Rapport de Stage Master 2

Antonin Calba

Supervisé par
Jeremy Fix et Alain Dutech

21 septembre 2020

Table des matières

Chapitre 1

Introduction

1.1 Sujet du stage

Auto-organisation pour l'apprentissage du support dans le cadre de l'approximation de fonction et application à l'apprentissage par renforcement.

1.2 Modification

Mon stage devait s'effectuer dans les bureaux de l'équipe biscuit au Loria, cependant pour cause de confinement l'intégralité du stage fut réalisé en télé-travail depuis mon domicile.

1.3 Résultat

Une méthode d'interpolation proposée par Göppert fut implémentée et donna des résultats satisfaisants. L'apprentissage des points supports nécessaires à l'interpolation fut réalisé avec des résultats corrects et qui semble être meilleur avec l'utilisation du modèle SOM plutôt que DSOM. L'association des cartes auto-organisatrices et de l'algorithme de Q-learning ont donné des résultats satisfaisants pour résoudre le problème de Mountain Car. Cette fois, il semble que ce soit l'utilisation de DSOM qui permet d'obtenir les meilleurs politiques.

Chapitre 2

Loria et Cadre de travail

Mon stage fut effectué au sein de l'équipe biscuit au Loria. Le Loria est le laboratoire lorrain de recherche en informatique et ses applications. Il est situé à Vandœuvre-lès-Nancy et Villers-lès-Nancy, près de Nancy, sur le campus de l'université Henri-Poincaré. L'équipe biscuit a pour objectif d'étudier les propriétés qui émergent d'architectures de calcul non conventionnelles pour la réalisation ou le contrôle de systèmes autonomes persistants interagissant avec des environnements dynamiques complexes. Cependant, pour cause de confinement, l'intégralité du stage fut réalisé par télétravail.

Chapitre 3

Sujet et contexte

L'intelligence artificielle est depuis plusieurs années revenue sous les feux des projecteurs grâce à l'accomplissement de grandes avancées dans des domaines allant de la reconnaissance d'image à la voiture autonome en passant par la maîtrise de jeux de plus en plus complexes, dernièrement le jeu de GO. Cependant toutes ces avancées ont été majoritairement accomplies grâce à de l'apprentissage supervisé cependant d'autres méthodes d'apprentissage existent et pourraient se révéler encore moins dépendantes de l'homme. Ce stage a pour but d'explorer d'autres thématiques telles que l'auto-organisation et l'apprentissage non supervisé. Ce stage est la fusion de deux sujets explorant ces thématiques : le premier avait pour objectif d'appliquer l'auto organisation et l'apprentissage par renforcement dans le cas de l'approximation de fonction tandis que le second avait comme application le contrôle d'un agent grâce à ces mêmes thématiques.

Chapitre 4

Modèle

La première partie du stage est consacrée aux différents algorithmes d'auto-organisation classique, ainsi que du modèle DSOM [1], qu'il est possible d'utiliser dans un objectif d'interpolation de fonction et d'apprentissage d'un comportement d'un agent artificiel par des techniques d'apprentissage par renforcement [2].

Les cartes auto-organisatrices sont une classe de réseaux de neurones artificiels fondées sur des méthodes d'apprentissage non-supervisées. Les cas d'utilisation les plus courants consistent en de la cartographie d'espace réel dans le but de faire de la discrétisation, de la quantification vectorielle ou encore de la classification. Elles ont été conceptualisées par Teuvo Kohonen en 1984 et peuvent être nommées réseau de Kohonen. Nous allons dans la suite de ce stage chercher à les utiliser comme fonction d'interpolation et comme politique dans le cadre d'apprentissage par renforcement.

4.1 Neural Gas

Le processus d'apprentissage de *Neural Gas*[3] est un algorithme itératif avec un facteur d'arrêt dépendant du nombre d'itération t_{final}.

Initialement on crée un ensemble de neurones avec chacun un vecteur de référence w_i , avec $i \in [0, N]$, N le nombre de neurones. Les données d'entrée sont des vecteurs stockés dans l'ensemble Ω.

A chaque itération, un vecteur $v \in \Omega$ est présenté au système. Selon v on définit la fonction rang, $k_v : [0, N-1] \mapsto [0, N-1]$, fonction bijective telle que $k_v(i)$ donne la position du i-ième prototype dans la série ordonnée des distances entre les vecteurs référence w_j et l'entrée v tel que $\left\|v - w_{k_v^{-1}(0)}\right\| \leq \left\|v - w_{k_v^{-1}(1)}\right\| \leq ... \leq \left\|v - w_{k_v^{-1}(N-1)}\right\|$. Suite à cet ordre, on modifie les vecteurs référence w_i selon l'équation (4.1) :

$$w_i(t+1) = w_i(t) + \varepsilon(t) h_\lambda(t, k_v(i), v)(v - w_i(t)) \tag{4.1}$$

$h_\lambda(t, k_v(i), v)$ la fonction de voisinage décrit par l'équation (4.2),

$$h_\lambda(t, k_v(i), v) = e^{-\frac{k_v(i)}{\lambda(t)}} \tag{4.2}$$

$\lambda(t)$ et $\varepsilon(t)$ sont respectivement le facteur de voisinage et le taux d'apprentissage définis selon les équations (4.3) et (4.4),

$$\lambda(t) = \lambda_{initial}(\frac{\lambda_{final}}{\lambda_{initial}})^{t/t_{final}} \tag{4.3}$$

$$\varepsilon(t) = \varepsilon_{initial}(\frac{\varepsilon_{final}}{\varepsilon_{initial}})^{t/t_{final}} \tag{4.4}$$

4.2 Growing Neural Gas

Le processus d'apprentissage de *Growing Neural gas* [4] est un algorithme itératif avec comme facteur d'arrêt un nombre de neurone maximum ou un critère de performance.

Initialement on crée deux neurones avec chacun un vecteur de référence w_i, avec $i \in [0, N]$, N le nombre de neurones, ainsi qu'une valeur d'erreur $\xi_i = 0$. On crée aussi un ensemble C initialement vide représentant toutes les connexions du système, une connexion $c \in C$ est composée de deux neurones différents et d'un âge. Les données d'entrée sont des vecteurs stockés dans l'ensemble Ω.

A chaque itération, un vecteur $v \in \Omega$ est présenté au système. On recherche les deux neurones s_1 et s_2 possédant le vecteur de référence avec la distance la plus faible de v tels que $\|v - w_{s_1}\| \leq \|v - w_{s_2}\| \leq \|v - w_j\|, \forall j \neq s1, j \neq s2$. Toutes les connexions comprenant s_1 voient leur valeur d'âge s'incrémenter d'une valeur constante tout au long de l'exécution, arbitrairement de 1 . Puis on modifie la valeur d'erreur de s_1 selon l'équation (4.5).

$$\xi_{s_1}(t+1) = \xi_{s_1}(t) + \|w_{s_1} - v\|^2 \tag{4.5}$$

On modifie le vecteur de référence de s_1 selon l'équation (4.6) et les vecteurs de référence des neurones n aillant une connexion avec s_1 selon l'équation (4.7) avec ϵ_b et ϵ_n les facteurs d'apprentissage constants ($\epsilon_b \geq \epsilon_n$).

$$w_{s_1}(t+1) = w_{s_1}(t) + \epsilon_b(v - w_{s_1}) \tag{4.6}$$

$$w_n(t+1) = w_n(t) + \epsilon_n(v - w_n) \tag{4.7}$$

Si il n'existe pas de connexions entre s_1 et s_2, il faut en créer une nouvelle avec l'âge de la connexion initialisé à 0. Toutes les connexions c avec $a_c \geq a_{max}$ sont supprimées.

Toutes les λ itérations un nouveau neurone r est créé. Pour savoir quel est son vecteur de référence et son erreur, il est nécessaire de trouver le neurone q tel que $\xi_q \geq \xi_j, \forall j \neq q$ ainsi que le neurone f, voisin de q, avec l'erreur la plus grande. On décroît les valeurs d'erreur ξ_q et ξ_f selon les équations (4.8) et (4.9), avec α une constante. On initialise r selon les équations (4.10) et (4.11) [1]. La connexion entre f et q est supprimée et remplacée par deux nouvelles entre r,f et r,q.

$$\xi_q(t+1) = \alpha\xi_q(t) \tag{4.8}$$

$$\xi_f(t+1) = \alpha\xi_f(t) \tag{4.9}$$

$$w_r = \frac{w_q + w_f}{2} \tag{4.10}$$

$$\xi_r = \frac{\xi_q + \xi_f}{2} \tag{4.11}$$

Pour finir toutes les erreurs ξ_i décroissent selon l'équation (4.12), avec β une constante.

$$\xi_i(t+1) = \xi_i(t) - \beta\xi_i(t) \tag{4.12}$$

4.3 Self Organizing Map (SOM)

Le fondement de SOM[5] repose sur une carte neuronale avec une structure où chaque neurone est relié par des connexions à des voisins (généralement au sein d'une grille régulière). Le processus d'apprentissage de SOM est un algorithme itératif avec un facteur d'arrêt dépendant du nombre d'itération t_{final}.

Initialement, on crée un ensemble de neurones avec chacun un vecteur de référence w_i, avec $i \in [0, N]$, N le nombre de neurones, et les connexions entre ces différents neurones. Les données d'entrée sont des vecteurs stockés dans l'ensemble Ω.

A chaque itération, un vecteur $v \in \Omega$ est présenté au système. On recherche le neurone s, aussi appelé BMU (*Best Matching Unit*), possédant le vecteur de référence avec la distance la plus faible de v tel que $\|v - w_s\| \leq \|v - w_j\|, \forall j$.

Ensuite, on modifie la valeur du vecteur de référence de chaque neurone selon l'équation (4.13),

$$w_i(t+1) = w_i(t) + \varepsilon(t)h_\lambda(t, i, s)(v - w_i(t)) \tag{4.13}$$

1. Dans son article [4] fritzke propose $\xi_r = \xi_q$

avec $h_\lambda(t,i,s)$ la fonction de voisinage décrit par l'équation (4.14) où $D(i,s)$ correspondant à la distance, généralement Euclidienne, entre les neurones i et s dans la carte neuronale.

$$h_\lambda(t,i,s) = e^{-\frac{D(i,s)^2}{2\lambda(t)^2}} \tag{4.14}$$

$\lambda(t)$ et $\varepsilon(t)$ sont respectivement le facteur de voisinage et le taux d'apprentissage définis selon les équations (4.15) et (4.16).

$$\lambda(t) = \lambda_{initial}(\frac{\lambda_{final}}{\lambda_{initial}})^{t/t_{final}} \tag{4.15}$$

$$\varepsilon(t) = \varepsilon_{initial}(\frac{\varepsilon_{final}}{\varepsilon_{initial}})^{t/t_{final}} \tag{4.16}$$

4.4 Dynamic Self Organisation Map (DSOM)

Comme pour SOM, DSOM [1] repose sur une carte neuronale avec une structure où chaque neurone est relié par des connexions à des voisins (généralement au sein d'une grille régulière). Le processus d'apprentissage de DSOM est un algorithme itératif qui ne dépend pas du temps, le facteur d'arrêt est donc généralement un critère de performance. DSOM a pour avantage, contrairement à SOM, de ne pas dépendre du temps et donc offre des possibilités de *tracking* et d'être bien moins tributaire de la densité des échantillons présentés à la carte.

Initialement, on crée un ensemble de neurones avec chacun un vecteur de référence w_i, avec $i \in [0,N]$, N le nombre de neurones, et les connexions entre les différents neurones. Les données d'entrée sont des vecteurs stockés dans l'ensemble Ω.

A chaque itération, un vecteur $v \in \Omega$ est présenté au système. On recherche le neurone s possédant le vecteur de référence avec la distance la plus faible de v tels que $\|v - w_s\| \leq \|v - w_j\|, \forall j$. Ensuite, on modifie la valeur du vecteur de référence de chaque neurone selon l'équation (4.17),

$$w_i(t+1) = w_i(t) + \varepsilon \|v - w_i(t)\|_\Omega h_\eta(i,s,v)(v - w_i(t)) \tag{4.17}$$

avec $h_\eta(i,s,v)$ la fonction de voisinage décrit par l'équation (4.18) où $D(i,s)$ correspond à la distance entre les neurones i et s dans la carte neuronale

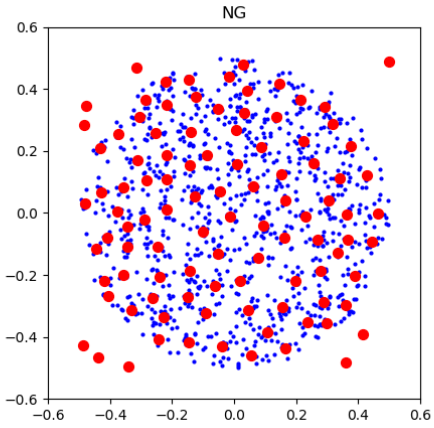
$$h_\eta(i,s,v) = e^{-\frac{1}{\eta^2}\frac{D(i,s)^2}{\|v-w_s\|_\Omega^2}} \tag{4.18}$$

η étant le facteur d'élasticité, ε le taux d'apprentissage et $\|x\|_\Omega$ la norme euclidienne normalisée sur l'ensemble Ω telle que $\|x\|_\Omega = \frac{\|x\|}{max_{y,z\in\Omega}\|y-z\|}$.

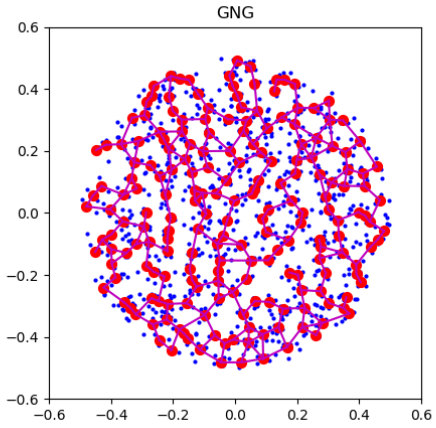
Dans le cas où $v = w_s$, on fixe $h_\eta(i,s,v) = 0$.

4.5 Exemple d'apprentissage des modèles

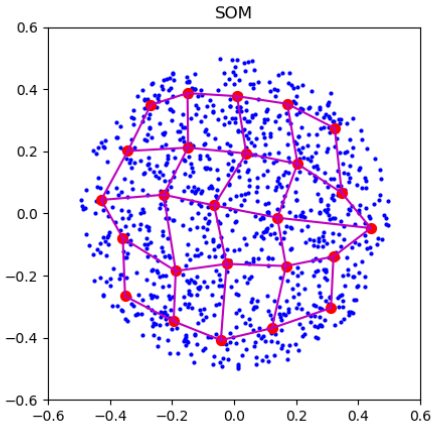
Dans le but d'illustrer ces différents algorithmes, nous proposons une expérience illustrative. Le protocole consiste à initialement tirer un ensemble de 1000 point de 2 dimension repartis de manière uniforme dans un disque de rayon 0,5 centré en [0 ;0]. Puis, on sélectionne aléatoirement un point parmi la totalité de l'ensemble. Le point tiré est présenté aux modèles. Les modèles actualisent les neurones selon leurs spécificités. Puis on répète pendant 10000 itérations. Suite à quoi on peut récupérer la position des vecteurs référence des différents modèles superposé à l'ensemble des 1000 points on obtient les figures 5.1 avec en bleu les points de l'ensemble, en rouge les neurones et en vert les éventuelles connections entre les neurones.



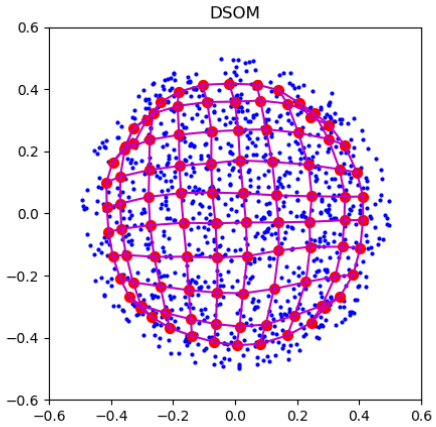
(a) Neural Gaz
$N = 100$, $\lambda_{initial} = 1.0$, $\lambda_{final} = 0.01$,
$\varepsilon_{initial} = 0.5$, $\varepsilon_{final} = 0.01$



(b) Growing Neural Gas
$\varepsilon_b = 0.2$, $\varepsilon_n = 0.0006$, $a_{max} = 10$,
$\lambda = 10$, $\alpha = 0.5$, $\beta = 0.995$



(c) Self Organizing Map
grille 5x5, $\lambda_{initial} = 1.0$, $\lambda_{final} = 0.01$,
$\varepsilon_{initial} = 0.5$, $\varepsilon_{final} = 0.01$



(d) Dynamic Self Organisation Map
grille 10x10, $\eta = 20$, $\varepsilon = 0.1$

FIGURE 4.1 – Application des algorithmes NG, GNG, SOM et DSOM sur un même ensemble de donnée

Chapitre 5

Interpolation de fonction grâce aux cartes auto-organisatrices

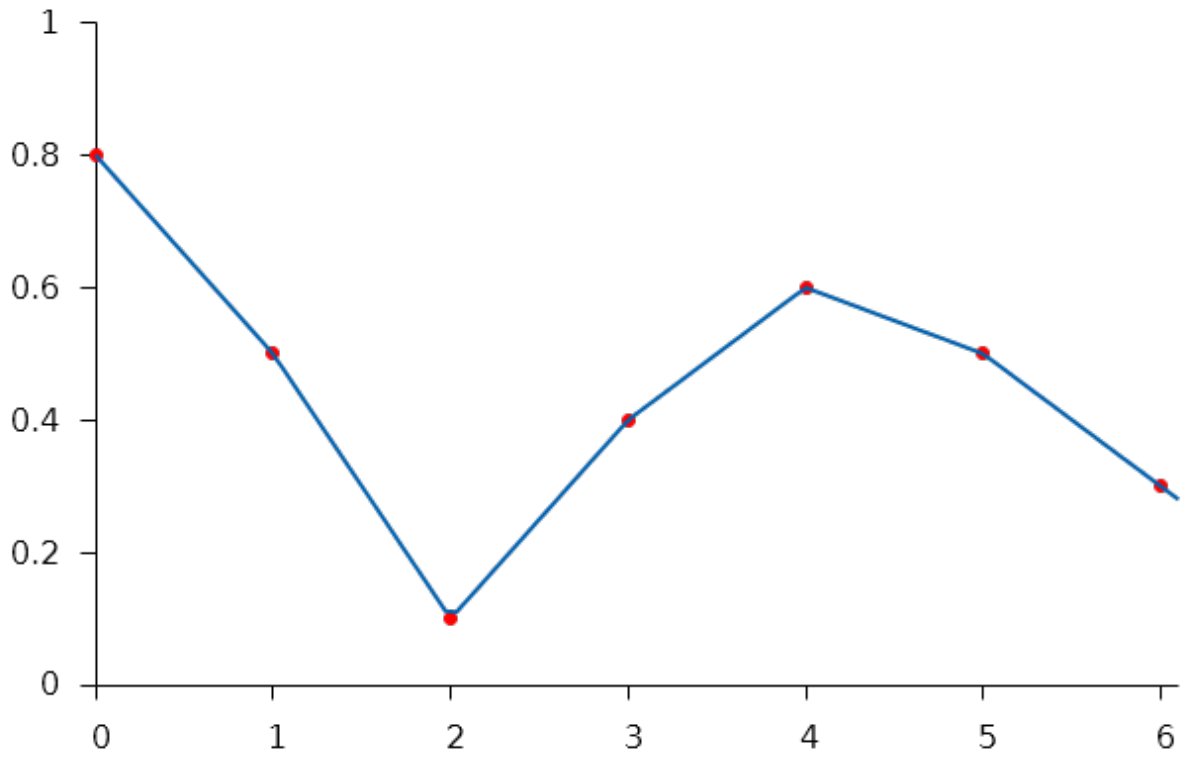


FIGURE 5.1 – Exemple interpolation

Suite à cette introduction, cette seconde partie est consacrée à la recherche d'une méthode pour réaliser de l'interpolation en utilisant les cartes de Kohonen. L'interpolation est une opération permettant de calculer une estimation de fonction lorsqu'on ne connaît qu'un nombre fini de points. Les points sont utilisés comme support pour l'estimation de la fonction. Dans la suite de ce rapport, ces points seront nommés "points support" en référence à ce rôle. Ils sont composés de deux parties : la première correspondant aux entrées de la fonction est appelée *input* (abrégé en *in* dans les formules) et la seconde qui correspond aux sorties est appelée *output* (abrégé en *out* dans les formules). Il existe plusieurs méthodes pour réaliser une interpolation : la plus simple est l'interpolation linaire consistant à relier les points par des segments de droite comme on peut le voir sur la figure 5.1.

Le sujet de recherche de cette partie est consacré à la recherche d'un moyen d'appliquer les mécanismes d'apprentissage des cartes de Kohonen à l'interpolation de fonction ; dans l'objectif de réussir à avoir de bonnes capacités de généralisation pour approximer une fonction à partir d'un nombre de points support réduit.

Parmi la littérature scientifique, l'article qui semble le plus intéressant pour cette problématique est celui de Josef Göppert et Wolfgang Rosentiel [6]. Pour valider ses algorithmes et ses résultats, je propose une implémentation et une réplication.

5.1 Interpolation continue avec des cartes auto-organisées

Göppert dans son article [6] propose une méthode d'interpolation utilisant des cartes de kohonen. Cette méthode appelée "*continuous interpolating self organizing map*" abrégée en CI-SOM fonctionne grâce à l'utilisation de points support de la fonction que l'ont souhaite interpoler qui ont été préalablement appris grâce à une carte auto adaptative.

Pour réaliser cette méthode d'interpolation, on peut noter deux grandes phases. La première phase consiste à apprendre une carte de Kohonen, de dimension D, en utilisant l'ensemble des données connues pour obtenir une généralisation utilisable comme points supports. Durant l'apprentissage de la carte de Kohonen, les vecteurs d'entrées de la carte correspondent à une concaténation des input et output de la fonction à interpoler. Les vecteurs prototypes des neurones de la carte de Kohonen apprennent cette concaténation lors de l'apprentissage. La seconde phase utilise les vecteurs prototypes des neurones comme points supports pour l'application des d'algorithmes d'interpolation.

5.1.1 I-SOM

CI-SOM est inspiré d'une méthode d'interpolation antérieure nommée I-SOM [7]. Le concept d'I-SOM est de construire un système localement linéaire et la fonction interpolée passant exactement par chaque point support. On donne un vecteur d'entrée X et l'ensemble des points supports $w_i \in W, i \in [0, N]$, ainsi que les liens de voisinage des neurones utilisés comme points supports lors de l'apprentissage de la carte de Kohonen. X et $w_i^{(in)}$ sont présents dans l'espace d'entrée de la fonction et possèdent le même espace vectoriel $E_{(in)}$ de dimension $D_{(in)}$. Tandis que $w_i^{(out)}$ possèdent un autre espace vectoriel $E_{(out)}$ de dimension $D_{(out)}$ qui est commun avec le vecteur de sortie Y que l'on cherche à interpoler.

Pour calculer d'après la méthode I-SOM le vecteur sortie Y associé à X, on recherche le point support le plus proche de X, $w_{BMU}^{(in)}$. Une première approximation pourrait être de prendre la valeur de sortie de ce point support $w_{BMU}^{(out)}$, on aurait le même comportement que celui de l'algorithme "*Plus proche voisin*" équation 5.1.

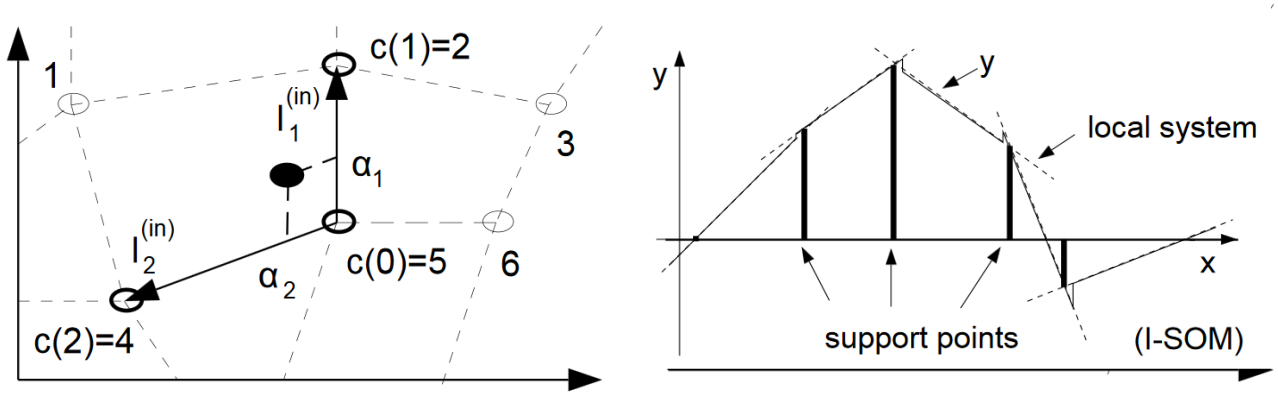
$$Y(X) = w^{(out)}_{\text{argmin}_i \|X - w_i^{(in)}\|^2} = w_{BMU}^{(out)} \tag{5.1}$$

Cependant, grâce aux voisinages utilisés lors de l'apprentissage, il est possible de récupérer les points supports voisins dans chaque dimension de $E_{(in)}$. Les voisins de w_{BMU} sont généralement aux nombres de deux dans chaque dimensions $d \in [0, D_{(in)}]$. Parmi ces deux voisins on recherche le plus proche de X, que l'on note $c(d)$ et ayant un vecteur référence $w_{c(d)}^{(in)}$. On peut donc calculer la distance entre $w_{c(d)}^{(in)}$ et $w_{BMU}^{(in)}$ nommée $l_d^{(in)}$ et on reproduit cela pour chaque dimension pour finir par obtenir le vecteur $L^{(in)}$. Une représentation est proposé par Göppert pour visualiser la signification des $l^{(in)}$ et de α figure 5.2. On peut résumer cela avec l'équation 5.2.

Grâce à $L^{(in)}$, il est possible de faire une approximation de la valeur α selon l'équation 5.3. Cette équation permet de calculer des droites affines passant par les deux points dans chaque dimension, α est la solution du problème de minimisation $\text{argmin}_\alpha \|X - (w_{BMU}^{(in)} + L^{(in)}\alpha)\|_2^2 + \lambda\|\alpha\|_2^2$. Ensuite, de la même manière que pour $L^{(in)}$ on calcule $L^{(out)}$, équation 5.4. Pour finir, on calcule la sortie Y selon l'équation 5.5 en utilisant α calculé précédemment.

$$l_d^{(in)} = w_{c(d)}^{(in)} - w_{BMU}^{(in)}; L^{(in)} = \left[l_0^{(in)} l_1^{(in)} ... l_D^{(in)}\right] \tag{5.2}$$

$$\alpha = (L^{(in)T} L^{(in)} + \lambda I)^{-1} L^{(in)T} (X - w_{BMU}^{(in)}) \tag{5.3}$$



(a) Linéarisation par un système local dans un cas bidimensionnel

(b) Exemple d'entrée unidimensionnelle (x) et de sortie unidimensionnelle (y). Interpolation linéaire des points support.

FIGURE 5.2 – Schéma d'interpolation I-SOM, Figure 1 dans l'article de Göppert [6]

$$l_d^{(out)} = w_{c(d)}^{(out)} - w_{BMU}^{(out)}; L^{(out)} = \left[l_0^{(out)} l_1^{(out)} ... l_D^{(out)}\right] \tag{5.4}$$

$$Y_{I-SOM} = w_{BMU}^{(out)} + L^{(out)}\alpha \tag{5.5}$$

Dans son article [6], Göppert propose de visualiser les propriétés de I-SOM et CI-SOM par l'intermédiaire d'un problème synthétique. Ce problème possède une seule dimension d'entrée et une seule de sortie ($D_{(in)} = 1$ et $D_{(out)} = 1$). L'ensemble des points supports est composé de 6 points ($w_i^{(in)}$,$w_i^{(out)}$) $\in$ [(0,0),(1,0),(4.5,1),(5.5,-1),(9,0),(10,-3)]. Si on applique la méthode I-SOM à cet ensemble, on obtient la courbe noire sur la figure 5.3 (qui correspond à la figure 2a de [6]). Sur cette figure 5.3 les barres horizontales colorées en bas de cette image correspondent à différents intervalles. La première barre (1*) correspond aux intervalles où un point est le plus proche. Chaque intervalles d'entrée est ainsi coloré de la couleur du point support qui est le BMU. La seconde barre (2*) correspond aux intervalles où un point est le second plus proche. Cette seconde barre permet de visualiser les points supports qui définissent le référentiel local, défini par la matrice $L^{(in)}$. La dernière barre correspond à la droite affine utilisée pour calculer de manière approchée la fonction (couleurs des intervalles 1* et 2* associé aux points support, 3* aux droites). On peut constater que les segments de droite utilisés lors de l'interpolation par la méthode I-SOM correspondent aux segments de droite formés par les couples [points les plus proches, seconds points les plus proches].

5.1.2 CI-SOM

Comme on le voit avec cet ensemble sur la figure 5.3, cette méthode pose des gros problèmes de discontinuité lors du changement de point support. La solution apportée par Göppert est de ne plus seulement utiliser les deux points supports les plus proches de chaque dimension de la carte de Kohonen. Mais d'utiliser tout les points supports dans un intervalle autour de X avec comme condition que si $X = w_i^{(in)}$ alors $Y = w_i^{(out)}$ sinon l'importance de la contribution des points support est pondéré selon leurs distances à X. Pour cela, il propose d'utiliser la formule de Shépard permettant de calculer l'influence d'un point support selon sa distance à X. La méthode de Shepard est une technique d'interpolation spatiale dans la famille dite de pondération inverse à la distance. La méthode de Shepard introduit une fonction ϕ pour chaque point support. Il en existe deux variantes (eq 5.6 et eq 5.7) pour calculer la valeur ϕ grâce à cette formule de Shepard, une qu'on peut qualifier de globale qui prend en compte tous les points supports (eq 5.6) et une locale qui ne prend en compte que les points qui sont dans un intervalle fixé autour de X(eq 5.7)

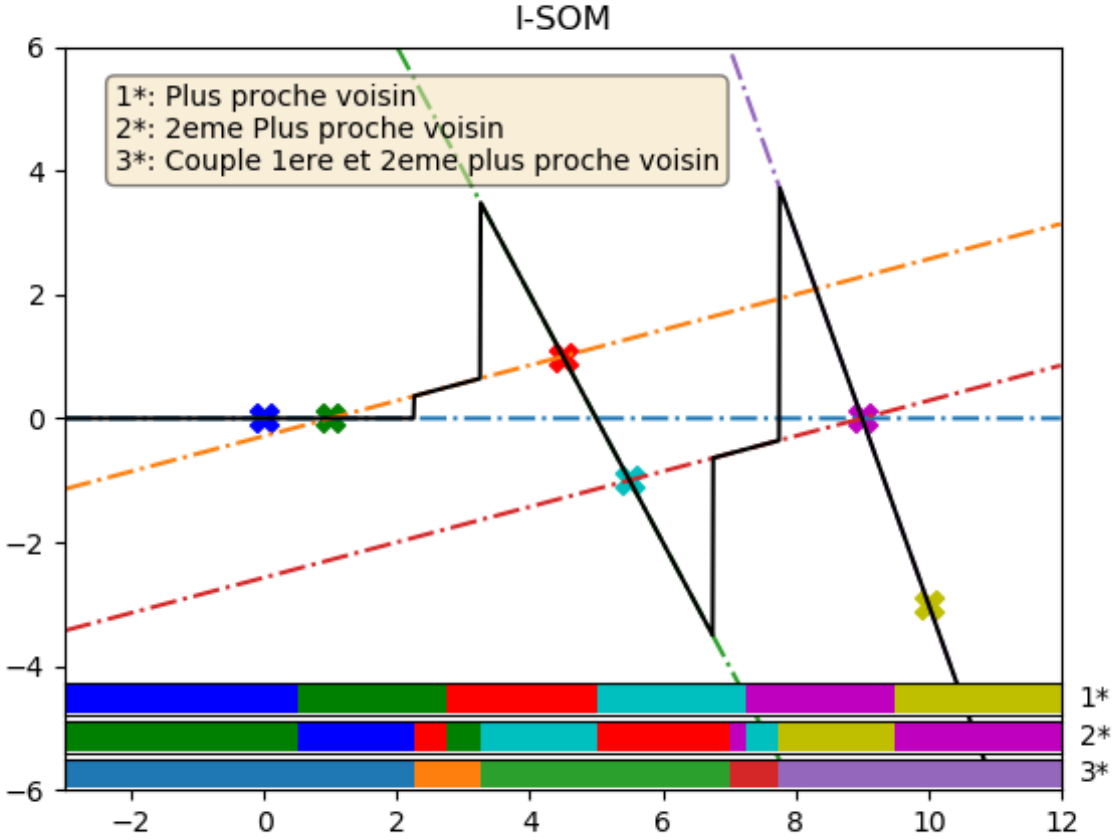


FIGURE 5.3 – Application de l'algorithme I-SOM courbe noire. Représentation des droites entre les couples des plus proches points

$$\phi_G(w, W, X) = \frac{1}{d(w, X)^\mu} / (\sum_{i=1}^{N} \frac{1}{d(w_i, X)^\mu}) \tag{5.6}$$

$$\phi_L(w, W, X) = [\frac{(R - d(w, X))+}{R * d(w, X)}]^\mu / (\sum_{i=1}^{N} [\frac{(R - d(w_i, X))+}{R * d(w_i, X)}]^\mu) \tag{5.7}$$

Dans ces deux formules, $d(w, X)$ correspond à la distance euclidienne entre w et X (eq 5.8).

$$d(w, X) = \sum_{i=1}^{N} (w_i^{(in)} - X_i) \tag{5.8}$$

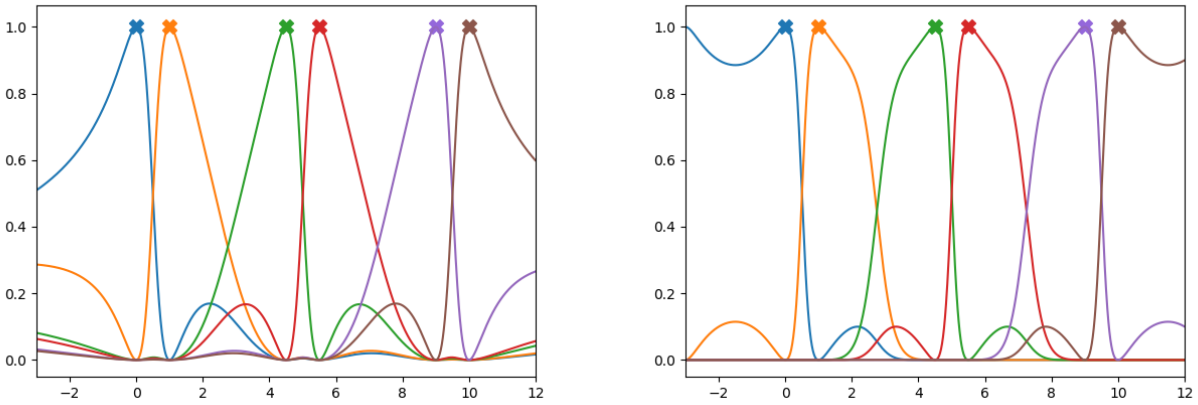
On peut voir sur la figure 5.4 les courbes des ϕ des différents points supports selon les deux méthodes. Le calcul des l^{in} et l^{out} sont adapté (eq 5.9 et 5.10) de manière à prendre en compte l'influence de chaque voisin selon la distance au point support observé. Ainsi, alors que I-SOM ne considère que le vecteur support BMU et les vecteurs supports, voisin de la BMU, les plus proches dans chaque dimension, CI-SOM considère le vecteur support BMU ainsi que l'ensemble des vecteurs supports voisins dans chaque dimension, notés w_{d+} et w_{d-}. Puisque L^{out} et Y_{I-SOM} utilisent ces nouvelles équations incluant tous les points supports, il est nécessaire de pondérer les sorties obtenue selon chaque point support par la formule de Shépard pour obtenir la sortie Y_{CF-SOM} (eq 5.11).

$$l^{(in)}(w, W, X) = \phi_G(w_{d+}, W, X) * (w_{d+}^{(in)} - w^{(in)}) - \phi_G(w_{d-}, W, X) * (w_{d-}^{(in)} - w^{(in)}) \tag{5.9}$$

$$l^{(out)}(w, W, X) = \phi_G(w_{d+}, W, X) * (w_{d+}^{(out)} - w^{(out)}) - \phi_G(w_{d-}, W, X) * (w_{d-}^{(out)} - w^{(out)}) \tag{5.10}$$

$$Y_{CF-SOM}(X, W) = \sum_{i=1}^{N} \phi(w_i^{(in)}, W, X) * Y_{I-SOM}(W, X, i) \tag{5.11}$$

Suite à ces modifications, avec l'ensemble des points précédents, on obtient la figure 5.5. On peut y voir les résultats suite aux changements avec l'utilisation de la formule globale et locale pour calculer l'influence des points supports lors de l'équation 5.11. L'utilisation de manière locale donne de bien meilleurs résultats. Cela se remarque surtout sur les bords de l'ensemble qui continuent de manière linéaire. Alors qu'avec la manière globale on finit par atteindre les valeurs d'une droite affine définie par l'ensemble des points supports qui n'est pas nécessairement une bonne approximation.



(a) Phi global équation 5.6

(b) Phi local équation 5.7

FIGURE 5.4 – Représentation de l'influence de chaque point de l'ensemble proposé par Göppert selon la formule utilisée pour calculer ϕ.

Göppert utilise un autre problème pour visualiser les performances de cette méthode d'interpolation. Ce problème est l'approximation d'une gaussienne en deux dimensions de variance 0,1 (figure 6.9). Les points support sont idéalement placés de manière équidistante tel que représenté par la figure 5.6a. Les figures 5.6c et 5.6e représentent l'application de la méthode CI-SOM selon les deux variantes de formule global et local, on peut constater une variation plus importante avec la méthode global comme pour le problème en une dimension. Cela est encore plus visible lorsque l'observe avec des points support toujours disposés de manière équidistante mais d'un nombre pair comme montré sur la figure 5.6b avec les figures 5.6d et 5.6f , résultats de la méthode CI-SOM.

La réplicabilité de cette méthode semble acquise, l'utilisation de la formule de Shépard local donne de bien meilleurs résultats que la formule globale. Cependant, si on cherche à interpoler trop loin d'un point support, la formule locale n'est plus capable de donner de valeur de sortie. On peut dans ce cas soit appliquer la formule globale soit appliquer la méthode linéaire Y_{I-SOM}.

5.2 Expérimentation CI-SOM avec apprentissage SOM et DSOM

Suite à l'implémentation de cette méthode, plusieurs expérimentations sont proposé. Dans l'article [6], Göppert n'utilise que des points supports fixés manuellement et non appris par une carte de Kohonen. La première expérimentation proposé est de réaliser l'apprentissage des points supports. Le problème utilisé est celui de la gaussienne présentée précédemment (une gaussienne à deux dimensions et de variance 0,1, figure 6.9). Dans un premier temps, en utilisant SOM puis DSOM, les deux modèles

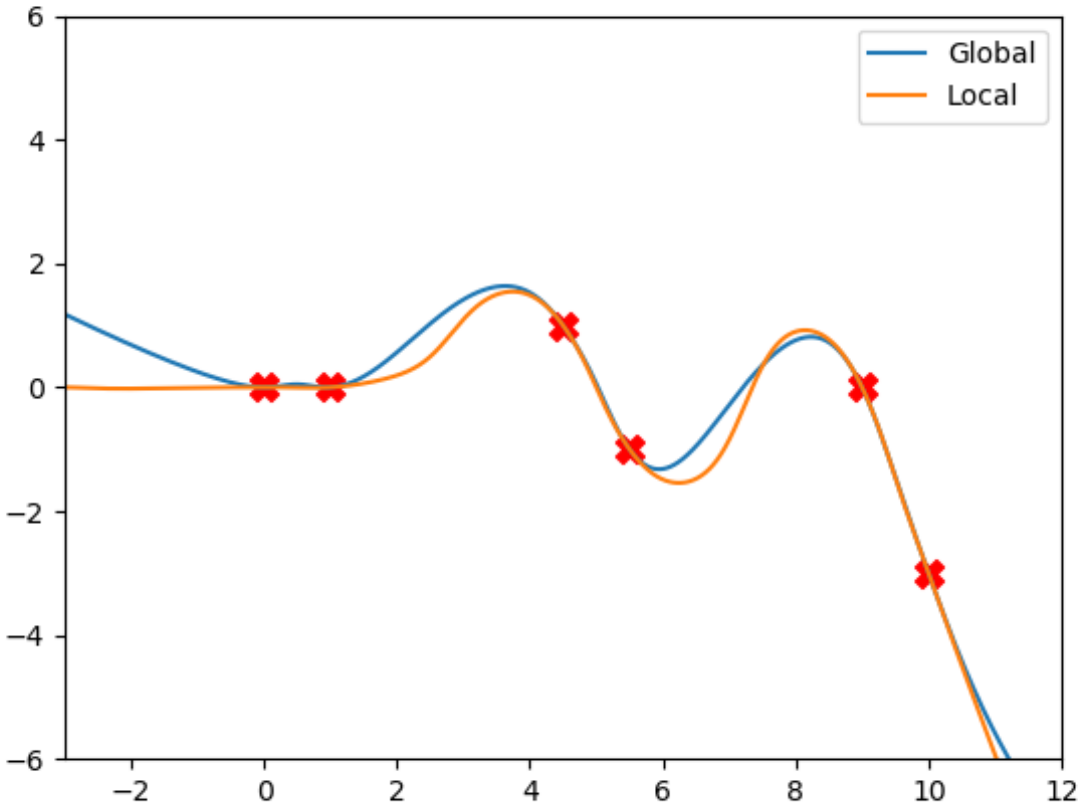


FIGURE 5.5 – Application la méthode interpolation sur problème d'interpolation unidimensionnel proposé par Göppert

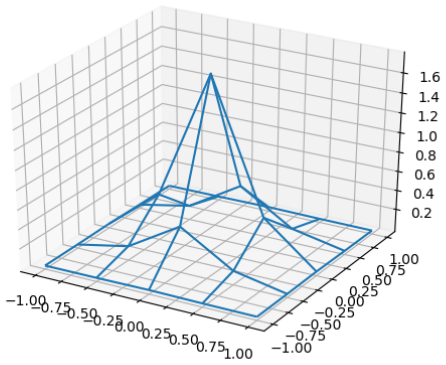
sont comparés selon 3 configurations de données et selon des tailles de grilles variables. Ces comparaisons attribuent de bien meilleurs résultats à SOM, deux alternatives de DSOM sont expérimentés pour essayer d'améliorer un peu ses performances.

La première expérimentation proposé est de ne plus utiliser des points supports fixés, mais des points supports appris grâce à un apprentissage d'une carte de Kohonen selon différentes configurations. Le modèle utilisé est dans un premier temps SOM. Les données d'entrées utilisées sont des vecteurs obtenus en concaténant [input, output] de la fonction gaussienne, l'output correspond à l'image exacte de la fonction gaussienne à interpoler avec les valeurs d'input comme antécédent. Les valeurs d'input sont définies selon 3 configurations différentes. Chacune d'entre elles a pour objectif de pouvoir observer le comportement de la méthode CI-SOM selon différents niveaux d'information. On distinguera deux zones différentes, le pic correspondant à une zone de forte variation et le plateau correspondant à une zone de faible variation.

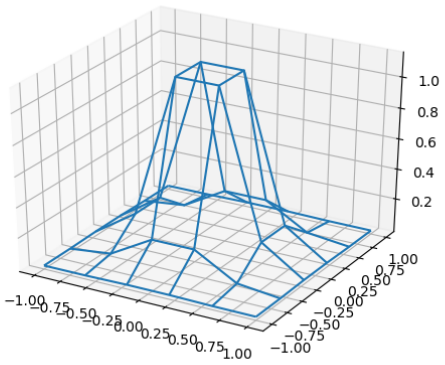
La première configuration des input est une grille régulière d'une taille de 30 sur 30 pour un total de 900 entrées différentes, cette configuration a pour caractéristique d'avoir des points équidistants et de comprendre dans l'ensemble un maximum proche de celui de la gaussienne. Cette configuration possède une information importante sur la gaussienne. On attend de cette configuration une interpolation très proche de la fonction initiale.

Les autres configurations sont aléatoires avec des densités différentes mais toujours avec 900 entrées différentes lors des apprentissages. La seconde configuration présente une densité importante autour du pic de la gaussienne mais en contre partie très peu sur le plateau. Cette configuration possède plus d'information sur la zone du pic mais très peu sur le plateau, éloigné du pic. On s'attend donc à un pic de la gaussienne très bien interpolé mais avec beaucoup plus d'erreurs sur le plateau.

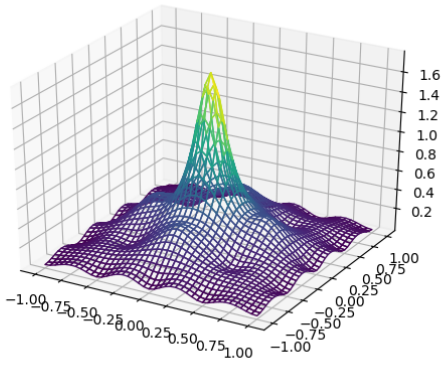
La troisième et dernière configuration présente à l'opposé de la seconde une densité faible sur le



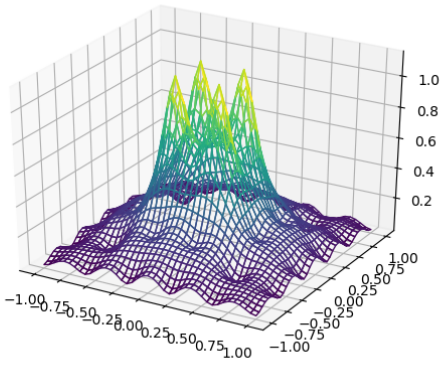
(a) Objectif carte 5 sur 5



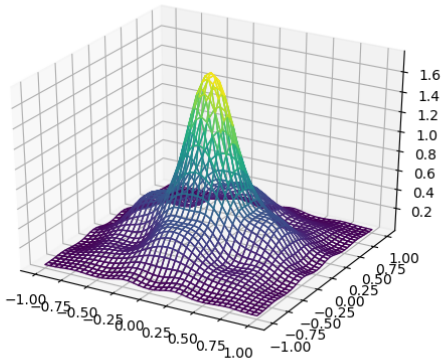
(b) Objectif carte 6 sur 6



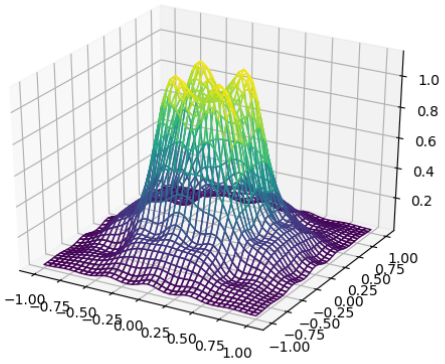
(c) Application de la méthode avec la formule de Shépard global 5 sur 5



(d) Application de la méthode avec la formule de Shépard global 6 sur 6

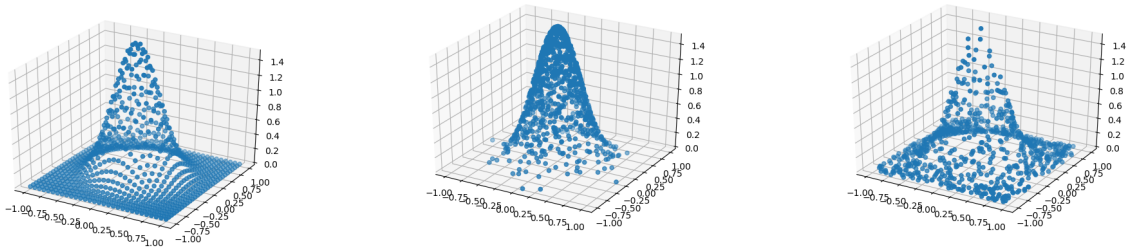


(e) Application de la méthode avec la formule de Shépard local 5 sur 5



(f) Application de la méthode avec la formule de Shépard local 6 sur 6

FIGURE 5.6 – Application de la méthode à deux configurations de gaussienne en 2D.



(a) Distribution en grille

(b) Distribution Gaussienne (sur le pic)

(c) Distribution Inverse Gaussienne (évitant le pic)

FIGURE 5.7 – Distribution donnée d'apprentissage gaussienne

pic, avec une probabilité nulle d'avoir la valeur maximum de la gaussienne, mais une densité beaucoup plus importante sur le plateau. On s'attend donc à obtenir le résultat opposé de la seconde configuration, un pic mal interpolé mais un plateau avec peu d'erreur. La figure 6.9 représente un exemple de chaque configuration. Les résultats de ces trois configurations sont comme prévu assez différents. La configuration la mieux apprise est sans surprise celle de la grille régulière qui permet d'obtenir des résultats bien dépliés pour SOM et qui donne des résultats assez bon après l'interpolation (figure 5.8).

Pour les deux autres configurations, l'apprentissage permet de mettre en évidence que même avec peu de points dans les zones de faible variation, l'interpolation ne fait que des erreurs minimes et confirme que l'absence de points support autour de la valeur maximale du pic amène à fortement sous-estimer cette dernière (figure 5.9 et 5.10. Une bonne répartition des points semble préférable. Cependant un ensemble de point plus réduit concentré sur les zones avec de forte variation semble pouvoir obtenir de très bon résultats.

Ensuite, les mêmes expériences sont entreprises avec DSOM comme modèle d'apprentissage. Les résultats sont de même nature dans les trois configurations. Cependant, SOM semble obtenir de meilleurs résultats après interpolation. En moyenne, l'erreur de DSOM semble deux fois plus importante que SOM. Pour vérifier cette hypothèse, nous proposons une comparaison avec un nombre plus important d'apprentissage indépendant selon la taille de la grille de neurone.

Pour réaliser cette comparaison, 10 apprentissages sont entrepris pour chaque modèle selon des taille de grille de neurones allant de 2x2 à 14x14. La configuration avec les données d'apprentissage répartis selon la grille régulière est privilégiée pour les apprentissages. Les apprentissages durent chacun 10.000 itérations, à la suite de quoi on applique la méthode d'interpolation de Göppert avec la variante des phi locaux. La comparaison est effectuée selon la moyenne des différences absolues entre les images calculée par interpolation et les vraies images de la gaussienne avec en input une grille uniforme de 40 x 40, la grille allant de -1 à 1 dans les deux dimensions, équation 5.12 avec x_i les points tests i $\in [0, N]$.

$$MoyErr = \frac{1}{N}\sum_{i}^{N} |(gauss(x_i) - Y_{CF-SOM}(x_i))| \tag{5.12}$$

Les résultats sont présentés sous la forme de boite moustache. Une boîte donne les limites du premier et dernier quartile, la barre du milieu donne la médiane, les moustaches vont jusqu'à la valeur la plus extrême dans la limite de 1.5 fois la hauteur de la boîte, et les points au-delà des moustaches sont représentés par des points isolés.

Sur les figures 5.11a et 5.11b, on peut voir l'évolution de l'erreur moyenne selon la taille de la grille après 10000 itérations. On peut constater que SOM à partir d'une taille de grille supérieur à 5x5 (>25 neurones) obtient des résultats avec une erreur deux fois plus faible que pour DSOM. On peut aussi constater que l'erreur moyenne ne varie plus beaucoup avec des grilles supérieur à 7x7, il ne semble donc pas utile d'utiliser de très grandes grilles de neurones pour l'interpolation dans ce problème.

Suite à cette comparaison, nous proposons d'expérimenter des alternatifs aux modèles de DSOM qui semblent surtout avoir du mal à apprendre les données qui sont sur la périphérie de l'ensemble des

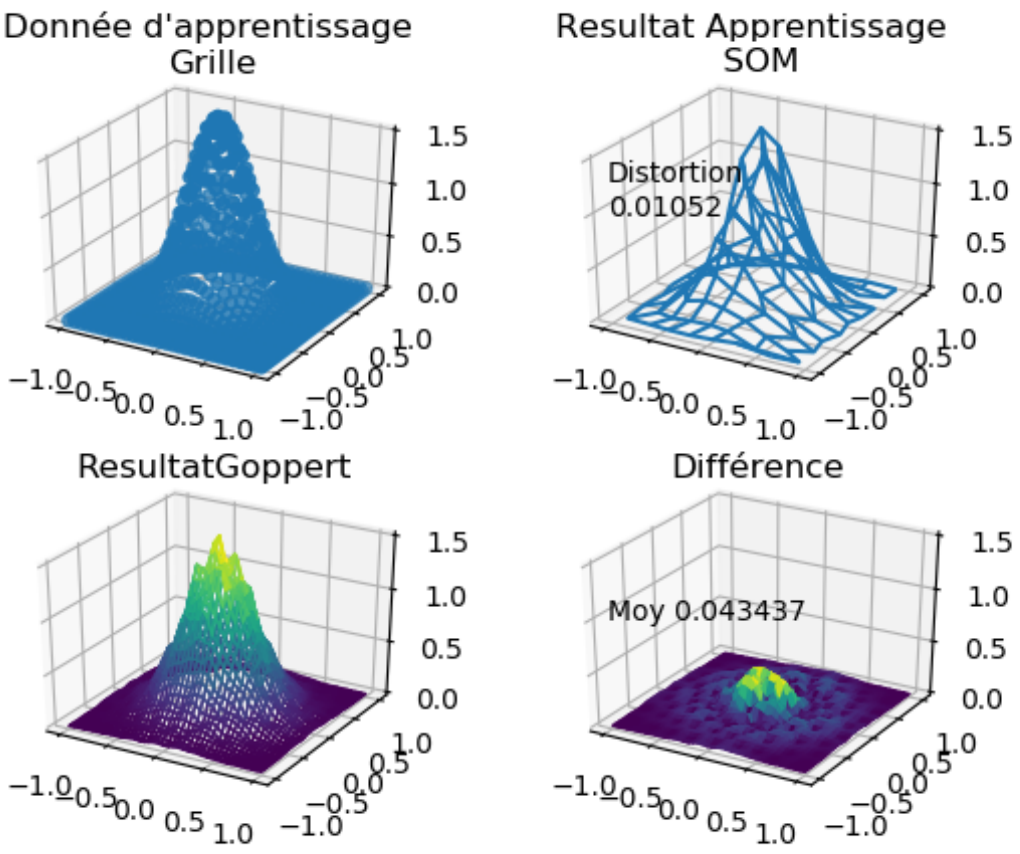


FIGURE 5.8 – Application méthode Göppert avec apprentissage SOM 10x10 selon un ensemble de donnée en grille

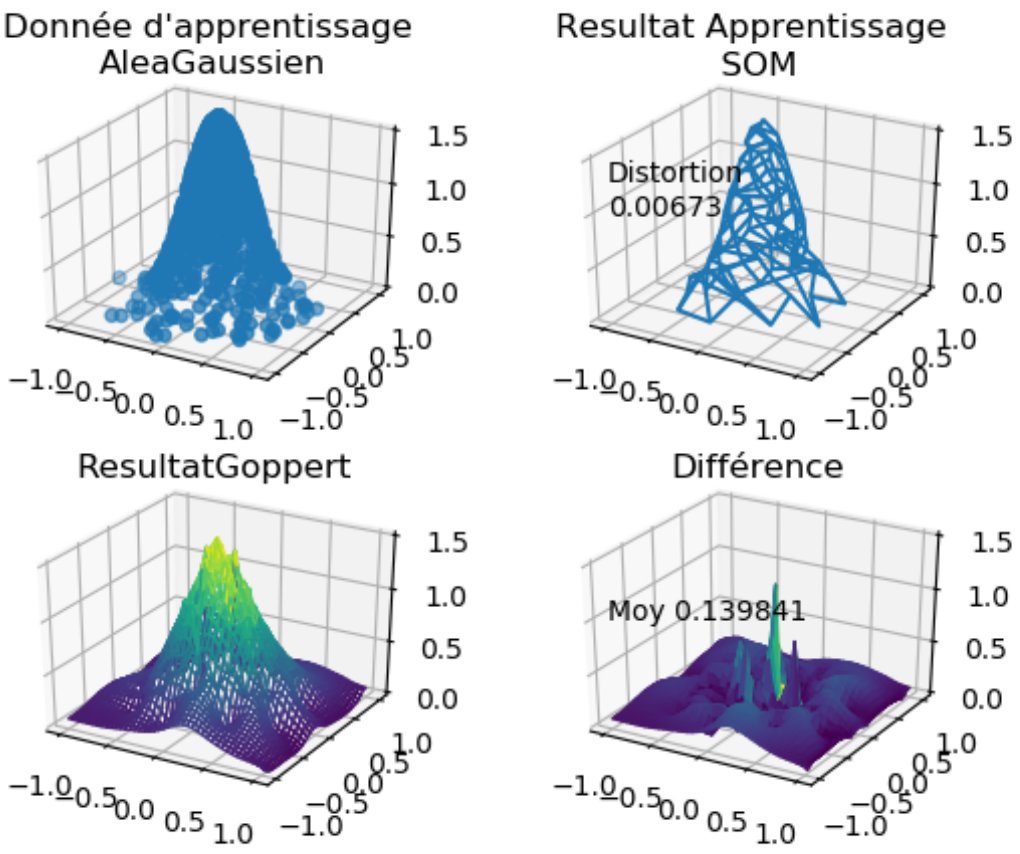


FIGURE 5.9 – Application méthode Göppert avec apprentissage SOM 10x10 selon un ensemble de donnée surtout sur le pic de la Gaussienne

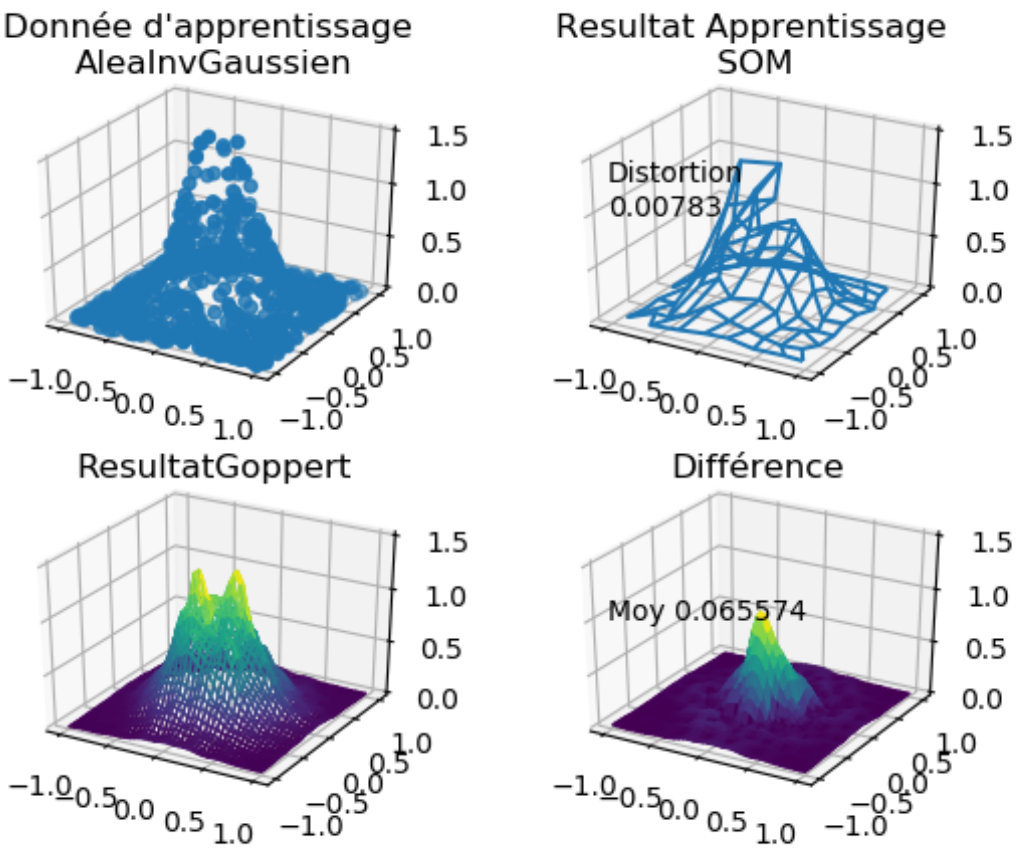
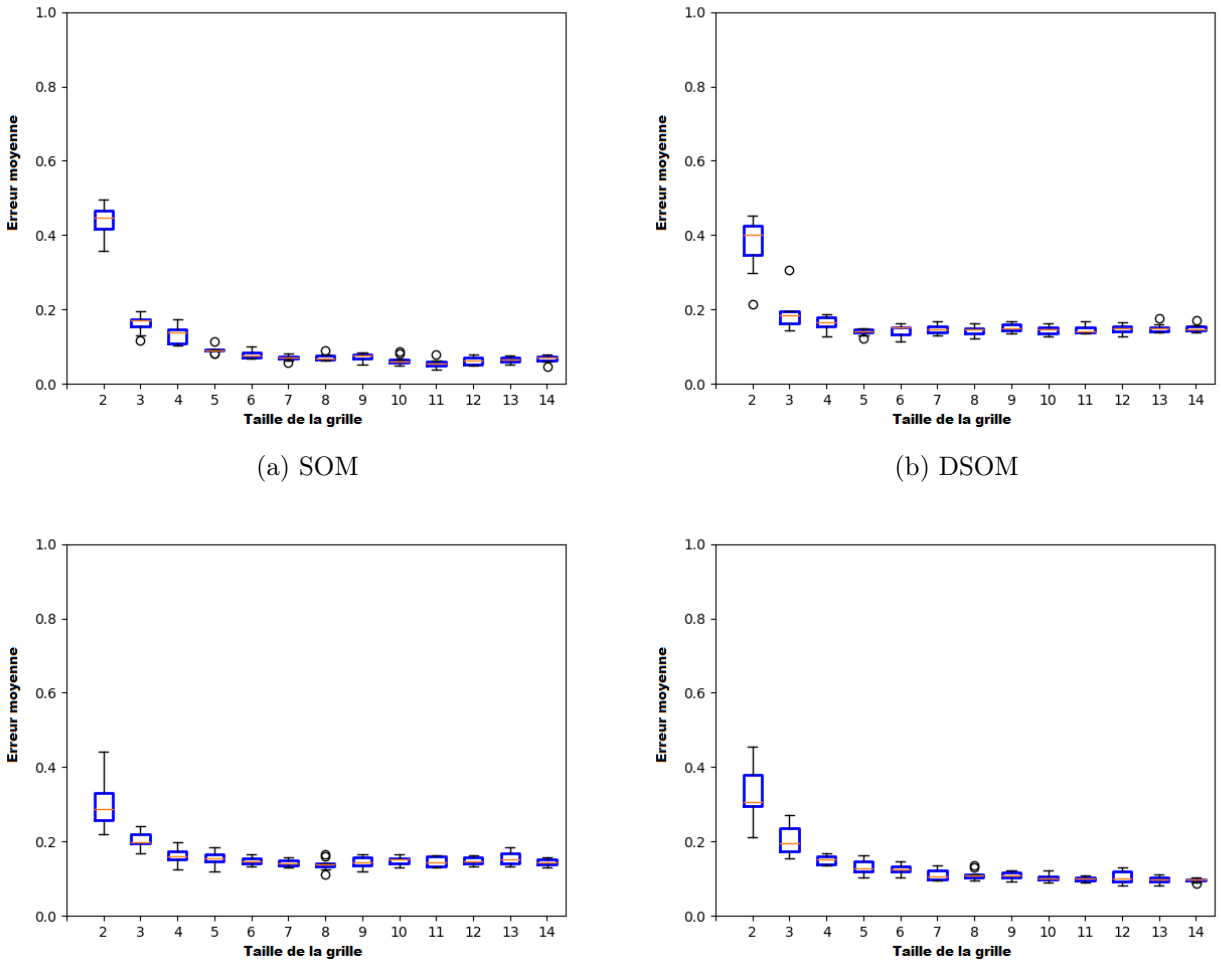


FIGURE 5.10 – Application méthode Göppert avec apprentissage SOM 10x10 selon un ensemble de donnée surtout en périphérie du pic de la Gaussienne



(a) SOM

(b) DSOM

(c) DSOM choix entrée

(d) DSOM Alternatif

FIGURE 5.11 – Comparaison de l'erreur moyenne de 4 modèle d'apprentissage, SOM, DSOM et deux alternative de DSOM. Chaque boite à moustache représente les résultat de 10 apprentissages réalisé de manière indépendantes avec un t_{final} de 10000 itérations, apprentissages effectués selon la configuration en grilles uniforme de taille 30 sur 30.

données. L'hypothèse explorée est qu'avec un meilleur apprentissage des données mal apprises, il serait possible d'améliorer l'apprentissage de la périphérie.

La première alternative consiste à ne présenter durant l'apprentissage que des entrées avec une distance au w_{BMU} supérieure à la distorsion moyenne du modèle. La distorsion correspond à la distance vectorielle entre une entrée et son BMU du modèle, la distorsion moyenne du modèle correspond à la moyenne de toutes les distorsions précédemment présentées aux modèles. L'hypothèse est que l'apprentissage devrait certainement être amélioré en ne présentant que des entrées avec des distorsions importantes. Cependant, suite au même protocole la différence n'est pas probante pour conclure que cette alternative permette un meilleur apprentissage : figure 5.11c.

La seconde alternative est de pondérer l'apprentissage de DSOM selon le rapport distance BMU / moyenne distorsion équation 5.13 avec d_{moy} correspondant à la distorsion moyenne du modèle. Plus le vecteur d'entrée est loin de son BMU plus on apprend et à l'opposée, plus il est proche moins on apprend. Cette alternative permet d'augmenter les performances obtenues sans pour autant réussir à égaler ceux de SOM, figure 5.11d.

$$w_i(t+1) = w_i(t) + \varepsilon \left\| v - w_i(t) \right\|_\Omega h_\eta(i,s,v)(v - w_i(t))(\frac{\|v - w_i(t)\|}{d_{moy}}) \tag{5.13}$$

Chapitre 6

Apprentissage par renforcement

La suite et dernière partie de ce stage est consacrée à l'utilisation des modèles SOM et DSOM comme support pour la fonction politique dans un processus de décision. En plus des modèles, le principe de Q-learning est utilisé pour apprendre la politique. Pour faire nos expérimentations, nous utilisons l'environnement "*Mountain Car*" proposé par *OpenAI Gym* [8].

6.1 Mountain car

L'environnement utilisé pour les expérimentations est Mountain car [9] et l'implémentation utilisée est celle proposée sur OpenAI.com (figure 6.1). L'agent (une voiture) commence en bas d'une vallée en 2D. L'objectif consiste à amener l'agent en haut de la montagne de droite. L'état de l'agent est défini par la position et la vitesse de la voiture. La position varie entre -1.2 à 0.5 et la vitesse entre -0.07 et 0.07. Lors de l'initialisation de l'environnement, la position de l'agent est comprise dans l'intervalle -0.6 et -0.4 avec une probabilité uniforme tandis que la vitesse est toujours nulle. Le but est situé à la position 0.5. Trois actions sont possibles à chaque état : accélérer à gauche, ne rien faire et accélérer à droite. La récompense est de 0 si l'agent parvient à atteindre le but (position > 0.5) sinon elle est de -1. Cependant, une gravité produit une accélération vers le fond de la vallée et empêche l'agent de sortir de la vallée en accélérant uniquement dans la direction du but. La politique optimale consiste en des allers retours entre les deux montagnes de plus en plus haut pour utiliser de l'énergie potentielle pour pouvoir grimper de plus en plus haut sur les montagnes.

6.2 Fondement théorique

Le type de problème étudié correspond à une suite de décisions où chaque décision influe sur le choix des décisions suivantes. On a donc un caractère séquentiel des décisions. Mountain car est un problème déterministe dans un espace continu. Cependant, les algorithmes que nous allons utiliser nécessite de le discrétiser, comme par exemple, avec l'approche tabulaire présentée plus loin. Cette discrétisation apporte de l'incertitude et donc les transitions du modèle doivent être probabilistes

L'une des approches existantes pour ce type de problème la plus classique correspond aux processus décisionnels de Markov (MDP) [10]. La base de cette approche consiste à rassembler les informations de l'agent sous la forme d'état, des actions qui influencent l'état au cours du temps et de récompenses associées aux transitions entre les états ; on utilisera respectivement s pour les états, a pour les actions et r pour les récompenses. Les MDP sont des chaînes de Markov visitant les états, contrôlées par les actions et évaluées par les récompenses. Résoudre un MDP consiste à choisir les actions de manière à maximiser son cumul de récompense. Les solutions aux MDP prennent la forme de politiques ou stratégies π qui sont des règles pour choisir l'action à entreprendre selon l'état de l'agent. Ces politiques sont construites grâce à des *fonctions valeurs*. Les *fonctions valeur* permettent de calculer l'espérance des récompenses. La *fonction valeur* que nous utiliserons associe une valeur à chaque couple [état, action] selon l'équation 6.1, concrètement on cherche à savoir quelle est l'espérance des récompense que l'on peut recevoir si on choisi d'abord l'action a et qu'ensuite on utilise la politique choisie. Et la politique optimale π^*, aussi appelée *greedy*, consiste, dans chaque état s, à choisir l'action qui maximise

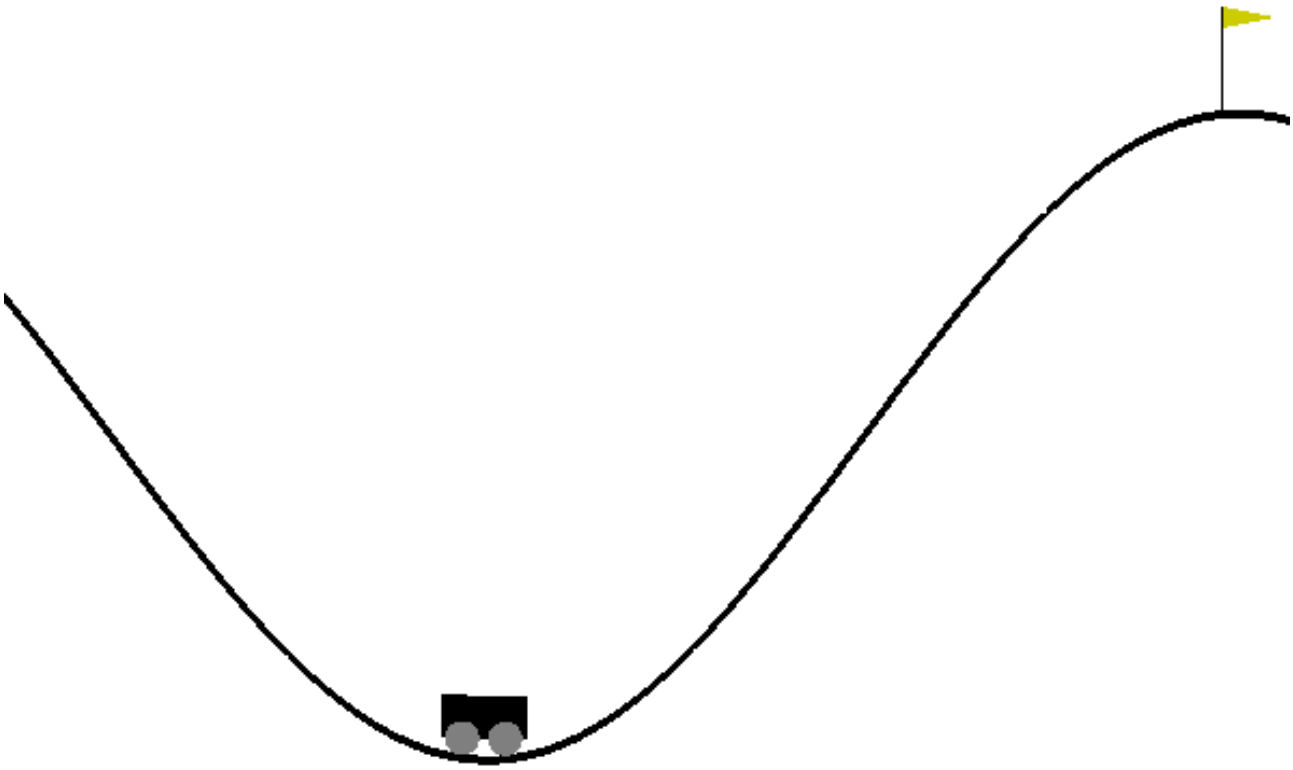


FIGURE 6.1 – Mountain Car exemple d'initialisation

$Q(s, a)$ (eq 6.2). Il ne faut cependant pas oublier qu'une même politique pourra avoir des résultats variés du fait de l'incertitude, dans le cas de Mountain car l'aléatoire de la position initiale de l'agent .

$$Q(s, a) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a] \tag{6.1}$$

$$\pi(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s, a) \tag{6.2}$$

L'apprentissage par renforcement se place dans le cadre de MDP où les transitions et récompenses ne sont pas connues *a priori*. Historiquement les algorithmes d'apprentissage par renforcement reposent sur des concepts issus de la cognition humaine ou animale, comme renforcer l'intérêt d'une action jugée positive ou renforcer l'intérêt d'une action fréquemment testée [11].

Parmi les différents algorithmes d'apprentissage par renforcement, nous utiliserons le Q-learning qui est un algorithme qui dans le cadre des MDP possède des preuves formelles de la convergence vers une fonction valeur optimale et donc une politique optimale [12]. La politique est donc apprise au cours du temps en utilisant une fonction valeur approchée grâce à l'équation 6.3, avec α le taux d'apprentissage (généralement vers 0.1), γ le facteur d'actualisation qui détermine l'importance des récompenses futures dans le choix de l'action (généralement égal à 0.9).

$$Q(s_n, a_n) \leftarrow Q(s_n, a_n) + \alpha[r_n + \gamma \max_a Q(s_{n+1}, a) - Q(s_n, a_n)] \tag{6.3}$$

L'actualisation de $Q(s_n, a_n)$ est calculée après modification de l'état s_n en l'état suivant s_{n+1} suite à l'action choisie par la politique utilisée a_n et grâce à la récompense r_n renvoyée par l'environnement. Cependant, pour pouvoir obtenir la politique optimale, il faut durant la phase d'exploration appliquer une politique qui passera par tous les états. La politique d'exploration la plus naïve consiste à choisir aléatoirement l'action avec une probabilité identique. Mais cette politique a pour inconvénient de demander une période d'exploration longue pour être sûr de bien parcourir tous les états suffisamment.

Une autre politique d'exploration possible est basée sur la méthode d'échantillonnage de Gibbs [13]. Cette politique échantillonne une action selon les probabilités définies par l'équation 6.4 avec v_i les valeurs de la Q-table dans l'état s pour les trois actions, cela aillant pour conséquence d'accélérer l'apprentissage car les actions sont choisies proportionnellement aux Q-valeurs associées, une action avec une Q-valeur proportionnellement plus importante que les autres aura plus de probabilité d'être choisie et menant au but en moins d'itération. La politique principale que nous utiliserons est une combinaison des deux précédentes avec à chaque choix 98% d'utiliser la politique de Gibbs et 2% d'utiliser la politique aléatoire. Dans la suite nous utiliserons π_A pour parler de la politique d'exploration aléatoire et π_{GA} pour parler de l'association de la politique aléatoire et de la politique de Gibbs.

$$P_{v_i} = \frac{e^{v_i}}{\sum_j e^{v_j}} \tag{6.4}$$

6.3 Expérimentation

6.3.1 Application dans un espace d'état discret

Afin d'introduire le problème et d'illustrer les concepts d'apprentissage par renforcement introduits précédemment et de disposer d'une base de comparaison pour les expériences des prochaines sections, on se propose d'appliquer l'algorithme de Q-learning avec une représentation tabulaire de la fonction valeur. L'espace d'état $[-1.2, 0.5] \times [-0.07, 0.07]$ est discrétisé en une grille régulière de $N_{sp} \times N_{sv}$ (ici 40×40). La Q fonction peut alors être tabulée, c'est-à-dire qu'elle se ramène à un ensemble fini de $N_{sp} \times N_{sv} \times N_a$ valeurs.

Le protocole d'apprentissage utilisé consiste en 1000 épisodes de 5000 itérations maximum, si on atteint le but durant un épisode on l'arrête et on passe au suivant, avec comme politique d'exploration π_{GA}, la valeur du taux d'apprentissage α dégressif selon l'équation 6.5 avec $\alpha_{Init} = 0.1$, $\alpha_{Final} = 0.001$ et $i \in [0, N]$ avec N le nombre maximum d'épisode ici 1000. On peut voir le résultat d'un apprentissage sur la figure 6.2.

On peut voir sur cette figure une évaluation de la politique π^* suite à l'apprentissage représenté par la courbe noire, elle aura nécessité 119 itérations pour atteindre le but. L'apprentissage a mis en évidence 3 zones distinctes, une zone où la meilleure action est d'accélérer à droite représenté par des triangles bleus orientés à droite, une zone où la meilleure action est d'accélérer à gauche représenté par des triangles rouges orientés à gauche et une zone qui n'a pas été explorée durant l'apprentissage représenté par des triangles gris, ces états étant probablement inatteignable durant l'apprentissage.

$$\alpha_i = \alpha_{Init} * (\alpha_{Final}/\alpha_{Init})^{(i/N)} \tag{6.5}$$

6.3.2 Application avec des cartes auto-organisatrices

On se propose maintenant d'étudier un algorithme qui repose sur une approximation de la fonction valeur en utilisant des cartes auto organisatrices, SOM et DSOM, avec la possibilité d'utiliser les algorithmes de Göppert introduits dans les chapitres précédents. Ces algorithmes conduisent à un apprentissage de la quantification de l'espace d'état plutôt que sur une discrétisation régulière fixée a priori.

Pour adapter SOM et DSOM à l'algorithme du Q-Learning, la solution utilisée est d'ajouter aux neurones en plus du vecteur référence une Q-table. La Q-table correspond à un ensemble de fonction valeur action qu'on associe à la position du BMU. L'utilisation de Q-tables associée à un BMU est équivalente à réaliser une approximation de la fonction valeur par une fonction constante par morceau.

Pour faciliter la compréhension des différents protocoles d'apprentissage qui vont être essayé, nous vous proposons ce schéma 6.3. On peut y voir en haut à gauche l'environnement et l'agent, dans notre cas Mountain Car. On en récupère l'état de l'agent, la position et la vitesse que l'on normalise entre [0, 1]. Cet état normalisé $\bar{s}$ est ensuite donné comme vecteur d'entrée à la carte auto organisatrice que l'on actualisera ou non (1). On en récupère la Q-table associée au neurone BMU_s (2) de l'état donné à la carte auto organisatrice. Cette Q-table fournit les fonctions valeur utiles (une pour chaque

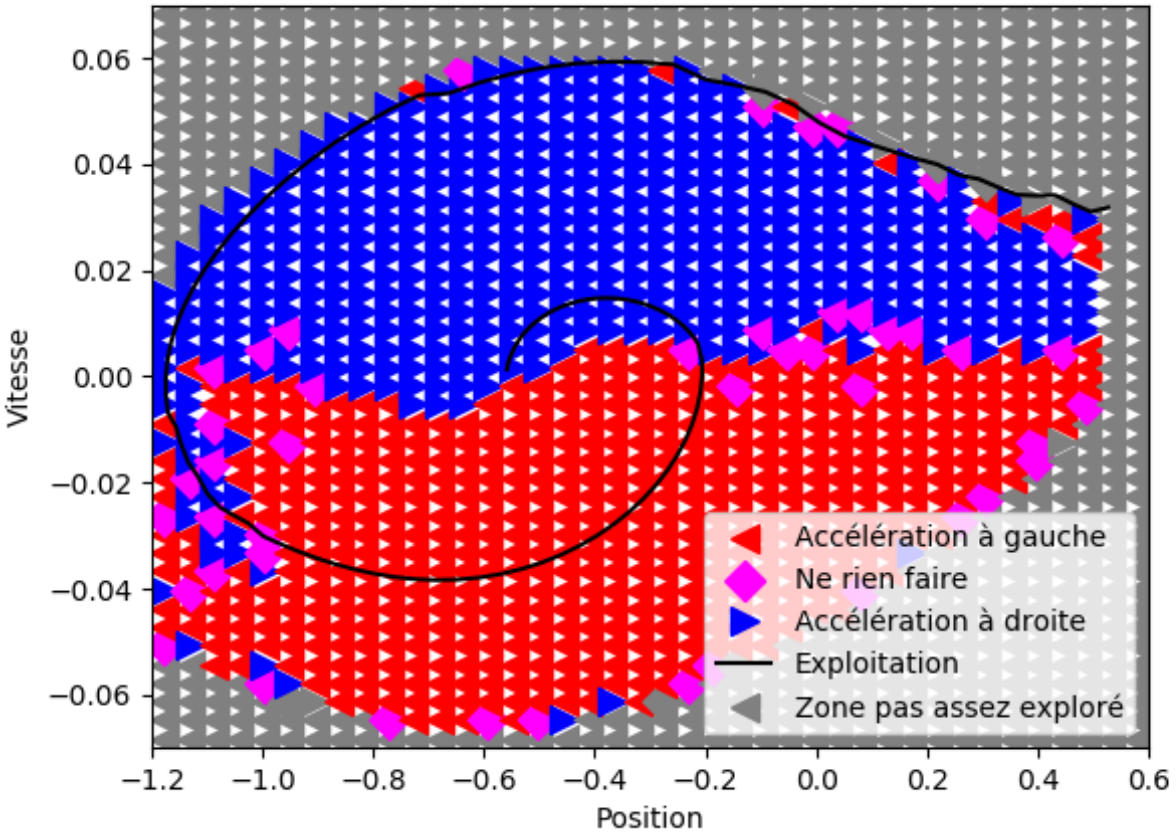


FIGURE 6.2 – Représentation de la politique optimale obtenu suite à un apprentissage par renforcement avec l'algorithme de Q-learning et une représentation tabulaire de la fonction valeur

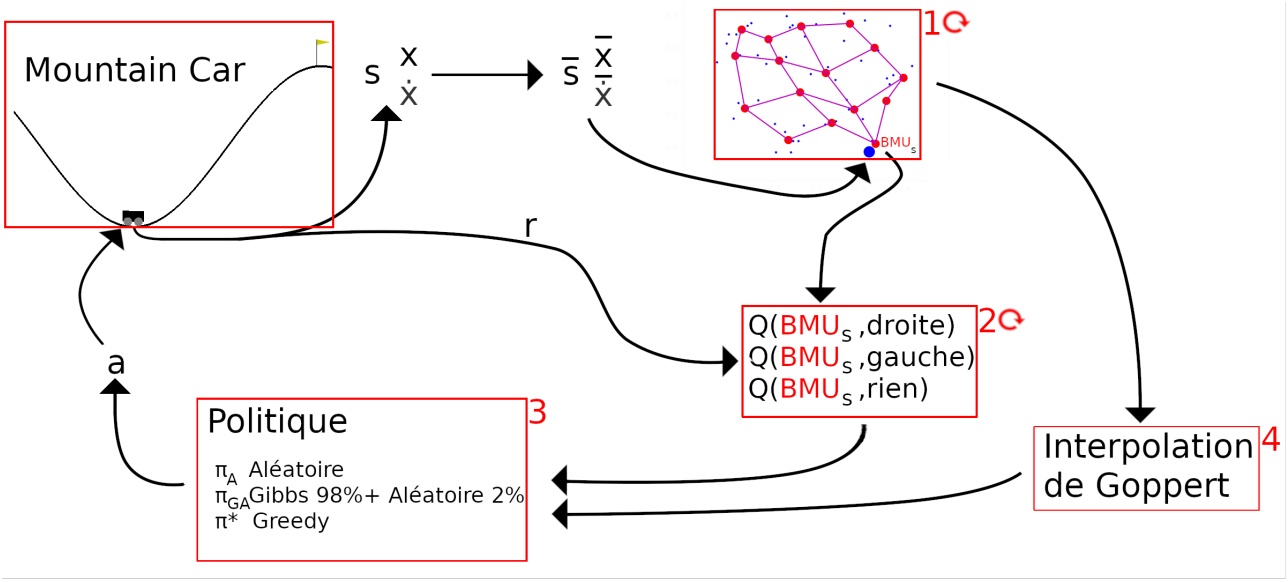
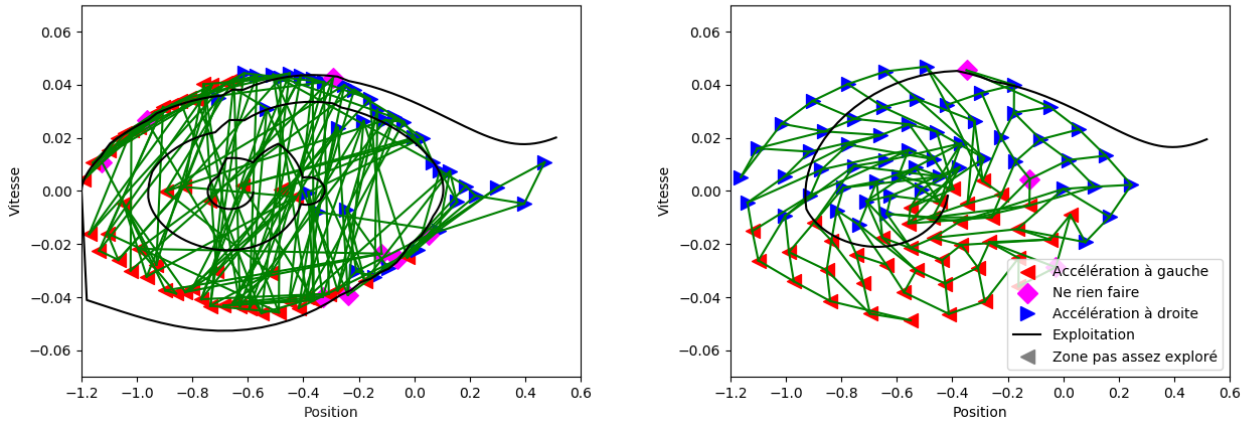


FIGURE 6.3 – Schéma des différentes étapes possible durant une itération



(a) SOM non normalisé

(b) SOM normalisé

FIGURE 6.4 – Politique π^* avec un exemple d'évaluation suite à un apprentissage de SOM selon des entrées non normalisées et normalisées.

action possible) pour y appliquer les politiques (3), durant l'exploration les politiques sont π_A et π_{GA}. Ensuite, on applique l'action choisie à l'environnement qui renvoie la récompense permettant d'actualiser la fonction valeur de l'action choisie dans la Q-table du neurone BMU_s grâce à l'équation du Q learning (eq 6.3), avec s_n le BMU_s, a_n l'action choisie et r_n la récompense, s_{n+1} étant le BMU de l'état obtenu. Durant l'évaluation plutôt que d'utiliser la Q-table du BMU (2), il est possible d'interpoler les fonctions de valeur de chaque action grâce à la totalité de la carte et des algorithmes de Göppert (4). La politique utilise les fonctions valeur interpolées pour définir l'action choisie.

Dans un premier temps, différents protocoles d'apprentissage seront expérimentés de manière empirique sur SOM, ensuite sur DSOM et pour finir une comparaison entre SOM et DSOM statistiquement représentative avec un protocole commun.

Application avec SOM

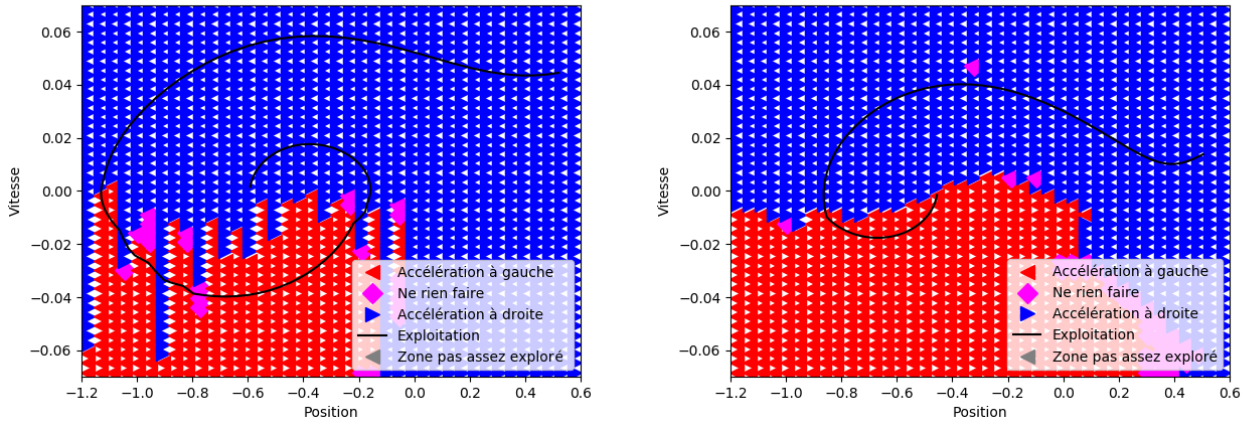
Le premier protocole qui est expérimenté consiste à réaliser l'apprentissage en deux phases avec comme répartition 1/3 des épisodes pour la première phase et les 2/3 restant pour la seconde avec 1000 épisodes de 5000 itérations maximum.

La première phase est consacrée à apprendre uniquement la carte SOM, la position des neurones dans l'espace d'entrée, l'agent étant contrôlé par la politique d'exploration aléatoire π_A. Suite à cette phase on ne modifiera plus les vecteurs références des neurones. Cependant, les Q-tables n'ont pas encore été apprises. La seconde phase consiste donc à apprendre les Q-tables de chaque neurone. On change la politique d'exploration pour π_{GA}. Si on reprend le schéma précédent on actualisait dans un premier temps que la partie 1 et dans un second que la partie 2. Durant l'exploitation, on utilise la Q-table du neurone le plus proche de l'observation et la politique π^*.

Dans une première version, les vecteurs d'entrées et l'espace de la carte auto organisatrice ne sont pas normalisés conduisant à des cartes avec très peu de neurones aux centres et beaucoup sur les observations extrêmes (fig 6.4a). Ce comportement semble être dû à une différence d'échelle trop importante entre les dimensions (amplitude de 1.8 pour la position, 0.14 pour la vitesse). Pour corriger ce problème, les différentes dimensions sont pour la suite normalisées entre 0 et 1 lors de l'apprentissage des cartes auto organisatrices ce qui permet d'éviter ces problèmes d'échelles (fig 6.4b).

Une fois l'apprentissage réalisé, il est possible d'appliquer l'interpolation Göppert (4) en utilisant toute la carte durant l'évaluation plutôt que d'utiliser uniquement la Q-table du BMU. Pour choisir l'action à effectuer, on interpole les fonctions valeurs de chacune des actions avec l'état actuel comme vecteur d'entrée et la carte apprise comme ensemble de points supports. La politique utilisée est la politique *greedy* π^*.

Comme pour l'apprentissage de la carte, dans une première version, les vecteurs d'entrée ainsi que



(a) Avec entrée non normalisée

(b) Avec entrée normalisée

FIGURE 6.5 – Politique π^* avec un exemple d'évaluation suite à une interpolation de Göppert selon des entrées non normalisées et normalisées.

les points supports ne sont pas normalisés dans $[0, 1]$, mais sont fournis dans l'espace d'état de Mountain car $[-1.2, 0.5] \times [-0.07, 0.07]$. On peut voir des lignes verticales se former lors de l'interpolation et nous pensons que ces effets sont causés par la différence d'échelle. Nous choisissons donc d'utiliser pour calculer la fonction valeur interpolée, d'utiliser l'état et les points supports normalisé exactement comme il l'est pendant l'apprentissage de la carte. On peut voir sur les résultats de l'interpolation de Göppert selon la carte de la figure 6.4b avec les données non normalisées et normalisées respectivement sur les figures 6.5a et 6.5b. Les résultats obtenus grâce à l'interpolation semblent similaires à l'utilisation de la Q-table du BMU. Cependant, le temps de calcul est beaucoup plus long, l'utilisation de la Q-table du BMU est donc privilégié.

Ce premier protocole qui apprend d'abord la position des neurones puis les Q-tables a comme avantage de ne pas voir les fonctions valeurs bouger dans l'espace de Mountain car. Cependant, cela oblige l'utilisation dans un premier temps de la politique π_A car aucune fonction valeur ne peut être exploitée avant la seconde phase.

Le second protocole d'apprentissage consiste à apprendre la carte SOM et les Q-tables de manière simultanée. A chaque itération la carte et la fonction valeur de l'action précédemment choisie est actualisées. L'agent est contrôlé par la politique d'exploration π_{GA}. L'apprentissage durant 1000 épisodes de 5000 itérations maximum. Durant l'exploitation, on utilise la Q-table du BMU et applique la politique π^*.

L'apprentissage simultané de la carte SOM et des Q-tables semble avoir des résultats similaires à l'apprentissage successif présenté précédemment.

Nous choisissons d'expérimenter un dernier protocole pour s'assurer que l'apprentissage de Q-table temporairement stable dans l'espace ne donnerais pas de meilleures politiques.

Le troisième protocole consiste à alterner l'apprentissage de SOM et celui des Q-tables, avec une répartition de 40 épisodes consacré à l'apprentissage de la carte SOM et ensuite 60 épisodes pour les Q-tables. Avec la politique d'exploration qui, elle aussi changeait. Durant les 40 épisodes de l'apprentissage de la carte SOM la politique utilisée est π_A. Tandis que lors des 60 épisodes de l'apprentissage des Q-tables la politique utilisée est π_{GA} Ce protocole ne semble pas donner de meilleure politique, il ne sera pas réutilisé.

Un dernier protocole est expérimenté avec un α fixé à 0.1 pour essayer d'enlever l'hyperparamètre α qui est décroissant dans les autres protocoles. Cependant, les résultats obtenus sont médiocres car la politique ainsi apprise renvoi toujours la même action : accélérer à gauche.

Les résultats obtenus selon les trois protocoles avec α dégressif semblent obtenir des résultats finaux équivalents. Cependant, le second protocole possède contrairement aux autres la possibilité de pouvoir

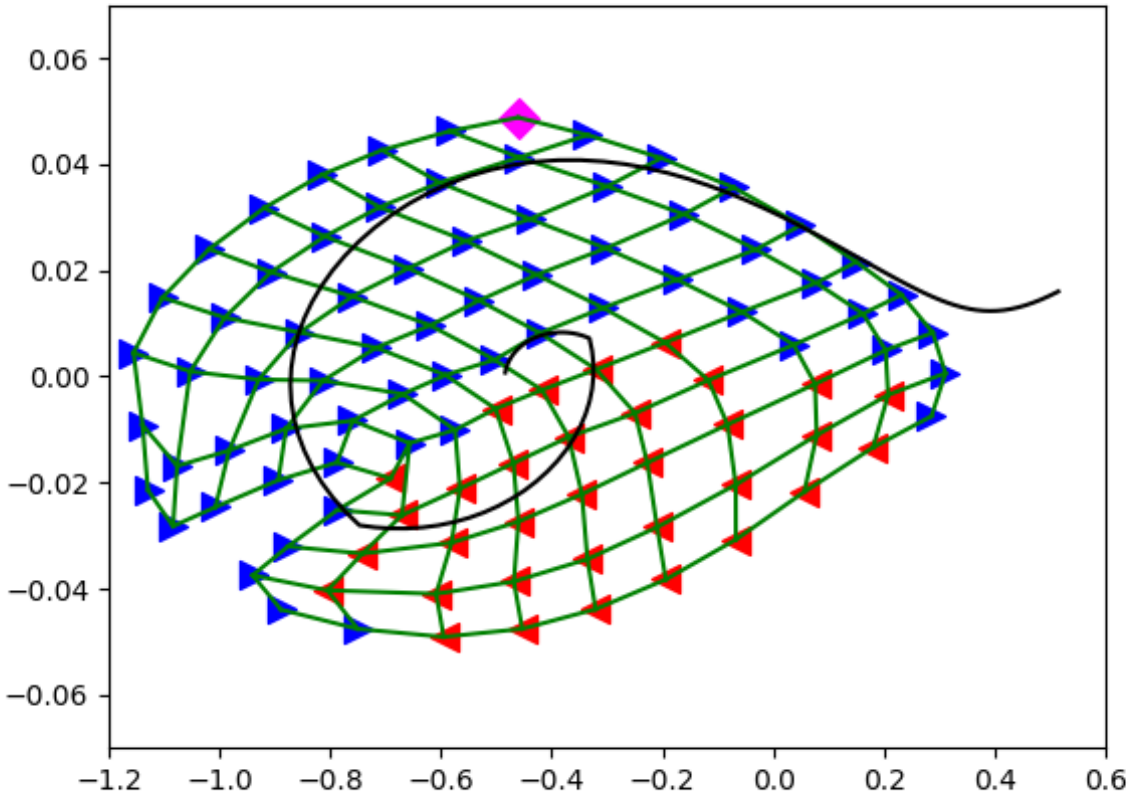


FIGURE 6.6 – Résultat de l'apprentissage par renforcement avec DSOM.

apprendre simultanément la carte SOM et les Q-tables, ce qui le rend plus simple à utiliser.

Un comportement est constatable au centre de la carte lors de tous les apprentissages de ces protocoles, la carte qui est pourtant bien dépliée semble se chiffonner au centre. Probablement dû aux nombres importants d'entrée provenant du fond de la cuvette.

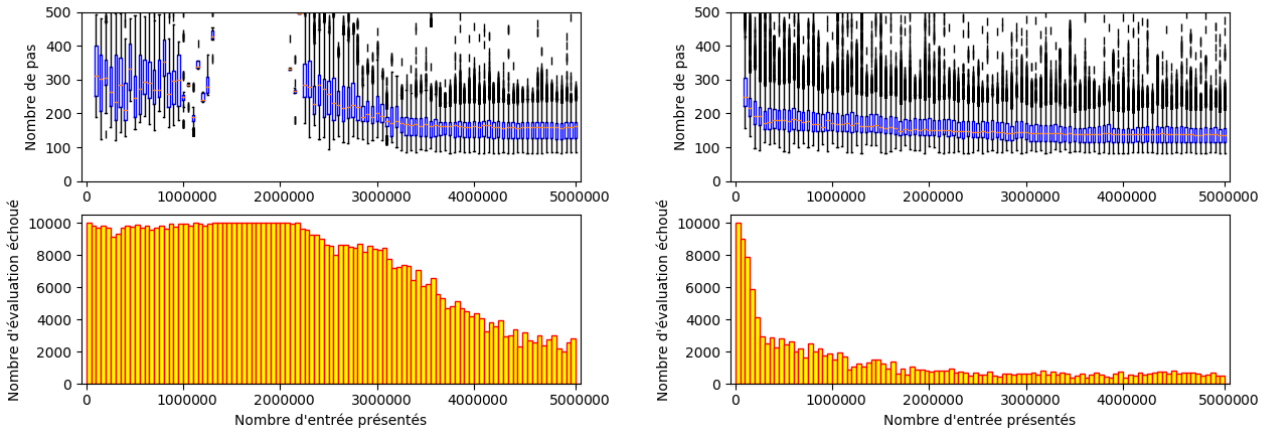
Suite à ces expérimentations sur SOM, il est décidé d'expérimenter avec DSOM pour observer d'éventuelles différences.

Application avec DSOM

Nous commençons par appliquer le même premier protocole que SOM : un apprentissage séparé en deux phases, une consacrée à l'apprentissage de la carte (1/3 des épisodes) et l'autre pour les Q-tables (2/3 des épisodes), avec 1000 épisodes de 5000 itérations maximum. Pour accélérer le temps de calcul, j'ai choisi de calculer le facteur $max_{y,z\in\Omega}\|y-z\|$ utilisé pour l'actualisation de la carte de DSOM uniquement à chaque début d'épisode et avec une limite arbitraire de 500000 données représentant un minimum de 100 épisodes complets. Les politiques d'explorations sont pour la première phase π_A et π_{GA} pour la seconde phase.

Les politiques apprises semblent équivalentes voir meilleures qu'avec SOM. Et on peut observer que les cartes DSOM apprises semblent mieux dépliées. Cette hypothèse est explorée dans la sous section 6.3.3.

Pour vérifier, si comme pour SOM apprendre la carte des neurones de DSOM et les Q-tables de manière simultanée aboutit à un résultat similaire au protocole précédent. Le même second protocole d'apprentissage est expérimenté. A chaque itération on actualise la carte des neurones et la fonction valeur correspondant à l'action choisi à l'itération précédente. La politique d'exploration appliqué est π_{GA}. Et, en effet, les résultats obtenus semble similaires. On peut voir sur la figure 6.6 le résultat d'un



(a) Résultat SOM

(b) Résultat DSOM

FIGURE 6.7 – Représentation des résultats des évaluations effectué suite à l'apprentissage de 100 modèles de SOM et DSOM en fonction du nombre d'itérations de l'apprentissage. Figure supérieur représentant la répartition du nombre de pas pour atteindre le but des évaluations réussite (<500 pas). Figure inférieur nombre d'évaluation qui ont échoué.

apprentissage avec ce second protocole.

Le comportement de la carte observé à la fin des apprentissages au centre de la carte SOM ne fut pas constaté lors de l'application avec DSOM.

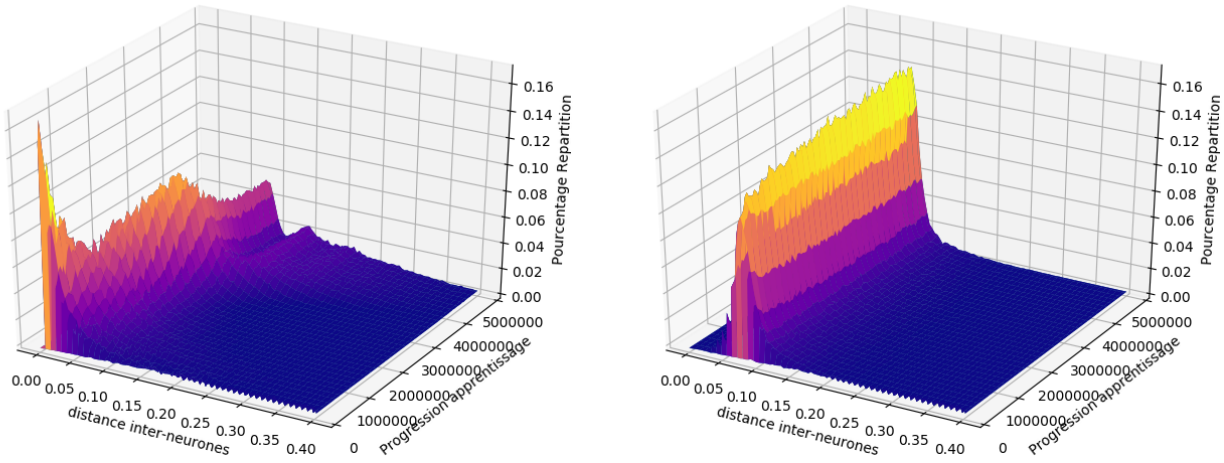
6.3.3 Comparaison des apprentissages SOM et DSOM

Une différence de performance semble exister entre SOM et DSOM dans cet environnement. Une comparaison statistiquement significative pour mettre en évidence ou réfuter cette hypothétique différence semble importante. Pour la réaliser, le protocole d'apprentissage que nous choisissons est celui apprenant simultanément la carte des neurones et les Q-tables. En effet, les protocoles semblant avoir des résultats équivalents et étant le protocole le plus simple avec le moins de contrainte, c'est le choix qui semble le plus logique. Cependant, pour être sûr que les apprentissages soient les plus semblables possible, le critère d'arrêt ne dépend plus du nombre d'épisode mais du nombre d'itérations total.

Pour cette comparaison, 100 cartes de neurones de chaque modèle sont appris avec comme temps limite d'apprentissage 5.000.000 itérations. L'apprentissage est constitué d'une succession d'épisodes durant au maximum 5000 itérations chacun. Une fois le nombre maximum d'itérations atteint, on stoppe immédiatement l'apprentissage sans attendre la fin de l'épisode en cours. Pour pouvoir suivre l'évolution des apprentissages au cours du temps, les modèles sont sauvegardés toutes les 50.000 itérations pour un total de 101 étapes sauvegardées. Le critère utilisé pour la comparaison est le nombre d'itérations nécessaire lors de l'évaluation pour atteindre le but. La position de l'agent étant initialisée aléatoirement entre -0.6 et -0.4, 100 évaluations sont effectuées pour chaque sauvegarde avec une limite de 500 actions maximum. Au-delà de 500 actions lors de l'évaluation on considère que l'agent a échoué.

Les résultats obtenus sont présentés par les figures 6.7a et 6.7b. On peut y voir les résultats présentés de chaque modèle selon deux figures. La première figure montre, grâce à des boites à moustaches, la répartition du nombre d'actions effectuées pour résoudre le problème en ne prenant en compte que les évaluations qui ont réussi, avec donc un nombre d'action inférieur à 500. La seconde figure montre le nombre d'évaluation qui ont échoué en fonction du nombre d'itérations de l'apprentissage.

La première chose marquante que l'on peut constater avec ces figures, c'est la différence du nombre d'évaluations qui échouent au cours du temps. Pour SOM plus de 3.500.000 itérations sont nécessaires pour commencer à avoir plus de 50% des évaluations qui réussissent. Alors que pour DSOM 300.000 itérations suffisent. Cette différence peut s'expliquer par la différence d'actualisation des cartes aux



(a) Histogramme de SOM

(b) Histogramme de DSOM

FIGURE 6.8 – Histogramme des distances inter neurones voisin des grilles des modèles en fonction du temps

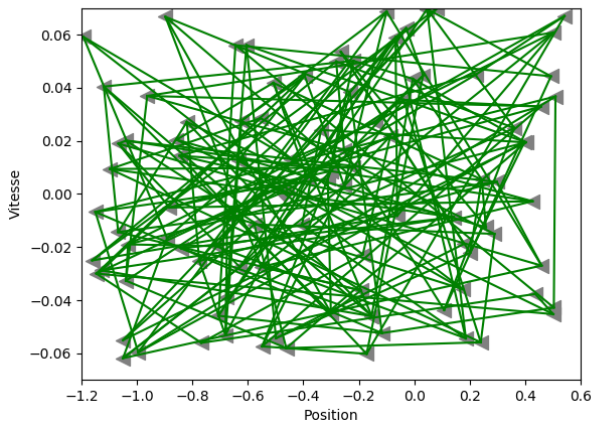
cours du temps, l'apprentissage des cartes selon SOM dépend du nombre d'itération maximum lors de l'actualisation des vecteurs références. Alors que DSOM ne possède pas cette contrainte. Malgré cette différence d'algorithme on peut en comparant uniquement la dernière sauvegarde à 5.000.000 itérations constater que DSOM possède presque 6 fois moins d'évaluations qui ont échoué que pour SOM. De plus, DSOM possède une répartition du nombre d'actions pour réussir une évaluation qui est inférieure à celle de SOM. DSOM semble être plus approprié que SOM dans le cadre de ce problème. Il possède de meilleures performances à nombre d'itération équivalente que ce sois durant ou à la fin de l'apprentissage.

Il faudrait tout de même comparer SOM et DSOM avec des temps de fin variables pour vraiment pouvoir comparer les performances finales selon le nombre d'itérations.

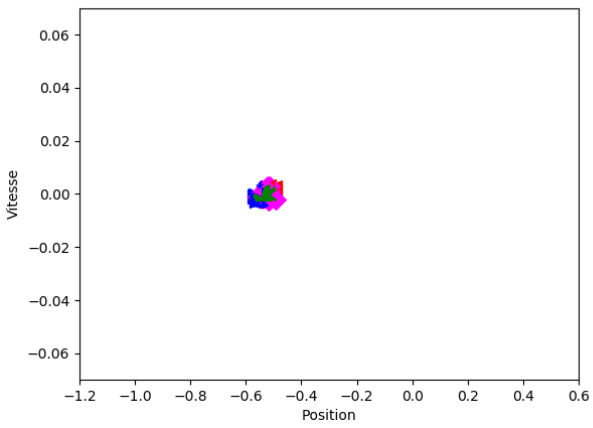
Pour essayer de visualiser le phénomène de chiffonnement qui se produit à la fin des apprentissages de SOM et la différence de performance au cours du temps, nous choisissons de représenter la variation des distances entre les neurones voisins au cours de l'apprentissage. On peut voir cette variation sur les figures 6.8a et 6.8b qui sont présentées sous la forme d'histogramme 3D avec comme axes : la distance inter-neurones dans l'espace normalisé $[0, 1]$ discrétisée en 100 sections dans le sous espace $[0, 0.4]$ pour plus de lisibilité et à cause de l'absence hors de l'initialisation de distances inter-neurones supérieures à 0.4, la progression de l'apprentissage et le pourcentage de la répartition. On peut constater que DSOM garde des distances similaires durant tout l'apprentissage entre chacun des neurones. Alors qu'on observe qualitativement plusieurs phases dans l'apprentissage utilisant SOM : une première où tous les neurones sont regroupés durant les premiers 500.000 itérations de l'apprentissage, une seconde jusque 3.000.000 itérations où la carte se déplie et une dernière où la carte semble s'enrouler au centre de la carte. Les figures 6.9 représentent des exemples de ces différentes phases.

Pour observer la différence de résultat lors de l'évaluation entre l'utilisation des fonctions valeurs de la Q-table du BMU ou de l'interpolation de Göppert utilisant toute la carte apprise. Nous proposons une comparaison entre les résultats d'évaluation obtenues avec les Q-tables précédemment utilisé pour comparer SOM et DSOM, et de nouveaux résultats d'évaluation utilisant exactement les mêmes réseaux de neurones entraîné, mais obtenu grâce à l'interpolation de Göppert. Le temps de calcul d'une évaluation avec l'interpolation étant significativement plus longue la comparaison sera effectuée uniquement sur la dernière sauvegarde à la fin des apprentissages, 100 évaluations sont effectuées par sauvegarde pour un total de 10.000 évaluations pour chaque, ils sont représentés de la même manière figure 6.10.

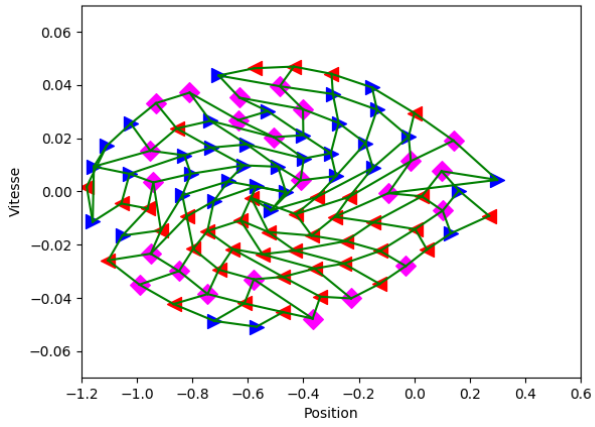
La première figure montre, grâce à des boites à moustaches, la répartition du nombre d'actions effectuées pour résoudre le problème en ne prenant en compte que les évaluations qui ont réussi, avec donc un nombre d'action inférieur à 500. La seconde figure montre le nombre d'évaluation qui ont



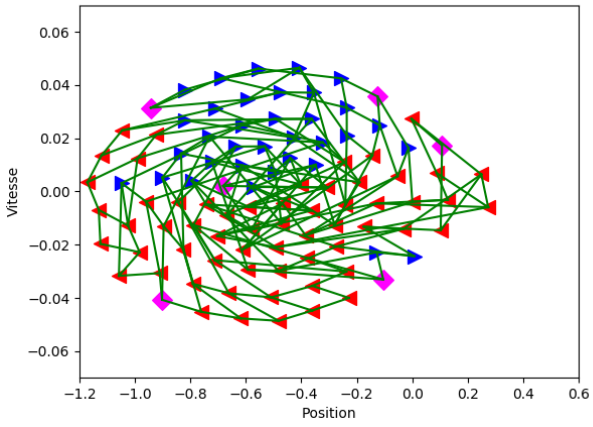
(a) 0 Itération



(b) 100.000 Itération



(c) 3.000.000 Itération



(d) 5.000.000 Itération

FIGURE 6.9 – Différentes phases de l'apprentissage SOM en fonction du nombre d'itération

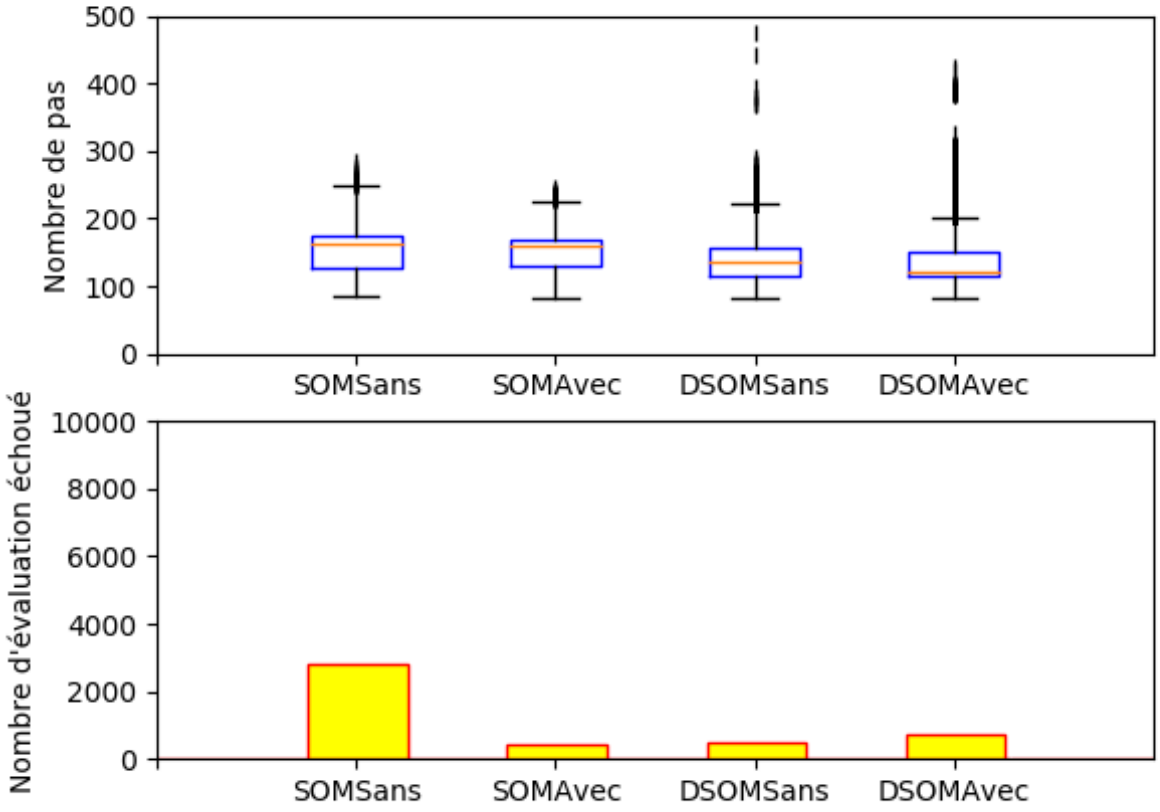


FIGURE 6.10 – Représentation des résultats des évaluations effectué suite à l'apprentissage de 100 modèles de SOM et DSOM, avec ou sans l'utilisation de l'interpolation de Göppert. Figure supérieur représentant la répartition du nombre de pas pour atteindre le but des évaluations réussite (<500 pas). Figure inférieur nombre d'évaluation qui ont échoué.

échoué en fonction du nombre d'itérations de l'apprentissage.

On peut observer que l'utilisation de l'interpolation avec SOM réduit très fortement le nombre d'échec, presque 7 fois moins, mais garde une répartition semblable. Tandis qu'avec DSOM l'interpolation obtient un nombre d'échec plus grand, x1.5 plus d'échec, mais la répartition est bien meilleure, la médiane est presque 20 pas inférieurs. L'hypothèse pour expliquer ces chiffres pour DSOM la plus probable selon moi étant que les évaluations qui ont échoué avec Goppert sont des évaluations qui nécessitent un nombre de pas important sans interpolation.

Il semble donc que l'utilisation de Göppert soit très utile suite à l'apprentissage de SOM. Mais pour DSOM cela semble aux mieux équivalent voir négatif dans ce problème.

Chapitre 7

Conclusion

Pour conclure, j'ai effectué mon stage de fin d'études de master 2 en tant que stagiaire dans l'équipe biscuit du Loria. Durant ce stage de 5 mois, j'ai pu approcher le travail de chercheur avec une reproduction des résultats d'un article et la recherche de réponses grâce à des expérimentations diverses.

Les méthodes d'interpolation de Göppert ont pu être reproduites et les résultats obtenus sont satisfaisants. L'apprentissage des points supports grâce aux modèles SOM et DSOM est possible et on a mis en évidence une supériorité de SOM significative sur le problème d'interpolation d'une gaussienne 3D. Il faudrait cependant réaliser plus de comparaisons pour pouvoir valider cette tendance. Ensuite, nous avons pu associer de l'apprentissage par renforcement en utilisant l'algorithme du Q-learning avec des cartes auto-organisatrices dans le but d'apprendre des fonctions de valeurs de manière adaptative sur un espace d'état continu. Pour expérimenter cette association, l'environnement de Mountain car fut privilégié. Suite à une comparaison des modèles SOM et DSOM, nous avons pu mettre en évidence une différence significative entre les résultats des algorithmes d'apprentissages des cartes de neurones dans la résolution de ce problème. DSOM obtient de meilleurs résultats au cours et à la fin de la comparaison que nous avons mené et pourrais être arrêter en cours d'apprentissage tout en étant utilisable. SOM est bien moins avantageux mais avec une interpolation les différences de résultat son minime.

Une poursuite des recherches sur ces associations semble néanmoins nécessaire pour confirmer ces résultats. Pour cela l'application à d'autre environnement avec plus de dimension me semble importante. De plus, l'interpolation de Göppert donnant de bon résultat après un apprentissage classique de SOM. Il pourrait être pertinent d'essayer d'en utiliser le principe durant l'apprentissage pour ne plus se baser sur le BMU et la méthode du plus proche voisin mais d'interpoler les fonctions de valeur à chaque consultation de la carte et de modifier le vecteur références des neurones selon leurs importances dans le calcul de l'interpolation.

Bibliographie

[1] Nicolas Rougier and Yann Boniface. Dynamic self-organising map. *Neurocomputing*, 74(11) :1840–1847, 2011.

[2] Olivier Sigaud and Olivier Buffet. Processus décisionnels de markov en intelligence artificielle, 2008.

[3] Thomas Martinetz, Klaus Schulten, et al. A "neural-gas" network learns topologies. 1991.

[4] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in neural information processing systems*, pages 625–632, 1995.

[5] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1) :59–69, 1982.

[6] Josef Göppert and W Rosentiel. The continuous interpolating self-organizing map. *Neural Processing Letters*, 5(3) :185–192, 1997.

[7] Josef Göppert and Wolfgang Rosenstiel. Topological interpolation in SOM by affine transformations. In *In Proceedings of the European Symposium on Artificial Neural Networks (ESANN'95*. Citeseer, 1995.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. https ://github.com/openai/gym, 2016.

[9] Andrew William Moore. Efficient memory-based learning for robot control. 1990.

[10] R. Bellman. *Dynamic programming.* Princeton University Press, Princeton, New-Jersey., 1957.

[11] Olivier Sigaud. Comportements adaptatifs pour des agents dans des environnements informatiques complexes. *Mémoire d'Habilitation à Diriger des Recherches de l'Université PARIS VI*, 2004.

[12] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[13] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6) :721–741, 1984.