

Evolutionary Neural Network Tools for Life Emergence and More

F. Furfaro

2021

Abstract

L'idée de vie artificielle remonte au année 1940 par principe d'autoréplication, meme époque ou les concepts sur l'intelligence artificielle sont créés. Nous proposons un modele à définition incomplete, mais utile pour etudier la fonctionnalisation des structure cerebrale lors de l'evolution. On combine l'apprentissage par renforcement et la modification aléatoire des connections intercouche. Pour tester notre modele, le jeu du chat et la souris est un exemple de choix pour etudier l'evolution des comportement basique de proie/predateurs, tout en etant assez complet pour une approche évolutive. Ces outils n'ont pas la prétentions de reproduire la vie sur ordinateurs, mais peuvent donner des idée, minime soit elle, pour l'optimisation des structure de reseau de neurones et d'avancer vers la creation de vie articiel. En effet, lors du processus d'apprentissage (evolution + descente de gradient), on on observe par l'apparition de structure et stratégie convergente. Ces structures sont conservé pendant un grand nombre de cycle alors meme que des reseau "challenger" sont toujours ajouter à chaque cycle de reproduction. En dehors, on observe que ce type d'algorithme peut s'averer efficace avec moins de neurone pour la classification 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 [5]]. Elle a commencer de maniere unicellulaire sans noyau [Buick [3] and Mojzsis et al. [8]], à un large panel de forme avec l'appartition du noyau [Hedges et al. [4] and Knoll et al. [6]] et de la multicellularité [Bonner [2]], et en parallele, l'apparition du potentiel d'action [Matthews [7]] suivi d'une explosion de des variétés des systeme nerveux [Ancil, Michel (2015)], avec plus recement les systeme centraux [Child, Charles Manning. (1921)]. Pour autant, il est assez difficile de definir ce qui est vraiment vivant et cette question est toujours débattu. 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]. L'evolution darwinienne est un principe fondamentale dans l'etude des systeme complexe disposant d'heredité, de variation et de selection. On ne le retrouve pas seulement en biologie, mais on peut l'observer dans l'evolution des courants musicaux, artistique, philosophique, technologique, societal et culturel [Daniel H. Chitwood 2014, Deborah S. Rogers 2007]. 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'homéostasie et d'apprentissage [Bartlett and Wong [1]]. Ce qui diminue localement l'entropie en favorisant l'auto-organisation. Meme si tout probleme n'est pas définissable, ces deux definitions se restreignent à un systeme "chimique", est ce qu'un robot dotée d'une intelligence, voient d'emotion humaine serait vivant dans cette définition ? Aujourd'hui, il existe un grand nombre de simulation de vie sur ordinateurs, comme par exemple Polyworld [Larry Yaeger 1995], et sont avec la construction de cellules minimaliste [Julius Fredens 2019], 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 environnement de jeu simple : le jeu du chat et la souris, alternant ainsi deux comportements : proie/prédateur. On se base sur deux outils : les automates cellulaires et les reseaux de neurones évolutifs. Les automates sont à l'origine des objets mathématique permettant de resoudre des probleme de decidabilité mathématique [Von Newman 1966]. On en retrouva ensuite des utilisation experimentale comme le plus connu "jeu de la vie" [Conway], 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 [Elwyn R. Berlekamp 2003]. 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 renforcement. 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 reseau.

STATISTIQUE : Les reseaux neuronaux ont ete construit à partir du modele de neurone biologique, le neurone formel [McCulloch & Pitts 1960], 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 [Karl Pearson 1901]. Les SVM, séparateurs à vaste marge, generalisant les classificateur linéaire par des vecteurs supports, mais formalisant la régularisation statistique et les concept de surapprentissage [Vapnik 1995]. Enfin Le K-mean qui divise un ensemble de points par minimisation d'un certain nombre de graine barycentrique [Hans-Hermann Bock 2007]. 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 en compétition [Hinton 2007].

APPRENTISAGE : Le question de l'apprentissage a été étudié au cours des année 1950, ou il a pu emerger la regles de Hebb [Hebb 1949] qui modifier simplemnt la valeurs des coefficient synaptique d'un reseau simple couche, il eue un declin en fin 1970 par l'impossibilité de reseont probleme non linéaire XOR ou connexité [Minsky 1969]. Mais ce probleme fut resolu par les perceptron multicouche, mais cette fois la modification des poids était plus compliqué. C'est le systeme de retropropagation du gradient [Rumelhart 1986] qu'on retrouve aussi dans le cerveau [Stuart 1997] et l'algorithme du gradient, par minimisation itérative de l'erreurs, qui permet de resoudre ce probleme. En parallele, inspirer par les neurone moteurs de la vue [Hubel 1968], on devellopa les reseau de neurone convolutif qui reduit fortement le calcul des poids en entrée du reseau, car n'est pas entierement connecté [Dan Cireşan 2012]. Le souci de ce genre de reseau est qu'il faut un grand nombre de donnée pour adapté les poids, pour reseoudre un probleme de decision markovien, il y eu l'apparition du Q-learning convergent par l'apprentissage des récompense [Dayan 1992] qui ont encore était amélioré par des reseau adversariaux "actor-critic" à l'image des reseau génératif, il permette d'accelerer la convergence des poids [Konda 2003].

FONCTIONNALISATION : Néanmoins, 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 1997], surtout 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 2002]. 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 [WOLFGANG MAASS 1996]. L'idée est donc de s'inspirer des donnée en neuroscience recente, ou il est montré le principe de fonctionnalisation cerebrale que le cerveau est pré-cablé pour voir certain danger ou encore marcher à la naissance [Rakison, D. H. & Derringer, J 2008] et il n'y a pas eu besoin d'avoir un temps d'apprentissage long, hormi le sommeil et son role dans la memoire [Björn Rasch 2013]. A cette image, on a donc develloper un algorithme ou 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 façon 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 debugage 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) [Gupta 2020]. Le perceptrons se represente par le produit scalaire entre le vecteurs d'entrée et le vecteurs poids des neurones, soit l'application multilinéaire suivante :

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

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). 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 [Juergen Schmidhuber 2014] 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 :

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

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 gradient (optimizer). Les algorithme les plus utilisé sont SGD "Stochastic Gradient Descent" plutot utilisé dans classification et "Adam" plutot utilisé au problème de regression [Sebastian Ruder 2016].

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 correspond 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 :

$$\vec{y} = \begin{bmatrix} w_{0,0} & & \\ & \ddots & \\ & & w_{m,n} \end{bmatrix} \cdot \begin{bmatrix} x_0 & \dots & x_m \end{bmatrix} + \begin{bmatrix} b_0 \\ \\ b_n \end{bmatrix}$$

$\vec{s} = f(\vec{y})$

Pour que l'algorithme mette à jours efficacement les poids du réseau, on utilise un certain lot de donnée d'apprentissage iterativement, qu'on appelle "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 vallée de stabilité, le taille du lot ne doit pas être trop petit non plus car augmente le bruit [Krzysztof C. Kiwiel 2001]. Dans ce cas, le vecteurs d'entrée devient une matrice " $\mathbf{m} \cdot \mathbf{b}$ ", et la matrice des poids devient un tenseurs d'ordre 3 " $\mathbf{n} \cdot \mathbf{m} \cdot \mathbf{b}$ ". La plupart des bibliothèques logicielles s'adapte automatiquement à la taille du batch, et n'a pas besoin d'être spécifié.

Code PyTorch :

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

Pour autant, les réseaux linéaire ne sont pas les seules à être employés et ne sont pas forcément les plus adaptés pour les problèmes d'images [Dan C. Ciresan 2011]. En effet, une image est une donnée 2D dont les données spatiales sont localement liées, mais les informations ne sont pas interconnectées, c'est pour cela qu'il existe les réseaux de convolution. Ces réseaux ont l'avantage d'être interconnectés mais par fenêtre "spatiale" ce qui réduit fortement le nombre de calculs pour l'actualisation des poids. Elle est composée de " \mathbf{N} " perceptrons, mais ne représente plus comme un produit tensoriel, mais comme un produit de convolution où les entrées sont moyennées pondérément par le glissement " \mathbf{C} " d'une fenêtre de taille " \mathbf{k} " sur les poids " \mathbf{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 considérer qu'il y a qu'une seule entrée par neurone, pour cela, le nombre d'entrée est égal au nombre de sortie du canal de la couche. Ensuite, la taille de la fenêtre est de " $\mathbf{1}$ " et le groupement de connexion doit être égal au nombre de neurones, qui est un nombre divisible de l'entrée et de la sortie par lui-même. Ce type de paramétrage est équivalent à \mathbf{N} neurones indépendants à 1 seule connexion chacune. Contrairement au réseau linéaire, la taille du batch doit être spécifiée 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 cerveau "modèle" pour construire des algorithmes neuronaux a toujours été présent dans le développement de nouvelles méthodes en *Intelligence Artificielle*. Dans ce projet, nous nous sommes inspirés de la spécialisation fonctionnelle du cerveau. 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 [Tooby, J. & Cosmides, 1992 ; Tooby, J. & Cosmides, 2015]. 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, J. C. et al. 2010]. La spécialisation fonctionnelle du cerveau est observée empiriquement par plusieurs troubles neuropsychologiques, comme par exemple avec le syndrome de Capgras, où une défaillance d'une région entraîne la perte de reconnaissance des visages [Thomas Antérion, 2008].

Dans notre cas, nous avons simplifié le plus possible un cerveau, où l'on a en entrée des neurones de "vision", suivis de couches séquentielles "fonctionnelles" et en sortie, des neurones "moteurs". On distingue des neurones stables, les neurones d'entrée et de sortie qui ne changeront pas pendant le processus évolutif. Les neurones de "visions" sont à l'image des cônes de la rétine c'est-à-dire des neurones connectés à des batonnets de vue unique [Dale Purves, G-J Augustine, 2005] qu'on représente dans notre cas comme une couche de convolution "simple sparse" : une seule entrée par neurone. Les neurones "moteurs" quant à eux sont analogues aux motoneurones qui activent la contraction musculaire et donc le mouvement des organismes [Fitzpatrick, D. (2001) The Primary Motor Cortex], on les représente dans notre cas comme une couche linéaire de neurones où les sorties seront liées à l'action du réseau de neurone complet.

La vraie particularité du projet est que les couches intermédiaires "fonctionnalisées" ne sont pas construites manuellement, mais adaptées par un processus évolutif de sélection. Ainsi, l'objectif est de faire émerger une structure intermédiaire fonctionnalisée à notre problème, ici le "jeu du chat et la souris". Les connexions intermédiaires sont initialement aléatoires avec au minimum une connexion entre la couche " $n+1$ " et " n ", pour avoir toujours un lien entre l'entrée et la sortie et ne pas avoir de problème lors du calcul des poids des neurones. Au cours de l'apprentissage, on s'attend à ce que le réseau soit de plus en plus structuré.

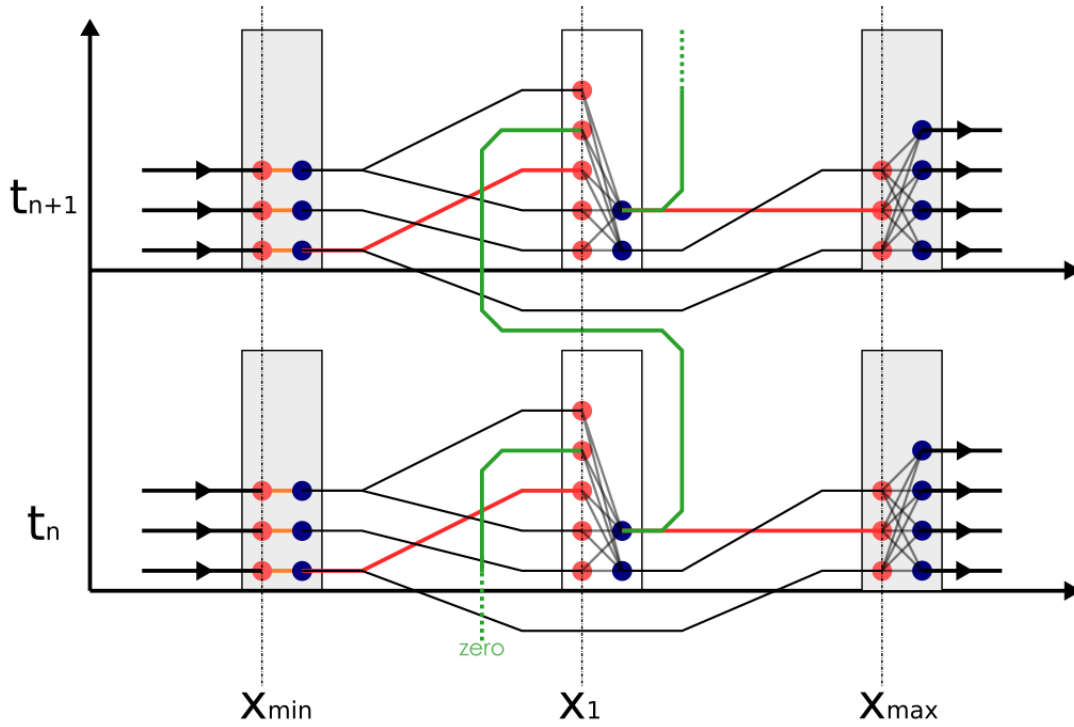


Figure 1: Réseau monocouche à connexion aléatoire; Le même réseau est représenté 2 fois à deux intervalle de temps à la suite, les nombres entrée/sorties sont pour l'exemple. De bas en haut : La commutation entre deux temps. De gauche à droite : L'entrée vers la sortie.

Comme les connexions entre l'entrée d'une couche X_n et la sortie d'une couche en X_{n+m} , avec $m \geq 0$, n'est pas possible pour le calcul de la rétropropagation à un temps donné, on se trouve avec la connexion $X_{n+m,t-1} \rightarrow X_{n,t}$. Ce type de connexion laisse apparaître une forme de réseau récurrent pour le calcul du gradient, mais comme nous voulons éviter la structure artificielle comme LSTM qui empêche le *vanishing gradient* des réseaux récurrents [Ref LSTM], nous avons considéré cette entrée à "t-1" comme une **entrée virtuelle**. Cette entrée virtuelle, n'est pas la solution la plus élégante, mais est la plus flexible dans le cas où les connexions sont complètement désordonnées et réduit le temps de calculs car la taille de l'entraînement est linéaire. Une couche connectée à une entrée virtuelle d'une couche elle-même connectée à une entrée virtuelle, permet de stocker une information à "t-2", la taille du stockage possible est proportionnelle à la taille du réseau. L'entrée virtuelle consiste au détachement de la sortie du graphe computationnel 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 entrées virtuelles.

1.2 Construction du graphe

La topologie du réseau présenté précédemment peut être vue comme un graphe orienté acyclique, en effet, les connexions entre l'entrée X_n et la sortie X_{n+m} sont vues comme des entrées virtuelles indépendantes de la rétropropagation. Les connexions intercouche peuvent être négligées pour la construction initiale du graphe, car elles sont complètement connectées (voir code PyTorch `nn.linear`). Pour les connexions entre les couches, on se retrouve face à plusieurs contraintes :

- Quel est le nombre minimal de connexions pour que le réseau soit complètement connecté ?
- Combien de connexions sont attribuées par couche ?
- Comment attribuer les connexions 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 entière du nombre d’agent par génération d’entraînement et ne peut dépasser 32.

Propriétés “Unicité des entrée”: Les liens/connections entre les différentes couches de neurones dépend 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 définir 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’intervall $[C_{min}, 2C_{min}]$. Puis le nombre de couche intermediaire du reseau “ N_L ” est attribué aléatoirement dans l’intervall $[1, C_{tot}]$. 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’écart 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’écart entre “ N_h ” et “ N_L ”.

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

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

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

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 attribué aléatoirement et sans remplacement dans l’intervall $[1; 31]$, 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 $P(X_k) = \begin{cases} 1 & X_k = X_{k+1} - 1 \\ 0 & \text{sinon} \end{cases}$ 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

$$f_X = \begin{cases} \frac{2}{3} & \text{pour } a \leq x \leq b \\ \frac{1}{3} & \text{pour } b \leq x \leq c \text{ dans ce cas.} \\ 0 & \text{sinon} \end{cases}$$

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 amont des connection en aval. La matrice à par définition la structure d'un graph orienté.

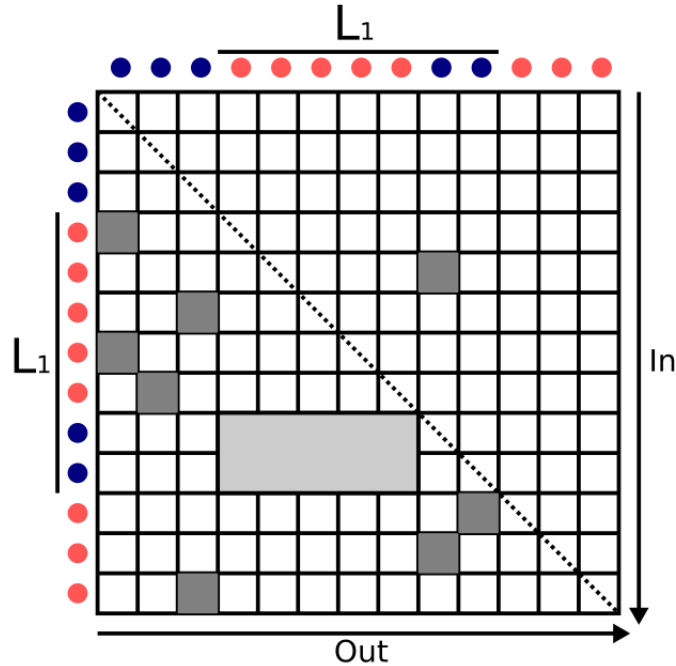


Figure 2: Matrice adjacente des liens du reseau de neurone précédent. En rouge, les noeud de connection (entrée d'une couche) et en bleu, les noeuds de perceptron (sortie de couche). Les case grisé correspondent à une connection de poids constants (1 par défauts).

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 reconstruira 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'être prédictif [ref]. 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'entraînement [ref]. Pour autant, dans le vivant, il n'y a pas besoin d'un tres grand nombre d'experience pour qu'un cerveau apprenne efficacement un probleme [ref]. 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 [ref] ou des mammifere juvénile proie [ref]. 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 precedement, on peut reconstruire les connections entre les différentes couches du *reseau 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; \mathbf{X} correspond au Tenseur d'entrée de taille $I \times \text{Batch}$. \mathbf{NET} correspond au information des neurones (index, position, liste adjacente). \mathbf{Trace} correspond à la mise en mémoire des sorties des couches modulés des liste de neurones \mathbf{Layers} (PyTorch : *nn.ModuleList*). \mathbf{h} correspond à la copie détaché de la liste des Tenseurs \mathbf{Trace} à t-1.

```

Batch size, Input size  $\leftarrow \dim(\mathbf{X})$ 
BATCH_  $\leftarrow \text{np.arange}(\text{len}(\mathbf{x}))$ 
Trace[-1]  $\leftarrow \mathbf{Layers}[-1](\mathbf{X}.\text{view}(\text{Batch size}, \text{Input size}, 1)).\text{view}(\text{Batch size}, \text{Input size})$ 
for i, network index ordered by position do :
    tensor  $\leftarrow []$ 
    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  $\leftarrow \text{torch.cat}(\text{tensor}, \text{dim}=1)$ 
    Trace[i]  $\leftarrow \mathbf{Layers}[i](\text{tensor\_in})$ 
    Trace[-2]  $\leftarrow \mathbf{Layers}[-2](\text{tensor\_in})$ 
for t in range(len(Trace)):
    h[t][BATCH_]  $\leftarrow \text{Trace}[t][\text{BATCH\_}].\text{detach}()$ 
return Trace[i], Trace[-2]
```

Une fois le reseau complètement câblé, 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, 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'évolution du systeme correspond à un processus de decision markovien, représenté 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) :

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

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 \mathbf{T} correspond à la matrice de transition et \mathbf{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 états 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 déterminées 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 dernière correspond à l'équation de Bellman et c'est elle qui est utilisée pour calculer la fonction de coût du réseau de neurone lorsqu'on réalise une action. La première équation ne peut être utilisée seule car elle nous donne l'efficacité de l'agent pour une action donnée, mais peut être utilisée en cas d'utilisation de réseau adversariaux [Ref actor-critic]. Dans notre cas, on réalise initialement une suite d'événements avec une matrice de transition aléatoire (réseau non entraîné), puis on calcule la fonction de perte entre la prédiction des valeurs Q obtenue au cours de l'expérience avant l'action et le résultat de l'équation de Bellman Q après l'action d'un batch d'entraînement donnée, puis on répète N_{cycle} fois cette étape 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 + (1-DONE)*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 réseau, et correspond à la probabilité d'action optimale. Normalement, à chaque pas de temps, on choisit la valeur maximale en sortie du réseau et l'on attribue une probabilité d'action aléatoire, c'est le **dilemme exploration-exploitation**. Ce dilemme permet au réseau de neurone d'explorer des paramètres et de ne pas se stabiliser dans une vallée non optimale. Mais dans notre cas, nous avons choisi d'attribuer une probabilité d'action à chaque "*pas*" de l'entraînement à partir de la sortie du réseau, qui est équivalent à un dilemme 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

À partir des différents modèles de réseau entraînés à temps courts, l'objectif qui suit est de sélectionner ceux qui ont eu le meilleur score d'entraînement, puis de modifier légèrement la structure du graphe neuronal. Le calcul des scores est relatif à l'environnement d'entraînement et est très utilisé dans les modèles évolutifs. L'attribution des points sera plus détaillée dans la partie 3. Le principe est le suivant : Lors du premier entraînement à temps court, on a N_r réseaux en parallèle et distincts, chacun va suivre le processus d'entraînement vu dans la sous-partie précédente. Ensuite, les N_b réseaux ayant le meilleur score sont sélectionnés, N_b réseaux gardent la même topologie pour le cycle suivant, N_m réseaux par N_b meilleurs réseaux héritent d'une mutation aléatoire. Enfin N_c nouveaux réseaux aléatoires sont introduits dans le processus d'entraînement suivant, ceux-ci peuvent hériter de paramètres optimisés des réseaux précédents, mais pas de la structure du réseau, c'est les réseaux "compétiteurs". Les nombres d'agents sont définis comme $N_r = (N_b + 1)^2$, de façon à ce que N_r est une racine entière, $N_m = N_b$, tel que le nombre total de mutations soit $N_{mtot} = N_b^2$ et enfin $N_c = N_b + 1$, de cette façon, 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 connexion : Ne change pas le nombre de couche neuronale, mais ajoute une connexion à l'une d'entre elles. Comme on n'ajoute pas de neurone, la liste des connexions possibles ne change pas et le nouveau nœud va être connecté aléatoirement à un neurone pré-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 vu que le reseau doit etre complet, on rajoute également une connection dans l'une des couches qui va se connecter exclusivement 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 différent ?), 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 quelconque 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 incomplet. 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 décalage 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)**

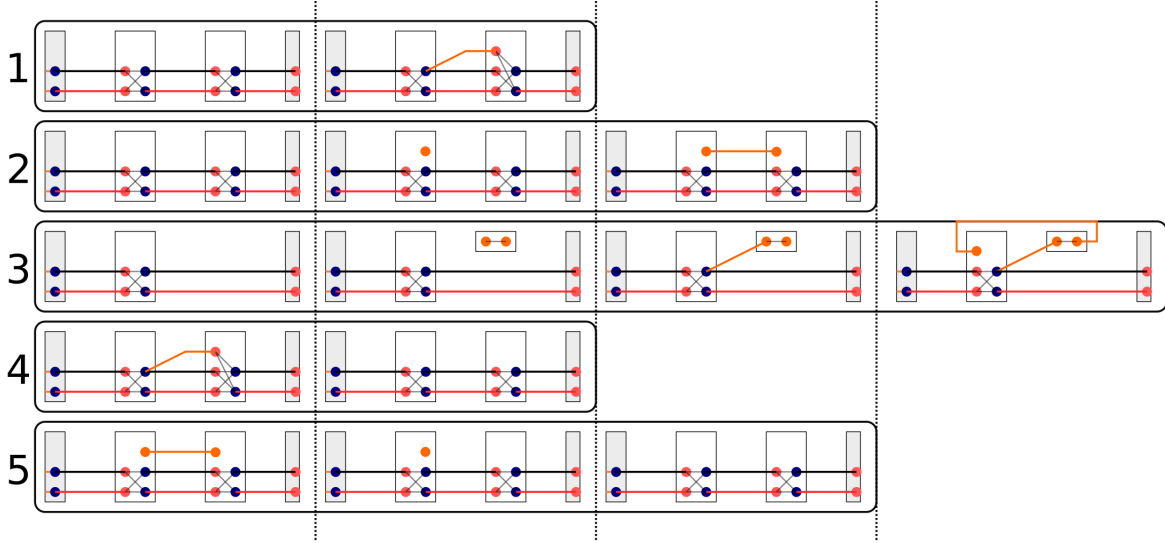


Figure 3: Algorithm visual des mutations. Respectivement les 5 types de mutations citer en paragraphes. De gauche à droite, l'etats initial vers l'etat final. Les propriétés du graphe sont à titre indicatives, cette representation tres simplifié ne reflète pas l'optimum d'une fonction. Les opérations $1 \leftrightarrow 4$ et $2 \leftrightarrow 5$ sont symétriques, l'opération 3, n'a pas de symetrie 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 vaisseau 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 [ref necessaire]. 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'**environnement** d'entrainement. Cet environnement contient des regles et s'apparente à un jeu, la bibliotques la plus utilisé en python est *OpenGym* et sa structure à servi d'exemple pour realiser l'environnement 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 prédatons 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 l'équation (**à implementer**) :

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

3. On revient à l'étapes 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 environnement de jeu. L'agent contient les informations de **vue** et d'**action** qui va transmettre à l'environnement. L'environnement contient les information de la position de l'agent et son adversaire, ainsi que les regles de jeu et comptage de points. L'environnement 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 environnement 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 environnement 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 :

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

Enfin, l'environnement 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" [ref necessaire et importante]. 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'environ 8 déplacement. Dans cette configuration, pour un calcul équilibré, 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.

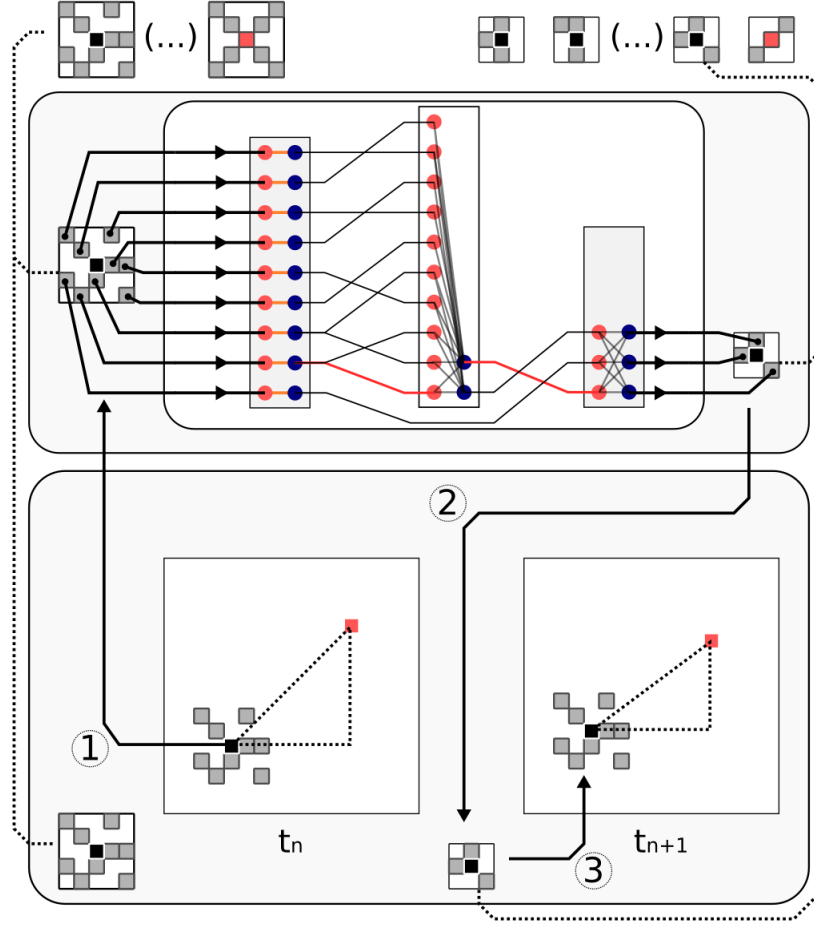


Figure 4: Les interactions entre un agent et un environnement de jeu. En haut, la génération aléatoire des positions relative possible des entree et sortie. Suivi des interactions entre l'agent en haut et l'environnement de jeu en bas. 1 : envoi de l'environnement des informations d'entrée (pixel) à l'agent; 2 : envoi de l'information de sortie de l'agent (mouvement) à l'environnement; 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% $\binom{9}{3} = 84$, 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 attribue 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 expérimentale, il est nécessaire 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 analogue à la génération precedente, 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 precedent. 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 représenter l'ensemble des environnement à un temps donnée par une grille $(5 \times 16) \times (5 \times 16)$, ce qui facilité la visualisation et l'interpretation des resultats. Il est possible aussi de représenter cette grille par décomposition de deux nombre premier si le nombre d'agents par cycle n'est pas le carré d'un nombre. Cette methode correspond à la verification de la parité puis recurrence des nombre impaire jusqu'à la racine entiere supérieur du module du nombre d'agent par génération :

$n \% \begin{cases} 2 \\ \text{impair}(\sqrt{n}) \end{cases}$. Cette derniere n'a pas été utilisé ici, mais est inclu dans le programme.

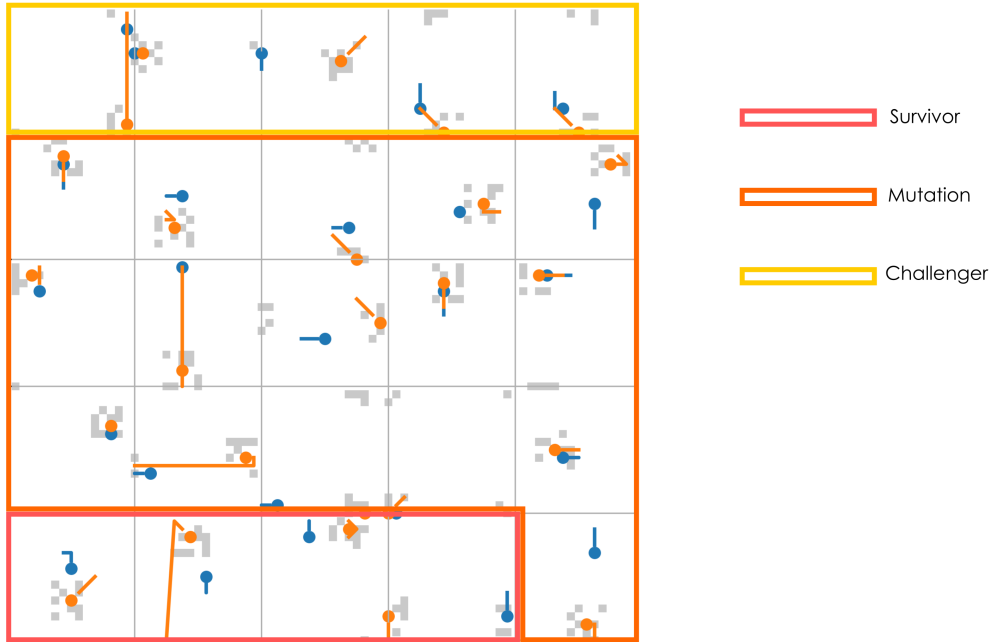


Figure 5: Representation d'une donnée expérimentale à un instant t. En encadré, les différentes catégories d'agent (survivant, mutation et compétiteur). En points bleu, les adversaires et en orange, l'agent intelligent. Les cases grisé correspondre à la vue de l'agent à un instant t.

Le nombre de génération n'est pas clairement défini, mais peut, tout comme le nombre d'agent par génération, être justifié par la loi des grands nombres, si l'on considère que la loi de probabilité à une espérance. 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 :

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

De cette façon, 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 experimentale 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 défini comme la combinaison de 9 cases parmi 25, sa sortie est défini comme la combinaison de 3 cases parmi 9 [Section §3]. On cherche à verifier l'existence 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

$$\begin{pmatrix} x_0 + x_c & y_0 + y_c \\ \vdots & \vdots \\ x_n + x_c & y_n + y_c \end{pmatrix} \cdot \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix}. \text{ La normalisation de la grille nous donne la proportion des meilleurs}$$

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'esperance d'une fonction d'une variable aléatoire X_n en fonction d'une intégrale convergente. Numériquement, il correspond à l'agorithme de *Monte-Carlo*, 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 représenter ces deux densités sur des grilles à deux dimensions.

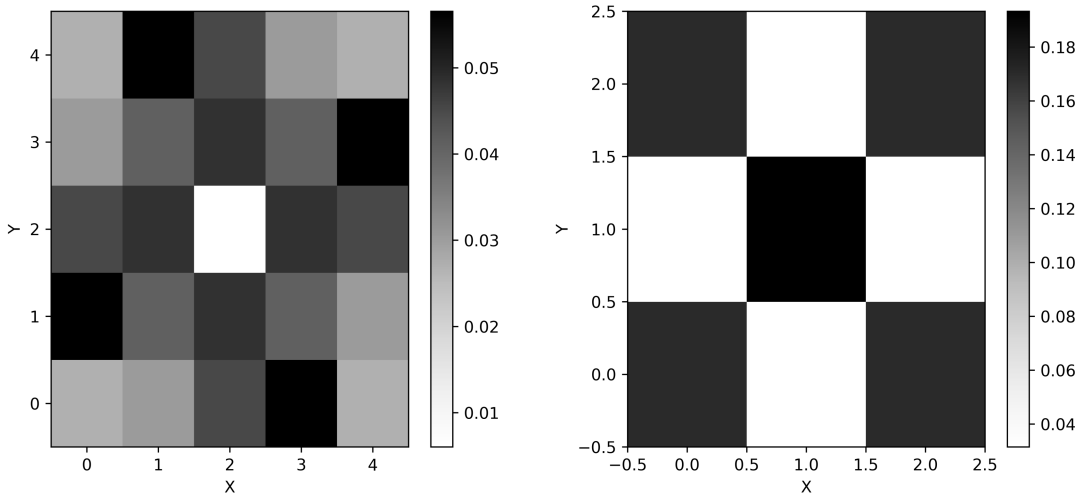


Figure 6: Mesure de la densité positionnel de succes des entree/sorties. A gauche, l'entree correspondant à la vue de l'agent et à droite la sortie, correspondant au mouvement relatif de l'agent. En nuance de gris le niveau de probabilité d'avoir plus de succes avec cette case.

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 implique 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 $(1, 6, 9) \iff (\nwarrow, \emptyset, \swarrow)$, 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 (\swarrow) , 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 éloigné 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.

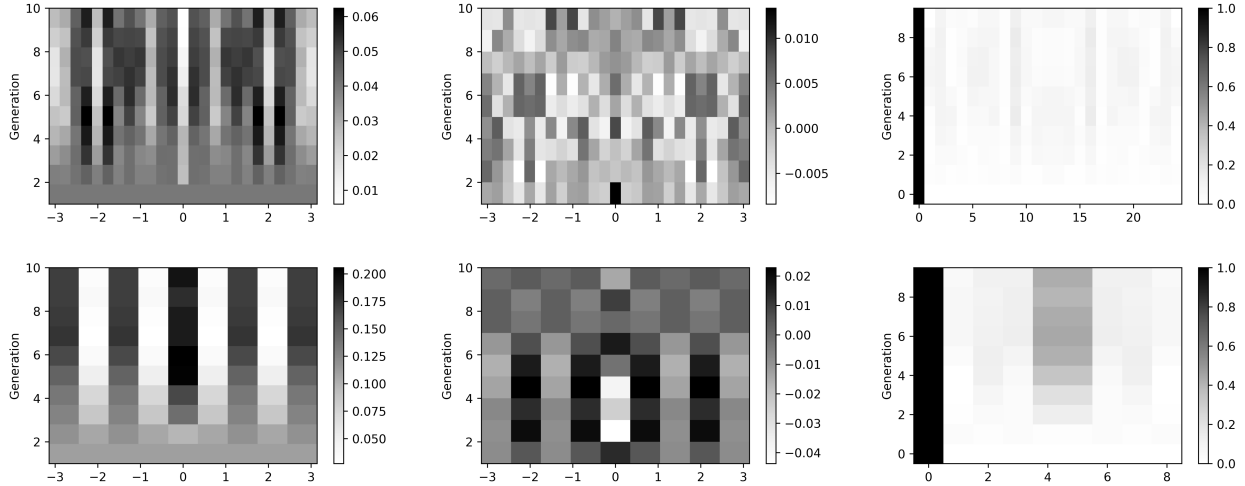


Figure 7: Mesure de la densité positionnel projeté de succes des entree/sorties en fonction du temps. À gauche, l'entrée et à droite la sortie. En haut, la densité de probabilité avec en nuance de gris le niveau de probabilité d'avoir plus de succes avec cette case. Au milieu, la variation de probabilité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

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

Comme la densité de sortie semble être un invariant, et que l'entrée oscille entre quelques positions d'instabilité, on peut considérer qu'il n'influencera pas trop fortement le processus de sélection de réseau une fois stabilisé. De même l'écart fréquentiel des positions défini 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 challengers qui hériteront de la densité de probabilité d'I/O ne devraient pas influencer leur victoire par leurs propriétés d'entrée/sortie, mais de la nouvelle structure aléatoire. Cette hypothèse forte, sera vérifiée dans les parties suivantes et plus discutée en fin de partie.

5 Caractérisation de la connectivité du réseau de neurone

Le réseau de neurone est défini comme un graphe orienté acyclique [1.2]. Comme celui-ci est généré aléatoirement, il est nécessaire d'explorer l'étude de certains paramètres de graphes pour caractériser s'il existe ou non des propriétés dominantes au cours de la sélection.

Contrairement à la mesure de la densité, au cours de la sélection, les agents ayant les meilleurs scores sont sélectionnés, mais, les nouvelles structures sont définies aléatoirement comme à l'instant initial. Les agents sélectionnés subissent 5 mutations aléatoires, on cherche à mesurer l'effet de la mutation sur la structure du réseau et la sélection. Pour cela, on peut représenter l'ensemble des matrices adjacentes ayant eu les meilleurs résultats au cours des générations. Le graphe étant dirigé acyclique, il n'y a pas de problème de *clique* à prendre en compte.

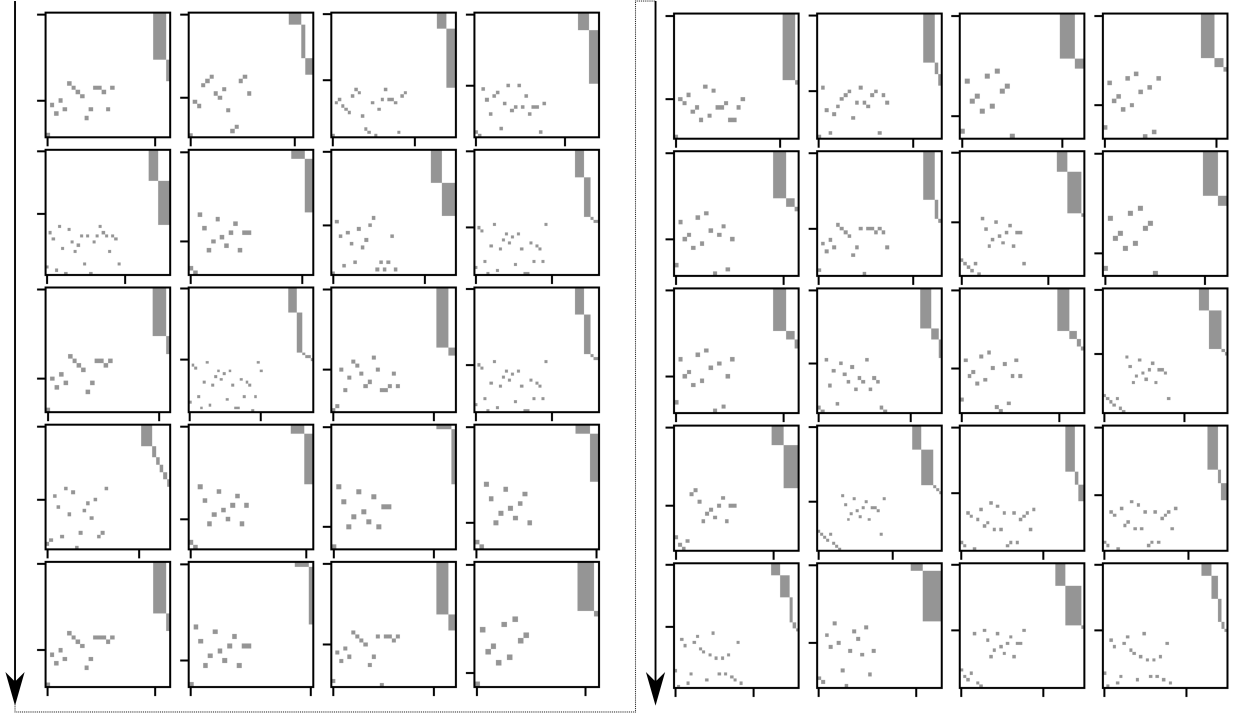


Figure 8: Evolution des réseaux des 4 meilleurs agents par générations. À gauche, de la première génération à la 5ème et à droite, de la 5ème à la dernière génération.

On obtient ainsi un ensemble de grille de matrices adjacentes de taille $N_b \times N_g$, avec N_g , le nombre de générations et N_b , le nombre de meilleurs agents par génération. On n'observe pas de structure très organisée dominante au dixième cycle, le réseau ne semble pas converger vers une structure d'équilibre. Pour autant, la matrice adjacente n'est pas suffisante pour avoir les informations des graphes, il est nécessaire de vérifier certains paramètres si l'on veut caractériser le réseau. Pour cela, on va comparer les **indicateurs de centralité**, tels que les "*degrés*", "*proximité/excentricité*", "*intermédiarité*" et "*centralité propre*" entre les 4 meilleurs réseaux et les 4 pires réseaux 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 $x_v = \frac{1}{\lambda} \sum_t x_t$.

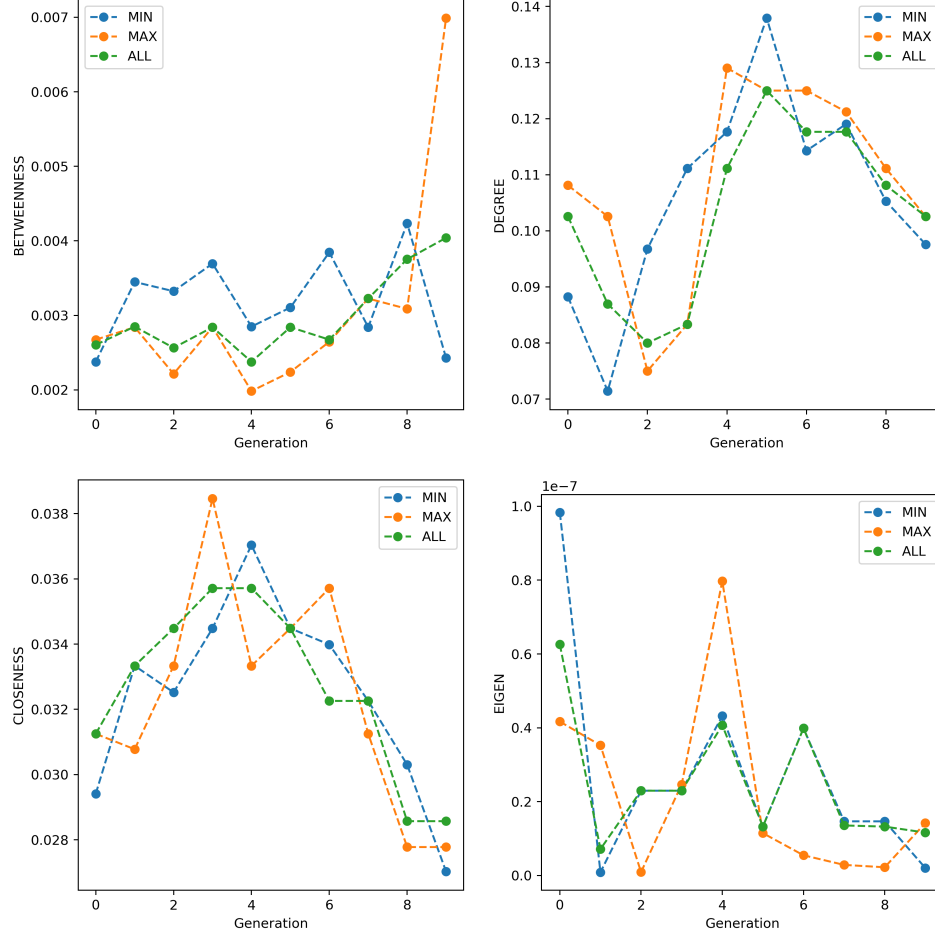


Figure 9: Comparaison des indicateurs de centralité entre les meilleurs et les pire agents. Pour chaque graphe, en abscisse le temps, en ordonnée le parametre “degree”, “betweenness”, “closeness” et “eigen”. Attention, les points sont tres proches, les effets de courbures sont accentué.

Ces différent indicateurs nous donne 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 indépendamment des algorithmes d'apprentissage. On obtient ainsi 4 courbes temporelles. On remarque que les courbes de centralité propre et de degré 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 diminue pour les agents ayant obtenu le plus grands score. Ces deux indicateurs sont plus lié à la structure des chemins qu'au flux de réseau.

Ces deux observations implique que les reseaux les plus optimaux au dernier instant sont plus efficace 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 phylogénèse. Chacun des noeuds 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.

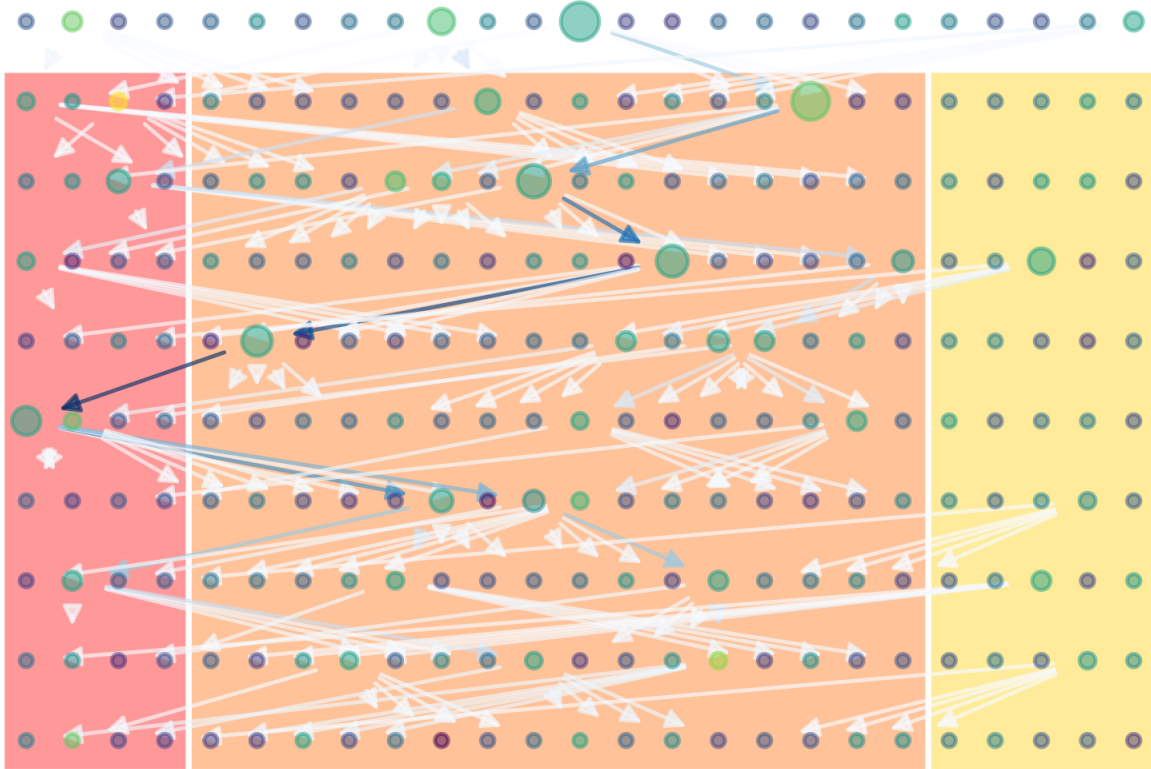


Figure 10: Arbres phylogenetique des agents en fonction de leurs score. De haut en bas, les generation successive. Les couleurs des noeud corresponde au score de l'agent à la fin de l'experience, leurs taille est proportionnele à leur nombre de descendance. En rouge le meilleurs scores. Les encadrés 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'à 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 réussi à se distinguer et se maintenir jusqu’à la dernière génération. Seul deux agents de la 4eme générations se sont maintenu jusqu’à la dernière, mais ceux-ci domine à 50% la population d’agent à la dernière. Même après 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 réussi à ce maintenir autant. Est-ce qu’il y a une strategies mis en place par les agents pour rester le plus longtemps proie, et le moins longtemps prédateurs ? Est ce qu’il y a une stratégie 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 dernière génération ayant le plus grands ancetre, les mouvement recurant des agents par la dispersion des trajectoire et la périodicité des actions.

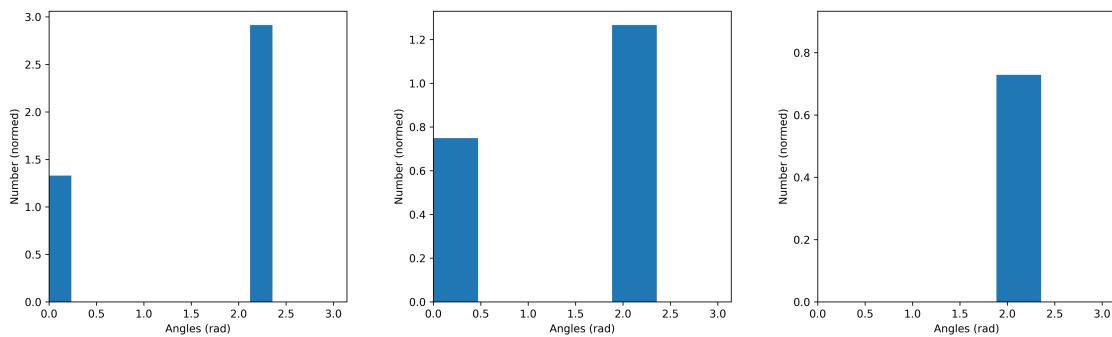


Figure 11: Convergence des strategie en fonction des embranchement les plus long : mesure de la dispersion moyenne, position moyenne et schéma recurrence (mouvement repeter de l’agent). De gauche à droite, de celui qui a le plus d’ancetre à celui en a le moins.

On mesure l’angle des vecteurs de déplacement 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’évolution. Néanmoins, 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 déplaçant qu’en diagonale. D’autant plus que la condition limites periodique $\text{mod}(v, N)$ permet d’envoyer l’agent à l’opposé de l’adversaire sans que lui puisse le suivre car ne peut franchir la limite.

Ces dernières observations montre bien qu’il y a une stratégie dominante, mais qu’un reseau quelconque 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’études 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 expérimentales de problème de régression, classification et partitionnement (corrélation de gène, transcriptome, etc, [source]), l'autre dans l'inspiration pour créer des modèles d'apprentissage liés au réseau de neurone. Ici on a vu un problème qui lie les deux, comment les réseaux de neurones peuvent se structurer au cours de l'évolution ? Cette problématique permettrait de modéliser l'apparition de vie intelligente, tout en créant de nouveaux outils pour l'apprentissage non supervisé.

CONTRAINTE GÉOMÉTRIQUE : Néanmoins cette étude peut comporter plusieurs contraintes limitantes. La première est directement liée à la géométrie des problèmes d'automates cellulaires dans des grilles 2D. Lorsque les bords du haut sont reliés à ceux du bas, et ceux de gauche, sont reliés à ceux de droite, on obtient des conditions limites périodiques de type géométrie torique. Cette géométrie simple à mettre en œuvre comporte plusieurs problèmes "LISTER LES PROBLÈMES + REF". Une autre façon, serait soit d'avoir des liaisons entre les limites, mais cela changerait complètement la symétrie du problème, tel que d'avoir une grille avec des limites décalées comme un pavage, ou encore d'avoir un rayon de limite, de façon à ce que le problème corresponde à un cercle, plutôt qu'à un carré.

VALLÉE DE STABILITÉ LIMITÉE : L'autre contrainte est liée au modèle de construction du réseau : il ne peut y avoir de suppression de couche de neurone. Ce choix est motivé par le fait de ne pas déstructurer complètement la topologie du réseau à chaque suppression de couche. Mais en faisant cela, on se peut retrouver avec des couches avantageuses à une génération donnée, mais inutile voire désavantageuses à une génération future. À cette génération, il est plus probable que l'agent soit plus efficace qu'un agent généré aléatoirement, on se retrouve avec un problème connu en biologie évolutive : l'apparition de structure vestigiale. Ces structures ne sont pas les plus optimales pour atteindre la vallée de stabilité du problème "physique" la plus basse, mais leur suppression nécessiterait trop de mutation, ce qui ferait sortir l'agent de la vallée de stabilité obtenue par le réseau de neurone.

COMPORTEMENT DE TRICHE : Malgré cela, on a obtenu une stratégie convergente au cours du processus évolutif et cela pour deux réseaux de neurone différents. Néanmoins le fait que le problème soit de condition limite périodique 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 critère est donc à prendre en considération pour ne pas surinterpréter les résultats. Cette observation de "triche" du réseau de neurone dans le cadre d'apprentissage par renforcement n'est pas quelque chose d'inconnu. En effet, plusieurs modèles, comme le NEAT peuvent générer des structures qui exploitent les failles d'un jeu, comme le cas où l'agent tourne en rond entre l'arrivée et le départ dans un jeu de course, ou encore l'araignée qui marche à l'envers pour optimiser sa marche [ref].

VERS UNE SYSTÉMISATION : L'optimisation de la structure du réseau est un problème qui pourrait être plus envisagé dans l'avenir des réseaux de neurones. Comme on l'a vu, il existe déjà des modèles qui optimisent la structure comme NEAT, mais il en existe d'autres. Pour ne citer que lui, il y a le modèle GNN, *Graph Neural Network*, celui-ci est par contre réservé à l'analyse des graphes. Le modèle GNN va construire un sous-graphe résultant de la convolution de plusieurs graphes à étudier, ce sous-graphe correspond au réseau de neurone le plus optimisé pour donner les informations d'un graphe de grande taille [Jie Zhou 2020]. Dans notre cas, nous avons utilisé un processus évolutif qui va changer les connexions entre couches uniquement, pour n'importe quel type d'information d'entrée, même les graphes [articles maximilien, vectorisation graph]. Néanmoins, il est possible de rendre notre modèle modulaire, en effet, il peut y avoir un niveau supplémentaire de connexion si celui-ci contient plus de fonction, ou que les entrées changent au cours du temps, comme par exemple le cas de l'étude de l'évolution de la morphologie [Michał Joachimczak 2015 : passe science trouvaille]. On aurait plusieurs fonctions définies évolutivement via notre modèle, ceux-ci seraient liés à des sorties d'action, mais aussi quelque connexion entre modules neuronaux, elle-même générée par un processus évolutif similaire. L'ensemble formerait ce qu'on pourrait appeler un système.

VERS DES MODÈLES COMPLETS : Notre modèle répond à la question : Comment construire un réseau de neurone où aussi bien la structure que le processus d'apprentissage pour que soit fonctionnalisé ? Cela reste dans le cadre d'un système capable d'apprentissage évolutif, mais la pour le compléter, il faudrait que le système s'auto-entretienne, et pour cela, il faudrait ajouter la notion de nourriture/nutriment. De plus, nos agents sont indépendants les uns des autres, ce qui ne reproduit pas ce qu'il se passerait dans un écosystème réel. On pourra ainsi avoir une mesure de la valeur sélective, ou *fitness* en anglais. Pour compléter l'approche, on faudrait créer un environnement où l'ensemble des agents y serait contenu et où leur survie et reproduction serait incluse dans les règles du jeu. On peut penser le cas, où il y a des agents et de la nourriture distribuée aléatoirement avec des "tailles/ressources" différentes. Lorsqu'un agent prend de la nourriture, il gagne en taille jusqu'à 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'à sa disparition et réciproquement. L'environnement contiendrait un

certain nombre limité d'agent, ou lorsque se nombre diminue, 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'environnement 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 ségrégation lié à l'erreur fondamental d'attribution [Polygon, SCHELLING 1971 + Attribution], les mouvement collectif et l'exploration [Vicsek], le dilemme du prisonnier lié à la coopération [Donnant-donnant], les contrainte géographique et de langues [Braudel, Marshall], ou encore, l'héritage culturel [Bourdieu]. Ces outils pourrait etre interessant pour des simulations de vie et pour mieux comprendre l'emergence des structure sociale [Strauss, Sim's].

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 partitionnement. 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 élevé 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'entraînement est un probleme ouvert et pourra etre abordé dans le cadre d'un futur projet.

References

- Bartlett, Stuart and Michael L. Wong (Apr. 2020). "Defining Lyfe in the Universe: From Three Privileged Functions to Four Pillars". In: *Life* 10.4. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 42. DOI: 10.3390/life10040042. URL: <https://www.mdpi.com/2075-1729/10/4/42> (visited on 05/12/2021).
- Bonner, John Tyler (1998). "The origins of multicellularity". In: *Integrative Biology: Issues, News, and Reviews* 1.1, pp. 27–36. ISSN: 1520-6602. DOI: [https://doi.org/10.1002/\(SICI\)1520-6602\(1998\)1:1<27::AID-INBI4>3.0.CO;2-6](https://doi.org/10.1002/(SICI)1520-6602(1998)1:1<27::AID-INBI4>3.0.CO;2-6). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291520-6602%281998%291%3A1%3C27%3A%3AAID-INBI4%3E3.0.CO%3B2-6> (visited on 05/12/2021).
- Buick, Roger (Aug. 27, 2008). "When did oxygenic photosynthesis evolve?" In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 363.1504. Publisher: Royal Society, pp. 2731–2743. DOI: 10.1098/rstb.2008.0041. URL: <https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2008.0041> (visited on 05/12/2021).
- Hedges, S. Blair et al. (Jan. 28, 2004). "A molecular timescale of eukaryote evolution and the rise of complex multicellular life". In: *BMC Evolutionary Biology* 4.1, p. 2. ISSN: 1471-2148. DOI: 10.1186/1471-2148-4-2. URL: <https://doi.org/10.1186/1471-2148-4-2> (visited on 05/12/2021).
- Huxley, J. (1942). "Evolution. The Modern Synthesis." In: *Evolution. The Modern Synthesis*. Publisher: London: George Allen & Unwin Ltd. URL: <https://www.cabdirect.org/cabdirect/abstract/19432202794> (visited on 05/12/2021).

- Knoll, Andrew et al. (2006). "The Ediacaran Period: a new addition to the geologic time scale". In: *Lethaia* 39.1. _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1080/00241160500409223>, pp. 13–30. ISSN: 1502-3931. DOI: <https://doi.org/10.1080/00241160500409223>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1080/00241160500409223> (visited on 05/12/2021).
- Matthews, Gary G. (Dec. 27, 2000). *Neurobiology: Molecules, Cells and Systems*. Wiley. 604 pp. ISBN: 978-0-632-04496-2.
- Mojzsis, S. J. et al. (Nov. 1996). "Evidence for life on Earth before 3,800 million years ago". In: *Nature* 384.6604. Number: 6604 Publisher: Nature Publishing Group, pp. 55–59. ISSN: 1476-4687. DOI: 10.1038/384055a0. URL: <https://www.nature.com/articles/384055a0> (visited on 05/12/2021).