Society, pp. 2731–2743.

[16] T. E. I. Cheatham, J. L. Miller, et al. "Molecular Dynamics Simulations on Solvated Biomolecular Systems: The Particle Mesh Ewald Method Leads to Stable Trajectories of DNA, RNA, and Proteins". In: *Journal of the American Chemical Society* 117.14 (Apr. 1, 1995). Publisher: American Chemical Society, pp. 4193–4194.

[17] C. M. Child. "Nervous centralization and cephalization in evolution". In: *The origin and development of the nervous system: From a physiological viewpoint*. Chicago, IL, US: University of Chicago Press, 1921, pp. 142–154.

[18] D. H. Chitwood. "Imitation, Genetic Lineages, and Time Influenced the Morphological Evolution of the Violin". In: *PLOS ONE* 9.10 (Oct. 8, 2014). Publisher: Public Library of Science, e109229.

[19] D. Ciresan, U. Meier, et al. "Flexible, High Performance Convolutional Neural Networks for Image Classification." In: International Joint Conference on Artificial Intelligence IJCAI-2011. July 16, 2011, pp. 1237–1242.

[20] D. Ciresan, U. Meier, et al. "Multi-column Deep Neural Networks for Image Classification". In: *arXiv:1202.2745 [cs]* (Feb. 13, 2012).

[21] J. Confer, J. Easton, et al. "Evolutionary Psychology Controversies, Questions, Prospects, and Limitations". In: *The American psychologist* 65 (Feb. 1, 2010), pp. 110–26.

[22] L. Deng. "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]". In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012). Conference Name: IEEE Signal Processing Magazine, pp. 141–142.

[23] S. Fowler, R. Roush, et al. "Evidence of Evolution". In: (Apr. 25, 2013). Book Title: Concepts of Biology Publisher: OpenStax.

[24] J. Fredens, K. Wang, et al. "Total synthesis of Escherichia coli with a recoded genome". In: *Nature* 569.7757 (May 1, 2019), pp. 514–518.

[25] K. P. F.R.S. "LIII. On lines and planes of closest fit to systems of points in space". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (Nov. 1, 1901). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/14786440109462720, pp. 559–572.

[26] M. Garwicz, M. Christensson, et al. "A unifying model for timing of walking onset in humans and other mammals". In: *Proceedings of the National Academy of Sciences* 106.51 (Dec. 22, 2009), pp. 21889–21893.

[27] R. Geirhos, J.-H. Jacobsen, et al. "Shortcut learning in deep neural networks". In: *Nature Machine Intelligence* 2.11 (Nov. 2020), pp. 665–673.

[28] F. Godin, J. Degrave, et al. "Dual Rectified Linear Units (DReLUs): A Replacement for Tanh Activation Functions in Quasi-Recurrent Neural Networks". In: *Pattern Recognition Letters* 116 (Dec. 2018), pp. 8–14.

[29] S. B. Hedges, J. E. Blair, et al. "A molecular timescale of eukaryote evolution and the rise of complex multicellular life". In: *BMC Evolutionary Biology* 4.1 (Jan. 28, 2004), p. 2.

[30] M. Henaff. *Claude Levi-Strauss and the Making of Structural Anthropology*. U of Minnesota Press, 1998. 312 pp.

[31] G. E. Hinton. "Learning multiple layers of representation". In: *Trends in Cognitive Sciences* 11.10 (Oct. 1, 2007), pp. 428–434.

[32] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 15, 1997), pp. 1735–1780.

[33] D. H. Hubel and T. N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of Physiology* 195.1 (Mar. 1968), pp. 215–243.

[34] J. Huxley. “Evolution. The Modern Synthesis.” In: *Evolution. The Modern Synthesis.* (1942). Publisher: London: George Alien & Unwin Ltd.

[35] W. James. “Lecture II: What pragmatism means”. In: *Pragmatism: A new name for some old ways of thinking.* New York, NY, US: Longmans, Green and Co, 1907, pp. 43–81.

[36] M. Joachimczak, R. Suzuki, et al. “Fine Grained Artificial Development for Body-Controller Coevolution of Soft-Bodied Animats”. In: July 30, 2014, pp. 239–246.

[37] E. E. Jones and V. A. Harris. “The attribution of attitudes”. In: *Journal of Experimental Social Psychology* 3.1 (Jan. 1, 1967), pp. 1–24.

[38] J. G. Kemeny. “Theory of Self-Reproducing Automata. John von Neumann. Edited by Arthur W. Burks. University of Illinois Press, Urbana, 1966. 408 pp., illus. $10”. In: *Science* 157.3785 (July 14, 1967). Publisher: American Association for the Advancement of Science Section: Book Reviews, pp. 180–180.

[39] K. C. Kiwiel. “Convergence and efficiency of subgradient methods for quasiconvex minimization”. In: *Mathematical Programming* 90.1 (Mar. 1, 2001), pp. 1–25.

[40] A. Knoll, M. Walter, et al. “The Ediacaran Period: a new addition to the geologic time scale”. In: *Lethaia* 39.1 (2006). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1080/00241160500409223, pp. 13–30.

[41] T. Kohonen. “Self-organized formation of topologically correct feature maps”. In: *Biological Cybernetics* 43.1 (Jan. 1, 1982), pp. 59–69.

[42] V. R. Konda and J. N. Tsitsiklis. “OnActor-Critic Algorithms”. In: *SIAM Journal on Control and Optimization* 42.4 (Jan. 2003), pp. 1143–1166.

[43] S. Leijnen and F. v. Veen. “The Neural Network Zoo”. In: *Proceedings* 47.1 (2020). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, p. 9.

[44] S. Liu, S. De Mello, et al. “Learning Affinity via Spatial Propagation Networks”. In: *arXiv:1710.01020 [cs]* (Oct. 3, 2017).

[45] W. Maass. “Networks of spiking neurons: The third generation of neural network models”. In: *Neural Networks* 10.9 (Dec. 1, 1997), pp. 1659–1671.

[46] A. Malinowski, J. Zurada, et al. “Minimal training set size estimation for neural network-based function approximation”. In: *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS ’94.* Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS ’94. Vol. 6. May 1994, 403–406 vol.6.

[47] A. Markov. “An Example of Statistical Investigation of the Text Eugene Onegin Concerning the Connection of Samples in Chains”. In: *Science in Context* (2006).

[48] T. Marshall. *Prisoners of Geography: Ten Maps That Explain Everything About the World.* Google-Books-ID: 46cxDQAAQBAJ. Simon and Schuster, Oct. 11, 2016. 320 pp.

[49] G. G. Matthews. *Neurobiology: Molecules, Cells and Systems.* Wiley, Dec. 27, 2000. 604 pp.

[50] W. S. McCulloch and W. Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1, 1943), pp. 115–133.

[51] N. Metropolis and S. Ulam. “The Monte Carlo Method”. In: *Journal of the American Statistical Association* 44.247 (1949). Publisher: [American Statistical Association, Taylor & Francis, Ltd.], pp. 335–341.

[52] M. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry.* Expanded ed. Cambridge, Mass: MIT Press, 1988. 292 pp.

[53] A. K. Modrell and P. Tadi. “Primitive Reflexes”. In: *StatPearls.* Treasure Island (FL): StatPearls Publishing, 2021.

[54] S. J. Mojzsis, G. Arrhenius, et al. “Evidence for life on Earth before 3,800 million years ago”. In: *Nature* 384.6604 (Nov. 1996). Number: 6604 Publisher: Nature Publishing Group, pp. 55–59.

[55] G. E. Moore. “Progress in digital integrated electronics [Technical literaiture, Copyright 1975 IEEE. Reprinted, with permission. Technical Digest. International Electron Devices Meeting, IEEE, 1975, pp. 11-13.]” In: *IEEE Solid-State Circuits Society Newsletter* 11.3 (Sept. 2006). Conference Name: IEEE Solid-State Circuits Society Newsletter, pp. 36–37.

[56] R. G. Morris. "D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949". In: *Brain Research Bulletin* 50.5 (Dec. 1999), p. 437.

[57] O. H. Mowrer. *Learning theory and behavior*. Learning theory and behavior. Pages: xii, 555. Hoboken, NJ, US: John Wiley & Sons Inc, 1960. xii, 555.

[58] *NASA - Life's Working Definition: Does It Work?* Publisher: Brian Dunbar. URL: `https://www.nasa.gov/vision/universe/starsgalaxies/life%27s_working_definition.html` (visited on 06/25/2021).

[59] H. A. Orr. "Fitness and its role in evolutionary genetics". In: *Nature Reviews Genetics* 10.8 (Aug. 2009). Bandiera_abtest: a Cg_type: Nature Research Journals Number: 8 Primary_atype: Reviews Publisher: Nature Publishing Group, pp. 531–539.

[60] I. P. Pavlov. *Lectures on conditioned reflexes: Twenty-five years of objective study of the higher nervous activity (behaviour) of animals*. Red. by W. H. Gantt. Lectures on conditioned reflexes: Twenty-five years of objective study of the higher nervous activity (behaviour) of animals. Pages: 414. New York, NY, US: Liverwright Publishing Corporation, 1928. 414 pp.

[61] W. H. Press and F. J. Dyson. "Iterated Prisoner's Dilemma contains strategies that dominate any evolutionary opponent". In: *Proceedings of the National Academy of Sciences* 109.26 (June 26, 2012), pp. 10409–10413.

[62] D. Purves, G. J. Augustine, et al. "The Primary Motor Cortex: Upper Motor Neurons That Initiate Complex Voluntary Movements". In: *Neuroscience. 2nd edition* (2001). Publisher: Sinauer Associates.

[63] D. H. Rakison and J. Derringer. "Do infants possess an evolved spider-detection mechanism?" In: *Cognition* 107.1 (2008). Place: Netherlands Publisher: Elsevier Science, pp. 381–393.

[64] B. Rasch and J. Born. "About Sleep's Role in Memory". In: *Physiological Reviews* 93.2 (Apr. 1, 2013). Publisher: American Physiological Society, pp. 681–766.

[65] D. Rogers and P. Ehrlich. "Natural selection and cultural rates of change". In: *Proceedings of the National Academy of Sciences of the United States of America* 105 (Apr. 1, 2008), pp. 3416–20.

[66] S. Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747 [cs]* (June 15, 2017).

[67] D. E. Rumelhart, G. E. Hinton, et al. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (Oct. 1986). Bandiera_abtest: a Cg_type: Nature Research Journals Number: 6088 Primary_atype: Research Publisher: Nature Publishing Group, pp. 533–536.

[68] T. C. Schelling. "Dynamic models of segregation". In: *The Journal of Mathematical Sociology* 1.2 (July 1, 1971). Publisher: Routledge _eprint: https://doi.org/10.1080/0022250X.1971.9989794, pp. 143–186.

[69] J. Schmidhuber. "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61 (Jan. 2015), pp. 85–117.

[70] J. M. Smith, R. Burian, et al. "Developmental Constraints and Evolution: A Perspective from the Mountain Lake Conference on Development and Evolution". In: *The Quarterly Review of Biology* 60.3 (Sept. 1, 1985). Publisher: The University of Chicago Press, pp. 265–287.

[71] K. O. Stanley and R. Miikkulainen. "Evolving Neural Networks through Augmenting Topologies". In: *Evolutionary Computation* 10.2 (June 2002). Conference Name: Evolutionary Computation, pp. 99–127.

[72] A. J. Stewart and J. B. Plotkin. "Extortion and cooperation in the Prisoner's Dilemma". In: *Proceedings of the National Academy of Sciences* 109.26 (June 26, 2012), pp. 10134–10135.

[73] G. Stuart, N. Spruston, et al. "Action potential initiation and backpropagation in neurons of the mammalian CNS". In: *Trends in Neurosciences* 20.3 (Mar. 1, 1997). Publisher: Elsevier, pp. 125–131.

[74] C. Thomas Anterion, P. Convers, et al. "An odd manifestation of the Capgras syndrome: loss of familiarity even with the sexual partner". In: *Neurophysiologie Clinique = Clinical Neurophysiology* 38.3 (June 2008), pp. 177–182.

[75] J. Tooby and L. Cosmides. "The Theoretical Foundations of Evolutionary Psychology". In: Nov. 1, 2015.

[76] V. Vapnik. *The Nature of Statistical Learning Theory*. 2nd ed. Information Science and Statistics. New York: Springer-Verlag, 2000.

[77] A. Vaswani, N. Shazeer, et al. "Attention Is All You Need". In: *arXiv:1706.03762 [cs]* (Dec. 5, 2017).

[78] T. Vicsek, A. Czirok, et al. “Novel type of phase transition in a system of self-driven particles”. In: *Physical Review Letters* 75.6 (Aug. 7, 1995), pp. 1226–1229.

[79] R. Wainwright. “Conway’s Game of Life: Early Personal Recollections”. In: *Game of Life Cellular Automata*. Ed. by A. Adamatzky. London: Springer, 2010, pp. 11–16.

[80] C. J. C. H. Watkins and P. Dayan. “Q-learning”. In: *Machine Learning* 8.3 (May 1, 1992), pp. 279–292.

[81] R. Wei. *’Civilization’ and ’Culture’*. SSRN Scholarly Paper ID 2182608. Rochester, NY: Social Science Research Network, Nov. 29, 2012.

[82] L. Yaeger. “Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context”. In: (Mar. 23, 1995).

[83] J. Zhou, G. Cui, et al. “Graph neural networks: A review of methods and applications”. In: *AI Open* 1 (Jan. 1, 2020), pp. 57–81.