République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE DE MSILA

FACULTE DE TECHNOLOGIE

DEPARTEMENT D'ELECTRONIQUE

MEMOIRE DE MASTER

DOMAINE : SCIENCES ET TECHNOLOGIE

FILIERE : GENIE ELECTRIQUE

OPTION : CONTROLE INDUSTRIEL

Thème

APPRENTISSAGE PAR RENFORCEMENT EN UTILISANT LES RESEAUX DE NEURONES ARTIFICIELS: SIMULATION DE LA NAVIGATION D'UN ROBOT MOBILE

Présenté par :
DILMI Abdallah

Encadré par :
MEZAACHE Hatem

N° d'ordre: 2012/08/85/125/

Promotion : JUIN 2012

République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique

UNIVERSITE DE MSILA

FACULTE DE TECHNOLOGIE

DEPARTEMENT D'ELECTRONIQUE

MEMOIRE DE MASTER

DOMAINE : SCIENCES ET TECHNOLOGIE

FILIERE : GENIE ELECTRIQUE

OPTION : CONTROLE INDUSTRIEL

Thème

APPRENTISSAGE PAR RENFORCEMENT EN UTILISANT LES RESEAUX DE NEURONES ARTIFICIELS: SIMULATION DE LA NAVIGATION D'UN ROBOT MOBILE

Présenté par :
DILMI Abdallah

Encadré par :
MEZAACHE Hatem

N° d'ordre: 2012/08/85/125/

Promotion : JUIN 2012

REMERCIEMENTS

Je remercie tout d'abord Dieu de m'avoir prêté santé et volonté pour mener à terme ce mémoire.

Un remerciement particulier, à l'être le plus chère dans ma vie, à celle qui m'a donné la vie, celle qui s'est sacrifiée durant de longues années, celle qui a tant donnée… sans demander en revanche…. Les mots s'épuisent maman !! mais même en remplissant des pages entières, je demeurerai ingrate à ton égard. Je te dis tout simplement que tu es la perle qui orne ma vie et que ma réussite est la tienne !!!

Ce travail a été réalisé sous la direction de mon encadreur Monsieur **HATEM MEZAACHE**, Je le remercie beaucoup, qui m'a guidé, dirigé et fournit la documentation et les ressources.

Je tiens aussi à remercier, **les membres du jury** qui ont accepté de juger ce travail.

DILMI ABDALLAH

Dédicace

Après de longues années d'études et de travail, sachant l'importance de l'aide des êtres qui n'aiment, je voudrais humblement leurs, dédier ce modeste travail tout en avant qu'ils méritent le meilleur qui soit.

Je dédie ce travail :

À ma chère mère NOURA

qui a fait tant de sacrifice pour m'élever et m'instruire et qui ma encourage tout le long de mon parcours scolaire et académique.

À mon cher père IBRAHIM

m'encourager et suer et à tant travail pour pouvoir m'instruire.

À mon grand père

À ma grande mère

Tout en lui souhaitant bonne santé et longue vie.

À mes chers enseignants qui m'ont dirigé et aidé et surtout soutenu.

À tous mes amis et camarades d'études

« La chique de la pente de l'exprès promo de l'électronique »

DILMI ABDALLAH

SOMMAIRE

CHAPITRE III

ROBOTIQUE MOBILE ET CONCEPTION D'UN SYSTEME D'APPRENTISSAGE PAR RENFORCEMENT

CHAPITRE IV

APPLICATION ET RESULTATS DE SIMULATION

Introduction Générale

Introduction Générale

L'apprentissage des machines est l'un des domaines les plus récents. Pour avoir un système de commande Intelligent, il existe plusieurs outils comme la logique floue, les réseaux de neurones artificiels et les algorithmes génétiques, de plus il existe trois grandes familles d'apprentissage qui sont : apprentissage supervisé, apprentissage non supervisé et apprentissage par renforcement ou semi supervisé ce dernier type est l'un des l'objectifs de notre étude.

Ce type d'apprentissage est basé sur l'interaction de l'agent (robot) avec son environnement de travail. De plus la résolution du problème de ce type d'apprentissage utilise plusieurs méthodes.

Imaginons un robot, doté de capteurs et d'effecteurs, qui va choisir ses actions en fonction de sa perception de l'environnement et dont le but de maximiser une récompense qu'il obtient en fonction des actions qu'il réalise.

Dans le cadre de notre mémoire nous avons utilisé la méthode d'Apprentissage par renforcement en utilisons les réseaux de neurones artificiels de type MLP basée sur l'algorithme ***Q-Learning*** pour assurer la navigation d'un robot mobile en évitant les obstacles qui se trouvent dans son environnement de travail. et pour réaliser notre simulation nous avons utilisé du logiciel **Webots 6.4.1** et aussi Matlab.

Le travail présenté dans ce mémoire est donc organisé en quatre chapitres qui sont structurés comme suit :

Le premier chapitre donne une présentation générale sur l'Apprentissage par renforcement et aussi les méthodes utilisées pour résoudre ce type de méthode d'apprentissage.

L'objectif du deuxième chapitre est de présenter l'un des outils utiliser Pour avoir un système de commande Intelligent qui sont les réseaux de neurones artificiels où on a présenté quelques généralités et aspects théoriques.

Par la suite le chapitre trois on la partager en deux parties principales, Dans la première partie on a donné des généralités sur la robotique mobile, et dans la deuxième partie : on a donné la conception d'un système d'apprentissage par renforcement.

Le quatrième chapitre est le fruit de notre travail, où on a présenté la démarche de notre application et ces résultats de simulation, on a proposé un système d'apprentissage par renforcement qui utilise un réseau de neurones artificiels de type **MLP.**

Et à la fin nous terminons notre travail par une conclusion générale.

Chapitre I:

Apprentissage Par Renforcement

Apprentissage Par Renforcement

I.1-Introduction :

L'apprentissage par renforcement est une méthode qui permet de trouver, par un processus d'essais et d'erreurs, l'action optimale à effectuer pour chacune des situations que le robot va percevoir afin de maximiser une récompense.

Nous commencerons par la présentation du principe de l'apprentissage par renforcement, puis nous donnons une définition sur l'approche développementale de l'apprentissage que nous souhaitons appliquer à la robotique. Enfin nous introduirons brièvement sur les outils théoriques.

I.2- Principes de l'apprentissage par renforcement :

Une solution utilisée couramment pour mettre en œuvre un apprentissage par renforcement est de considérer un agent au sein d'un environnement et qui doit prendre des décisions en fonction de sa situation actuelle (état) dans l'environnement.

L'agent peut interagir avec l'environnement par le biais d'actions et l'environnement lui rend une récompense (positive, négative ou nulle) en fonction de celles-ci. Plus formellement, l'apprentissage par renforcement fait référence à une classe de problèmes d'apprentissage automatique, dont le but est d'apprendre, à partir d'expériences, ce qu'il convient de faire en différentes situations, de façon à optimiser une récompense numérique au cours du temps.

L'agent va donc chercher à maximiser les récompenses obtenues en effectuant différentes expériences et types d'expériences (exploration, suivi d'une politique, ...) au sein de l'environnement. La nature et le choix des expériences à

effectuer à un moment précis de l'apprentissage varie en fonction de l'algorithme d'apprentissage choisi. **[1]**

I.3- Approche développementale de l'apprentissage :

Les principes de l'approche développementale de l'apprentissage sont inspirés de l'observation du comportement humain et plus particulièrement celui des très jeunes enfants Figure (I.1). Cette méthode s'inspire des mécanismes de développement présent chez les enfants, et identifiés par la psychologie du développement, les neurosciences cognitives, et la linguistique cognitive. La partie qui intéresse les chercheurs dans ce domaine est le mécanisme d'apprentissage où l'enfant prend progressivement conscience du monde qui l'entoure, de ses propres capacités et appréhende l'environnement qui l'entoure tout en continuant de faire évoluer ses capacités. **[1]**

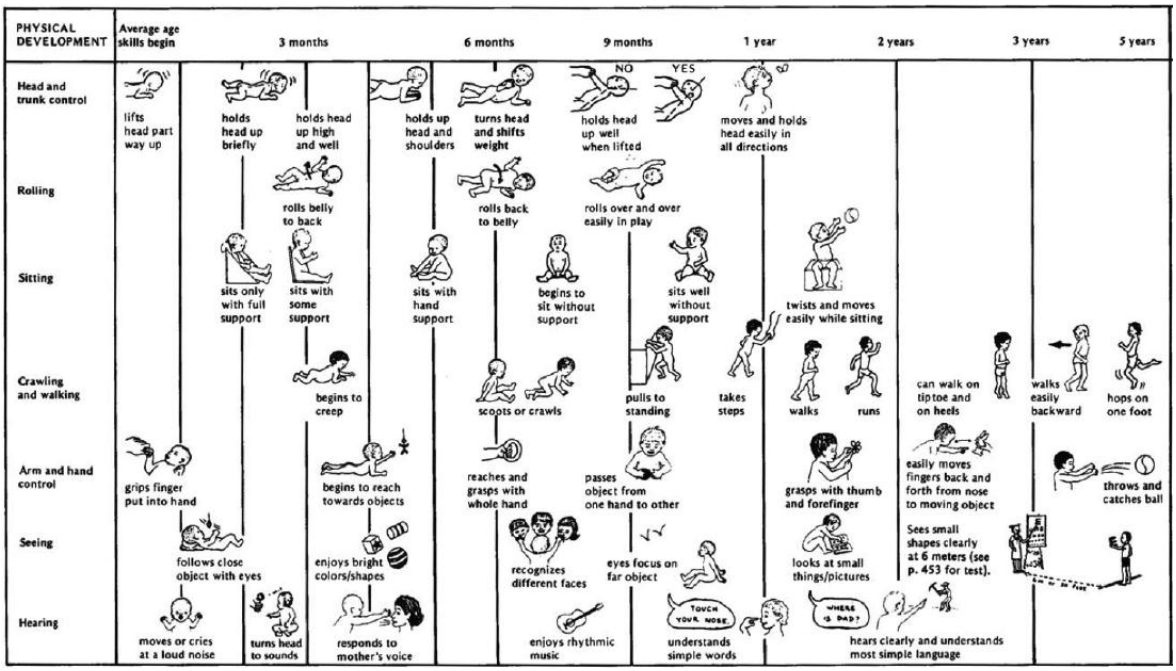


Figure I.1 : Evolutions constatés chez l'enfant dans ses premières années.

Par exemple, quelques grands défis de l'apprentissage développemental et social sont :

- Comment découvrir et apprendre à contrôler un corps inconnu et possiblement changeant ?
- Comment apprendre à utiliser ce corps pour manipuler des objets ?

- Comment apprendre de nouveaux savoir-faire sensorimoteurs au cours d'interactions sociales ?
- Comment apprendre les rudiments du langage ?
- Comment permettre des interactions sociales naturelles et intuitives avec d'autres personnes, et comment ces interactions peuvent-elles changer la nature du problème ?

I.4- Méthodes d'apprentissage :

On distingue trois familles d'apprentissage en fonction de la nature des informations disponibles et du but recherché : l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement.

I.4.1- Apprentissage supervisé :

Dans ce cas, il est nécessaire de disposer d'un ensemble de couples de données {entrées du réseau ; sorties désirées}, appelées base d'exemples ou base de données. **[2]**

La différence entre la sortie calculée par un réseau de neurones et la sortie désirée donne ainsi une mesure d'erreur quantitative sur le calcul effectué par le réseau. Cette erreur est utilisée pour réaliser l'adaptation.

I.4.2- Apprentissage non supervisé :

Contrairement à l'apprentissage supervisé, seules les informations en entrée sont fournies au système. Celui-ci doit donc déterminer ses sorties en fonction des similarités détectées entre les différentes entrées, c'est-à-dire en fonction d'une règle auto - organisatrice. Le système est appelé donc à découvrir les régularités présentes dans ces configurations qui peuvent servir à les diviser en groupes de configurations semblables. **[2]**

I.4.3- Apprentissage par renforcement :

Il existe beaucoup de problèmes où les sorties désirées que l'apprentissage supervisé exige sont difficile à spécifier. On ne dispose souvent que d'une information qualitative permettant l'évaluation de la réponse calculée.

L'apprentissage par renforcement utilise cette évaluation pour améliorer les performances du système. Cette forme d'apprentissage constitue une méthode d'apprentissage par essais et erreurs. **[2]**

I.5-Processus de décision de Markov :

I.5.1-Formalisme :

Un processus de décision de Markov (MDP) est un modèle stochastique qui peut être vu comme une chaîne de Markov contrôlée, c'est à dire à laquelle on ajoute une composante décisionnelle. C'est un outil mathématique qui permet de formaliser l'apprentissage par renforcement. Un MDP vérifie la propriété de Markov qui peut être résumée de la manière suivante : prédire le futur à partir du présent est tout aussi efficace que le prédire en possédant des informations concernant le passé.

Plus formellement la propriété de Markov peut être définie de la manière suivante :

Soient $X_0 \ldots X_n$ des variables aléatoires réelles, nous avons donc

$$\langle S, A, T, R \rangle \; \forall n \geq 0, \forall (i_0, ..., i_{n-1}, i, j) \in E^{n+2}$$

$$P(X_{n+1} = j | X_0 = i_0, ..., X_{n-1} = i_{n-1}, X_n = i) = P(X_{n+1} = j | X_n = i) \quad \text{.....1.1}$$

Un processus de décision de Markov est un quadruplet $\langle S, A, T, R \rangle$ qui peut être défini de la façon suivante :

- $S = s_0, ..., s_{|S|}$ est l'ensemble fini discret des états possibles du système à contrôler.
- $A = a_0, ..., s_{|A|}$ est l'ensemble fini discret des actions que l'on peut effectuer pour contrôler le système.

- $T : S \times A \times S \rightarrow [0; 1]$ est la fonction de transition du système T et donne la probabilité que le système passe d'un état à un autre en ayant choisi une action donnée.
- $R : S \times A \times S \rightarrow$ □ est la fonction de récompense, elle indique la valeur réelle obtenue lorsque l'on effectue l'action a dans l'état s pour arriver dans l'état s'.

Dans le cadre des MDP on appelle politique $\pi : S \rightarrow A$ une fonction qui indique pour chaque état quelle est l'action à effectuer. Il s'agit là d'une politique déterministe où, contrairement à une politique stochastique, il n'y a pas d'ambiguïté sur l'action à effectuer. **[1]**

I.6- Programmation dynamique :

La programmation dynamique est un ensemble de méthodes permettant de calculer une politique optimale dans un MDP connu, en utilisant les propriétés des équations de Bellman. Nous supposons donc dans cette section que le modèle de l'environnement est connu. La programmation dynamique va chercher à estimer la fonction de valeur optimale V^* afin d'en déduire une politique optimale. **[3]**

I.6.1- Évaluation d'une politique :

La première étape de la programmation dynamique est l'estimation de la fonction de valeur pour une politique donnée π. Cela peut se faire soit en résolvant le système des équations de Bellman, soit en utilisant une procédure itérative, que nous préférerons car elle s'applique plus naturellement, notamment en cas de contraintes temps-réel. Cette procédure utilise le fait que la fonction de valeur V^π est le point fixe de l'équation de Bellman :

$$V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')] \quad \text{.....1.2}$$

Nous pouvons ainsi utiliser cette équation comme étape de mise-à-jour permettant de calculer une suite de fonctions V^k qui convergera vers V^π :

$$V^{k+1}(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^k(s')] \text{.....1.3}$$

La méthode de programmation dynamique va donc utiliser cette mise-à-jour tant que les modifications $\max_s(|V^{k+1} - V^k)$ seront supérieures à un seuil donné pour fournir une approximation de V^π.

I.6.2- Amélioration d'une politique :

Après l'évaluation d'une politique, il va être possible de calculer un meilleur politique à partir de sa fonction de valeur associée. En effet, pour une politique donnée, il n'y a aucune raison que la fonction de valeur satisfasse l'équation d'optimalité de Bellman, c'est à dire que l'on peut avoir :

$$\pi(s) \neq \arg\max_a E(r_{t+1} + \gamma V^\pi(s_{t+1}) \| s_t = s, a_t = a) \text{.....1.4}$$

Par contre, on peut prouver que la politique π' définie par :

$$\pi'(s) = \arg\max_a E(r_{t+1} + \gamma V^\pi(s_{t+1}) | s_t = s, a_t = a) \text{.....1.5}$$

est meilleure ou équivalente à la politique π, ce qui nous permet d'améliorer notre politique initiale.

De plus, si la politique π' ainsi définie n'est pas meilleure que π (c'est à dire si $V^{\pi'} = V^\pi$), la définition de π' l'équation (1.5) est l'équation d'optimalité de Bellman, qui prouve donc que la politique obtenue est optimale.

I.6.3- Algorithmes d'apprentissage :

L'évaluation et l'amélioration de politique peuvent être utilisées de différentes manières pour estimer une politique optimale pour un MDP donné.

Le première méthode, l'itération de politique utilise simplement ces deux phases de manière itérative :

$$\pi_0 \to V^{\pi_0} \to \pi_1 \to V^{\pi_1} \ldots \pi^* \to V^{\pi^*} \text{.....1.6}$$

Dans ce processus, cependant, l'évaluation de politique est elle-même un processus itératif, qui ne va converger qu'à une erreur donnée près et de plus être potentiellement très long.

Une autre méthode pour converger vers la politique optimale est d'améliorer la politique avant même d'avoir une estimation correcte de sa valeur. On peut par exemple faire un nombre fixe d'itérations d'évaluation avant de faire une amélioration. Le cas où on ne fait qu'une itération d'évaluation de la politique est l'algorithme d'itération de valeur, pour lequel les deux étapes se réduisent à une seule :

$$V^{k+1}(s) = \max_a \sum_{s'} p^a_{ss'} \left[\Re^a_{ss'} + \gamma V^k(s') \right] \text{.....} \mathbf{1.7}$$

et qui converge vers V^*.

Pour les problèmes avec de très grands espaces d'états, le fait de parcourir tout les états pour la mise-à-jour peut être difficile en soit. Il est dans ce cas possible d'utiliser une méthode de programmation dynamique asynchrone qui réalise la mise-à-jour de l'itération de valeur pour un état sélectionné au hasard, ou en fonction du comportement de l'agent. Cette méthode converge également vers V^* à condition de visiter à la limite tout les états un nombre infini de fois. Elle possède l'avantage de fournir rapidement une approximation de la fonction de valeur.

I.7- Méthodes de Monte-Carlo :

Le fait de connaître l'environnement pour apprendre une stratégie rend les méthodes de programmation dynamique peu utiles en robotique. Les méthodes de Monte-Carlo que nous allons voir dans cette section vont utiliser les mêmes idées (estimer la fonction de valeur puis améliorer la politique), mais en ayant recours à des expériences réalisées dans l'environnement plutôt qu'à un modèle. **[3]**

I.7.1- Évaluation d'un politique :

L'estimation de la fonction de valeur va se réaliser à partir d'un ensemble de séquences "état-action-récompenses-état-action ...." réalisées par l'agent. Pour les états de ces séquences, il est alors possible d'estimer V simplement par la moyenne des revenus :

$$V(s) = \frac{1}{N(s)} \sum \text{Re}venu(s) \text{.....} \mathbf{1.8}$$

Où N(s) est le nombre de séquences ou apparaît s et Revenu(s) est le revenu après la première visite de l'état s, c'est à dire la somme pondérée des récompenses après cette visite.

Cette méthode de Monte-Carlo a de plus l'avantage d'estimer la valeur de chaque état indépendamment, contrairement à la programmation dynamique qui doit estimer simultanément tout les états, ce qui permet par exemple de se concentrer sur des zones de l'espace d'états plus importantes pour l'objectif du robot.

Cette méthode s'applique de la même façon pour une fonction$Q(s,a)$, ce qui est encore plus intéressant, car pour trouver la politique optimale à partir de V^* il faut disposer d'un modèle de l'environnement, ce qui n'est pas le cas en utilisant Q^*. L'utilisation de Q et de la méthode de Monte-Carlo permet donc de découvrir la politique optimale sans aucun modèle de l'environnement, en utilisant uniquement des expériences réalisée dans cet environnement.

I.7.2- Besoin d'exploration :

La méthode de Monte-Carlo présentée doit estimer les valeurs Q(s,a) à partir des récompenses obtenues après avoir réalisé l'action a dans l'état s. Ceci suppose donc que tout ces couples (s,a) soient rencontrés une infinité de fois à la limite. Ceci est particulièrement problématique, car toutes les politiques ne peuvent garantir cette propriété. Il faut donc ajouter au comportement défini par la politique un comportement d'exploration qui va assurer que toutes les actions seront réalisées avec un certaine probabilité (même faible).

Deux solutions existent pour résoudre ce problème. La première consiste à contraindre les politiques pour qu'elles associent au moins une faible probabilité proportionnelle à un paramètre ε à toutes les actions. L'apprentissage converge ainsi vers la politique optimale au sein de cette classe. Cette probabilité garanti une exploration exhaustive et peut être diminuée au cours du temps lorsque les données suffisantes pour évaluer la politique ont été recueillies. Cette méthode s'appelle

contrôle "on policy" car elle modifie la politique effectivement suivie par l'agent et évalue cette politique modifiée.

La seconde méthode est une méthode "Off Policy" car elle évalue une politique tandis que l'agent en suit une autre. Cette autre politique choisit en général l'action de la politique originale avec une probabilité $1-\varepsilon$ et une action aléatoire avec une probabilité ε. Pour évaluer la politique originale, l'évaluation de Monte-Carlo utilise simplement les parties finales des séquences pour lesquelles le choix d'action correspond au choix qui aurait été fait par la politique originale, mais modifie les pondérations des récompenses pour compenser les différences de probabilité de choix des actions entre les deux politiques.

I.7.3- Algorithmes d'apprentissage :

L'apprentissage utilisant une méthode de Monte-Carlo se fait en alternant l'évaluation d'une politique et son amélioration en prenant l'action maximisant le revenu dans chaque état. Cette alternance peut se réaliser de plusieurs manières, comme pour la programmation dynamique. Soit l'évaluation est complète avant de réaliser une amélioration, soit il est possible d'alterner une évaluation utilisant une seule séquence, puis une amélioration.

I.8- Les méthodes de différence temporelle :

Nous allons nous tourner à présent vers les méthodes de différence temporelle, qui combinent l'incrémentation de la programmation dynamique avec le recours à l'expérience des méthodes de Monte-Carlo**.[4]**

I.8.1- L'algorithme TD(0) :

L'algorithme élémentaire d'apprentissage par renforcement, dit algorithme de «différence temporelle », s'appelle TD. Nous le notons ici TD(0) pour des raisons qui apparaîtront quand nous présenterons les traces d'éligibilité. Cet algorithme repose sur une comparaison entre la récompense que l'on reçoit effectivement et la récompense que l'on s'attend à recevoir en fonction des estimations construites précédemment. Si les estimations des fonctions de valeur aux états s_t et s_{t+1}, notées V (s_t) et V (s_{t+1}), étaient exactes, on aurait **[4]** :

$$V(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{1.9}$$

$$V(s_{t+1}) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{1.10}$$

Donc on aurait :

$$V(s_t) = r_t + \gamma V(s_{t+1}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \textbf{1.11}$$

On voit que l'erreur de différence temporelle δ_k mesure l'erreur entre les valeurs effectives des estimations V (s) et les valeurs qu'elles devraient avoir.

La méthode de différence temporelle consiste à corriger peu à peu cette erreur en modifiant la valeur de $V(s_t)$ selon une équation de type Widrow-Hoff, que l'on utilise dans le domaine des réseaux de neurones :

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)] \ldots\ldots\ldots\ldots\ldots\ldots \textbf{1.12}$$

Cette équation de mise à jour permet de comprendre immédiatement comment les algorithmes de différence temporelle combinent les propriétés de la programmation dynamique avec celles des méthodes de Monte-Carlo. En effet, elle fait apparaître les deux caractéristiques suivantes :

- comme dans les algorithmes de programmation dynamique, la valeur estimée de $V(s_t)$ est mise à jour en fonction de la valeur estimée de $V(s_{t+1})$. Il y a donc propagation de la valeur estimée à l'état courant à partir des valeurs estimées des états successeurs.
- comme dans les méthodes de Monte-Carlo, chacune de ces valeurs résulte d'une estimation locale des récompenses immédiates qui repose sur l'expérience accumulée par l'agent au fil de ses interactions avec son environnement.

On voit donc que les méthodes de différence temporelle et, en particulier, TD(0), reposent sur deux processus de convergence couplés, le premier estimant de plus en plus précisément la récompense immédiate reçue dans chacun des états et le second approchant de mieux en mieux la fonction de valeur résultant de ces estimations en les propageant de proche en proche.

Dans le cas de TD(0), les mises à jour se font localement à chaque fois que l'agent réalise une transition dans son environnement, à partir d'une information se limitant à son état courant s_t, l'état successeur s_{t+1} et la récompense r_t reçue suite à cette transition. Une preuve de convergence de l'algorithme a été proposée par Dayan et Sejnowski.

En revanche, il faut noter que, comme TD(0) estime la fonction de valeur de chacun des états d'un problème, faute d'un modèle des transitions entre les états, l'agent est incapable d'en déduire quelle politique suivre, car il ne peut réaliser un pas de regard en avant pour déterminer quelle est l'action qui lui permettra de rejoindre l'état suivant de plus grande valeur. Ce point explique que l'on préfère avoir recours aux algorithmes qui travaillent sur une fonction de valeur associée aux couples état-action plutôt qu'à l'état seul.

I.8.2- L'algorithme Sarsa :

la forme de l'équation de Bellman V = LV n'est pas satisfaisante pour en dériver directement un algorithme adaptatif de résolution. Pour cela, Watkins a introduit la fonction de valeur Q, dont la donnée est équivalente à celle de V lorsque l'on connaît la fonction de transition p. **[4]**

DÉFINITION 1.1. – Fonction de valeur Q

A une politique π fixée de fonction de valeur V^{π}, on associe la nouvelle fonction :

$$\forall s \in S, a \in A \quad Q^{\pi}(s,a) = r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) V^{\pi}(s') \text{.....} \mathbf{1.13}$$

L'interprétation de la valeur $Q^{\pi}(s,a)$ est la suivante : c'est la valeur espérée du critère pour le processus partant de s, exécutant l'action a, puis suivant la politique π par la suite. Il est clair que $V^{\pi}(x) = Q^{\pi}(x, \pi(x))$, et l'équation de Bellman vérifiée par la fonction Q^{*} devient :

$$\forall s \in S, a \in A \quad Q^{*}(s,a) = r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{b} Q^{*}(s',b) \text{...} \mathbf{1.14}$$

On a alors :

$$\forall s \in S \quad V^*(s) = \max_a Q^*(s,a), \text{.....} \mathbf{1.15}$$

$$\pi^*(s) = \arg\max_a Q^*(s,a). \text{.....} \mathbf{1.16}$$

L'algorithme SARSA est similaire à l'algorithme TD(0) à ceci près qu'il travaille sur les valeurs des couples (s,a) plutôt que sur la valeur des états. Son équation de mise à jour est identique à celle de TD(0) en remplaçant la fonction de valeur par la fonction de valeur d'action :

$$Q(s_n,a_n) \leftarrow Q(s_n,a_n) + \alpha\left[r_n + \gamma Q(s_{n+1},a_{n+1}) - Q(s_n,a_n)\right]. \text{.....} \mathbf{1.17}$$

L'information nécessaire pour réaliser une telle mise à jour alors que l'agent réalise une transition est le quintuplât $(s_n, a_n, r_n, s_{n+1}, a_{n+1})$ d'où découle le nom de l'algorithme.

Effectuer ces mises à jour implique que l'agent détermine avec un pas de regard en avant quelle est l'action a_{n+1} qu'il réalisera lors du pas de temps suivant, lorsque l'action a_n dans l'état s_n l'aura conduit dans l'état s_{n+1}.

Il résulte de cette implication une dépendance étroite entre la question de l'apprentissage et la question de la détermination de la politique optimale. Dans un tel cadre, il n'existe qu'une seule politique qui doit prendre en compte à la fois les préoccupations d'exploration et d'exploitation et l'agent est contraint de réaliser cet apprentissage uniquement sur la base de la politique qu'il suit effectivement. On dit d'un algorithme tel que SARSA qu'il est on-Policy. La dépendance que cela induit entre l'exploration et l'apprentissage complique considérablement la mise au point de preuves de convergences pour ces algorithmes, ce qui explique que de telles preuves de convergence soient apparues beaucoup plus tard que pour les algorithmes dits off-policy tels que Q-Learning, que nous allons voir à présent.

I.8.3- L'algorithme Q-Learning :

L'algorithme Q-Learning se présente comme une simplification de l'algorithme SARSA par le fait qu'il n'est plus nécessaire pour l'appliquer de déterminer un pas de temps à l'avance quelle sera l'action réalisée au pas de temps suivant. **[4]**

Son équation de mise à jour est la suivante :

$$Q(s_n,a_n) \leftarrow Q(s_n,a_n) + \alpha\left[r_n + \gamma \max_a Q(s_{n+1},a) - Q(s_n,a_n) \right] \text{.....} \mathbf{1.18}$$

La différence essentielle entre SARSA et Q-Learning se situe au niveau de la définition du terme d'erreur. Le terme $Q(s_{n+1}, a_{n+1})$ apparaissant dans l'équation (1.17) a été remplacé par le terme $\max_a(s_{n+1}, a)$ dans l'équation (1.18). Cela pourrait sembler équivalent si la politique suivie était gloutonne (on aurait alors $a_{n+1} = \text{argmax}_a Q(s_{n+1}, a)$. Toutefois, compte tenu de la nécessité de réaliser un compromis entre exploration et exploitation, ce n'est généralement pas le cas. Il apparaît donc que l'algorithme SARSA effectue les mises à jour en fonction des actions choisies effectivement alors que l'algorithme Q-Learning effectue les mises à jour en fonction des actions optimales mêmes si ce ne sont pas ces actions optimales que l'agent réalise, ce qui est plus simple.

Cette simplicité a fait la réputation de l'algorithme Q-Learning. Il s'agit sans doute de l'algorithme d'apprentissage par renforcement le plus connu et le plus utilisé en raison des preuves formelles de convergence qui ont accompagné sa publication.

Algorithme 1.1 : Le Q-Learning

/* α_n est un taux d'apprentissage */

Initialiser(Q_0)

Pour $n \leftarrow 0$ jusqu'à N_{tot} -1 faire

$s_n \leftarrow$ Choix-Etat

$a_n \leftarrow$ Choix-Action

$(s_n', r_n) \leftarrow Simuler(s_n, a_n)$

{mise à jour de Q_n :}

Début

$Q_{n+1} \leftarrow Q_n$

$\delta_n \leftarrow r_n + \gamma \max_b (s_n', b) - Q_n(s_n, a_n)$

$Q_{n+1}(s_n, a_n) \leftarrow Q_n(s_n, a_n) + \alpha_n(s_n, a_n)\delta_n$

fin

retourner $Q_{N_{tot}}$

Tableau I.1 : L'algorithme *Q-Learning*

Le principe de l'algorithme Q-Learning, défini formellement par l'algorithme 1.1, est de mettre à jour itérativement, à la suite de chaque transition (s_n, a_n, s_{n+1}, r_n), la fonction de valeur courante Q_n pour le couple (s_n, a_n), où s_n représente l'état courant, a_n l'action sélectionnée et réalisée, s_n' l'état résultant et r_n la récompense immédiate. Cette mise à jour se fait sur la base de l'observation des transitions instantanées et de leur récompense associée.

Dans cet algorithme, N_{tot} est un paramètre initial fixant le nombre d'itérations. Le taux d'apprentissage α_n (s, a) est propre à chaque couple (s, a), et décroît vers 0 à chaque passage. La fonction Simuler retourne un nouvel état et la

récompense associée selon la dynamique du système. Le choix de l'état courant et de l'action à exécuter est effectué par les fonctions Choix-Etat et Choix-Action et sera discuté plus loin. La fonction Initialiser revient la plupart du temps à initialiser les composantes de Q_0 à 0, mais il existe des initialisations plus efficaces.

Il est immédiat d'observer que l'algorithme Q-Learning est une formulation stochastique de l'algorithme de value itération pour les MDP En effet, ce dernier peut s'exprimer directement en termes de fonction de valeur d'action:

$$V_{n+1}(s) = \max_{a\in A} \overbrace{\left\{ r(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) V_n(s') \right\}}^{Q_n(s,a)}$$

$$\Rightarrow Q_{n+1}(s,a) = r(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) V_{n+1}(s')$$

$$\Rightarrow Q_{n+1}(s,a) = r(s,a) + \gamma \sum_{s'\in S} p(s'|s,a) \max_{a'\in A} Q_n(s',a'). \text{.....} \mathbf{1.19}$$

Le Q-Learning est alors obtenu en remplaçant le terme $r(s,a) + \sum_{s'} p(s'|s,a) \max_{a'\in A} Q_n(s',a')$

Par son estimateur sans biais le plus simple construit à partir de la transition courante

$r_n + \max_{a'\in A} Q_n(s',a')$ La convergence de cet algorithme est établie (la fonction Q_n converge presque sûrement vers Q*) sous les hypothèses suivantes :

- finitude de S et A.
- chaque couple (s,a) est visité un nombre infini de fois .
- $\sum_n \alpha_n(s,a) = \infty$ et $\sum_n \alpha_n{}^2(s,a) < \infty$.
- $\gamma < 1$ ou si $\gamma = 1$
- pour toute politique, il existe un état absorbant de récompense nulle.

Rappelons que cette convergence presque sûre signifie que $\forall s,a$ la suite $Q_n(s,a)$ converge vers $Q^*(s,a)$ avec une probabilité égale à 1. En pratique, la suite $\alpha_n(s,a)$ est souvent définie comme $\alpha_n(s,a) = \frac{1}{n_{sa}}$

I.9- Les algorithmes TD(λ), Sarsa(λ) et Q(λ) :

Les algorithmes TD(0), SARSA et Q-Learning présentent le défaut de ne mettre à jour qu'une valeur par pas de temps, à savoir la valeur de l'état que l'agent est en train de visiter. Comme il apparaît sur la figure (I.2), cette procédure de mise à jour est particulièrement lente. En effet, pour un agent ne disposant d'aucune information a priori sur la structure de la fonction de valeur, il faut au moins n expériences successives pour que la récompense immédiate reçue dans un état donné soit propagée jusqu'à un état distant du premier de n transitions. En attendant le résultat de cette propagation, tant que toutes les valeurs sont identiquement nulles, le comportement de l'agent est une marche aléatoire. **[4]**

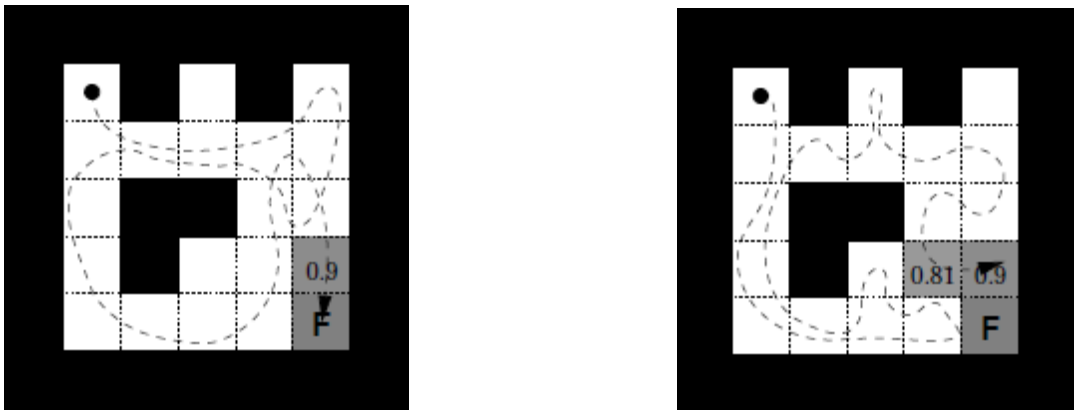


Figure I.2 Navigation d'un agent dans un environnement des pièces par l'utilisation de l'algorithme *Q-Learning.*

Étant initialement nulles, la propagation de valeurs non nulles ne se fait qu'une fois que l'agent a trouvé une première fois la source de récompense et ne progresse que d'un pas à chaque essai de l'agent.

Une façon d'améliorer cet état de fait consiste à doter l'algorithme d'une mémoire des transitions effectuées au cours d'une expérience afin d'effectuer toutes les propagations possibles à la fin de cette expérience. Cette mémoire des transitions effectuées précédemment est appelée une trace d'éligibilité. Ainsi, Sutton et Barto ont proposé une classe d'algorithmes appelés « TD(λ) » qui généralisent l'algorithme TD(0) au cas où l'agent dispose d'une mémoire des transitions. Plus tard, les algorithmes SARSA et Q-Learning ont été généralisés en

SARSA (λ) et Q (λ), le second l'ayant été de deux façons différentes par deux auteurs différents.

Un premier procédé naïf pour accélérer l'apprentissage consiste à stocker directement une liste des couples état-action parcourus par l'agent puis, à chaque fois que l'agent reçoit une récompense, propager celle-ci en parcourant la mémoire des transitions en marche arrière depuis la récompense reçue. Avec un tel procédé, plus la mémoire des transitions est longue, plus une récompense reçue est propagée efficacement. Il apparaît donc un compromis entre la quantité de mémoire mobilisée pour apprendre et la vitesse d'apprentissage. Mais une telle méthode ne fonctionne pas sur un horizon infini.

I.10- Les architectures acteur-critique :

Historiquement, les architectures acteur-critique ont été les premières architectures informatiques imaginées par des chercheurs pour modéliser l'apprentissage par renforcement. **[4]**

Nous avons dit que, dans TD(0), on met à jour la fonction de valeur en se fondant sur l'erreur de différence temporelle $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ mais, si on ne dispose pas d'un modèle de la fonction de transition, on ne sait pas comment exploiter la fonction de valeur pour choisir une action. La solution proposée par les architectures acteur-critique consiste à tenir à jour en parallèle une structure représentant la fonction de valeur, appelée « le critique », et une structure représentant la politique, appelée « l'acteur ».

Le critique est nécessaire pour calculer la valeur de δ_t, qui dépend de la fonction de valeur. Mais le terme δ_t est aussi utilisé pour mettre à jour la politique. Si ce terme est positif, l'action réalisée mérite d'être renforcée. S'il est négatif, au contraire, il faut diminuer la tendance de l'acteur à réaliser cette action.

Cette spécification est très générale, elle impose très peu de contraintes sur les formalismes de représentation respectifs de l'acteur et du critique. La seule contrainte forte est que l'acteur soit capable de prendre en compte le signal d'erreur δ_t pour modifier sa propension à réaliser telle ou telle action dans un contexte

donné. Quant au critique, tout formalisme capable de fournir une approximation de la fonction de valeur fait l'affaire.

Cependant, un grand nombre d'architectures acteur-critique s'appuient sur des réseaux de neurones formels, en raison de la relative plausibilité biologique de cette architecture. Le terme δ_t vient alors modifier le poids des connexions qui gouvernent la propension d'un réseau à réaliser telle ou telle action.

1.11-Conclusion:

Plusieurs techniques sont utilisées pour la résolution du problème d'apprentissage par renforcement et l'acquisition d'un comportement optimal dans un environnement.

Du fait de toutes ces caractéristiques, l'apprentissage par renforcement est une méthode particulièrement adaptée à la robotique.

Chapitre II:

Réseaux de Neurones Artificiels

Réseaux de Neurones Artificiels

II.1-Introduction :

Dans ce chapitre nous allons donner quelque aspects théoriques liaient au réseau de neurones artificiel, à titre d'exemple nous citons le neurone biologique, le neurone formel, fonction d'activation, types de réseaux de neurones artificiels, modèles des réseaux de neurones artificiels et nous présentons l'une des propriétés désirables pour les réseaux de neurones artificiels qui est l'apprentissage et à la fin nous donnons une description pour l'algorithme de rétropropagation.

II.2- Définition d'un réseau de neurones artificiel :

Un réseau de neurones artificiel est un assemblage de cellules interconnectés chacune d'elles appelée neurone. Chaque neurone fonctionne indépendamment par rapport aux autres d'une manière parallèle ; de telle sorte que l'ensemble forme un système qui permet de réaliser des fonctions bien déterminer où des tâches complexes dans des différents types d'applications. On traite les informations qu'il reçoit par l'entraînement du réseau grâce à des méthodes d'apprentissages et non par sa programmation.

II.3-Neurone Biologique :

Le neurone biologique est une cellule vivante spécialisée dans le traitement des signaux électriques. Les neurones sont liés entre eux par des liaisons appelées ***axones*.** Ces axones vont eux-mêmes jouer un rôle important dans le comportement logique de l'ensemble. Ces ***axones*** conduisent les signaux électriques de la sortie d'un neurone vers l'entrée d'un autre neurone où cette entrée est appelée ***synapse***.

Les neurones font une sommation des signaux reçus en entrée et en fonction du résultat obtenu vont fournir un courant en sortie. **[2]**

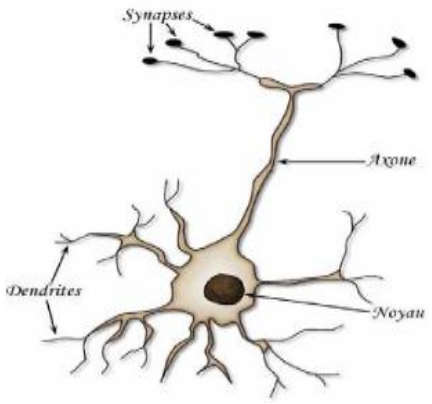


Figure II.1 : Neurone Biologique.

La structure d'un neurone se compose en quatre parties :

- Le corps cellulaire, qui contient le noyau de la cellule nerveuse ; c'est en cet endroit que prend naissance l'influx nerveux qui représente l'état d'activité du neurone.
- Les Dendrites, ramifications tubulaires courtes formant une espèce d'arborescence autour du corps cellulaire ; ce sont les entrées principales du neurone qui captent l'information venant d'autres neurones.
- L'axone, longue fibre nerveuse qui se ramifie à son extrémité ; c'est la sortie du neurone et le support d'information vers les autres neurones.
- La synapse, qui communique l'information, en la pondérant par un poids synaptique, à un autre neurone elle est essentielle dans le fonctionnement du système nerveux**. [2]**

II.4-Neurone formel :

Après avoir vu brièvement le neurone biologique, voyons maintenant son homologue formel.

Le premier neurone formel est apparu en 1943 par les professeurs Mac Culloch et Pitts, leur modèle peut être représenté par la figure suivante :

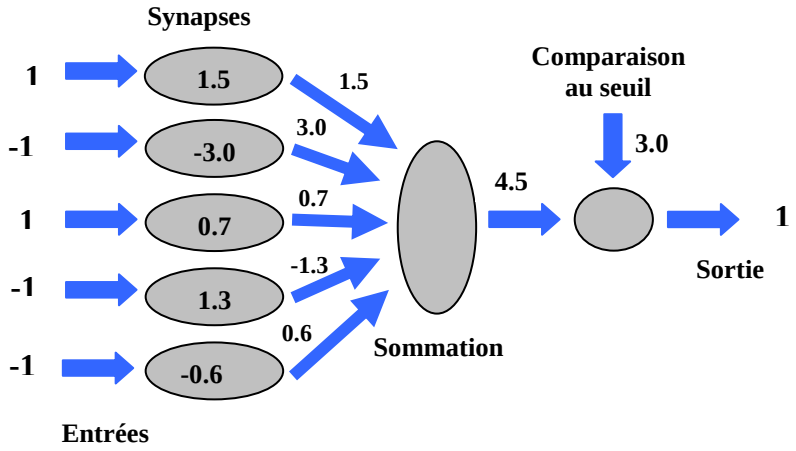


Figure II.2 : Neurone Formel.

Donc, le neurone formel est une modélisation mathématique qui reprend les grands principes du fonctionnement d'un neurone biologique, et particulièrement la sommation des entrées. Cette modélisation est représentée par le tableau suivant qui nous permet d'avoir clairement la transition entre le neurone biologique et le neurone formel.

Neurone Biologique	Neurone Formel
Synapses	Poids de Connexion
Axones	Signal de Sortie
Dendrite	Signal d'Entrée
Corps cellulaire	Fonction d'Activation

Tableau II.1 : Analogie entre le neurone Biologie et le neurone Formel.

II.4.1- Poids de connexion :

Un poids d'un neurone représente l'efficacité d'une connexion.

II.4.2- Les entrées :

Elles peuvent êtres :

- Booléennes.
- Binaires (0 1) ou Bipolaire (-1 1).
- Réelles.

II.4.3- Fonction d'activation :

Cette fonction permet de définir l'état interne du neurone en fonction de ces entrées.

A titre d'exemple nous citons quelques fonctions souvent utilisées :

II.4.3. a- Fonction Seuil :

Fonction Heaviside définie par :

$$H(x)=\begin{cases}1 & si \quad x\geq 0\\ 0 & si \quad non\end{cases} \qquad \text{.....2.1}$$

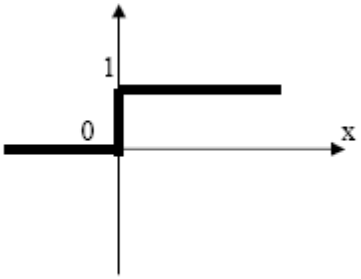


Figure II.3 : Fonction Heaviside.

Fonction Signe définie par :

$$Sgn(x)=\begin{cases}1 & si \quad x\geq 0\\ -1 & \sin on\end{cases} \qquad \text{.....2.2}$$

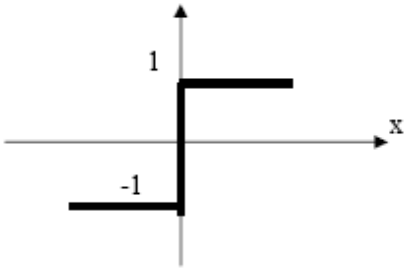


Figure II.4 : Fonction Signe.

Le seuil introduit un non linéarité dans le comportement du neurone, cependant il limite la gamme des réponses possibles à deux valeurs.

II.4.3. b- Fonction linéaire :

C'est l'une des fonctions d'activation les plus simples, elle est définie par :

$$F(x) = x \quad \text{.....2.3}$$

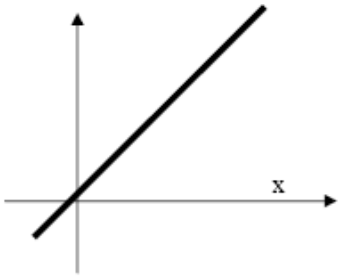


Figure II.5 : Fonction Linéaire.

II.4.3. c- Fonction Linéaire à seuil :

Cette fonction d'activation est définie comme suit :

$$F(x) = \begin{cases} x & si \quad x \in [u \quad v] \\ v & si \quad x \geq v \\ u & si \quad x \leq u \end{cases} \quad \text{.....2.4}$$

Elle représente un compromis entre la fonction linéaire et la fonction seuil, entre ses deux barres de saturation, elle confère au neurone une gamme de réponses possibles. En modulant la pente de la linéarité, on affecte la plage de réponse du neurone.

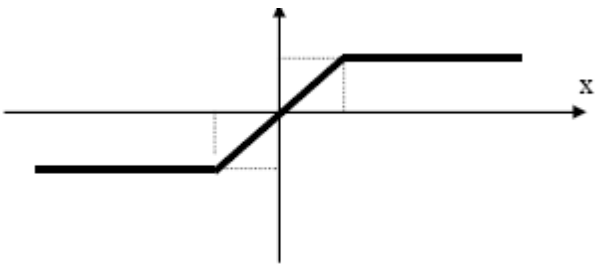


Figure II.6 : Fonction Linéaire à seuil.

II.4.3. d- Fonction sigmoïde :

Elle est l'équivalant continu de la fonction linéaire. Etant continu, elle est dérivable, d'autant plus que sa dérivée est simple à calculer. Cette fonction est définie par :

$$F(x)=\frac{1}{1+e^{-x}} \text{ ……..……2.5}$$

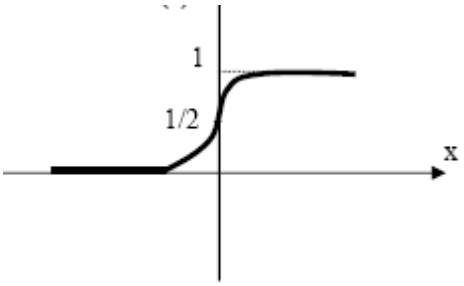


Figure II.7 : Fonction sigmoïde.

II.4.4- Fonction de sortie :

Elle calcule la sortie d'un neurone en fonction de son état d'activation. En général cette fonction est considérée comme la fonction identité. Elle peut être, binaire ou réelle.

II.5- Description mathématique :

On considère le cas général d'un neurone formel à ***m*** entrées, auquel on doit donc soumettre les ***m*** grandeurs numériques ou signaux, notées x_1 à x_m. Un modèle de neurone formel est une règle de calcul qui permet d'associer aux *m* entrées une sortie : c'est donc une fonction à ***m*** variables et à valeurs réelles. Chaque entrée ***m*** est associée un poids synaptique, c'est-à-dire une valeur numérique notée de w_1 pour l'entrée **1** jusqu'à w_m pour l'entrée ***m***. La première opération réalisée par le neurone formel consiste en une somme des grandeurs reçues en entrées, pondérées par les coefficients synaptiques, c'est-à-dire la somme :

$$w_1x_1 + w_2x_2 + ... + w_mx_m = \sum_{j=1}^{m} w_jx_j \text{ …….....2.6}$$

Cette grandeur est comparée à un seuil θ. Le résultat est alors transformé par une fonction d'activation non linéaire ***F***. La sortie associée aux entrées x_1 à x_m est ainsi donnée par :

$$F\left(\sum_{j=1}^{m} w_jx_j - \theta\right) \text{ ……..…..…..…2.7}$$

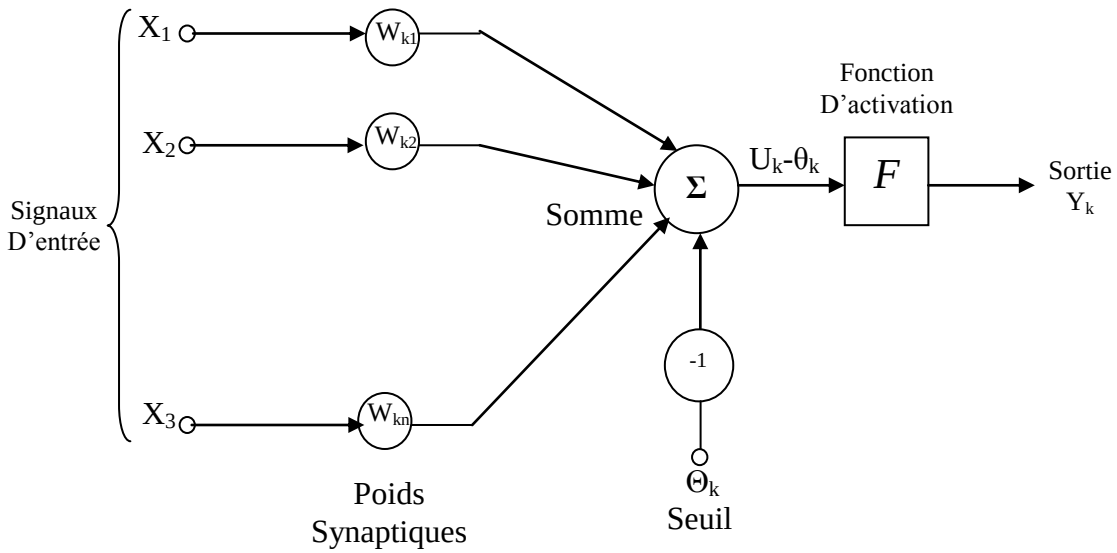


Figure II.8 : Modèle général d'un neurone.

Le neurone formel est l'unité élémentaire des réseaux de neurones artificiels (RNA) dans lesquels il est associé à ses semblables pour calculer des fonctions arbitrairement complexes, utilisées pour diverses applications.

II.6-Réseaux de neurones artificiels :

Un réseau de neurone artificiel est un ensemble de neurone formels associer en couches ou sous-groupes et fonctionnent en parallèle. Dans un réseau chaque sous-groupe fait un traitement indépendant des autres et transmet le résultat de son analyse au sous-groupe suivant. L'information donnée au réseau va se propager couche par couche, de la ***couche d'entrée*** à la ***couche de sortie***, en passant soit aucune, une ou plusieurs couches intermédiaires dites ***couches cachées***. Il est à noter qu'en fonction de l'algorithme d'apprentissage, il est possible d'avoir une propagation de l'information à reculons (Back propagation). Généralement chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et la couche suivante « sauf les couches d'entrée et de sortie ». **[2]**

II.7-Différentes types de réseaux de neurones artificiels :

Nous entendrons par architecture ou topologie d'un réseau de neurones artificiels la manière selon laquelle les neurones sont organisés. Les structures qui peuvent être utilisées sont très variées mais beaucoup moins complexes que celles des réseaux de neurones biologiques.

D'une façon générale, l'architecture des réseaux de neurones artificiels peut aller d'une connectivité totale (chacun des neurones du réseau est relié à tous les autres neurones) à une connectivité locale où les neurones ne sont liés qu'a leurs plus proches voisins. **[2]**

Deux classes différentes d'architectures de réseaux de neurones peuvent être distinguées :

1- Les réseaux proactifs (feed forward).

2- Les réseaux récurrents (feedback).

II.7.1- Les réseaux proactifs :

Cette classe se distingue par l'absence de toute boucle de rétroaction de la sortie vers l'entrée, d'où l'appellation « feed-forward ». En d'autres termes, la propagation des signaux s'y fait uniquement dans le sens de l'entrée vers la sortie.

Ce type de réseaux comprend deux groupes d'architectures : les réseaux monocouches et les réseaux multicouches. Il différent par l'existence ou non des neurones intermédiaires appelés neurones cachées entres les unités d'entrées et les unités de sorties appelées aussi nœuds d'entrées et nœuds de sorties respectivement. **[2]**

II.7.1. a- Les réseaux proactifs monocouches :

Ce type de réseaux possède une couche d'entrée recevant les stimuli à traiter par l'intermédiaire des nœuds d'entrées. Cette couche se projette en une couche de sortie composée de neurones de calcul transmettant les résultats du traitement au milieu extérieur.

La figure (II.9) montre par exemple, un réseau proactif monocouche à quatre neurones d'entrée et trois neurones de sortie. La désignation monocouche est attribuée à la couche de sortie. La couche d'entrée n'est pas comptée dans ce sens vu qu'il n'y a pas de calcul qui se fait au niveau de ces nœuds, ils servent uniquement à recevoir les signaux d'entée et à les transmettre à la couche suivante. Un exemple classique de réseau monocouche est le perceptron. **[2]**

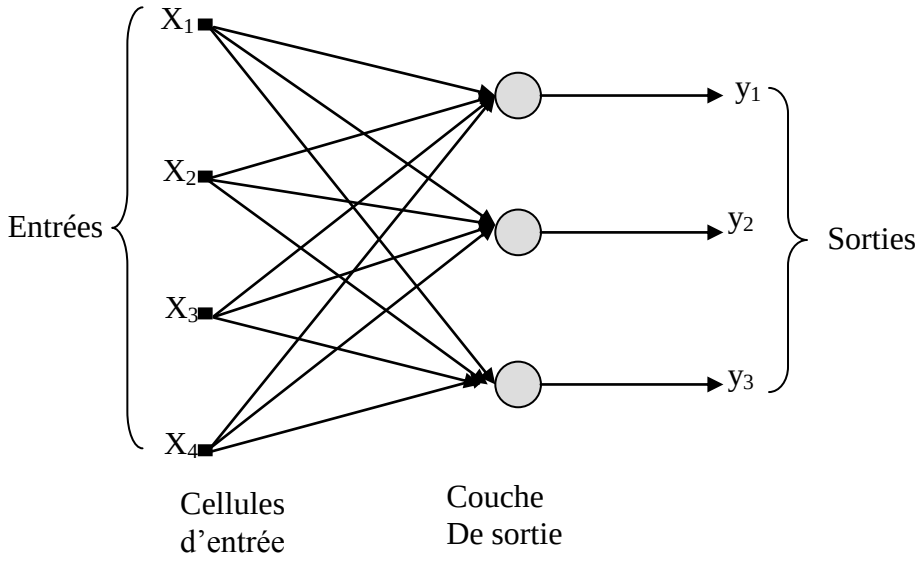


Figure II.9 : Réseau proactif monocouche (Perceptron).

II.7.1. b- Réseaux proactifs multicouches :

Ce type de réseaux proactifs se caractérise par la présence d'une ou plusieurs couches cachées, où les nœuds de calcul correspondants s'appellent neurones cachés. Les couches cachées s'interposent entre l'entrée du réseau et sa sortie. Leur rôle est d'effectuer un prétraitement des signaux d'entrées, reçu par la couche d'entrée en provenance du milieu extérieur, et de transmettre les résultats correspondants à la couche de sortie où sera déterminée la réponse finale du réseau avant quelle soit transmise au milieu extérieur.

Ce rôle de prétraitement fait que, en ajoutant une ou plusieurs couches cachées, le réseau est capable d'extraire plus de propriétés statistiques que celles extraites d'un réseau similaire ayant moins de couches cachées. Ceci est utile pour réaliser des fonctions plus complexes que de simples séparations linéaires.

Dans ce type de réseaux, les entrées des neurones d'une couche particulière proviennent uniquement des sorties de la couche adjacente précédente. Les réseaux les plus fréquemment utilisés de cette catégorie sont les perceptron multicouches (PMC).**[2]**

La figure (II.10) Montre un réseau à une seule couche cachée qui contient quatre neurones, huit entrées et une sortie à trois neurones (réseau 8-4-3). Ce réseau est dit complètement connecté où tous les neurones de la couche actuelle sont connectés à tous les neurones de la couche adjacente suivante. Le réseau et partiellement connecté si des connexions manquaient entre des neurones de deux couches voisines.

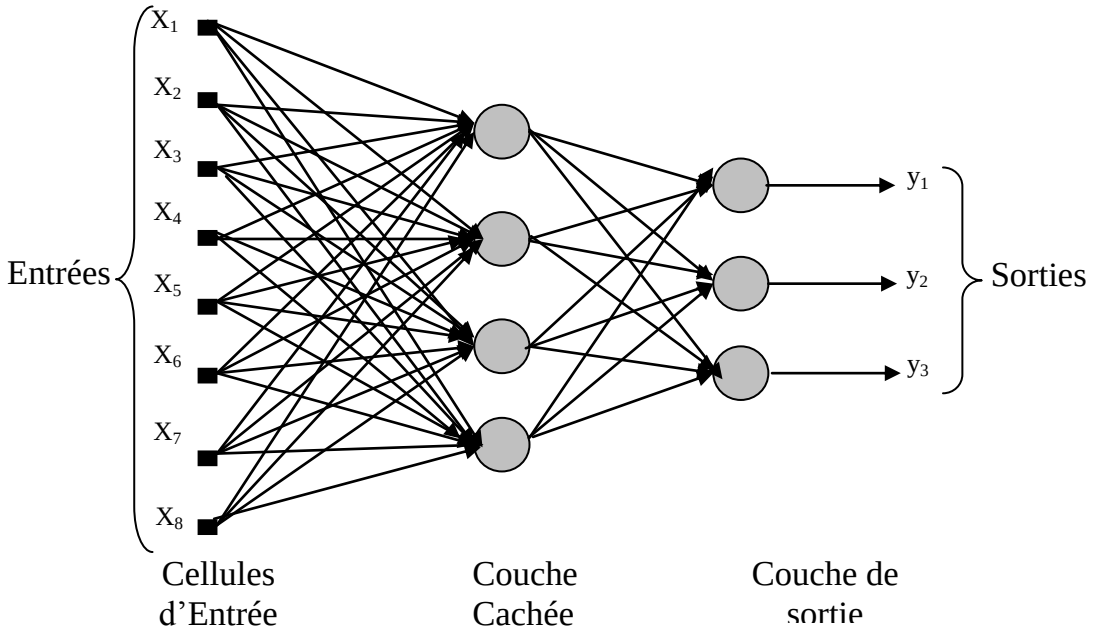


Figure II.10 : Réseau proactif complètement connecté avec une seule couche cachée.

II.7.2- Réseaux récurrents :

Les réseaux récurrents se distinguent des réseaux proactifs par le fait qu'ils contiennent au moins une boucle de contre-réaction des neurones de sortie vers les entrées ou au mois d'une couche vers une couche précédente, adjacente ou non. **[2]**

La figure (II.11) Présente un exemple d'un réseau de neurone récurrent qui à deux entrés, trois neurones dans la couche cachée et deux neurones dans la couche de sortie. Les connexions de rétroaction de ce réseau proviennent aussi bien des neurones de la couche cachée que des neurones de la couche de sortie. **[5]**

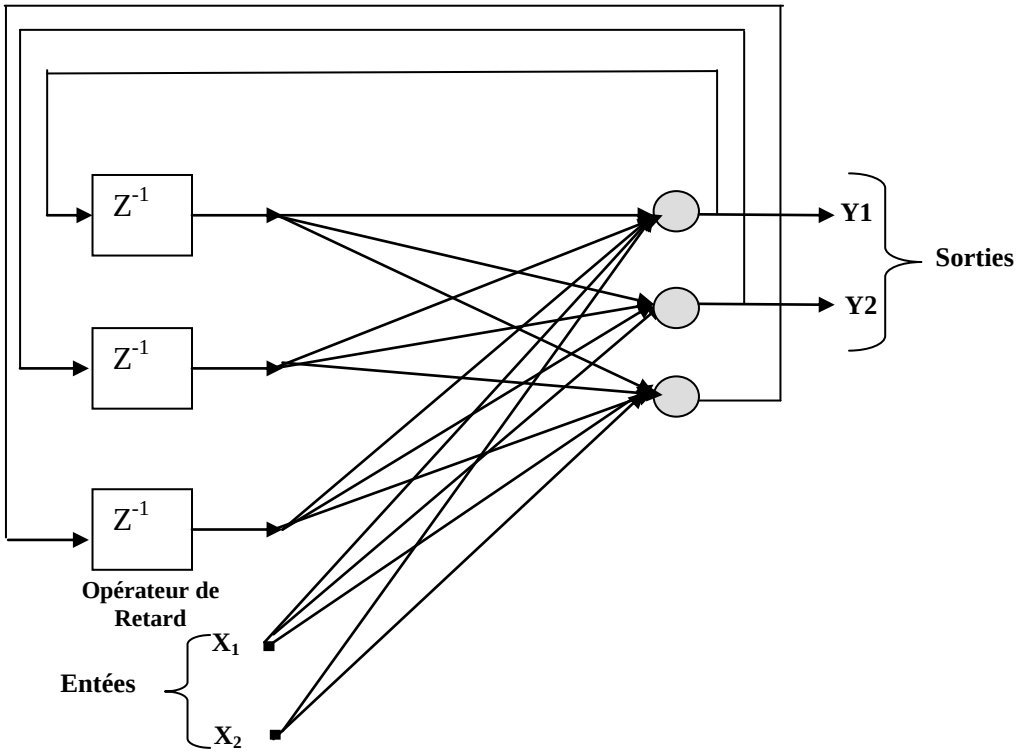


Figure II.11 : Réseau récurent avec neurones cachés.

II.8- Quelques Modèles des Réseaux de Neurones Artificiels :

Il existe plusieurs modèles de réseaux de neurones artificiels dans ce paragraphe en va cité quelques un à titre d'exemple.

II.8.1- Les cartes auto organisatrices de Kohonen :

Ce sont des réseaux à apprentissage non supervisé qui établissent une carte discrète, ordonnée topologiquement, en fonction des entrées. Le réseau forme ainsi une sorte de treillis dont chaque nœud est un neurone associé à un vecteur de poids. La correspondance entre chaque vecteur de poids est calculée pour chaque entrée. Par la suite, le vecteur de poids ayant la meilleure corrélation, ainsi que certains de ses voisins, vont être modifiés afin d'augmenter encore cette corrélation. **[5]**

II.8.2- Les réseaux de Hopfield :

Les réseaux de Hopfield sont des réseaux récurrents et entièrement connectés qui est présenté en 1982. Dans ce type de réseau, chaque neurone est connecté à chaque autre neurone et il n'y a aucune différenciation entre les neurones d'entrée et de sortie. Ce modèle de Hopfield utilise l'architecture des réseaux entièrement connectés et récurrents, les sorties sont en fonction des entrées et du dernier état pris par le réseau. Ils fonctionnent comme une mémoire associative non linéaire et sont capables de trouver un objet stocké en fonction de représentations partielles ou bruitées. L'application principale des réseaux de Hopfield est l'entrepôt de connaissances mais aussi la résolution de problèmes d'optimisation. Le mode d'apprentissage utilisé ici est le mode non supervisé. **[5]**

II.8.3- Les réseaux ART :

Les réseaux ART ("Adaptative Resonance Theory") sont développés par Carpenter et Gross Berg en 1987, ce sont des réseaux à apprentissage par compétition. Le problème majeur qui se pose dans ce type de réseaux est le dilemme « stabilité/plasticité ». En effet, dans un apprentissage par compétition, rien ne garantit que les catégories formées vont rester stables. La seule possibilité, pour assurer la stabilité, serait que le coefficient d'apprentissage tende vers zéro, mais le réseau perdrait alors sa plasticité. Les ART ont été conçus spécifiquement pour contourner ce problème. Dans ce genre de réseau, les vecteurs de poids ne seront adaptés que si l'entrée fournie est suffisamment proche, d'un prototype déjà connu par le réseau. On parlera alors de résonance. A l'inverse, si l'entrée s'éloigne trop des prototypes existants, une nouvelle catégorie va alors se créer, avec pour prototype, l'entrée qui a engendrée sa création. Il est à noter qu'il existe deux principaux types de réseaux ART : les ART-1 pour des entrées binaires et les ART-2 pour des entrées continues. Le mode d'apprentissage des ART peut être supervisé ou non. **[5]**

II.8.4- Le modèle Adaline :

C’est un réseau présenté par Windrow et Hoff, ce réseau contient trois couches, une d’entrée, une couche cachée et une couche de sortie. Ce modèle est similaire au modèle de perceptron seulement la fonction d’activation change, mais reste toujours linaire. Ce réseau est utilisé généralement pour la classification c.à.d la séparation linéaire entre les données qui seront présenté par classe.

II.8.5- Le perceptron monocouche :

Historiquement c’est le premier RNA, c'est le Perceptron de Rosenblatt. C'est un réseau simple, puisque il ne se compose que d'une couche d'entrée et d'une couche de sortie. Cependant, il peut aussi être utilisé pour faire de la classification et pour résoudre des opérations logiques simples (telle "ET" ou "OU"). Sa principale limite est qu'il ne peut résoudre que des problèmes linéairement séparables.

II.8.6-Les perceptrons multicouches :

Les perceptron multicouches (Multi Layer Perceptron) appartient aux réseaux multicouches, donc ils ne possèdent pas de boucle de retour, ils sont « Freed-forward ». Ce sont une extension du perceptron monocouche, avec une ou plusieurs couches cachées entre l'entrée et la sortie. Chaque neurone dans une couche est connecté à tous les neurones de la couche précédente et de la couche suivante (sauf pour les couches d'entrée et de sortie) et il n'y a pas de connexions entre les cellules d'une même couche. Les fonctions d'activation utilisées dans ce type de réseaux sont principalement les fonctions à seuil ou sigmoïdes. Il peut résoudre des problèmes non linéairement séparables et des problèmes logiques plus compliqués, et notamment le fameux problème du XOR.

II.9-Apprentissage :

L’une des propriétés désirables pour les réseaux de neurones artificiels, et la plus fondamentale est sûrement sa capacité d’apprendre de son environnement. Mais qu’est-ce que l’apprentissage ? Malheureusement, il n’existe pas de définition

générale, universellement acceptée, car ce concept touche à trop de notations distinctes qui dépendent du point de vue l'on adopte. **[6]**

Pour les réseaux de neurones artificiels on peut donner la définition suivante :

L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré. **[7]**

Dans la plupart des architectures des réseaux de neurones artificiels, l'apprentissage se traduit par un changement de valeurs des poids qui relient les neurones d'une couche à l'autre. Soit le poids w_{ij} reliant le neurone **i** à son entrée **j**. au temps **t**, un changement $\Delta w_{ij}(t)$ de poids peut s'exprimer par l'équation suivante :

$$\Delta w_{ij}(t) = w_{ij}(t+1) - w_{ij}(t) \quad \textbf{2.8}$$

Par conséquent :

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \quad \textbf{2.9}$$

Où :

$w_{ij}(t)$: représente l'ancienne valeur de poids w_{ij}.

$w_{ij}(t+1)$: représente la nouvelle valeur de poids w_{ij}.

Il existe un ensemble de règles bien définies qui permet de réaliser l'adaptation des poids constitue des algorithmes d'apprentissage du réseau. A titre d'exemple on peut citer : apprentissage par la règle de Hebb, apprentissage par la correction d'erreur.

II.10-Algorithme de rétro propagation du gradient:

II.10.1- Principe :

La rétropropagation du gradient consiste à propager « à l'envers » (de la couche de sortie vers la couche d'entrée) l'erreur obtenue sur les exemples de la base d'apprentissage. On utilise pour cela l'erreur quadratique « le carré de la différence entre ce qu'on obtient et ce qu'on désire ».

Si on calcule la dérivée partielle de l'erreur quadratique par rapport aux poids des connexions (gradient), il est possible de déterminer la contribution des poids à l'erreur générale, et de corriger ces poids de manière à se rapprocher du résultat souhaité. La correction par itération en corrige plus ou moins fortement les poids par l'intermédiaire d'un coefficient η . **[8]**

Après un certain nombre d'itérations, où on n'est satisfait du classement des exemples de notre base d'apprentissage, on fixe les poids qui constituent aussi des frontières entre les classes.

II.10.2- Algorithme :

II.10.2. a- Définition du réseau :

Soit un réseau multicouche définit par:

- Une couche d'entrée à m cellules d'entrées. Ces cellules ne sont pas des neurones mais simplement des entrées $x_i = e_i$ du réseau.
- Une couche cachée qui contient n neurones qui ont une fonction d'activation y_j
- Une couche de sortie à p neurones qui ont une fonction d'activation z_k .
- $n \times m$ connexions entre la couche d'entrée et la couche cachée, chacune pondérée par v_{ji} .
- $m \times p$ connexions entre la couche cachée et la couche de sortie, chacune pondérée par w_{kj} .

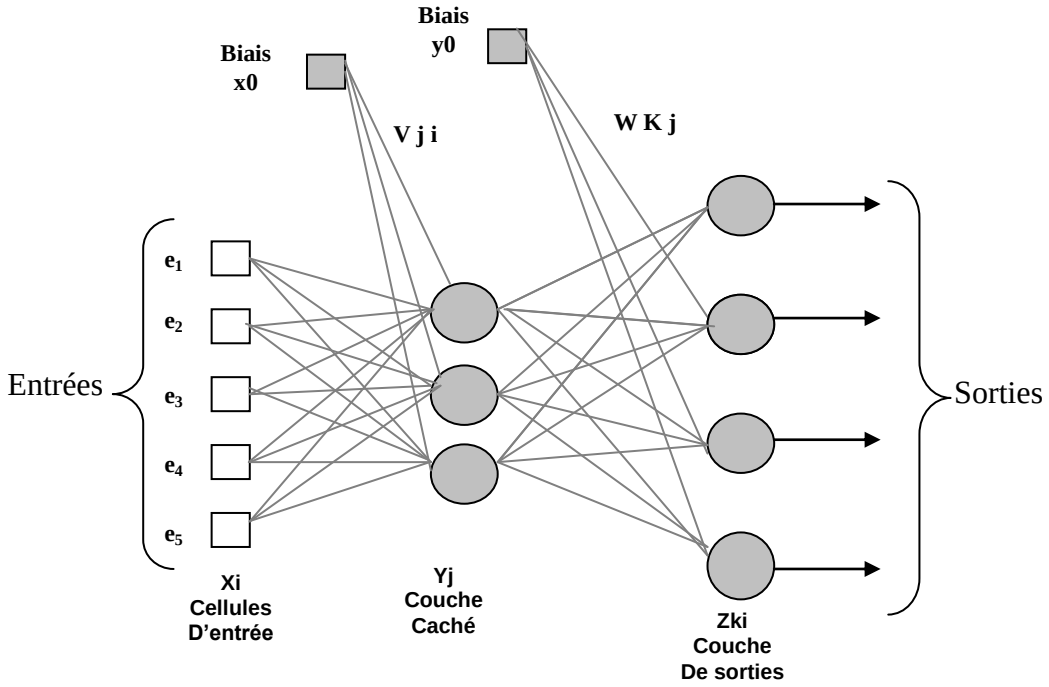


Figure II.12 : Exemple de réseau MLP à une couche cachée avec 5 entrées, 3 neurones dans la couche cachée, et quatre 4 sorties.

II.10.2.b- Algorithme de Rétropropagation de l'erreur :

1. ***Initialisation au hasard ou aléatoire des poids des connexions entre les couches (Entrée - Cachée) et (Cachée - Sortie).***
2. ***Propagation des entrées*** $x_i = e_i$ ***de la couche d'entrée vers la couche de sortie en passant par la couche cachée.***
3. ***rétropropagation de l'erreur des neurones de la couche de sortie vers la couche cachée puis vers la couche d'entrée.***
4. ***Correction des poids des connexions entres la couche d'entrée et la couche cachée ; la couche cachée et la couche de sortie.***
5. ***Boucler à 2 jusqu'à un critère d'arrêt à définir.***

Tableau II.2 : Algorithme Rétropropagation de l'erreur.

II.11- Application des réseaux de neurones :

Les réseaux de neurones servent dans aujourd'hui à toutes sortes d'applications dans divers domaines. On peut citer par exemples :

- Autopilotage des avions.
- Système de guidage des automobiles.
- Lecture automatique des chèques bancaires et d'adresses postales.
- Production des systèmes de traitement signal et pour la synthèse de la parole.
- Les réseaux de neurones ils sont utilisés aussi pour les systèmes de vision par ordinateur.
- Ils sont utilisés en robotique et en télécommunication.
- Ils sont aussi utilisés dans les domaines de finance.
- Ils sont utilisés pour les diagnostiques médical.

II.12- Conclusion :

Dans ce chapitre nous avons donné une brève description sur les réseaux de neurones artificiels, et aussi les différents types d'architectures et de modèle qui existent. Nous avons aussi présenté une définition de l'apprentissage des réseaux de neurones, puis une description générale de l'algorithme de rétropropagation, et aussi les domaines d'utilisation des réseaux de neurones.

Chapitre III:

Robotique Mobile et Conception d'un système d'apprentissage par renforcement

Robotique Mobile et Conception d'un système d'apprentissage par renforcement

III.1-Introduction :

Ce chapitre est constitué deux parties principales, dans la première partie on va essayer de donner des généralités sur la robotique mobile (type des robots mobiles, les capteurs) et à la fin des exemples sur les applications dans la vie.

Dans la deuxième partie on va donner une description sur le système d'apprentissage par renforcement et ces fonctions et aussi les facteurs influent sur ce système.

III.2- Généralités sur la robotique mobile :

III.2.1-Définition du Robot Mobile :

Un robot mobile est une machine équipée de capacités de perception, de décision et d'action qui lui permettent d'agir de manière autonome dans son environnement en fonction de la perception.

Nous présentons rapidement les grandes classes de robots mobiles et leurs modèles

III.2.2- Robots mobiles de type unicycle :

Le modèle qui nous concerne dans notre mémoire est bien ce type de modèle.

III.2.2.1- Description :

On désigne par unicycle un robot actionné par deux roues indépendantes et possédant éventuellement un certain nombre de roues folles assurant sa stabilité. Le schéma des robots de type uni cycle est donné à Figure (III.1)

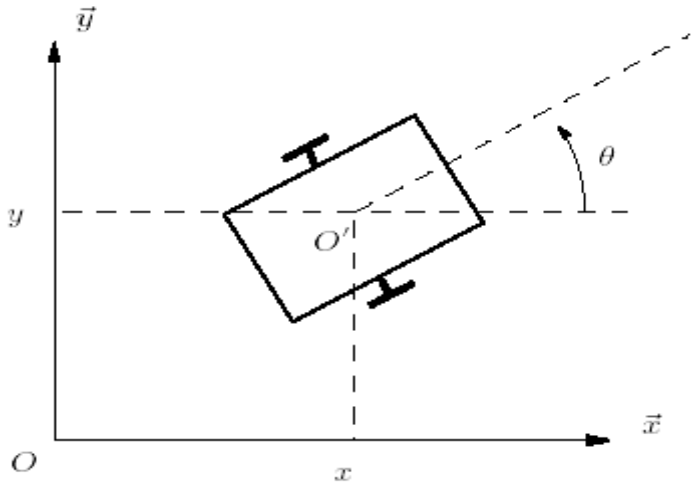
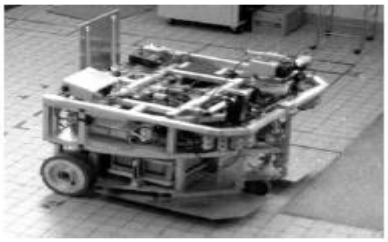


Figure III.1 : Repérage d'un robot mobile.

Ce type de robot est très répandu en raison de sa simplicité de construction et de propriétés cinématiques intéressantes.

La figure (III.2) présente différents robots de type unicycle, depuis Hilare, en 1977, jusqu'aux modèles actuels, qui, à l'instar du robot Khepera,



Hilare, LAAS-CNRS, Toulouse, 1977

Khepera II, K-team, EPFL, Lausanne, 2002

Figure III.2: Evolution des robots mobiles de type uni cycle.

III.2.3- Robots mobiles omnidirectionnels :

Un robot mobile est dit omnidirectionnel si l'on peut agir indépendamment sur les vitesses : vitesse de translation selon les axes x et y et vitesse de rotation autour de z.

D'un point de vue cinématique on montre que cela n'est pas possible avec des roues fixes ou des roues centrées orientables. **[9]**

On peut en revanche réaliser un robot omnidirectionnel en ayant recours à un ensemble de trois roues décentrées orientables ou de trois roues suédoises disposées aux sommets d'un triangle équilatéral Figure (III.3)

Du point de vue de la transmission du mouvement, ceci ne va pas sans poser de problème.

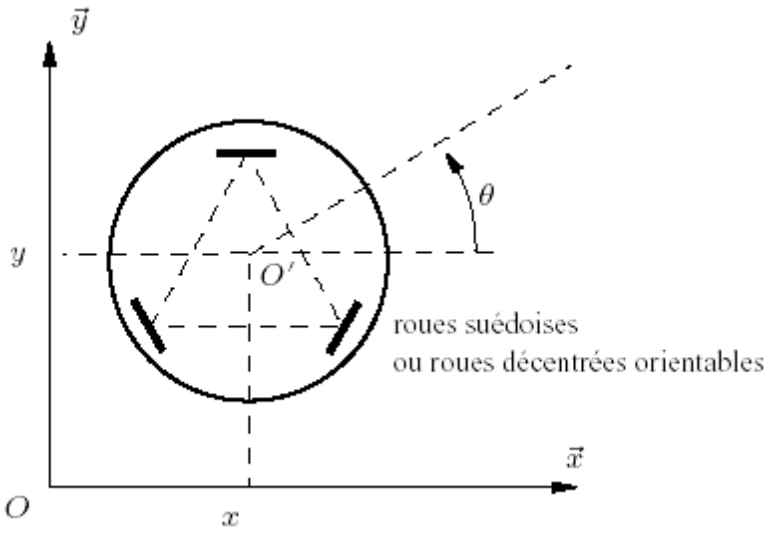


Figure III.3 : Représentation d'un robot mobile omnidirectionnel.

Figure III.4: Robot mobile omnidirectionnel Nomadic XR4000.

III.2.4 -Robots mobiles de type tricycle et de type voiture :

Considérons tout d'abord le cas du tricycle, représenté à Figure (III.5), ce robot est constitué de deux roues fixes de même axe et d'une roue centrée orientable placée sur l'axe longitudinal du robot.

Le mouvement est conféré au robot par deux actions : la vitesse longitudinale et l'orientation de la roue orientable.

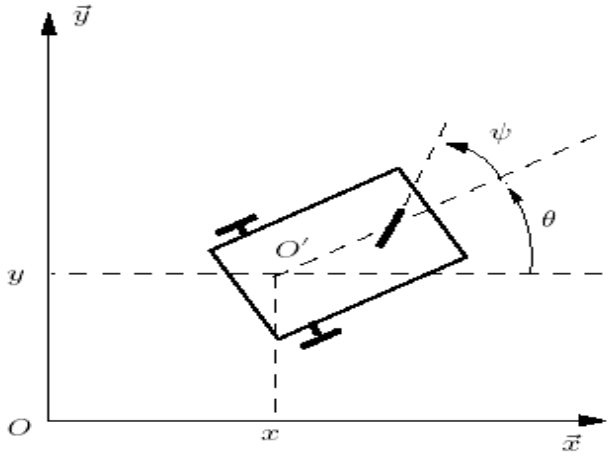


Figure III.5 : Robot mobile de type tricycle.

φ : angle de braquage.

De ce point de vue, il est donc très proche d'une voiture, et l'intérêt pratique de ce type de robot voir une augmentation concrète dans les laboratoires de recherche en robotique ces dernières vingtièmes années.

Le cas du robot de type voiture est très similaire à celui du tricycle, la différence se situe au niveau du train avant, qui comporte deux roues au lieu d'une.

Cela va de soit, on rencontre beaucoup plus souvent ce type de systèmes, on parle de robot dès lors que la voiture considérée est autonome **[10]**, donc sans chauffeur ni télé pilotage.

Il s'agit là d'un des grands défis issus de la robotique mobile.

Deux réalisations sont montrées à la Figure (III.6), basées sur des voitures de série instrumentées.

Figure III.6: Projets de voitures autonomes à l'université de Carnegie Mellon.

III.3- Quelque Capteurs en robotique mobile :

Le choix des capteurs dépend bien évidemment de l'application envisagée, et pour se focaliser sur le problème de navigation.

En robotique mobile, on classe traditionnellement les capteurs en deux catégories selon qu'ils mesurent l'état du robot lui-même ou l'état de son environnement.

L'étude détaillée des capteurs, qui relève à la fois de la physique, de l'électronique et du traitement du signal, ne sera pas vue ici.

Nous nous contenterons d'expliquer les type des capteurs utilises dans le domaine robotique.

III.3.1-Capteurs infrarouges :

Le modèle qui nous concerne dans notre mémoire est bien ce type de modèle.

III.3.1.1-Description :

Les capteurs infrarouges sont constitués d'un ensemble émetteur/récepteur fonctionnant avec des radiations non visibles, dont la longueur d'onde est juste inférieure à celle du rouge visible.

La mesure des radiations infrarouges étant limitée et, en tout état de cause, la qualité très dégradée au-delà d'un mètre, ces dispositifs ne servent que rarement de télémètres.

On les rencontrera le plus souvent comme détecteurs de proximité, ou dans un mode encore plus dégrade de présence.

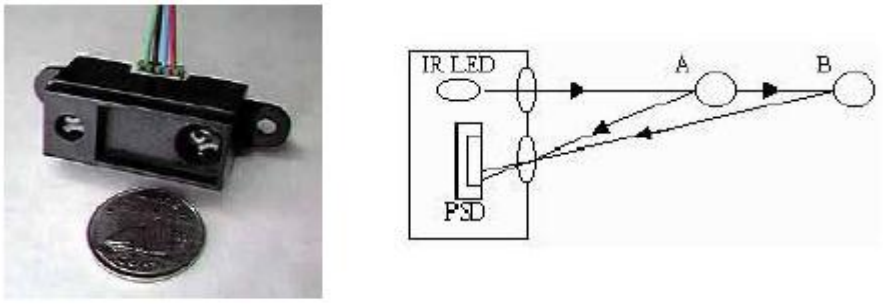


Figure III.7: Télémètres infrarouges Sharp.

III.3.2-Capteurs ultrasonores :

Les capteurs ultrasonores utilisent des vibrations sonores dont les fréquences ne sont pas perceptibles par l'oreille humaine.

Les fréquences couramment utilisées dans ce type de technologie vont de 20 kHz à 200 kHz.

Les ultrasons émis se propagent dans l'air et sont réfléchis partiellement lorsqu'ils heurtent un corps solide, en fonction de son impédance acoustique.

L'écho en retour prend la forme d'une onde de pression à l'image des vaguelettes circulaires déformant la surface de l'eau lorsqu'on y jette une pierre.

La distance entre la source et la cible peut être déterminée en mesurant le temps de vol séparent l'émission des ultrasons du retour de l'écho.

Figure III.8: Télémètres ultrasonores Polaroïd USP 3 et Migatron RPS 409 IS.

III.3.3- Télémètres laser :

Les télémètres laser sont à ce jour le moyen le plus répandu en robotique mobile pour obtenir des mesures précises de distance.

Leur principe de fonctionnement est le suivant :

A un instant donné, une impulsion lumineuse très courte est envoyée par l'intermédiaire d'une diode laser de faible puissance, la réflexion de cette onde donne un écho qui est détecté au bout d'un temps proportionnel à la distance capteur - obstacle.

La direction des impulsions est modifiée par rotation d'un miroir, l'angle de balayage couvrant généralement entre 100 et 180 degrés sur des produits commerciaux. **[11]**

Figure III.9: La famille de télémètre laser Sick.

III.4-Exemples d'applications :

Aujourd'hui, le marché commercial de la robotique mobile est toujours relativement restreint en dehors des robots aspirateurs vendus à plusieurs millions d'exemplaires. Cependant, il existe de nombreuses perspectives de développement qui en feront probablement un domaine important dans le futur. Les applications des robots peuvent se trouver dans de nombreuses activités "ennuyeuses, salissantes ou dangereuses" (3 D's en anglais pour Dull, Dirty, Dangerous), mais également pour des applications ludiques ou de service, comme l'assistance aux personnes âgées ou handicapées.

Figure III.10– Exemples de robots commerciaux ou de recherche.

Parmi les domaines d'applications possibles de la robotique, citons :

- La robotique de service (hôpital, bureaux, maison),
- La robotique de loisir (jouets, robot 'compagnon'),
- La robotique industrielle ou agricole (entrepôts logistiques, récolte de productions agricoles, mines),
- La robotique en environnement dangereux (spatial, industriel, militaire, catastrophes naturelles).

III.5-Description des fonctions du système d'apprentissage par renforcement:

III.5.1-Principe :

Le modèle standard du système d'apprentissage par renforcement est généralement basé sur des interactions entre l'agent et l'environnement. Depuis une situation réelle « s » dans l'environnement, l'agent choisit et exécute une action « a » qui provoque une transition vers l'état « s' ». Il reçoit en retour un signal de renforcement « r » négatif de type pénalité si l'action conduit à un échec ou positif

de type récompense s'il l'action est bénéfique ; un signal nul signifie une incapacité à attribuer une pénalité ou une récompense.

L'agent utilise alors ce signal pour améliorer sa stratégie, c'est à dire la séquence de ses actions, afin de maximiser le cumul de ses récompenses futures. Dans ce but, il doit trouver un équilibre entre exploration et exploitation. L'exploration consiste à tester de nouvelles actions, pouvant conduire à des gains supérieurs, mais avec le risque qu'ils soient inférieurs tandis que l'exploitation consiste à appliquer la meilleure stratégie acquise jusqu'alors (celle-ci pouvant ne pas être optimale).

L'interaction entre l'agent et l'environnement est représentée par le diagramme de la Figure (III.11):

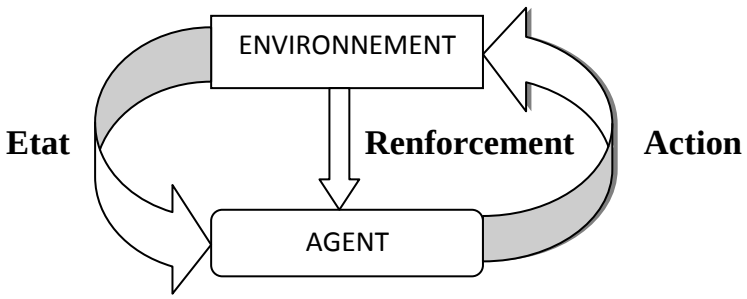


Figure III.11 : Apprentissage par renforcement: diagramme d'interaction agent / environnement.

III.5.2-Facteurs d'influence sur un système d'apprentissage par renforcement :

L'apprentissage par renforcement est basé sur des éléments qui ont une grande influence ces élément sont : **[2]**

- **Le temps**

L'espace de temps a des formes différentes, il peut être :

- Discret ou continu.
- Fini ou infini.
- Déterminé ou aléatoire.

- **Les états**

Les états caractérisent les situations d'un agent et de l'environnement à chaque instant, ils peuvent se décomposer en trois formes :

  - Une situation relationnelle d'agent par rapport à l'environnement (position, etc.).
  - Une situation propre à l'environnement (modifications du milieu).
  - Une situation interne à l'agent (sa mémoire, ses capteurs, etc.).

Les trois formes d'état peuvent être présentes en même temps en fonction du problème traité.

- **Les actions**

Un agent choisit une action parmi les actions possibles à chaque instant t, cette action peut être instantanée ou durer jusqu'au prochain instant. A chaque état de l'espace d'état est associé un ensemble d'actions possibles de l'espace d'action, cette relation est représentée sur la Figure (III.12)**.**

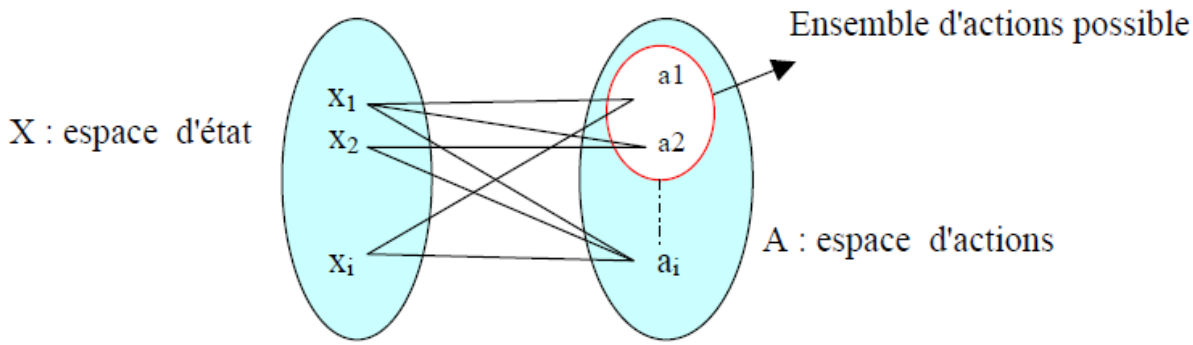


Figure III.12 : Relation états, actions.

III.5.3-Environnement :

Les principaux qualificatifs qui caractérisent l'environnement et l'interaction agent / environnement **[12]**, et qui sont utiles pour choisir une méthode d'apprentissage sont les suivants:

- **Accessible / inaccessible :** L'agent a-t-il accès à l'état complet de l'environnement, ou certaines informations restent-elles non ou mal connues ?
- **Déterministe / non-déterministe :** Le prochain état de l'environnement est-il complètement déterminé par son état courant et l'action sélectionnée par l'agent ?

  Remarque : un environnement déterministe peut ne pas l'être du point de vue de l'agent, du fait d'une connaissance insuffisante de l'état de l'environnement.
- **Episodique / non-épisodique :** On parle d'expérience épisodique si les interactions de l'agent avec l'environnement peuvent être divisées en « épisodes ». Chaque épisode consiste en une succession d'étapes de perception / action, le résultat de l'action ne dépendant que de l'épisode courant.
- **Statique / Dynamique :** Un environnement est dit dynamique si son état peut changer en fonction du temps pendant la prise de décision de l'agent. Il est dit statique dans le cas contraire.
- **Discret /Continu :** S'il existe un nombre limité de perceptions et d'actions possibles, on parle d'environnement discret

III.5.4-Fonction de renforcement :

A chaque instant, l'interaction produit une valeur de renforcement r_t, valeur numérique bornée, qui mesure la justesse de réaction de l'agent. Le but de l'agent est de maximiser le « cumul » de ces renforcements dans le temps. Pour prendre en compte l'horizon de temps, il suffit de considérer la somme des valeurs de renforcement qu'il recevra dans le futur :

$$R_t = r_{t+1} + r_{t+2} + ... + r_T \quad \text{.....} \quad \textbf{3.1}$$

Où T est un instant terminal qui met fin à l'interaction. Dans bien des cas cependant, l'interaction n'a pas de limite (T = ∞). Pour éviter que le critère ci-dessus ne diverge, on retient la somme pondérée sur le long terme :

$$R_t = r_{t+1} + \gamma.r_{t+2} + \gamma^2.r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad \text{.....3.2}$$

Où $\gamma \in [0\ 1[$, est un facteur de pondération.

$$R_t = r_{t+1} + \gamma.R_{t+1} + (1-\gamma).0 \quad \text{.....3.3}$$

Ce facteur de pondération détermine la valeur présente d'une récompense future : une récompense reçue dans k pas de temps vaut γ^k fois ce qu'elle vaudrait si elle était reçue immédiatement. Ce critère présente un intérêt fonctionnel : il incite à maximiser les récompenses tout en diminuant le temps pour les obtenir. L'influence du temps est d'autant moins négligeable que γ est proche de 1.

A contrario, si γ vaut 0, on dit que l'agent est glouton : son but revient alors, à chaque instant, à maximiser sa récompense immédiate sans se préoccuper de celles qui suivent. L'utilisation d'un signal de récompense (local dans le temps) pour définir une tâche peut paraître a priori limitant par rapport à un critère global calculé sur toute la trajectoire des états. De nombreuses applications dans la littérature ont néanmoins montré que cette approche était pratique et flexible.

III.5.5-Approximation de la fonction valeur :

On sait que l'agent réagit dans son environnement d'une manière au hasard ; le choix des actions qu'il doit exécuter est fait d'une manière aléatoire. Le premier objectif de l'agent est de trouver une projection topographique correcte entre son état (situation) et l'action qu'il a choisie. Une fois cette projection est complétée, il est possible d'extraire la politique optimale. Pour illustrer cette procédure, on utilise les notations suivantes :

$V^*(s_t)$ est la fonction valeur optimale avec s_t est le vecteur d'état.

$V(s_t)$ est l'approximation da la fonction de valeur.

$r(s_t)$ le signal de renforcement.

γ est un facteur compris entre 0 et 1.

Généralement $V(s_t)$ est initialisée d'une manière aléatoire, et elle ne contient pas des informations sur la fonction optimale $V^*(s_t)$.À l'instant t l'erreur d'approximation de la fonction de valeur $e(s_t)$ est présentée par la différence entre la fonction de valeur optimale $V^*(s_t)$ et son approximation $V(s_t)$ La relation entre ces trois grandeurs est donnée par l'équation suivante :

$$V(s_t) = V^*(s_t) + e(s_t) \quad \text{.....} \quad \textbf{3.4}$$

L'utilisation de la définition de la fonction de valeur optimale est donnée par l'équation :

$$V^*(s_t) = r(s_t) + \gamma V^*(s_{t+1}) \quad \text{.....} \quad \textbf{3.5}$$

Cette relation représente l'équation de Bellman, où La plupart des algorithmes d'apprentissage par renforcement sont basés sur elle.

L'approximation de la fonction de valeur a la même relation que l'équation précédente, où elle est donnée par :

$$V(s_t) = r(s_t) + \gamma V(s_{t+1}) \quad \text{.....} \quad \textbf{3.6}$$

Substituant l'équation (3.4), (3.5) dans l'équation (3.6), on obtient la formule suivante:

$$e(s_t) + V^*(s_t) = r(s_t) + \gamma(e(s_{t+1}) + V^*(s_{t+1})) \quad \text{.....} \quad \textbf{3.7}$$

Donc :

$$e(s_t) + V^*(s_t) = r(s_t) + \gamma e(s_{t+1}) + \gamma V^*(s_{t+1}) \quad \text{.....} \quad \textbf{3.8}$$

L'utilisation des équations (3.2) et (3.5) nous conduit à obtenir l'équation suivante :

$$e(s_t) = \gamma e(s_{t+1}) \quad \text{.....} \quad \textbf{3.9}$$

Qui représente la relation entre les erreurs d'approximation de la fonction valeur entre deux états (situations) successifs s_t et s_{t+1}.Où le facteur γ est compris entre 0 et 1.

Finalement, un système d'apprentissage par renforcement doit être capable d'apprendre un comportement voulu de manière à ce que l'erreur d'approximation de la fonction de valeur tend vers zéro.

III.5.6-Fonction de valeur :

L'une des parties fondamentales du système d'apprentissage par renforcement est la fonction valeur. Celle-ci permet à l'agent d'apprendre comment choisir les bonnes actions et comment mesurer leurs utilités. Et comme nous l'avons présenté dans le chapitre un, il existe deux types de fonction valeurs : Fonction « Valeur - état » notée par $V^{\pi}(s)$ et Fonction « valeur état-action » notée par $Q^{\pi}(s,a)$.**[2]**

L'utilisation de l'une de ses deux fonctions conduit a synthétisé un algorithme qui pourra faire une bonne approximation à l'une de ces deux fonctions valeur.

III.5.6.1- Tableau de consultation :

Il est possible de représenter l'approximation de la fonction de valeur par un tableau de consultation [Touzet 1997] dans lequel chaque case corresponde à une approximation de la fonction de valeur optimale pour une configuration fixe de l'état ou de la paire état/action. **[2]**

Le concepteur du système d'apprentissage par renforcement peut fixer la forme de la fonction $V^{*}(s_t)$ suivant les objectifs voulus, donc l'approximation de la fonction de valeur optimale peut être obtenue après l'initialisation aléatoire et mis à jour par l'application de l'équation (3.6) en balayant tout les états s_t . Pour que l'algorithme converge il faut que l'erreur d'approximation donnée par l'équation suivante :

$$\Delta V(s_t) = V^{*}(s_t) - V(s_t) \quad \textbf{3.10}$$

Tend vers zéro.

Comme exemple on peut citer le travail présenté par C. Touzet en 1997 où il a donné une présentation générale de l'algorithme Q Learning proposé par Watkins

en 1989, la Figure (III.13) présente le modèle général de l'algorithme d'apprentissage par renforcement « Q Learning »

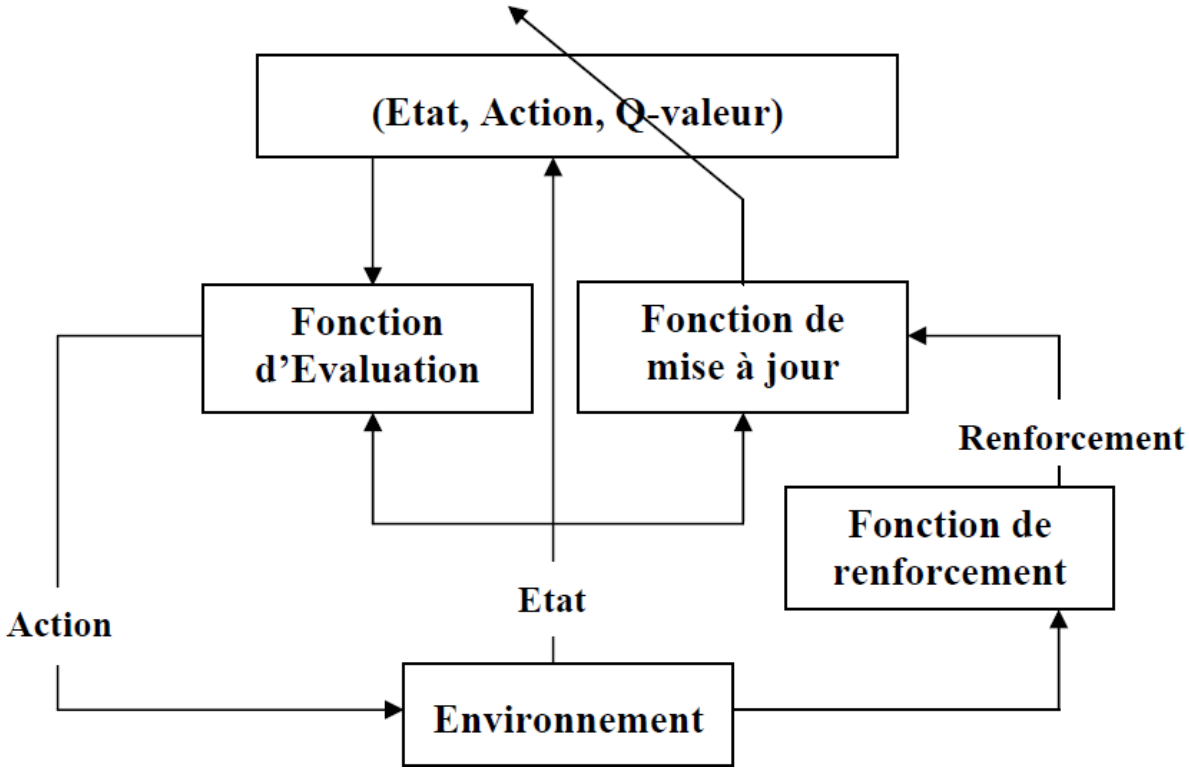


Figure III.13 : Modèle général pour les algorithmes d'apprentissage par renforcement, (ici le Q-Learning).

Un tableau à deux dimensions représente les états de l'environnement et les actions possibles. Les lignes correspondent aux états et les colonnes aux actions. Les couples déjà essayer (état visité, action exécutée) correspondent au cases non vides. La fonction de renforcement fournit pour chaque état une évaluation qualitative de son intérêt par rapport au comportement désiré, le tableau (III.1) représente le signal de renforcement reçus par l'agent (+1 : bon, - 1 : mauvais, 0 : neutre).

r	a_1	a_2	a_3	a_4	a_5	a_6
S_1					+1	
S_2		+1				
S_3		0	0			
S_4			-1		0	
S_5			-1			
S_6			0			-1

Tableau III.1 : Renforcement reçus par l'agent.

La fonction d'évaluation parcourt, pour l'état présent, les valeurs des Q associées aux actions et sélectionne celle de plus grande utilité. Le tableau (III.2) représente Les valeurs d'utilité Q calculées grâce à l'application de l'équation (1.17) de mise à jour indiquée dans le chapitre un.

Q	a_1	a_2	a_3	a_4	a_5	a_6
S_1	0	0	0	0	+0.7	0
S_2	0	+0.8	0	0	0	0
S_3	0	0.4	0.3	0	0	0
S_4	0	0	-1	0	0.8	0
S_5	0	0	+0.9	0	0	0
S_6	+0.1	0	0	0	0	-0.7

Tableau III.2 : Valeurs de *Q* calculées.

III.6-Conclusion :

Dans ce chapitre nous avons donné des généralités sur la robotique mobile où on a présenté quelques types des robots mobiles, le matérielle utiliser surtout pour la perception (Capteurs) et aussi l'intervention des robots mobiles dans notre vie quotidienne, à la nous avons donné une description pour tous les blocs qui constituent un système d'apprentissage par renforcement, le rôle et l'influence de chacun dans ce système que nous allons l'utilisé dans notre simulation qui sera l'un des objectifs du chapitre suivant.

Chapitre IV:

Application et Résultats de Simulation

Application et Résultats de Simulation

IV.1-Introduction :

Ce chapitre est consacré spécifiquement à la mise en œuvre de la technique d'apprentissage par renforcement pour la navigation d'un robot mobile dans un environnement qui contient des obstacles, où nous utilisons un réseaux de neurone artificiel de type MLP pour la génération de la fonction valeur, ce système est basée sur la perception de l'agent (robot) pour son environnement de travail, cette perception représente l'état de l'environnement où elle seras utilisée comme entrée pour notre réseau de neurone artificiel, une fonction de sélection permet de déterminer l'action qui doit être exécuter par l'agent (robot) et une fonction de renforcement qui est un scalaire +1 récompense, -1 punition. Tous ces blocs seront expliqués en détail par la suite.

IV.2-Architecture du système Apprentissage par renforcement:

L'un des objectifs de cette proposition est d'avoir un système d'apprentissage qui permet à un robot de ce déplacé dans un environnement qui contient des obstacles et en les évitant.

L'algorithme que nous avons choisie pour notre système d'apprentissage par renforcement est le ***Q-Learning*** que nous l'avons présenté dans le chapitre un. Dans la version classique du ***Q Learning*** les valeurs de la fonction ***Q*** sont stockées dans une table à deux dimensions : Etats et Actions. Il est possible de représenter la fonction ***Q*** avec un ***réseau de neurone artificiel***, où ces entrées sont les lectures des capteurs associés au robot pour la perception de son ***environnement***, le choix de l'action que l'agent doit exécuter est faite par une fonction appelée ***fonction de sélection***.

La figure (IV.1) représente la structure du système d'apprentissage par renforcement basé sur un réseau de neurone.

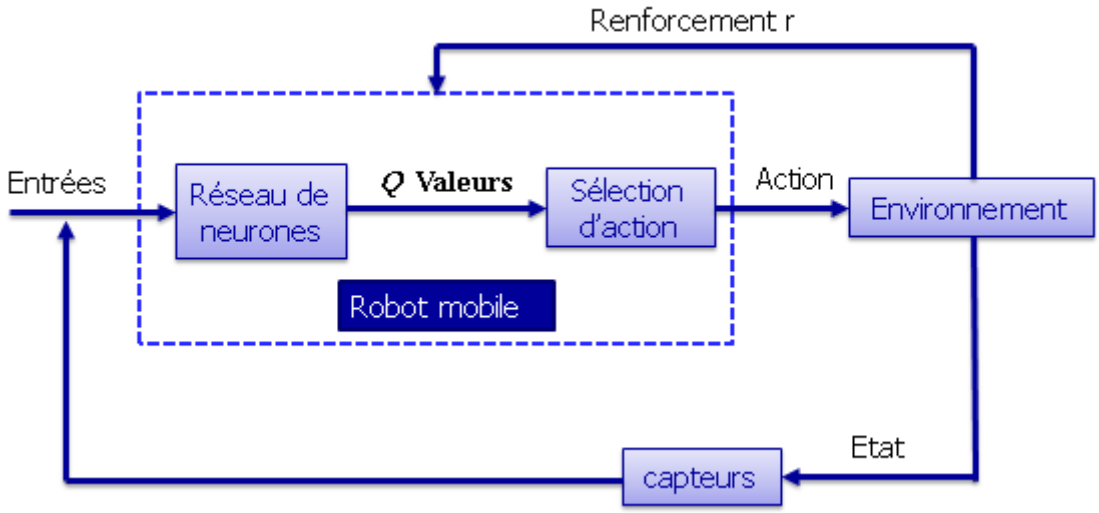


Figure IV.1 : Structure du système d'apprentissage par renforcement.

IV.2.1- Environnement :

La lecture de l'état de l'environnement est faite grâce à des capteurs qui sont placés sur les trois côtés du robot deux à gauche, trois en avant et deux à droite. Les capteurs utilisés peuvent êtres de type de détection de distance. Le vecteur d'état S est choisi de manière à avoir des informations sur l'existence d'obstacles sur les trois côtés du robot.

Ce vecteur est composé de sept variables binaires s_i, i=1,7. Le choix de ces variables est fait de manière à avoir des informations tout ou rien. C'est-à-dire, si par exemple si=1 alors il existe un obstacle près du robot, si si=0 pas d'obstacle prés du robot.

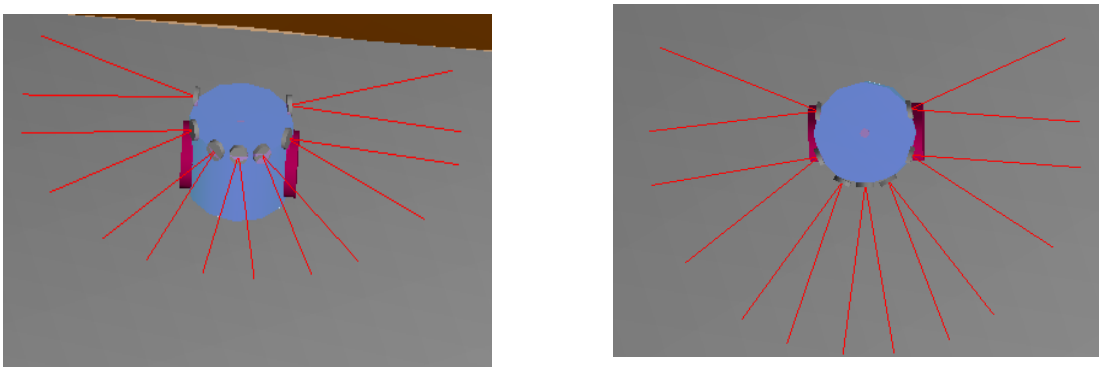
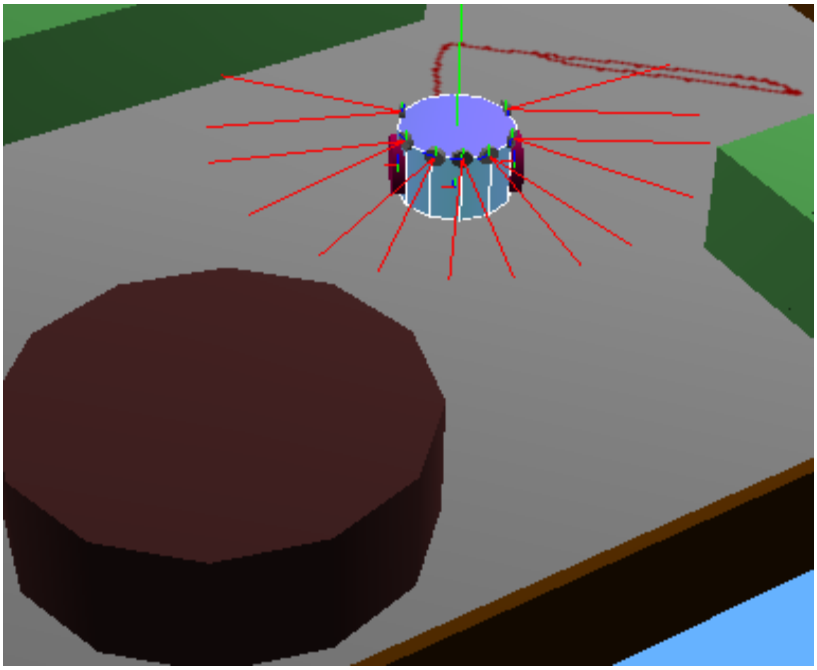


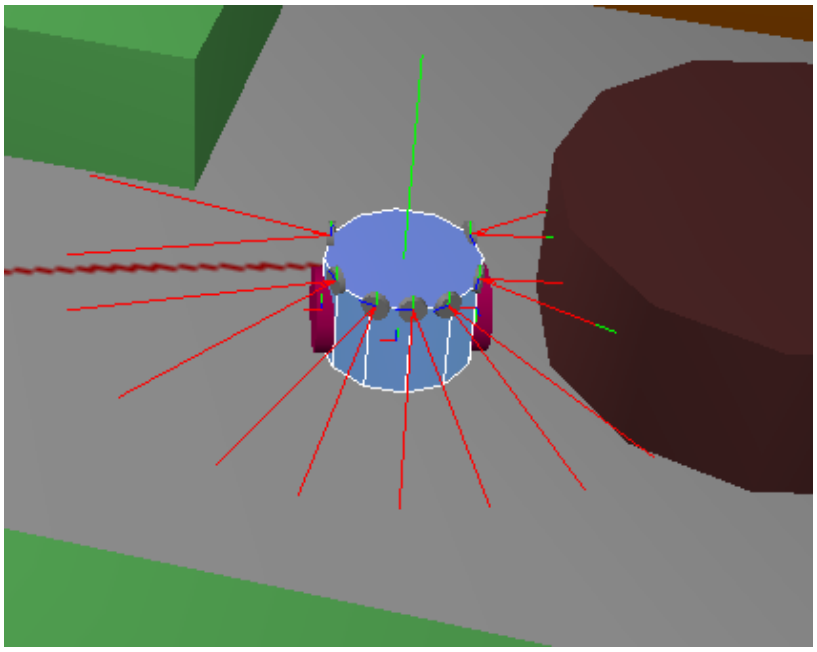
Figure IV.2 : Robot mobile de type unicycle a des capteurs infrarouges.

Les figures suivantes représentent quelques Etat de l'environnement qui donne la valeur du vecteur d'état S correspondante.



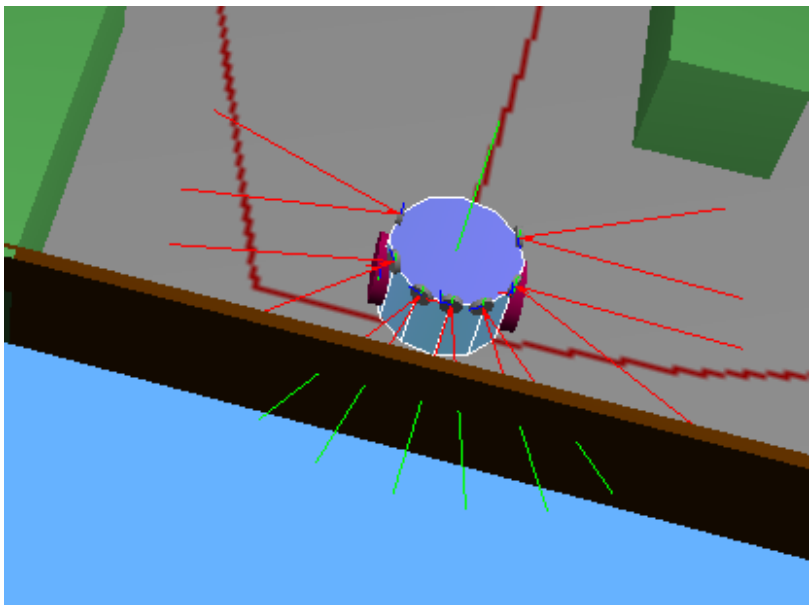
S=[0 0 0 0 0 0 0]

Figure IV.3 : 1er Etat de l'environnement.



S=[1 1 0 0 0 0 0]

Figure IV.4 : 2^{em} Etat de l'environnement.



S=[0 0 1 1 1 0 0]

Figure IV.5 : 3^{em} Etat de l'environnement.

IV.2.2- Fonction du renforcement :

Pour chaque état où se trouve le robot, une évaluation de l'action effectuée est jugée par un signal appelé signal de renforcement. Cette fonction de

renforcement permet au robot d'explorer son environnement, Ce signal est lié aux valeurs des mesures des capteurs qui indiquent l'existence ou l'absence des obstacles dans les trois directions, à gauche, devant et à droite, qui représentent l'état de l'environnement. La valeur de cette fonction ou ce signal de renforcement est **-1** lorsque le robot percute un obstacle, et **+1** lorsque le robot évite l'obstacle.

IV.2.3- Fonction de valeur :

Comme nous l'avons indiqué précédemment, la fonction de valeur ***Q*** est une projection topologique des paires états / actions. La génération de la fonction de valeur est faite par une implémentation d'un réseau de neurone artificiel de type ***MLP***.

Le réseau choisi est caractérisé par sept cellules d'entrée liés aux mesures des capteurs placés sur les trois côtés du robot ***(Gauche, Avant, Droite)*** deux à gauche, trois en avant et deux à droite, trois autres cellules sont liées aux actions possibles ***(Tourner à Gauche, Avancer, Tourner à Droite)***, un neurone dans la couche de sortie qui présente la valeur de Q avec une couche cachée qui contient un nombre nc de neurones.

IV.2.4- Architecture et paramètres des R.N.A utilisés :

IV.2.4.1- Description de l'architecture envisagée :

Le modèle de neuronal est un réseau à 2 couches avec fonction d'activation Sigmoïde pour la couche cachée, et fonction d'activation Linéaire pour la couche de sortie.

- un vecteur à 10 entrées.
- 1^{ere} couche cachée à 05 neurones (S1=5)
- Un seul neurone dans la couche de sortie (S2=1)

La couche cachée et de sortie sont alors caractérisées par les matrices poids respectives [W1, B1], [W2, B2].

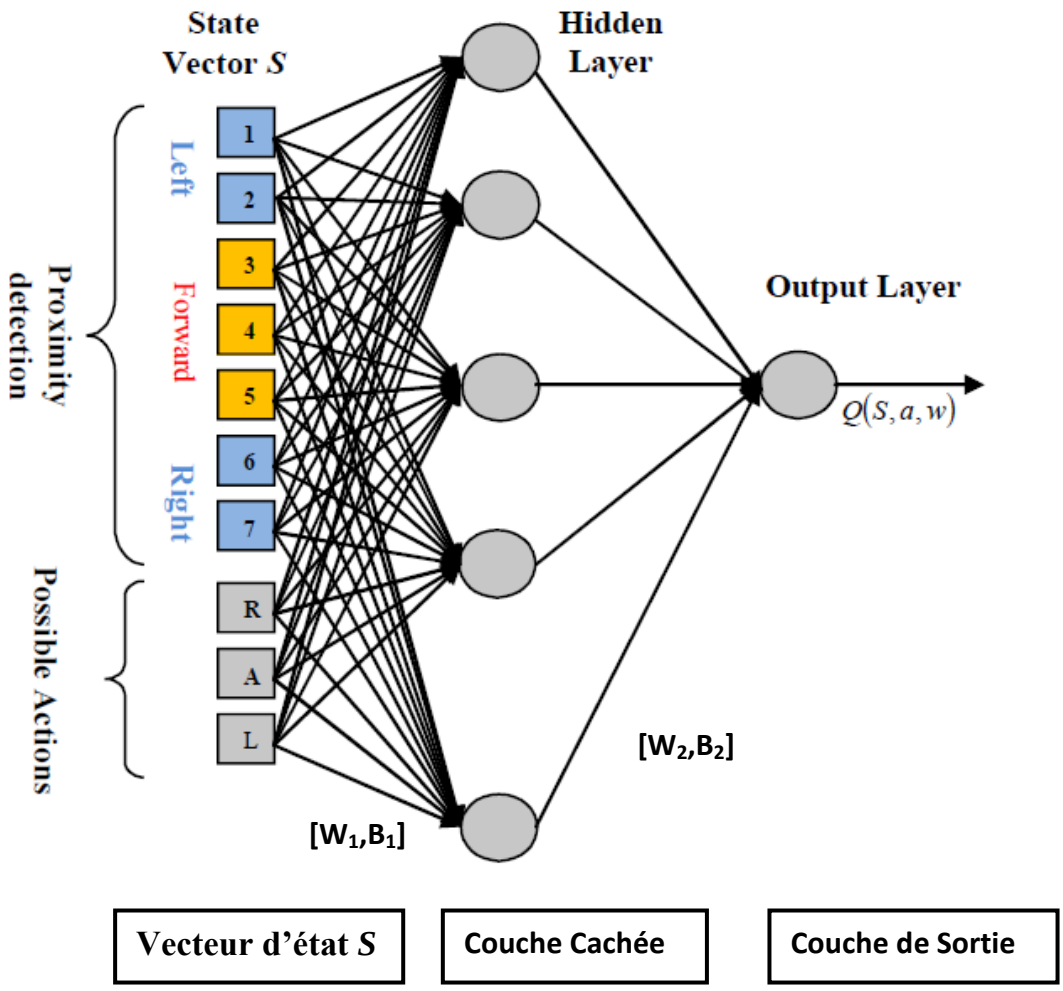


Figure IV.6 : Description de l'architecture envisagée.

Pour la matrice de poids [W1] le nombre de ligne égale le nombre de neurone dans la couche cachée S_1=5, et le nombre de colonne égale le nombre d'entrée du réseau.

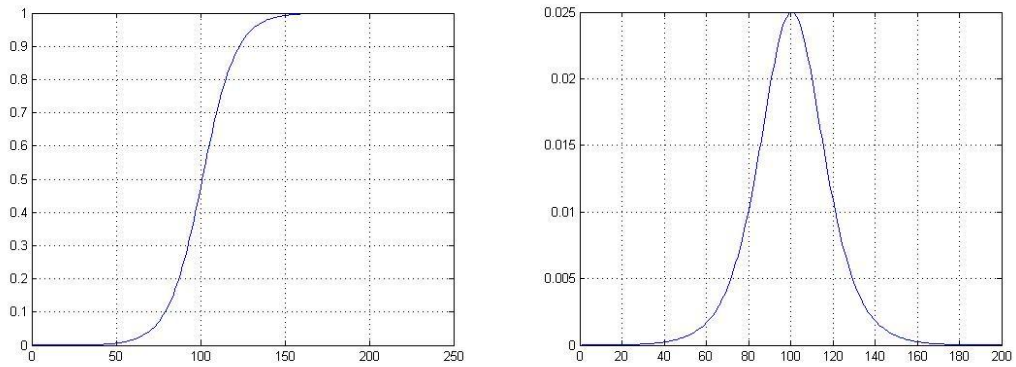
Pour la matrice de poids [W2] le nombre de ligne égale le nombre de neurone dans la couche de sortie S3=1, et le nombre de colonne égale le nombre de neurone dans la couche cachée S1=5.

Pour la biais (b1) le nombre de ligne corresponde au nombre de neurone dans la 1er couche cachée (S1=5) avec une seule colonne, pour le biais (b2) le nombre de ligne égale au nombre de neurone dans la couche de sortie (S3=1) .

- la fonction sigmoïde

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{.....4.1}$$

La représentation graphique de cette fonction et sa dérivée est représentée sur la figure (IV.7).



Fonction d'activation *f(x)* **Dérivée de *f(x)***

Figure IV.7 : Fonction d'activation et sa dérivée.

- La fonction linéaire :

C'est l'une des fonctions d'activation les plus simples, elle est définie par :

$$F(x) = x \quad \text{.....4.2}$$

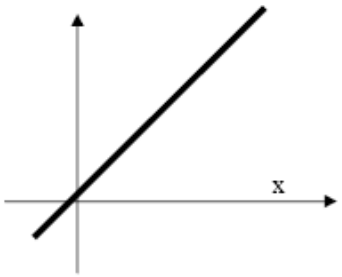


Figure IV.8 : Fonction Linéaire.

IV.2.4.2-Apprentissage du réseau :

L'apprentissage du réseau est basé sur la mise à jour ou l'ajustement des matrices des poids du réseau de neurones.

IV.2.4.2.a- Données d'apprentissage :

Les données d'apprentissage du R.N.A sont obtenues comme suit :

- On définit les éléments d'entrées ***states***,
- On leurs associe les éléments désirées $\boldsymbol{Q}_{target}$;
- On fixe le nombre de neurones de la couche cachée S1 et S2 ;

Où :

$\boldsymbol{Q}_{target}$ représente la simplification de l'équation d'optimalité de BELLMAN qui est donnée par l'équation suivante:

$$Q_{t\,\mathrm{arg}et} = r(s_t, a_t) + \gamma \max Q(S, a_t, w)$$

States représente le vecteur d'état qui contient le vecteur de capteur et le vecteur de l'action possible.

IV.2.5-Fonction de sélection d'action :

Notre réseau de neurones nous permet de générer la fonction de valeur ***Q***.

L'ensemble d'actions possible est donné par $A=\{a_1,a_2,a_3\}$ où :

- $\mathbf{a}_1$: Action tourné à Gauche.
- $\mathbf{a}_2$: Action Avancer.
- $\mathbf{a}_3$: Action tourné à Droite.

La sélection de l'action que le robot doit exécuter est basée sur la politique d'exploration/exploitation (PEE) pour cette raison on a utilisé la méthode ε-gloutonne (ε-greedy) qui consiste à choisir l'action gloutonne avec une probabilité ε et à choisir une action au hasard avec une probabilité 1- ε , soit :

- $p \in [0,1]$ Un nombre au hasard.
- $p < \varepsilon$ on choisit une action a au hasard (***Exploration***), où $a \in A$ l'ensemble de l'action possible.

- Si $p > \varepsilon$ on choisit $a(S) = Arg\ \max_{b \in a} q(S,b,w)$ (***Exploitation***).

$$a(S) = Arg\ \max_{b \in a} q(S,b,w) \text{ .....} \mathbf{4.3}$$

IV.3-Modèle Géométrique d'un robot du type unicycle :

La conception d'une commande pour un système dynamique est généralement basée sur sa modélisation. La figure (IV.9) représente le type du robot que nous avons choisi pour notre application, ce robot est actionné par deux roues indépendantes, de rayon **R** et séparée entres elles par une distance **L**.

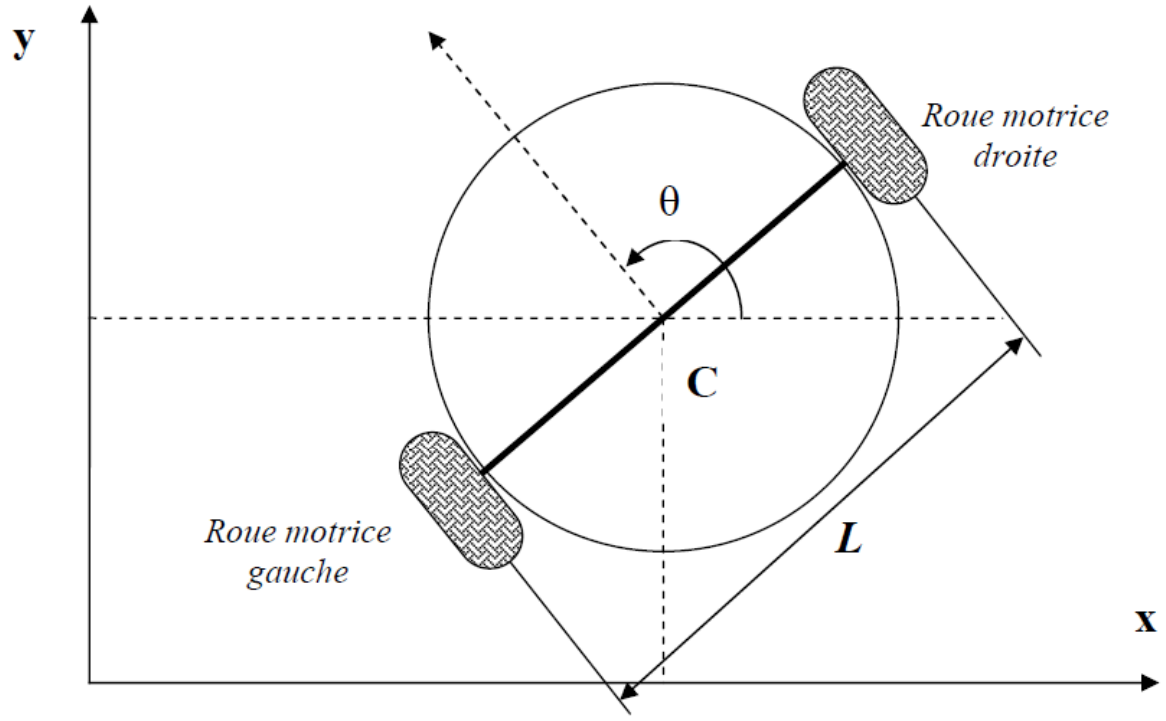


Figure IV.9 : Type du robot utilisé

Ce robot est caractérisé par les équations cinématiques suivantes :

IV.3.1-La position du robot :

$$x_r(k+1) = x_r(k) + v(k) * \cos\left(\theta(k) + \frac{\Delta\theta(k)}{2}\right) \text{.....} \mathbf{4.4}$$

$$y_r(k+1) = y_r(k) + v(k) * \sin\left(\theta(k) + \frac{\Delta\theta(k)}{2}\right) \text{.....} \mathbf{4.5}$$

$x_r(k)$ et $y_r(k)$: sont les abscisses et l'ordonnée du robot dans le repère **(ox, oy).**

IV.3.2-L'orientation du robot :

$$\theta(k+1) = \theta(k) + \Delta\theta(k) \quad \textbf{4.6}$$

Avec :

$\theta(k)$: Est la position angulaire du robot dans le repère (ox, oy).

IV.3.3-la vitesse linière du robot :

$$v(k) = \frac{1}{2}(\theta_r(k) + \theta_l(k)) \quad \textbf{4.7}$$

Avec :

$\theta_r(k)$: La vitesse angulaire de la roue droite du robot. (ox, oy).

$\theta_l(k)$: La vitesse angulaire de la roue gauche du robot. (ox, oy).

IV.3.4-La variation de thêta :

$$\Delta\theta(k) = \frac{1}{2*L}(\theta_r(k) - \theta_l(k)) \quad \textbf{4.8}$$

Où :

L : est la distance entre la roue droite et la roue gauche.

IV.4-Algorithme et résultat de simulation :

4.4.1-Algorithme :

Algorithme 4.1

- Initialisation aléatoire des poids W du réseau de neurone ;
- Donner la position initiale du robot [**(Xr(0), Yr(0), θr (t))**];

Pour t=1 jusqu'a t itération

- Lecture de l'état St de l'environnement par les trois capteurs (**G, A, D**) ;
- Pour un état fixe, calcul de la fonction ***Q*** par le réseau de neurone pour les trois actions possible (**TG, AV, TD**);
- Détermination de l'action optimale c'est l'action qui correspond à la plus grande valeur de Q (Action_optimale → Max (Q)) ;
- Exécuter l'action optimale avec une probabilité ε ou d'une action aléatoire avec une probabilité 1- ε ;
- Lecture du nouvel état $\mathbf{S_{t+1}}$ de l'environnement par les Sept capteurs (**G, A, D**) ;
- Renforcement ;
- Mise à jour des poids et des biais du réseau de neurone.
- Teste du renforcement

  Si r=-1 (il y a un obstacle)

  - Retour à l'état précédent.

  Fin si

- ε= ε*0.99 pour diminuer ε progressivement ;

Fin Pour

IV.4.2-Quelques Résultats de simulation :

Dans ce paragraphe nous présentons quelques résultats obtenus on utilisant l'algorithme présenté précédemment.

La figure (IV .10) représente une allure du signal de renforcement avec un choix arbitraire des paramètres (5 neurones dans la couche cachée, et le coefficient d'apprentissage γ =0.8, avec 2000 d'itération). D'après cette figure, on remarque que le robot à effectuer des mauvaises actions au début ceci est normal car le robot n'a pas d'informations sur son environnement.

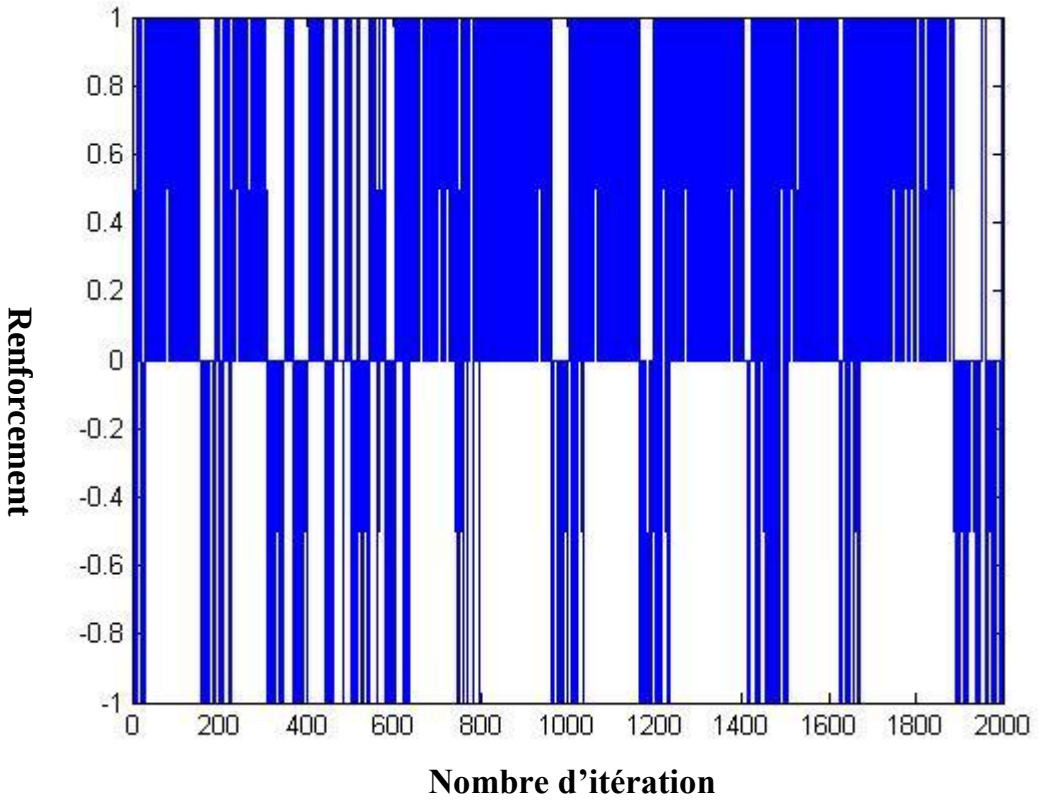


Figure IV.10 : Exploration et Exploitation de l'environnent.

La figure (IV.11) représente l'exploration du robot pour son environnement avec un choix arbitraire des paramètres (5 neurones dans la couche cachée, et le coefficient d'apprentissage γ =0.8) d'après cette figure, on remarque que le robot choisi des actions aléatoires et optimales en évite les obstacles.

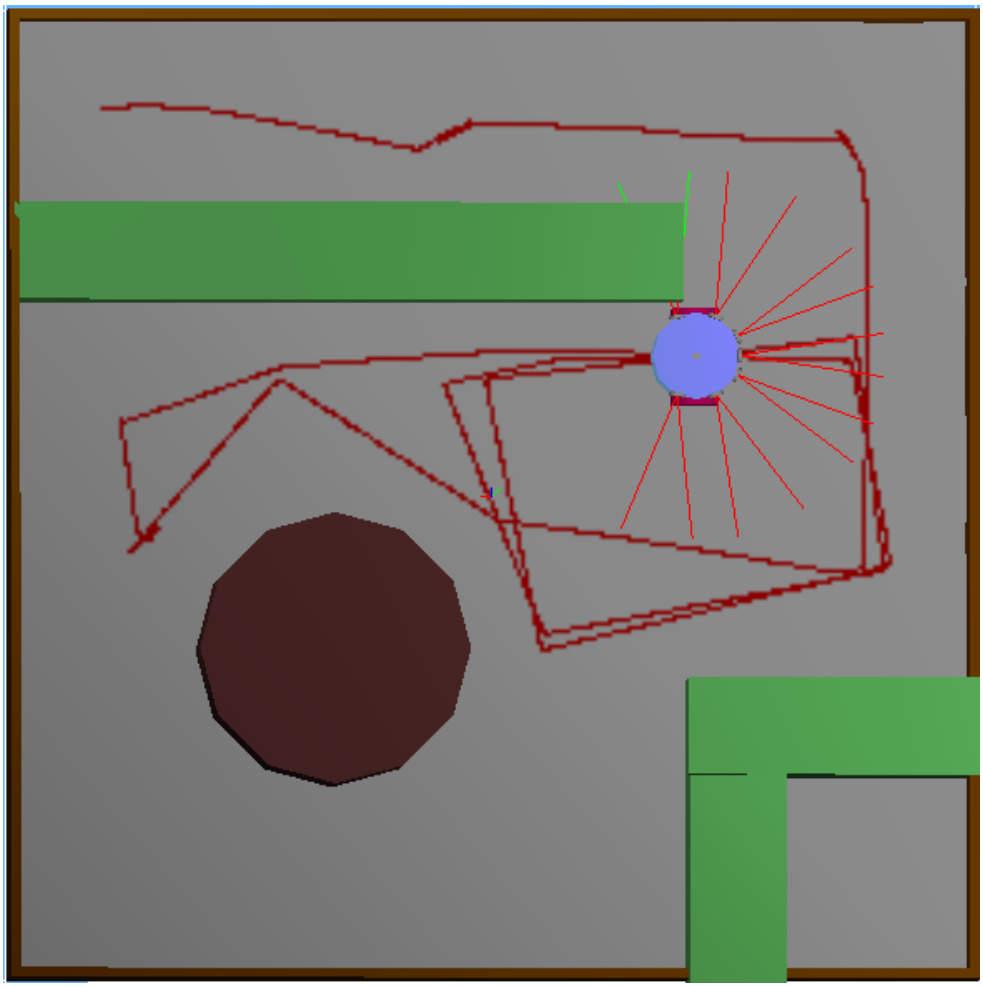


Figure IV.11 : Navigation d'un robot mobile avec un choix arbitraire des paramètres : une seule couche cachée a cinq neurones, γ =0.8.

La figure (IV.12) représente l'exploration du robot pour son environnement avec un choix arbitraire des paramètres (3 neurones dans la couche cachée, et le coefficient d'apprentissage γ =0.8) d'après cette figure, on remarque que le robot en début il est facile pour tourner quand trouve l'obstacle en avant ou à côtes, mais âpres un peu de temps il est raccroché dans le dernier coin.

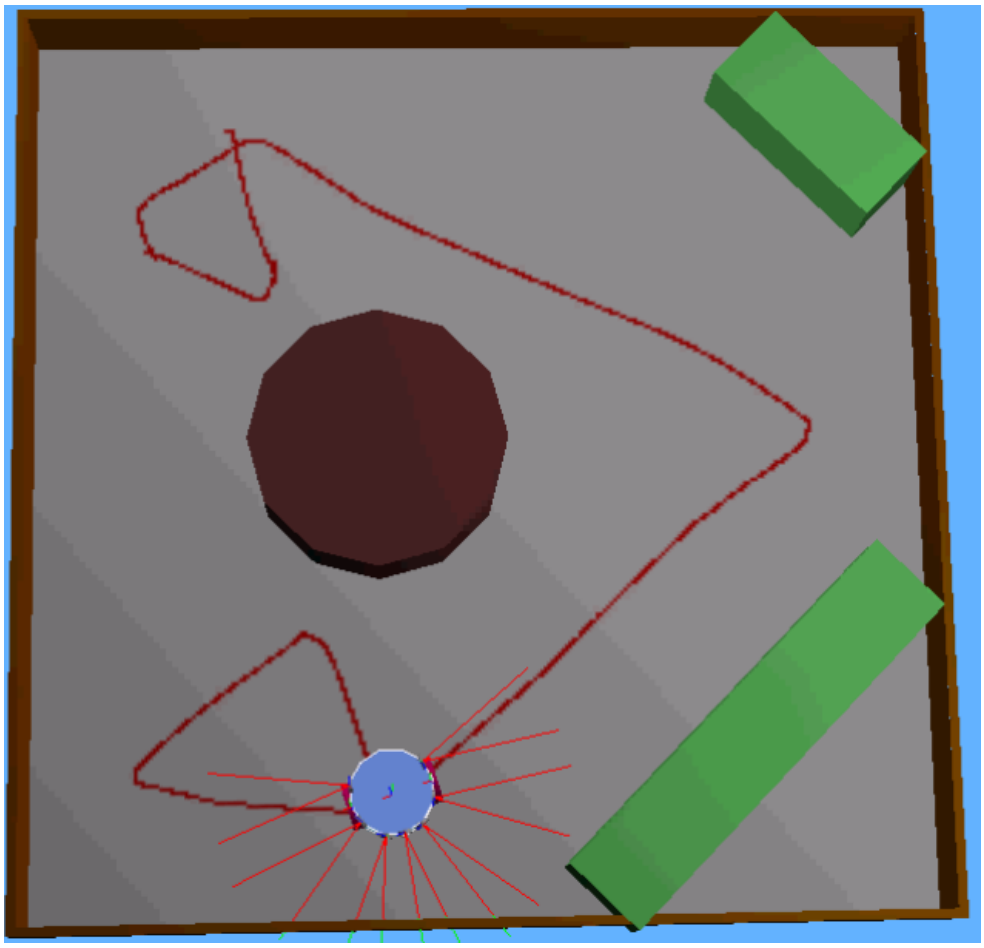


Figure IV.12 : Navigation d'un robot mobile avec un choix arbitraire des paramètres : une seule couche cachée a trois neurones, γ =0.8.

La figure (IV.13) représente l'exploration du robot pour son environnement avec un choix arbitraire des paramètres (5 neurones dans chacun les deux couches cachées, et le coefficient d'apprentissage γ =0.9) d'après cette figure, on remarque que le robot évite Les obstacles tout faciles.

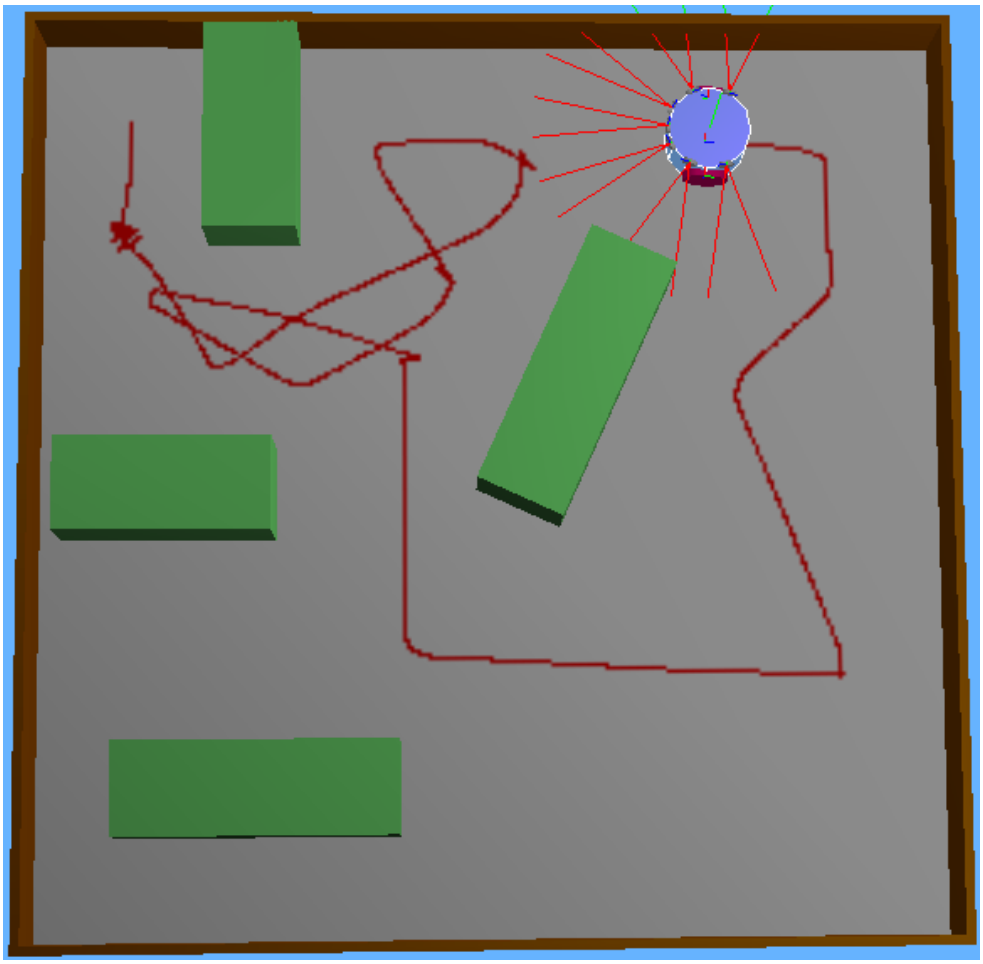


Figure IV.13 : Navigation d'un robot mobile avec un choix arbitraire des paramètres : 5 neurones dans chacun les deux couches cachées, γ =0.9.

La figure (IV.14) représente l'exploration du robot pour son environnement avec un choix arbitraire des paramètres (5 neurones dans chacun les deux couches cachées, et le coefficient d'apprentissage γ =0.1) d'après cette figure, on remarque que le robot fait des mauvaises transitions entre les actions possibles par rapport l'expérience précédente (Figure IV.13).

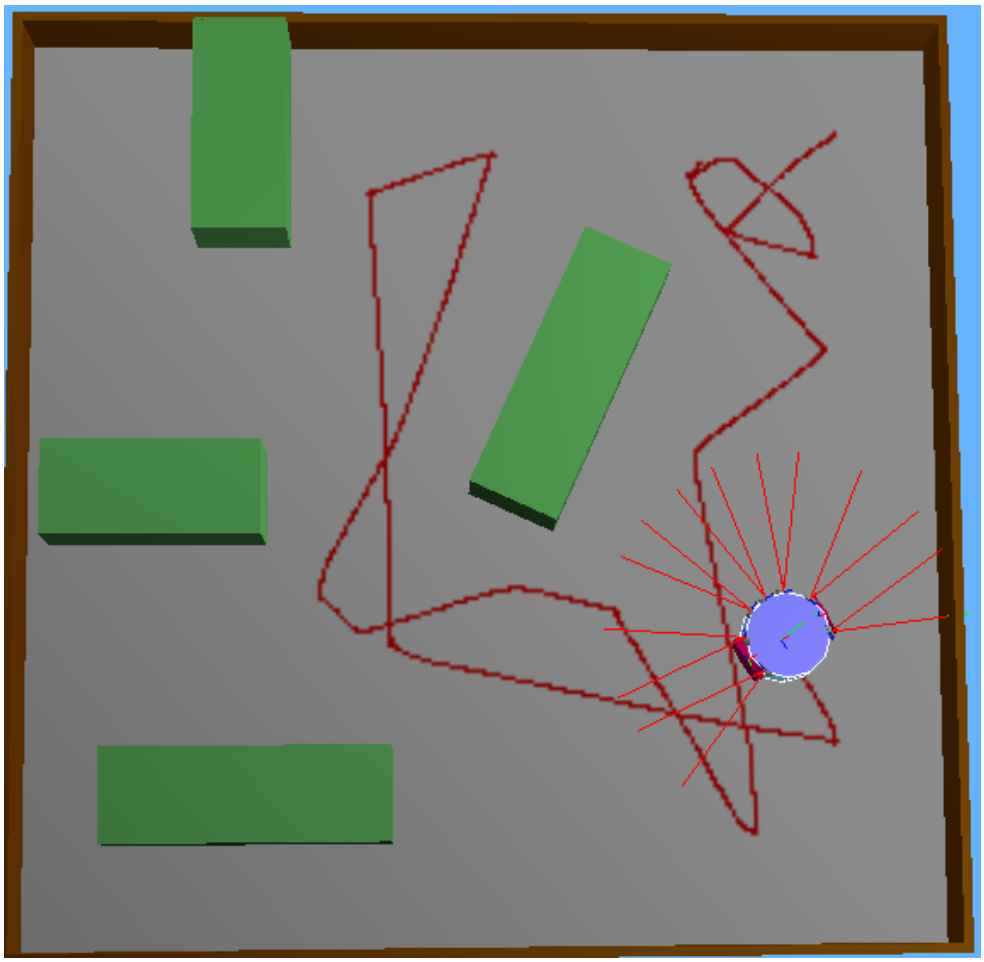


Figure IV.14 : Navigation d'un robot mobile avec un choix arbitraire des paramètres : 5 neurones dans chacun les deux couches cachées, γ =0.1.

IV.5-Conclusion:

Ce dernier chapitre a fait l'objectif d'une étude en simulation de la mise en œuvre de la technique d'apprentissage par renforcement en utilisons réseaux de neurone artificiel de type MLP où on l'a appliquées au domaine de la robotique mobile spécialement la navigation et l'évitement d'obstacles.

Les résultats obtenus sont acceptables et nous ont permis de visualiser la trajectoire suivie par le robot mobile en évitent les obstacles, de plus on peut indiquer que les paramètres suivant :

- Pas d'apprentissage.
- Nombre de neurones dans la couche cachée.

Peut influés sur les résultats obtenus.

Conclusion Générale

Conclusion Générale

Dans ce travail nous avons présenté les méthodes d'apprentissage automatique qui existent, où on a choisi la méthode d'***apprentissage par renforcement,*** cette méthode est basée sur l'interaction entre l'agent est son environnement de travail.

Pour la résolution de ce type d'apprentissage il existe plusieurs méthodes, dans ce mémoire on a choisi la méthode des ***différences temporelle***, où on a utilisé l'algorithme de ***Q-Learning***, cet algorithme est basé sur une fonction appelé fonction valeur « état - action », pour la génération de cette fonction on a utilisé un ***réseau de neurones artificiel*** de type ***MLP***.

L'utilisation du logiciel **Webots 6.4.1** et aussi Matlab nous ont permet de visualiser des trajectoires pour quelques environnements que nous avons proposées est assurée la navigation d'un robot mobile dans un environnement qui contient des obstacles en les évitent.

1-Introduction à Webots :

Webots est un logiciel destine à la simulation de robots en trois dimensions. Il est développe par la société Cyberbotics en collaboration avec l'EPFL. Le langage de Webots est basé sur un sous-groupe du langage de spécification 3D VRML97. Il est conçu pour simuler fidèlement des robots existants ou inexistants, ainsi que le monde qui les entoure. Il permet de créer simplement et rapidement des prototypes, puis de simuler leur comportement. Le portage d'un contrôleur du simulateur au robot réel généralement sans problèmes majeurs et est une des principales qualités de ce programme. Il offre, en plus, l'avantage de pouvoir réaliser des expérimentations en temps accélère, sans utiliser des robots.

Le logiciel utilise le moteur physique ODE (Open Dynamics Engine) pour gérer la dynamique, les frottements et les contacts. Cette librairie open source est capable de simuler de manière fiable est performante la dynamique de corps rigides.

La documentation de se logiciel se compose principalement de trois documents :

- Le guide utilisateur.
- Le manuel de référence.
- Le manuel d'utilisateur.

2- La notion de "monde"

Un monde, dans Webots est un environnement virtuel en trois dimensions dans lequel on peut créer des objets et des robots. Ce monde est donc la description du modèle qui est représente sous la forme d'un arbre dans lequel chaque objet (robot, sol etc..) est représente par un nœud et ses fils (qui sont ses paramètres).

3- La notion de contrôleur

Un contrôleur est un fichier exécutable qui permet de contrôler le robot. Il accède une API Webots qui est plus ou moins limitée au contrôle des moteurs (c'est ce que l'on cherche _a faire dans la majeure partie des cas). Le contrôleur utilisé par la simulation est défini dans le fichier "monde".

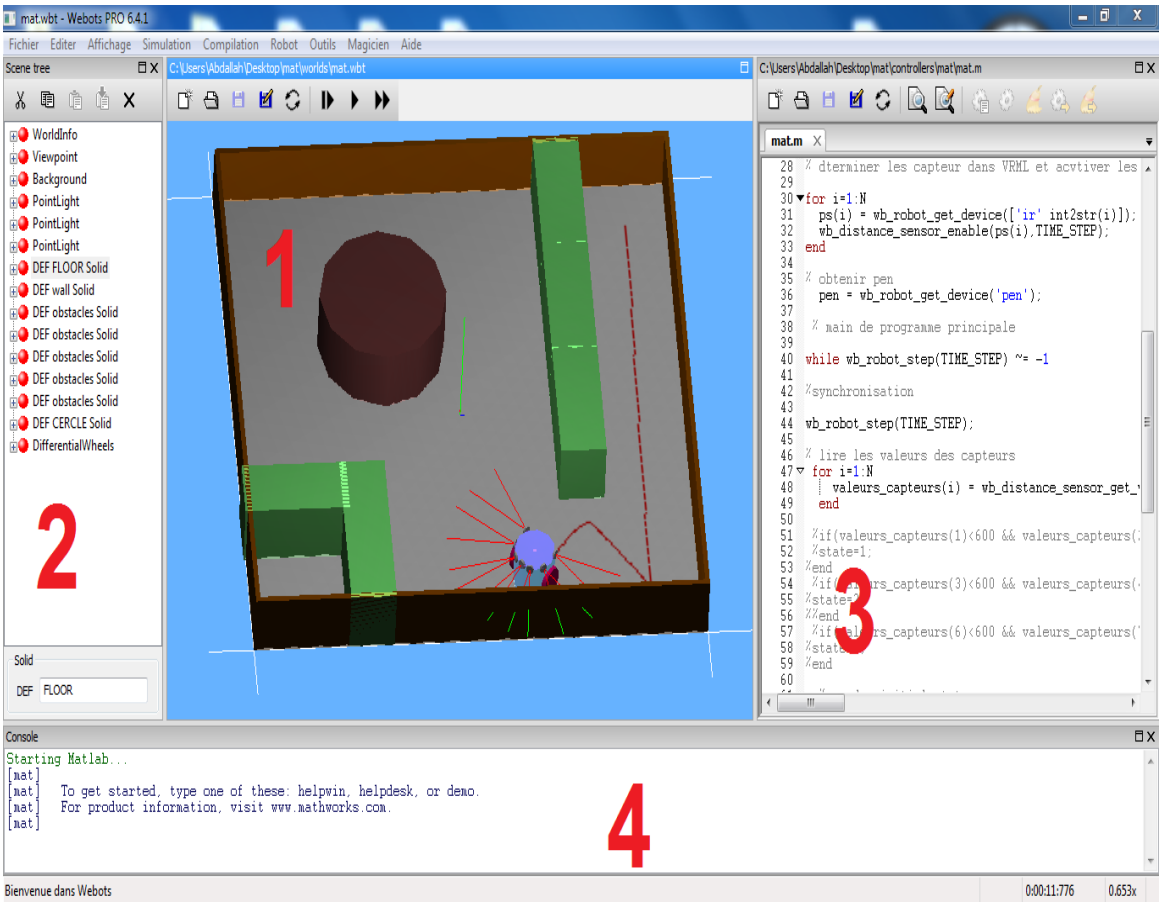


Figure 1 : Interface Webots

4- La notion de plugin physique

Un plugin physique est, tout comme un contrôleur, un fichier exécutable. Il permet l'accès direct au moteur physique ODE. C'est de cette manière que l'on peut ajouter des forces s'exerçant directement sur les objets du monde ou les lire. Un plugin physique est moins facile _a utiliser qu'un contrôleur mais beaucoup plus complet.

5- Interface Webots

La figure (1) montre l'interface de base du logiciel Webots. Elle possède trois fenêtres principales. La Fenêtre 4 étant un simple log des messages générés par les différents contrôleurs et le plugin physique.

1 La fenêtre de visualisation de la simulation. Elle permet de visualiser ce qui se passe durant les phases de modélisation et de simulation. C'est depuis

cette fenêtre qu'on peut ouvrir ou enregistrer les fichiers "monde" 1. Elle permet aussi de contrôler l'exécution de la simulation (relancer la simulation, avancer, avancer d'un pas de temps, arrêter et avancer en mode accélérer.

2 La fenêtre de modélisation. C'est dans cette dernière qu'on peut modifier le modèle (le fichier "monde"). On peut donc modifier ici les tailles, les paramètres physiques des différents segments, les forces que peuvent exercer les différents moteurs, le contrôleur utilise par un robot, le pas de temps de la simulation, etc…... Les modifications sont appliquées et peuvent être visualisées dans la fenêtre 1.

3 La fenêtre d'Edition permet d'éditer et de compiler les contrôleurs et plugins physiques.

Listes des figures

Liste des tableaux

BIBLIOGRAPHIE

[1] : Luc Sarzyniec « **Apprentissage par renforcement développemental pour la robotique autonome** » Université Henri Poincaré, France. 2010.

[2] : Mezaache Hatem « **Les réseaux de Neurones formels et les systèmes Neuro-Flous pour l'apprentissage par renforcement** » Université Hadj Lakhdar, Batna-Algérie. 2007.

[3] : David FILLIAT « **Robotique Mobile** » École Nationale Supérieure de Techniques Avancées Paris Tech, France.2011.

[4] : Site WEB: http://wwwdfr.ensta.fr/Cours/docs/C10-2/

[5] : Rachid LADJADJ **Les réseaux de neurones** 2002/2003 Site WEB: http://www-igm.univ-lv.fr/~dr/XPOSE2002/Neurones/index.php?rubrique=Accueil

[6] : Marc Parizeau « **Réseau de neurones** » Université Laval, Canada. 2004.

[7] : Claude Touzet « **Introduction au connexionnisme** » COURS, EXERCICES

ET TRAVAUX PRATIQUES Juillet 1992.

[8] : Clément Châtelain « **Les multi layer perceptron (MLP)** » novembre 2003.

[9] : G. Campion, G. Bastin et B. D'Andréa-Novel. « **Structural Properties and Classification of Kinematic and Dynamic Models of Wheeled Mobile Robots** » IEEE Transactions on Robotics and Automation, vol. 12, no. 1, pages 47–62, 1996.

[10] : Bilal KARABAGLI « **Commande d'un robot mobile, avec évitement d'obstacles par la logique floue** » UMK Biskra-Algérie.

[11] : Sick. **Scanners de mesure à laser**, 2004.

Site WEB: http://www.sick.fr/fr/produits/vision/lasermeasurementinterface/fr.html.

[12] : Olivier Buffet « **Apprentissage par renforcement dans un système multi-agents** » Université Henri Poincaré-Nancy1, France. 2000.

Résumé :

L'apprentissage est un processus visant à améliorer les performances d'un système en se basant sur ses expériences passées. Cette méthode intervient lorsque le problème paraît trop compliqué à résoudre en temps réel, ou lorsqu'il paraît impossible de résoudre le problème de manière classique et rigoureuse. Comme exemple de méthodes d'apprentissage on cite l'***apprentissage par renforcement***.

Cette méthode d'apprentissage est souvent utilisée dans le domaine de la robotique. Elle vise à déterminer une loi de commande pour un ***robot mobile*** dans un environnement inconnu. Ce genre de technique s'applique lorsqu'on suppose que la seule information sur la qualité des actions effectuées par le ***robot mobile***, est un signal scalaire qui présente une récompense ou une punition, la procédure d'apprentissage vise à améliorer le choix des actions afin de maximiser les récompenses.

L'un des plus algorithmes utilisés pour la résolution de ce problème d'apprentissage est l'algorithme ***Q-Learning*** qui est basé sur la Q-Fonction, et pour assurer la génération de cette dernière fonction et le bon fonctionnement du système d'apprentissage on utilise un ***réseaux de neurones artificiels*** car les états des environnements où évolue un ***robot mobile*** ont des grands espaces, l'action effectuée par le ***robot mobile*** dans son environnement est assurée par l'utilisation d'une fonction de sélection, cette action est évaluée par un signal scalaire qui vaut -1, 0 et 1.

***Mots Clé* :** Apprentissage par renforcement, Q-Learning, Réseaux de Neurones Artificiels, Robot Mobile.