“Réservoir computing” et Apprentissage par Renforcement Développemental

Alain Dutech

LORIA / INRIA
Campus Scientifique, BP 239, 54506 Vandoeuvre les Nancy
alain.dutech@loria.fr

Résumé : Dans cet article, nous présentons une architecture d'apprentissage par renforcement originale s'appuyant sur une carte neuronale auto-organisatrice dynamique pour intégrer une composante développementale à l'apprentissage. En suivant le schéma du “reservoir computing”, la carte neuronale permet d'apprendre une approximation de la fonction de valeur dans un cadre où l'espace *état* $\times$ *action* est d'une taille conséquente. Pour appréhender la complexité inhérente à la taille du problème, nous proposons une approche développementale dans laquelle nous faisons augmenter la richesse et la complexité des espaces moteurs et perceptifs au fur et à mesure que les performances de l'agent apprenant s'accroissent. Nous détaillons les apports de cette proposition ainsi que les questions qu'elle soulève et explicitons notre architecture en la testant sur une tâche de robotique relativement simple.

Mots-clés : “Reservoir Computing”, Apprentissage par Renforcement, Robotique Développementale, Dilemme exploration/exploitation, Problème de grande taille

1 Introduction

Le cadre de l'apprentissage par renforcement (Sutton & Barto, 1998) est particulièrement séduisant du fait qu'il permet à un agent artificiel d'apprendre un plan d'action pour résoudre un problème qui peut être mal connu ou incertain et ce en n'utilisant qu'un signal scalaire pour distinguer certaines bonnes ou mauvaises situations. L'apprentissage n'est pas supervisé et il n'est pas nécessaire qu'un expert ou un oracle soit disponible. De plus, le formalisme des processus décisionnels de Markov (Puterman, 1994) permet d'appréhender les aspects théoriques de nombreux algorithmes, garantissant, par exemple, l'existence d'une solution au problème d'apprentissage posé et la convergence vers cette solution optimale.

Bien que pouvant s'enorgueillir de succès applicatifs certains (backgammon, contrôle d'une flottille d'ascenseurs, pilotage d'un hélicoptère, *etc*.), l'utilisation pratique des outils de l'apprentissage par renforcement pour des problèmes réalistes n'est pas chose aisée. Les difficultés viennent généralement de la taille des problèmes à résoudre qui ne peuvent être décrits qu'à travers un très grand nombre d'états, ce qui rend alors computationnellement irréaliste l'utilisation des algorithmes d'apprentissage par renforcement recherchant une solution exacte. Des algorithmes cherchant des solutions approchées existent (Itération sur les valeurs avec approximation, LSPI, recherche directe de politique par montée de gradient, *etc*, voir les chapitres 1 et 3 du tome 2 du livre commun du Groupe PDMIA (2008)), mais ils sont loin d'apporter des réponses à toutes les difficultés rencontrées en pratique.

Nous nous intéressons particulièrement à l'utilisation de l'apprentissage par renforcement dans le cadre de la robotique autonome. Plusieurs difficultés y réduisent l'applicabilité des algorithmes d'apprentissage. Les données issues de capteurs sont à valeur continues, souvent bruitées, coûteuses à obtenir (en temps et en énergie) et très (trop) riches en information (par exemple dans le cas d'un flux vidéo). Il est certes possible et classique de prétraiter ces données mais deux situations peuvent se présenter. D'un côté, le pré-traitement peut être trop spécifique, offrant alors une représentation de l'espace perceptif d'une taille certes limitée mais trop spécifique et par trop *ad hoc* : le problème à apprendre y est trivial. D'un autre côté, une représentation plus générale court le risque d'être d'une taille déraisonnable, rendant la tâche d'apprentissage trop difficile. Le compromis entre ces deux extrêmes est difficile à trouver et ces pré-traitements limitent de fait l'autonomie du robot qui reste très dépendant du concepteur du robot.

Dans cet article, nous proposons une méthode d'apprentissage par renforcement qui tire parti des avantages liés aux deux cas que nous venons d'évoquer : apprentissage facilité car s'appuyant sur un espace perceptif de taille

réduite tout en ayant la possibilité de s'appuyer, à terme, sur un espace perceptif aussi complexe que nécessaire et ne limitant donc pas l'autonomie du robot. L'idée est de faire évoluer la taille de l'espace perceptif du robot au cours de la tâche d'apprentissage. Plus le robot devient performant, plus les représentations sur lesquelles il s'appuie pour prendre ses décisions deviennent complexes. Cette démarche s'étend aussi, et dans le même temps, aux capacités motrices du robot ainsi qu'à la difficulté du comportement que doit apprendre le robot. Nous qualifions cet apprentissage de "*développemental*" car il s'inspire des mécanismes de développement présents chez les jeunes enfants et identifiés, notamment, par la psychologie du développement et les neurosciences cognitives. Des recherches montrent comment les mécanismes d'apprentissage profitent de cette prise de conscience progressive de la complexité de son corps et de son environnement par l'enfant (Turkewitz & Kenny, 1985; Kuhl, 1999).

Dans un premier temps, cet article présente les principes essentiels de l'apprentissage par renforcement et détaille les difficultés pratiques liées à la taille et à la complexité des espaces sensorimoteurs d'un agent. Nous verrons ensuite quels éléments d'apprentissage développemental permettent de faciliter l'exploration de l'espace sensorimoteur avant de détailler l'architecture que nous proposons. Puis nous décrirons la plateforme expérimentale et les expériences réalisées, avant de discuter les résultats actuels et la suite des travaux. Une brève conclusion complète cet article.

2 Principes et Motivations

Notre approche d'apprentissage par renforcement développemental s'inspire largement de l'algorithme du *Q-learning* (Watkins, 1989) et cherche donc à estimer la fonction de valeur optimale dans l'espace $\mathcal{S} \times \mathcal{A}$. Le principe de l'algorithme est d'utiliser chaque échantillon d'apprentissage (s_t, a_t, s_{t+1}, r_t) fournit à l'instant t par une interaction de l'agent avec son environnement (percevant l'état s_t, l'agent effectue l'action a_t qui modifie l'environnement, alors perçu dans le nouvel état s_{t+1}, et reçoit une récompense r_t qui peut être nulle) pour mettre à jour l'estimation de la fonction de valeur Q^* pour le couple (s_t, a_t) en s'appuyant sur l'équation de Bellman (**??**) de la manière suivante :

$$Q^*_{t+1}(s_t, a_t) =\longleftarrow (1-\alpha)Q^*_t(s_t, a_t) + \alpha\left(r + \gamma \max_{a' \in \mathcal{A}} Q^*_t(s_{t+1}, a')\right), \tag{1}$$

où α est un coefficient d'apprentissage, qui peut dépendre de s et de a.

Une des principales limitations à l'utilisation pratique des méthodes d'apprentissage par renforcement est liée à la complexité des algorithmes. En théorie, pour garantir la convergence des algorithmes d'apprentissage par renforcement vers une solution optimale, il faut que chaque couple (*état*, *action*) du processus soit visité un nombre infini de fois. Même en se contentant d'un très grand nombre de visite, cette contrainte est l'une des principale limitation à l'utilisation pratique des techniques d'apprentissage par renforcement, et notamment dans le cadre de la robotique pour les raisons suivantes :

a) **Espace d'état continu**. L'espace perceptif d'un robot est généralement continu et, bien que le formalisme des MDP reste valide dans ce contexte, les algorithmes d'apprentissage classiques s'appuient sur un espace d'état discret et fini. Il est évidemment possible de discrétiser l'espace d'état, mais ce n'est pas toujours simple et il faut trouver un bon compromis entre une discrétisation trop fruste (la taille de l'espace d'état reste gérable mais les résultats de l'apprentissage peuvent être erronés) et une discrétisation trop détaillée (la taille de l'espace d'état peut alors être trop grand pour être computationnellement gérable). Une autre possibilité est d'utiliser des schémas d'approximation pour la fonction de valeur, les travaux en voguent s'appuient sur l'utilisation de régression linéaire au sens des moindres carrés (Lagoudakis *et al.*, 2002). La qualité de l'approximation repose en grande partie sur la qualité des échantillons utilisés, et finalement, on doit de nouveau gérer un compromis entre trop et trop peu d'échantillons, sans parler de la difficulté de générer des échantillons offrant une bonne couverture de l'espace d'état.

b) **Echantillon coûteux**. Obtenir un échantillon d'apprentissage, c'est-à-dire effectuer une action a dans un contexte perceptif s pour en observer le résultat, en terme de nouvel état s' et d'une éventuelle récompense r, n'est pas anodin en robotique. Au minimum, cela demande un certain temps et, dès lors, il devient impossible de générer des corpus d'apprentissage conséquent. C'est un problème d'autant plus crucial que le `Q-learning` ne s'appuie que sur des interactions effectives entre l'agent et son environnement. Des techniques adaptées à cette problématique où l'on ne dispose que de peu de donnée existent, notamment en mémorisant et en réutilisant plusieurs fois les mêmes données, mais là encore le problème reste largement ouvert.

c) **Exploration de l'espace d'état-action**. Qui plus est, si l'on ne dispose pas d'un simulateur ou d'un oracle, il est délicat de générer des échantillons d'apprentissage pertinents et couvrant de manière adéquate l'espace des (*état*, *action*). Le dilemme exploration/exploitation a été identifié depuis longtemps (par exemple (Thrun, 1992)), il reste clairement un problème actuel. La probabilité qu'un robot utilisant une politique aléatoire suivent une trajectoire récompensée est assez faible sauf si la récompense à été bien pensées (au risque de trop simplifier l'apprentissage, (Ng *et al.*, 1999)) ou si le robot est accompagné ou guidé dans son apprentissage (Buffet *et al.*, 2007). Une autre approche possible, inspirée de la psychologie développementale, est de doter le robot de capacités lui permettant d'explorer progressivement son environnement au fur et à mesure que son apprentissage sensorimoteur devient de plus en plus performant (Oudeyer *et al.*, 2007). La validité de cette démarche a été montrée par expérimentation sur des tâches avec 3 degrés de liberté, il reste à voir comment on peut l'étendre à des espaces d'action plus compliqués.

d) **Richesse de l'environnement**. Ce qui pourrait sembler un avantage est aussi, et surtout, une difficulté pour les algorithme d'apprentissage par renforcement. L'environnement d'un robot étant riche et varié, il regorge d'informations que le robot peut prendre en compte pour décider de son action. En conséquence, la taille de l'espace d'état peut croître de manière disproportionnée et, finalement, mettre à mal tout algorithme d'apprentissage. On retrouve le "*Frame Problem*" énoncé par (McCarthy & Hayes, 1969) qui reste un problème majeur dès lors qu'on ne veut pas limiter artificiellement – et souvent de manière *ad hoc* – l'espace perceptif de l'agent, ce qui revient à terme à l'empêcher d'augmenter son autonomie et limiter ses capacités. Cet aspect du problème de l'apprentissage est au cœur de nos préoccupations.

3 Approche développementale

Parmi les nombreux concepts et approches mis en avant dans le cadre de la robotique développementale (Lungarella *et al.*, 2003), nous voulons explorer plus avant les apports possibles d'un accompagnement progressif de l'agent apprenant pour lui permettre de gérer la complexité de la tâche d'apprentissage. Ici, cet accompagnement prend principalement la forme d'une augmentation progressive de la richesse des perceptions et des actions potentielles de l'agent au fur et à mesure que ses performances dans la tâche à apprendre progressent.

Ainsi, en débutant son apprentissage avec des perceptions et des actions très grossières, l'agent ne pourra pas résoudre des tâches très compliquées. Mais, pour une tâche adaptée, il n'aura qu'à explorer un espace de taille réduite pour trouver un comportement satisfaisant. L'idée est de capitaliser sur les comportements ainsi appris pour, en augmentant progressivement la richesse des perceptions et des actions, permettre à l'agent d'apprendre des tâches plus compliquées, tâches qui pourraient s'avérer impossible à apprendre directement.

De même, la difficulté de la tâche elle-même peut être amenée à augmenter au fur et à mesure de l'apprentissage. Le but principal de cette approche est de permettre à l'agent apprenant de tirer le maximum de la richesse et de la complexité du monde sans être pénalisé par cette trop grande richesse, répondant ainsi principalement à la problématique d) exposée à la section précédente (cf. 2).

Dans cette optique, plusieurs problèmes se présentent. Il faut d'une part que l'architecture d'apprentissage mise en place permettent de gérer de la manière la plus transparente (du point de vue de l'agent) et la plus autonome possible ces augmentations progressives de perception, motricité et difficulté de la tâche. D'autre part, nous voulons que l'agent puisse s'appuyer sur ce qu'il a déjà appris pour apprendre dans un environnement plus riche. Cette dernière problématique est particulièrement délicate, elle se rapproche de la problématique du transfert de connaissance, domaine de recherche encore largement ouvert (Caruana, 1997). Dans le cadre de l'apprentissage par renforcement, un élément central de l'apprentissage est l'estimation de la fonction de valeur sur l'espace sensorimoteur. Les problèmes que nous avons évoqués se transposent donc au niveau de l'estimation d'une fonction : comment tirer parti de l'estimation courante d'une fonction pour l'étendre à un espace d'entrée modifié (quand les perceptions s'enrichissent), comment utiliser l'estimation courant pour estimer une autre fonction que l'on suppose "proche" de l'estimation courante (quand on veut valuer une nouvelle action ou valuer une nouvelle tâche) ?

4 Architecture d'apprentissage développemental

Pour gérer les problèmes cités précédemment liés à la taille et la nature des espaces d'état et d'action (voir Section 2), nous nous appuyons sur une structure d'approximation de fonction. Cette structure d'approximation est développée autour d'une architecture connexioniste que nous pensons plus à même de pouvoir s'adapter à

un espace sensorimoteur variable et évolutif qui découle de l'aspect développemental de l'apprentissage. Ainsi, la nature même de cet approximateur de fonction est à l'origine de plusieurs travaux autour de la notion d'apprentissage par renforcement développemental, ou nous avons d'abord travaillé avec un ou plusieurs perceptrons multicouche avant d'explorer plus méthodiquement l'utilisation de cartes auto-organisatrices adaptatives que nous décrivons dans cet article (Sarzyniec *et al.*, 2011; Dutech, 2011),

Le principe de notre architecture est exposé sur la figure 1. Les perceptions du robot (nous reviendrons plus en détail sur ces perceptions à la section 5), notées s, nourrissent une carte auto-organisatrice dynamique (DSOM [1]). L'activité de cette carte auto-organisatrice constitue l'entrée d'un perceptron à une couche qui possède autant de neurones de sorties que d'actions possibles pour l'agent apprenant. Ainsi, pour le neurone de sortie associé à l'action a, un apprentissage par différences temporelles tirant parti de l'équation de Bellman permet d'apprendre la fonction de valeur $Q(s, a)$ dans un schéma de régression linéaire. Quant à la carte DSOM, elle s'adapte de manière non-supervisée aux entrées perceptives reçues (les détails sont donnés en Annexe, cf section 8).

Bien que nous n'ayons pas utilisé une architecture plus classique comme LSTD ou LSQ (Bradtke & Barto, 1996; Lagoudakis *et al.*, 2002), notre système en reste finalement assez proche. On peut en effet considérer que l'utilisation d'une carte DSOM projette les perceptions sur un ensemble de fonction de base $\Phi(s)$ qui évolue avec le temps, et que la régression linéaire estime la fonction de valeur comme une combinaison linéaire $w^T\Phi(s)$ de ces fonctions de base. La recherche des coefficients optimaux de cette combinaison linéaire se faisant par une descente de gradient stochastique et les fonctions de base étant adaptables, notre algorithme n'offre aucune garantie théorique de convergence.

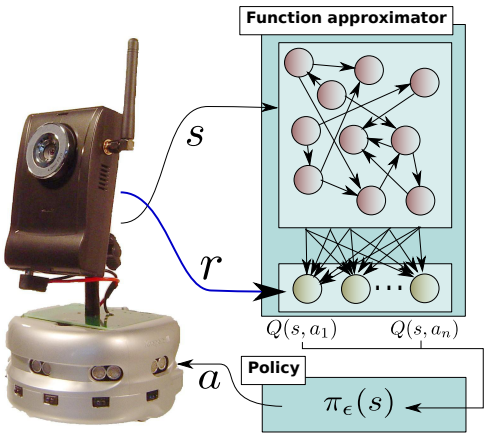

FIGURE 1 – **Architecture d'apprentissage**. Les perceptions du robot s nourrissent une carte auto-organisatrice dynamique. Une couche de sortie linéaire avec un neurone par action a permet d'apprendre $Q(s, a)$.

L'estimation courante de la fonction valeur permet de définir une politique gourmande qui peut être utilisée pour guider les explorations ultérieures de l'agent apprenant, en utilisant un schéma d'exploration epsilon-gourmand par exemple.

L'architecture d'apprentissage à base neuronale que nous utilisons permet de mettre en œuvre l'approche développementale que nous voulons expérimenter. L'espace d'action étant discret, sa richesse est étroitement liée à sa cardinalité. Augmenter la richesse de l'espace des actions peut se gérer en ajoutant des neurones à la coude de sortie. Ces nouveaux neurones bénéficient de la carte DSOM déjà adaptée au entrées, et qui peut continuer à s'adapter. L'apprentissage rapide des coefficients de la combinaison linéaire d'un nouveau neurone de sortie peut être accéléré en initialisant ces coefficients en tenant compte de neurones codant des actions "sémantiquement proches". Dans l'exemple que nous allons présenter plus loin (cf. section 5), un nouveau neurone lié à l'action `tourner-droite-lentement` sera initialisé avec les coefficients du neurone lié à l'action `tourner-droite`.

En ce qui concerne l'espace d'état qui est continu, une partie de sa richesse est liée à la dimension de cet espace. Augmenter la dimension de l'espace d'entrée n'est pas aussi simple que pour l'espace de sortie. Une solution est d'augmenter la dimension de tous les poids des vecteurs d'entrée sans modifier leur projection sur l'espace perceptifs précédent. Une autre solution que nous voudrions tester consiste à doter l'architecture de poids d'entrée de la dimension maximale de l'espace d'entrée et, tant que le vecteur perceptif est de dimension inférieur à ce

1. Dynamic Self-Organizig Map

maximum, à augmenter artificiellement sa dimension en clonant quelques une de ces valeurs. Cette dernière façon de procéder nous paraît plus à même de permettre de transférer ce qui est appris en faible dimension aux tâches de plus grande dimension.

5 Plateforme expérimentale

Nos expériences robotiques sont effectuées sur un robot KheperaIII de l'entreprise K-Team [2]. Bien qu'il soit doté d'un processeur, nous avons préféré déporter les traitements et les calculs sur une machine externe, les communications entre cette machine et le robot se font par wifi.

Le robot se déplace à l'aide de deux roues motrices qui lui permettent de tourner sur place. Le robot dispose d'actions discrètes, comme avancer ou reculer d'une certaine distance (en moyenne, car toute les actions du robots sont imprécises) ou tourner à droite ou à gauche d'un certain angle. Nous utilisons des actions discrètes car nous voulons explorer un schéma de type *Q-Learning* et que l'estimation d'une fonction de valeur pour des actions continues est un problème encore largement ouvert. L'espace des actions du robot est donc un espace discret et la richesse de cet espace sera caractérisé par le nombre des actions disponibles.

Parmi les capteurs du robot, nous n'utilisons les capteurs de distance infrarouge que pour stopper le robot s'il s'approche trop d'un obstacle. Le capteur principal que nous utilisons est une caméra wifi placée au dessus du robot (voir figure 2) et qui permet de capturer une image couleur 320x200 au format bitmap, image qui est ensuite traitée pour fournir une information sensorielle évolutive dans un espace de dimension moindre.

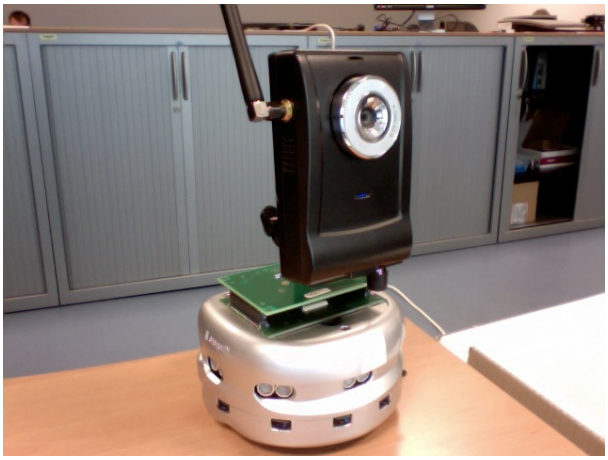

FIGURE 2 – Le robot Khepera III et sa caméra.

Ainsi, par le biais d'un capteur logique que nous appelons la "*rétine*", nous fournissons au robot une perception constituée d'un vecteur de dimension où chaque coefficient du vecteur, compris entre 0 et 1, correspond au taux d'une teinte particulière calculée sur une bande verticale de l'image. Le nombre et la position de ces bandes verticales est paramétrable et peut évoluer pendant les expériences. Sur l'exemple de la partie droite de la figure 3, le robot fait face à un losange bleu sur fond rouge, la rétine est configurée pour donner les taux de bleu sur des bandes verticales – nommées `B1`, `B2` et `B3` – positionnées aux abscisses 80, 160 et 240 de l'image. Les perceptions du robot dans ce cas sont donc $[0,42; 0,67; 0,33]$. L'espace des états du robot est donc un espace continu dans la richesse sera mesurée, dans le cadre des expériences que nous allons mener, par la dimension de cet espace (c'est-à-dire le nombre de bandes verticales).

Etant donné notre plateforme expérimentale, les tâches que nous voulons faire apprendre à notre robot sont des tâches de navigation dans des labyrinthes avec des indices visuels. Notre idée est de placer des indices visuels contrastés dans l'environnement pour guider le robot, par exemple des flèches qui lui indiquent dans quelle direction se diriger pour gagner la sortie. Si le robot a besoin d'une perception fine pour distinguer et reconnaître les flèches, il n'a pas besoin de cette finesse de perception pour être "attiré" par ces symboles contrastés. De plus, le robot peut apprendre des comportements "satisfaisant" avec seulement des actions grossières et affiner et préciser

2. Voir `http://www.k-team.com`.

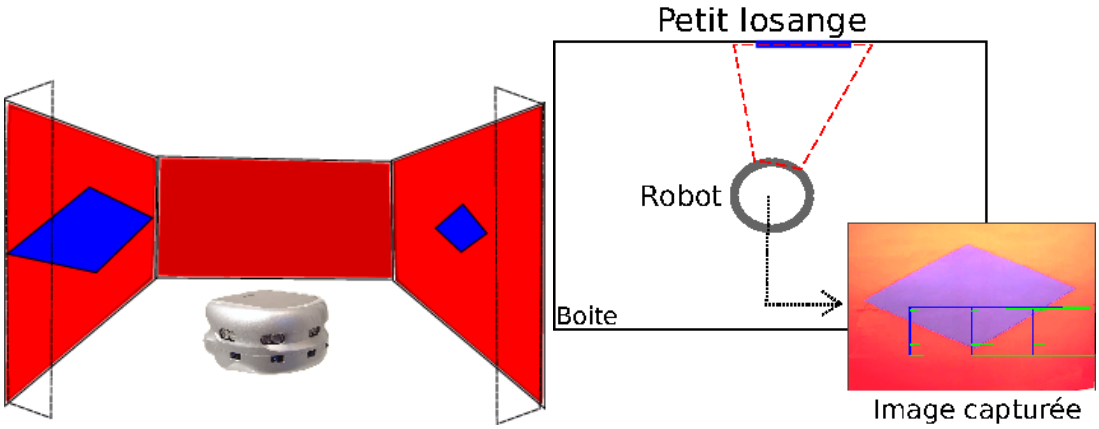

FIGURE 3 – Gauche : le robot dans son environnement. Droite : les perceptions données par la `rétine` du robot. Pour chacune des bandes verticales, en positions $(80, 160, 240)$, le taux de bleu est fourni, soit $[0,42; 0,67; 0,33]$.

ces comportements par la suite. Nous pensons que ce type de scénario, qui est décomposables en sous-tâches plus ou moins complexes, est propice à l'expérimentation des concepts que nous avons exposés ici.

6 Résultats expérimentaux

L'architecture d'apprentissage que nous proposons est testée en confrontant le robot à une tâche très simple. Cette expérience, qui ne met en jeu que l'augmentation de la richesse des actions du robot, est vue comme une première expérimentation, prélude à d'autres tests où l'augmentation de la richesse perceptive et de la difficulté des tâches seront testées.

Dans cette expérience, le robot est placé dans un environnement fermé. Les murs qui l'entourent sont de couleur rouge et un losange bleu (la cible) est placée sur un des murs. Le robot dispose de 5 actions discrètes : `tourner-droite` et `tourner-gauche` (de 34 degrés environs), `stop`, `tourner-droite-lentement` et `tourner-gauche-lentement` (de 20 degrés approximativement). La nature du robot rend ces actions largement bruitées. L'espace perceptif est constitué de 3 bandes verticales donnant le taux de bleu, ces bandes, nommées $B1$, $B2$ et $B3$ sont respectivement placée à l'extrémité gauche, au milieu et à l'extrémité droite de l'image caméra du robot, ce qui correspond à des angles relatifs de -16, 0 et 16 degrés approximativement. Le robot doit apprendre à se tourner vers la cible, et il reçoit une récompense s'il s'arrête alors que la cible est plus ou moins en face de lui. Si $0,7B1 + B2 + 0,7B3 > 1,4$ et que le robot s'arrête, alors il reçoit une récompense de $+1$, sinon sa récompense est nulle. Après une récompense, ou après 12 mouvements sans succès, le robot est replacé aléatoirement.

Les échantillons d'apprentissage sont générés sur la plateforme robotique, mais l'algorithme est encore trop gourmand en nombre d'échantillon pour que l'apprentissage puisse se faire en utilisant uniquement ces échantillons. Comme nous le verrons, il faut en moyenne 10 000 à 15 000 itérations pour converger. Nous nous sommes donc résolus à utiliser, et surtout réutiliser 2 000 échantillons issus de la plateforme robotique. Les échantillons d'apprentissage sont donc choisis aléatoirement dans ces échantillons réels, en respectant les scénarios d'apprentissage (on n'utilise que les échantillons compatibles avec les action courantes).

Trois scénarios d'apprentissage sont testés. Dans le premier, le robot ne dispose que des 3 actions `tourner-droite`, `tourner-gauche` et `stop`. Dans le second, le robot dispose dès le départ de toutes ses actions, c'est-à-dire qu'il peut aussi tourner lentement. Enfin, dans le scénario dit "*développemental*" le robot commence à apprendre avec 3 actions pendant N_B itérations puis il peut utiliser les 2 actions supplémentaires, avec une altération éventuelle de certains paramètres d'apprentissage.

Les résultats que nous présentons ont été obtenus en fixant les paramètres suivants : 64 neurones pour la carte DSOM, avec une organisation "petit monde" et au moins un arc par neurone ($\mathrm{nb}_{\mathrm{neur}} = 64$, $\mathrm{nb}_{\mathrm{link}} = 1$) ; $\epsilon_D = 0,5$ et $\eta = 1$ comme paramètres d'apprentissage de la carte DSOM ; $\alpha = 0,1$ et $\gamma = 0,9$ pour le problème d'apprentissage par renforcement ; la politique d'exploration utilisée par le robot est une politique epsilon-gourmande, avec $\epsilon_\pi = 0,25$. Le coefficient d'apprentissage ϵ_L du régresseur linéaire variera en fonction des expériences (généralement entre $0,001$ et $0,5$. Ce paramètre semble en effet jouer un rôle plus important sur la qualité des

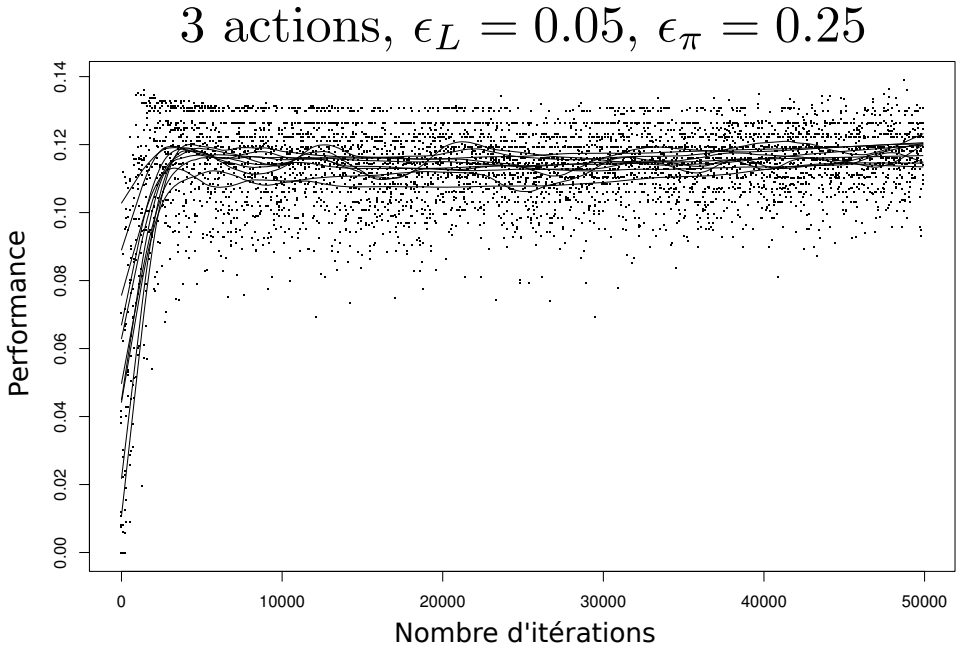

FIGURE 4 – Apprentissage avec 3 actions, les points représentent les performances mesurées toutes les 100 itérations sur 10 trajectoires, chaque trajectoire est représentée en étant lissée.

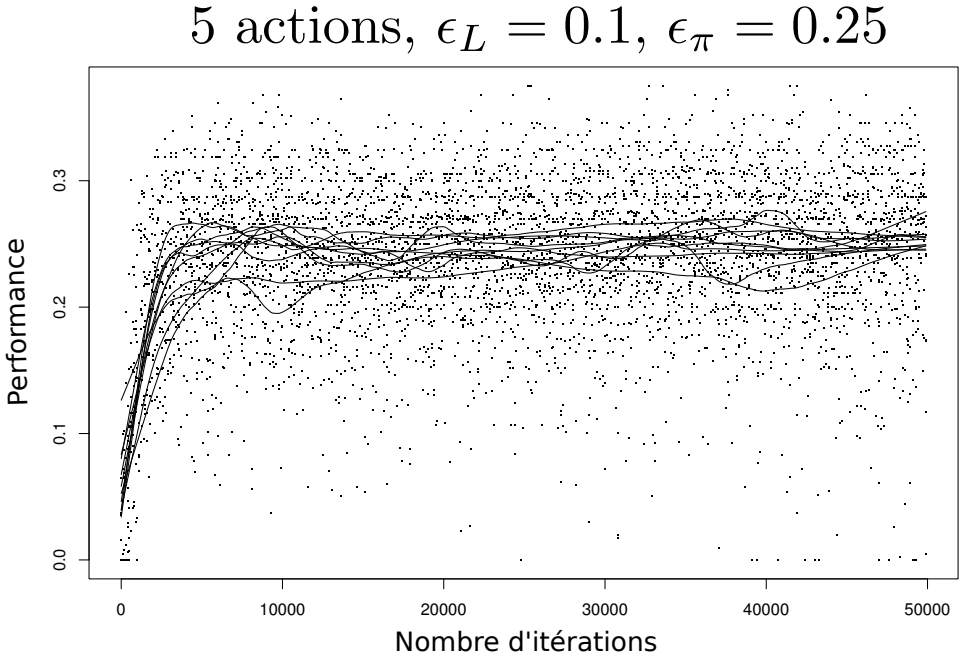

FIGURE 5 – Apprentissage avec 5 actions, les points représentent les performances mesurées toutes les 100 itérations sur 10 trajectoires, chaque trajectoire est représentée en étant lissée.

résultats et la vitesse de convergence de l'architecture. Ces paramètres ont été fixés par essais et erreur, de manière à présenter des résultats d'apprentissage satisfaisant, comme le suggère la représentation d'une politique apprise sur la figure 6. Il faudrait néanmoins explorer plus méthodiquement l'influence de chacun des paramètres

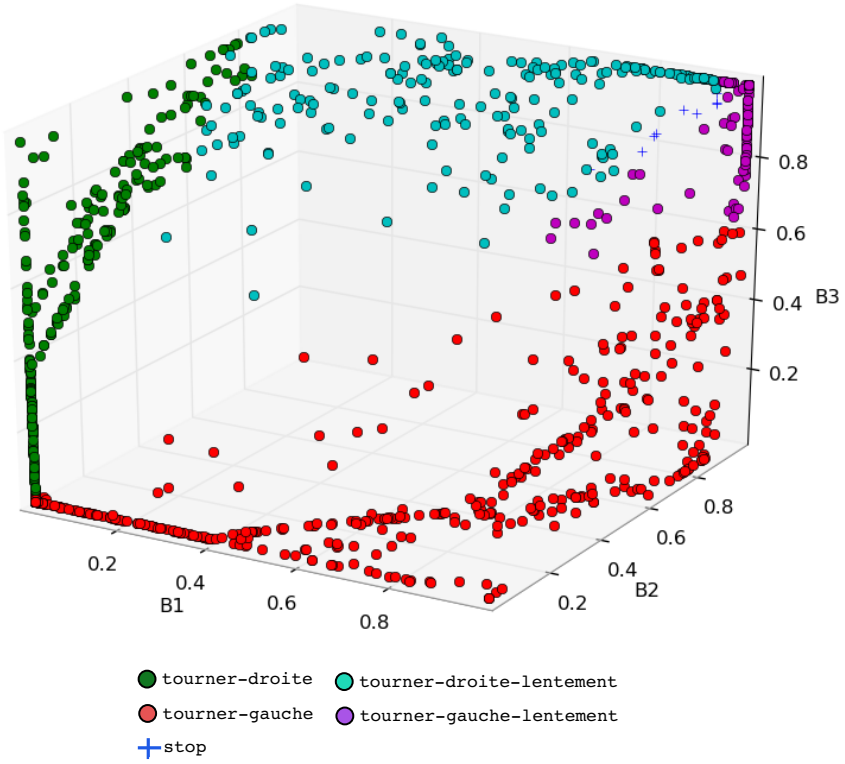

FIGURE 6 – Représentation d'une politique apprise avec 5 actions dans l'espaces sensorimoteur (les axes B1, B2 et B3 sont les dimensions de cet espaces perceptif). L'agent reste bien immobile face à la cible (ce qui arrive rarement) et tourne plus largement lorsque la cible est loin du centre (faibles valeurs de B1 ou B3).

Dans tous les cas, la qualité de l'apprentissage est évalué toutes les 100 itérations en testant la politique gourmande apprise par le robot sur un ensemble de positions de départs choisi de manière à bien distinguer les politiques. Les angles caractérisants ces points sont : 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 25, 30, 35, 40, 45, 50, 90, 180 et leur symétriques. La moyenne des récompenses obtenues depuis ces positions fixées caractérise la performance des politiques. Pour information, la meilleure politique utilisant les 3 actions de base à une performance théorique de $0,138$ et avec les 5 actions on peut optimalement obtenir une performance de $0,39$.

Les figures 4 et 5 donnent la performance de l'algorithme en fonction du nombre d'itérations en utilisant 3 et 5 actions avec $\epsilon_L = 0,05$ et $\epsilon_L = 0,1$. Il y a 10 trajectoires d'apprentissage par graphe. Les points sont les valeurs effectivement mesurées et les courbes sont des moyennes glissantes de ces trajectoires d'apprentissage. Pour ces valeurs des paramètres, comme pour l'essentiel des valeurs testées, on constate une grande variabilité dans la performance de l'algorithme au cours d'une même trajectoire. En 100 itérations, la performance de l'algorithme peut varier grandement. Cette variabilité peut être diminuée en prenant pour ϵ_L des valeurs très faibles, mais comme le montre la figure 7, la vitesse d'apprentissage s'en ressent fortement.

En parlant de cette vitesse de convergence, il est aussi intéressant de constater que l'utilisation d'une architecture neuronale s'appuyant sur une carte auto-organisatrice accélère notablement la vitesse d'apprentissage. Dans des travaux précédents mettant en œuvre un perceptron multi-couche et l'utilisation de traces d'éligibilité (Sarzyniec *et al.*, 2011), il fallait environs $100\,000$ itérations avant de converger alors qu'ici on est plutôt entre $5\,000$ et $15\,000$ itérations.

Le point qui nous intéresse le plus est l'apport d'une stratégie développementale pour l'apprentissage. La figure 8 permet de comparer les performances relatives de l'algorithme obtenues en utilisant directement 5 actions ou en

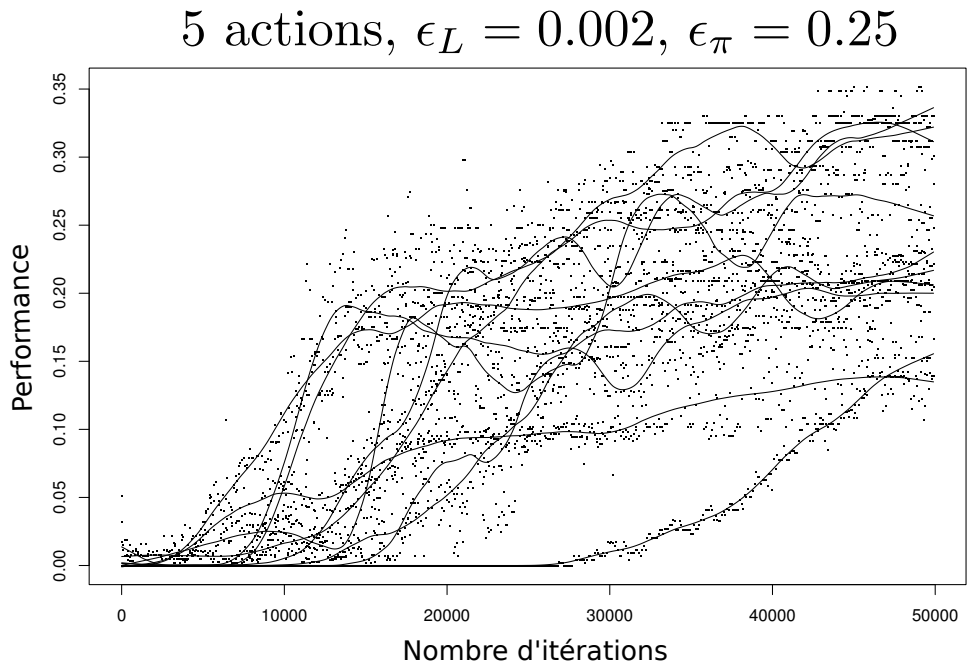

FIGURE 7 – Apprentissage avec 5 actions, les points représentent les performances mesurées toutes les 100 itérations sur 10 trajectoires, chaque trajectoire est représentée en étant lissée.

s'appuyant sur le scénario développemental. On y voit que les performances de l'approche directe progressent plus rapidement mais que l'approche développementale permet d'obtenir, relativement rapidement (à partir de 10 000 itérations), de meilleures performances. Il faut aussi noter que pour que les performances du scénario développemental restent élevées, le coefficient d'apprentissage ϵ_L du régresseur linéaire a été diminué de $0,1$ à $0,01$ lors de l'ajout des nouvelles actions. Si la valeur de ce coefficient reste la même $(0,1)$, on obtient les performances de la courbe rouge de la figure 9.

La figure 9 montre que le nombre d'itérations d'apprentissage *avant* de passer à 5 actions n'influence pas les performances atteintes ensuite. On remarque aussi que pour ce jeu de paramètres ($\epsilon_L = 0,1$ et $\epsilon_\pi = 0,25$, comme pour beaucoup d'autres, les performances d'apprentissage affichent une tendance à décroître avec le nombre d'itérations. La grande variabilité des performances et cette tendance semblent indiquer que la tâche d'apprentissage n'est pas aussi simple qu'il n'y paraît, une constatation à rapprocher du fait que l'utilisation d'approximateurs neuronaux dans un cadre d'apprentissage par renforcement reste délicat (Coulom, 2002).

7 Discussion

Rappelons pour commencer que les résultats présentés ici restent très préliminaires et relèvent d'un travail en cours. Il y a notamment de nombreux paramètres qui peuvent influencer le comportement de l'algorithme d'apprentissage et une exploration plus méthodique de ces influences est à mener. Certains semblent se "compenser", comme par exemple le nombre de neurones de la carte DSOM et les coefficients d'apprentissage de cette dernière. Avec un grand nombre de neurones, il n'est pas nécessaire que cette carte soit vraiment adaptative pour que les performances de l'algorithme soient satisfaisantes. Ce cas est d'ailleurs plus typique des architecture de "reservoir computing" étudiées et proposées classiquement.

Nous pensons que les propriétés caractéristiques des cartes DSOM, comme le fait de s'adapter continuellement aux entrées présentées et de ne pas être sensible à la distribution de probabilité sous-jacente à ces entrées, seront particulièrement intéressantes avec un espace perceptif évolutif. Comme pour les actions, nous envisageons d'augmenter la cardinalité de l'espace de perception (en ajoutant des bandes visuelles) mais aussi de déplacer des bandes existantes et de bruiter plus ou moins les taux de couleurs calculé sur ces bandes. Ceci reste évidemment à tester et à explorer.

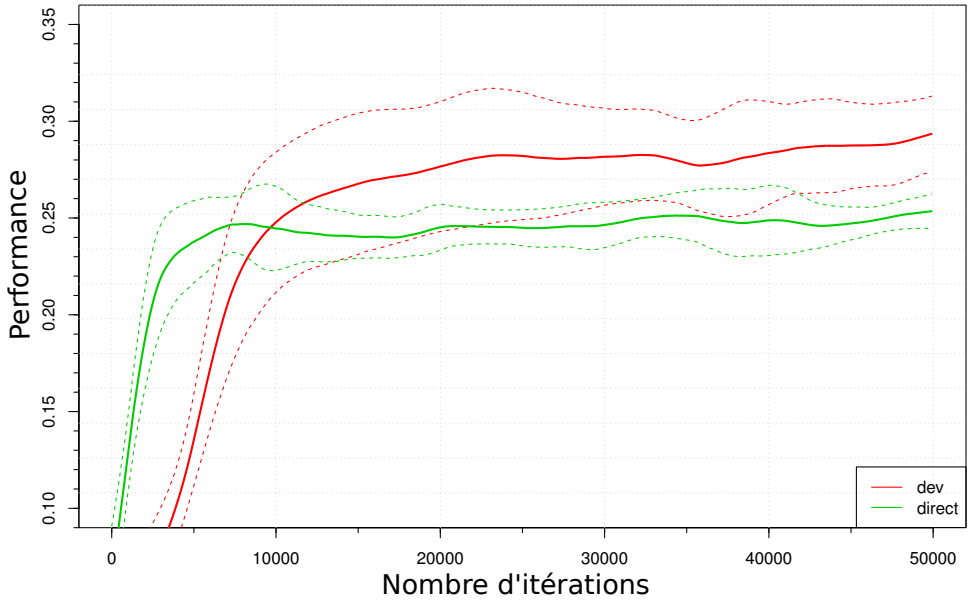

FIGURE 8 – **Comparaison approche développementale**. Comparaison entre les performances de l'approche directe avec 5 actions et un scénario développemental où 2 actions supplémentaires sont ajoutées après $N_B = 5\,000$ itérations (et dans ce cas, ϵ_L passe de 0,1 à 0,01). Les courbes montrent la moyenne et la variance pour 10 apprentissages.

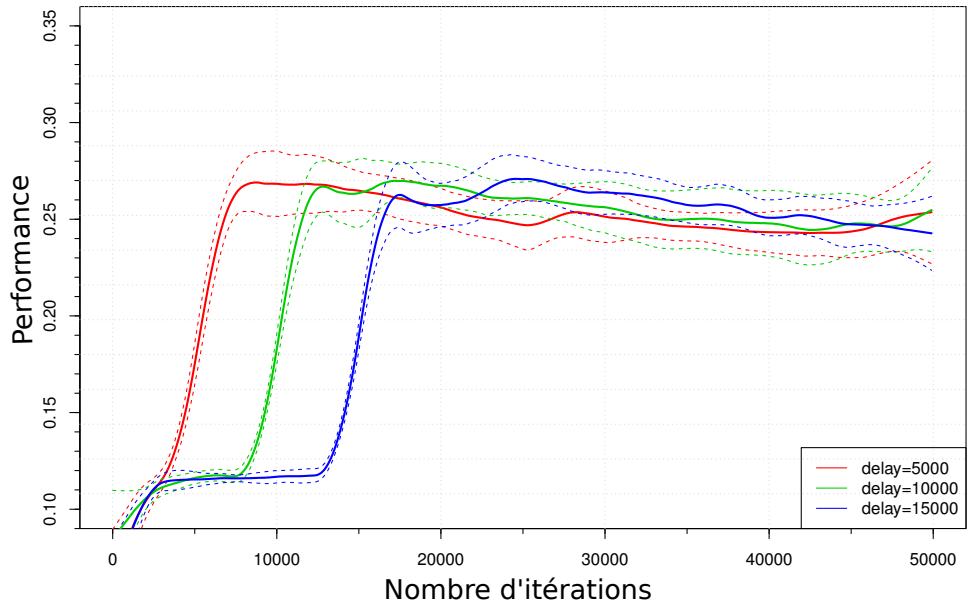

FIGURE 9 – **Influence du délai dans l'approche développementale**. Dans un scénario développemental, on compare les performances en fonction du nombre d'itérations avant l'ajout des 2 actions supplémentaires ($\epsilon_L = 0, 1$, $N_b = \{5\,000, 10\,000, 15\,000\}$). Les courbes montrent la moyenne et la variance pour 10 apprentissages.

Les performances d'apprentissage sont très instables, d'une itération à l'autre, l'évaluation de la fonction valeur, et donc la politique ϵ-gourmande associée peuvent varier grandement. Même quand la carte DSOM est stabilisée, ce qui arrive assez rapidement dans l'expérience menée car l'espace perceptif est stable, et que le coefficient d'apprentissage de la sortie linéaire est faible, les performances restent instables. Pour essayer de diminuer cette instabilité, tout en conservant des performances acceptables et une vitesse de convergence la plus grande possible, une étude systématique de tous les paramètres serait souhaitable mais difficile à mettre en œuvre car la prise en compte de ces différents critères est qualitative. Pourtant, une estimation plus quantitative de la performance serait non seulement intéressante pour mieux fixer les paramètres mais aussi automatiser le processus qui accroît la richesse de l'espace sensorimoteur, en s'appuyant sur cette mesure de performance. La performance de l'apprentissage pourrait alors guider le processus développemental, comme cela a été exploré dans (Baranes & Oudeyer, 2010) par le biais de l'exploration de nouveaux sous-buts comportementaux.

8 Conclusion

Dans cet article nous avons présenté les principes d'un apprentissage par renforcement développemental dans le but de faciliter l'exploration, et donc l'exploitation, d'espace sensorimoteur compliqué et de grande taille. Le cœur de cette architecture est un approximateur de fonction neuronal qui, dans le cadre du "reservoir computing", s'appuie sur une carte auto-organisatrice dynamique. Le principe général de notre algorithme, que nous avons détaillé, prend comme modèle le *Q-Learning*.

Les premières expérimentations sur une tâche robotique plutôt simple n'ont pas encore permis d'évaluer toutes les caractéristiques de notre architecture. Nos premiers résultats nous poussent à continuer nos expérimentations, notamment car cette architecture originale est capable d'apprendre une politique de qualité bien plus rapidement que celles testées dans des travaux précédents (Sarzyniec *et al.*, 2011; Dutech, 2011) quand l'espace des actions grossi pendant l'apprentissage. Outre une étude plus systématique des différents paramètres de l'algorithme, il nous faut maintenant passer à des tâches plus complexes, mettant en jeu des espaces perceptifs évoluant au cours du temps avant de pouvoir déterminer si cette architecture, et les concepts qui la sous-tendent, sont valides. Le plus gros du travail est devant nous.

Annexe : Détails de l'architecture

La partie "réservoire" de notre architecture est une carte DSOM (Rougier & Boniface, 2011) qui fonctionne de la manière suivante.

Une perception s est présentée. On cherche dans la carte DSOM le neurone le plus proche en terme de sa distance à s, on l'appelle v. Les poids d'un neurone i sont notés w_i.

$$w_v = \text{argmin}_{w_i, i \in \mathcal{N}}(||s - w_i||) \tag{2}$$

où $\mathcal{N}$ est l'ensemble des neurones de la carte DSOM.

Les poids du neurone "vainqueur" v et de ses voisins sont alors modifiés. La notion de voisinage s'exprime par un graphe d'adjacence dans l'espace des neurones. Une organisation classique est de distribuer les neurones sur une grille régulière 2D ou 3D et de calculer la distance entre voisin en terme de distance euclidienne. Ici, comme nous ne sommes pas sûr que la structure de l'espace d'entrée soit très régulière, nous préférons utiliser une organisation "petit monde" où un neurone possède au moins nb_{lien} voisins. La figure 10 illustre ces organisations.

Si p_i est la position du neurone i dans l'espace des neurones, on modifie les poids du neurone i de la manière suivante.

$$\delta w_i = \epsilon_D ||s - w_i|| h_\eta(i, v, s)(s - w_i) \tag{3}$$

où ϵ_D est un paramètre de la carte et h_η une fonction de voisinage qui pondère la modification en fonction de la distance entre un neurone et le neurone vainqueur (noté v) d'une part et le neurone vainqueur et l'entrée d'autre part.

$$h_\eta(i, v, s) = \exp -\frac{1}{\eta^2} \frac{||p_i - p_v||^2}{||s - w_v||^2} \tag{4}$$

avec p_i la position de chaque neurone et η un paramètre d'apprentissage.

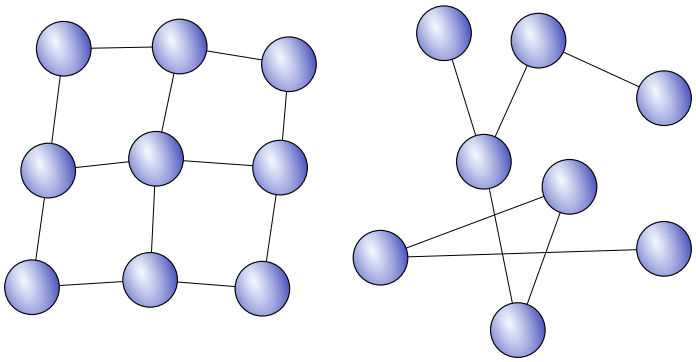

FIGURE 10 – **Voisinage dans la carte DSOM**. A gauche, on utilise une organisation classique sous forme d'une grille 2D où la distance entre 2 neurones est la distance euclidienne entre les neurones. A droite, on utilise une organisation "petit monde" où la distance est le nombre d'arêtes entre les neurones.

Dans le cadre "petit monde" des cartes DSOM que nous utilisons, la distance $||p_i - p_v||$ est fonction du nombre d'arc séparant les neurones varie dans $[0; 1]$ (nombre d'arcs entre le neurone i et le neurone v divisé par le maximum de cette distance), donc toujours inférieure à 1.0. Quant à $||s - w_v||$, c'est la distance euclidienne entre s et les poids du neurones v.

En ce qui concerne le perceptron monocouche de sortie, ses valeurs d'entrées x_j sont les activations des neurones de la carte DSOM et l'activité y_k du neurone de sortie k se calcule comme :

$$y_k = \sum_j w_{jk} x_j + \theta_k \tag{5}$$

où w_{jk} est le poids reliant le neurone d'entrée j au neurone de sortie k et θ_k est un coefficient qui sera aussi appris.

L'apprentissage suit une règle classique :

$$\delta w_{jk} = -\epsilon_L(\text{error}_k)x_j \tag{6}$$

$$\delta \theta_k = -\epsilon_L(\text{error}_k) \tag{7}$$

$$\tag{8}$$

où ϵ_L est un paramètre de l'algorithme et l'erreur associée au k-ème neurone de sortie (associé à l'action a_k), calculé à la suite d'une transition (s, a, s', r), est l'erreur de Bellman si $a_k = a$ et 0 sinon. Ainsi, si l'agent a effectué l'action a dans l'état s pour transiter vers l'état s' en recevant une récompense r, on a :

$$\text{error}_k = \begin{cases} \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] & \text{si} a_k = a \\ 0 & \text{sinon} \end{cases} \tag{9}$$

où α est un coefficient d'apprentissage, γ un coefficient de pondération.

Les paramètres de notre architecture sont donc :

- nb_{neur}, le nombre de neurones dans DSOM
- le type de voisinage dans DSOM, avec nb_{link} le nombre maximum de voisins par neurone dans l'hypothèse petit monde.
- ϵ_D et η, paramètres de la plasticité de DSOM
- ϵ_L, le coefficient d'apprentissage du perceptron de sortie

à cela il faut ajouter les paramètres classiques de l'apprentissage par renforcement (α, γ) et l'aléa ϵ_π de la politique ϵ_π-gourmande.

Références

BARANES A. & OUDEYER P.-Y. (2010). Maturationally-constrained competence-based intrinsically motivated learning. In *Proc. of IEEE Int. Conference on Development and Learning (ICDL 2010)*, Ann Arbor, Michigan, USA.

BRADTKE S. & BARTO A. (1996). Linear least-squares algorithms for temporal difference learning. **22**, 33–57.

BUFFET O., DUTECH A. & CHARPILLET F. (2007). Shaping multi-agent systems with gradient reinforcement learning. *Autonomous Agent and Multi-Agent System Journal (AAMASJ)*, **15**(2), 197–220.

CARUANA R. (1997). *Learning to Learn*, chapter Multitask Learning. Kluwer Academic Publishers.

COULOM R. (2002). *Reinforcement learning using Neural Networks, with Applications to Motor Control*. PhD thesis, Institut National Polytechnique de Grenoble.

DUTECH A. (2011). Dynamic reservoir for developmental reinforcement learning. In *Workhop on Development and Learning in Artificial Neural Networks (DevLeaNN)*, Paris.

GROUPE PDMIA (2008). *Processus Décisionnels de Markov en Intelligence Artificielle. (Edité par Olivier Buffet et Olivier Sigaud)*, volume 1 & 2. Lavoisier - Hermes Science Publications.

KUHL P. (1999). *The New Cognitive Neurosciences*, chapter 8 - Language, Mind, and Brain : Experience Alters Perception, p. 99–115. Bradford Books.

LAGOUDAKIS M., PARR R. & LITTMAN M. (2002). Least-squares methods in reinforcement learning for control. In *Proc ; of the 2nd Hellenic Conference on Artificial Intelligence (SETN-02)*, number 2308 in Lecture Notes on Artificial Intelligence, p. 249–260.

LUNGARELLA M., METTA G., PFEIFER R. & SANDINI G. (2003). Developmental robotics : a survey. *Connection Science*, **15**(4), 151–190.

MCCARTHY J. & HAYES P. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, **4**, 463–502.

NG A., HARADA D. & RUSSELL S. (1999). Policy invariance under reward transformations : Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning, ICML-99*, p. 278–287.

OUDEYER P.-Y., KAPLAN F. & HAFNER V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionnary Computation*, **11**(2), 265–286.

PUTERMAN M. (1994). *Markov Decision Processes : discrete stochastic dynamic programming*. John Wiley & Sons, Inc. New York, NY.

ROUGIER N. P. & BONIFACE Y. (2011). Dynamic Self-Organising Map. *Neurocomputing*, **74**(11), 1840–1847.

SARZYNIEC L., BUFFET O. & DUTECH A. (2011). Apprentissage par renforcement développemental en robotique autonome. In *Conférence Francophone d'Apprentissage (CAp 2011)*, Chambéry.

SUTTON R. & BARTO A. (1998). *Reinforcement Learning*. Bradford Book, MIT Press, Cambridge, MA.

THRUN S. (1992). *Efficient exploration in reinforcement learning.* Rapport interne CMU-CS-92-102, Computer Science Department, Carnegie Mellon University.

TURKEWITZ G. & KENNY P. (1985). The role of developmental limitations of sensory input on sensory/perceptual organization. *Developmental & Behavioral Pediatrics*, **6**(5), 302–306.

WATKINS C. (1989). *Learning from delayed rewards.* PhD thesis, King's College of Cambridge, UK.