REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université El Hadj Lakhdar Batna

Faculté des Sciences de l'Ingénieur
Département d'Electronique

Option : Robotique

Mémoire

Présenté par :
Mezaache Hatem
Ingénieur d'état en Electronique, Option Contrôle Université de Batna

Mémoire soumis en vue de l'obtention du Diplôme de

Magister

THEME

Les réseaux de Neurones formels Et Les systèmes Neuro-Flous Pour l'apprentissage par renforcement

---

Soutenu le : 03/07/2008

Jury d'Examen :

Dr : Lamir Saidi	*Maître de Conférences Université de Batna*	***Président.***
Dr : Foudil Abdessemed	*Maître de Conférences Université de Batna*	***Rapporteur.***
Dr : Yassine Abdessemed	*Maître de Conférences Université de Batna*	***Co -Rapporteur.***
Dr : Ahmed Louchene	*Maître de Conférences Université de Batna*	***Examinateur.***
Dr : Djamel Eddine Ameddah	*Maître de Conférences Université de Batna*	***Examinateur.***
Dr : Khair-Eddine Chafaa	*Maître de Conférences Université de M'sila*	***Examinateur.***

Année universitaire: *2007/2008*

REMERCIMENT

En premier lieu, je remercie DIEU de m'avoir aidé et donner la force et la volonté pour achever ce modeste travail.

Par la suite ce travail a été réalisé sous la direction du monsieur ***Foudil Abdessemed*** *qu'il trouve ici ma profonde reconnaissance et mes sincères remerciements, pour ses encouragements, son aide, ses conseils précieux et aussi ses idées pour la réalisation de ce mémoire.*

J'adresse aussi mes sincères remerciements à monsieur ***Ahmed Louchene*** *de m'avoir autoriser l'accès au* ***L****aboratoire d'****E****lectronique* ***A****vancée.*

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

-- M^{er}. ***Lamir Saidi****, Maître de Conférences à l'université de Batna, pour l'honneur qu'il me fait de présider le jury.*

- M^{er}. ***Foudil Abdessemed****, Maître de Conférences à l'université de Batna, d'avoir accepté la lourde tâche d'être rapporteur.*

- M^{er}. ***Yassine Abdessemed****, Maître de Conférences à l'université de Batna, d'avoir accepté la lourde tâche d'être co-rapporteur.*

- M^{er}. ***Ahmed Louchene****, Maître de Conférences à l'université de Batna, d'avoir accepté d'être examinateur.*

- M^{er}. ***Djamel Eddine Ameddah****, Maître de Conférences à l'université de Batna, d'avoir accepté d'être examinateur.*

- M^{er}. ***Khair-Eddine Chafaa****, Maître de Conférences à l'université de M'sila, d'avoir accepté d'être examinateur.*

Et enfin, je tien aussi à remercier ma femme pour ses encouragements et sa grande patience avec moi, pour la réalisation de ce modeste travail.

M. Hatem

Dédicace

A la mémoire de ma très, très, très chère mère.

A ma femme et mes très chers petits enfants :
Bahia Rayane, Oussama Islam et Ritedj Hibatou Allah.

A ma très chère grande mère et mon père.

A mes très chers frères Walid, Abdel Aziz , Larbi et sa petite Hafssa.

A toutes ma famille.

A tout mes Amis surtout D. Mohamed, A. Kamel, M. Abdelhamid, K. Fouzi, B. Abdessalem, D. Djalel, Z. Ghania et D. Salima.

Et à ma belle aimée

l'Algérie.

M. Hatem

Sommaire

Introduction Générale.

Chapitre 1

Apprentissage par renforcement

1.1- Introduction 01

1.2- C'est quoi l'apprentissage 01

1.2.1-L'apprentissage supervisé 01

1.2.2- L'apprentissage non supervisé 02

1.2.3- L'apprentissage par renforcement 02

1.3- Problème d'apprentissage par renforcement 02

1.3.1- Principe 02

1.3.2- Formalisation 03

1.3.3- Processus Décisionnels de Markov 03

1.3.3. a- Politique 04

1.3.3. b- Fonctions valeur 05

1.3.3. c- Fonction «Valeur - état» 05

1.3.3. d- Fonction «valeur état - action» 06

1.3.3. e- Fonction de valeur optimale 06

1.4- Système d'apprentissage par renforcement 07

1.4.1-Agent 07

1.4.2- Le temps 07

1.4.3- Les états 08

1.4.4- Les actions 08

1.4.5- Le signal ou la fonction de renforcement 08

1.4.6- L'environnement 09

1.5- Méthodes de résolution..... 10

1.5.1- Méthode de la Programmation Dynamique 11

1.5.1. a- Evaluation d'une politique..... 11

1.5.1. b- Amélioration d'une politique..... 12

1.5.2- Méthode de Monté Carlo..... 13

1.5.3- Méthode de Différence Temporelles..... 14

1.5.3. a -Q-Learning..... 15

1.6- Conclusion 17

Chapitre 2

Système d'inférence floue et le Q Learning floue

2.1-Introduction 18

2.2- Rappel sur la logique flou..... 18

2.2.1-Structure générale des systèmes d'inférence flou 19

2.2.1. a- Base de connaissance..... 19

2.2.1. b- Moteur d'Inférence 20

2.2.1. c- Interface de Fuzzification 20

2.2.1. d- Interface de Défuzzification 20

2.3-Type des Systèmes d'Inférence Flou (SIF) 20

2.3.1- SIF de type Mamdani 21

2.3.2- SIF de type Takagi – Sugeno 22

2.4-Caractéristiques d'un système d'inférence flou 23

2.4.1- Caractéristiques structurelles..... 23

2.4.2- Caractéristiques paramétriques..... 24

2.5-L'apprentissage par renforcement de système d'inférence flou..... 24

2.5.1- L'algorithme Fuzzy Q-Learning 24
2.5.2-Extraction de connaissance 25
2.5.3-Fusion des connaissances 26
2.5.4-Mise à jour des qualités 28
2.6-Exemple d'application 29
2.6.1- Système d'Inférence Floue 30
2.6.2-Modèle du robot 30
2.6.3-Fonction de renforcement 31
2.6.4- Q-Learning floue 31
2.7- Conclusion 32

Chapitre 3

Réseaux de neurones artificiels

3.1- Introduction 33
3.2- Neurone Biologique 33
3.3- Neurone formel 34
3.3.1- Poids de connexion 35
3.3.2- Les entrées 35
3.3.3- Fonction d'activation 35
3.3.3. a- Fonction Seuil 35
3.3.3. b- Fonction linéaire 36
3.3.3. c- Fonction Linéaire à seuil 37
3.3.3. d- Fonction sigmoïde 37
3.3.4- Fonction de sortie 37
3.4- Description mathématique 38
3.5-Réseaux de neurones artificiels 39

3.6-Différentes types de réseaux de neurones artificiels 39
3.6.1- Les réseaux proactifs 40
3.6.1. a- Les réseaux proactifs monocouches 40
3.6.1. b- Réseaux proactifs multicouches 41
3.6.2- Réseaux récurrents 42
3.7- Quelques Modèles des Réseaux de Neurones Artificiels 43
3.7.1- Les cartes auto organisatrices de Kohonen 43
3.7.2- Les réseaux de Hopfield 43
3.7.3- Les réseaux ART 44
3.7.4- Le modèle Adeline 44
3.7.5- Le perceptron multicouches 45
3.7.5. a- Le perceptron monocouche 45
3.7.5. b-Les perceptrons multicouches 45
3.8-Apprentissage 45
3.9-Algorithme de rétro propagation du gradient 46
3.9.1- Principe 46
3.9.2- Algorithme 47
3.9.2. a- Définition du réseau 47
3.9.2. b- Les Etapes de l'algorithme 48
3.9.3- Sommaire de l'algorithme Rétropropagation de l'erreur 49
3.10- Application des réseaux de neurones 49
3.11- Conclusion 50

Chapitre 4

Conception d'un système d'apprentissage par renforcement

4.1-Introduction 51

4.2-Description des fonctions du système d'apprentissage par renforcement 51
4.2.1- Environnement 51
4.2.2-Fonction de renforcement 52
4.2.3-Fonction de valeur 52
4.3- Approximation de la fonction valeur 52
4.4-Génération de la fonction valeur 54
4.4.1-Tableau de consultation 54
4.4.2- Réseau de Neurones Artificiel 56
4.5-Conclusion 60

Chapitre 5

Application et résultats

5.1-Introduction 62
5.2-Architecture du système Apprentissage par renforcement proposé 62
5.2.1- Environnement 63
5.2.2- Fonction du renforcement 67
5.2.3- Fonction de valeur 67
5.2.4-Fonction de sélection d'action 69
5.3-Apprentissage du réseau 70
5.3.1- La couche de sortie 70
5.3.2- La couche cachée 70
5.4-Modèle du Robot 71
5.4.1- La position du robot 72
5.4.2- L'orientation du robot 72
5.4.3- la vitesse linière du robot 72
5.4.4- La variation de thêta 72

5.5-Algorithme et résultat de simulation 73

5.5.1- Algorithme..... 73

5.5.2- Quelques Résultats de simulation..... 74

5.6- Conclusion 76

Conclusion Générale.

Liste des Figures.

Liste des Tableaux.

Bibliographie.

Introduction Générale

Au cours de ces derniers années, l'apprentissage des machines est l'un des domaines les plus ressens. Pour réaliser cette tâche, il existe plusieurs outils comme la logique floue, les réseaux de neurones artificiels et les algorithmes génétiques qui permets d'avoir un système de commande Intelligent, de plus il existe aussi plusieurs méthodes d'apprentissage, qui sont généralement liées aux sources d'informations utilisées par l'agent. En effet l'agent peut utiliser soit des exemples fournis par un professeur, soit un signal sous forme de récompenses et de punitions qui lui permet de corriger les actions qui a exécuté dans son environnement de travail.

Dans notre étude, nous nous sommes intéressés à cette dernière méthode qui est appelée ***apprentissage par renforcement***. Dans l'apprentissage par renforcement l'agent ne dispose plus d'exemples de commande ou de trajectoires de référence. L'apprentissage est réalisé grâce à une évaluation des résultats.

Dans le cadre de notre mémoire nous avons utilisé la méthode d'apprentissage par renforcement, basée sur l'algorithme ***Q-Learning*** avec les réseaux de neurones artificiels de type ***MLP***.

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

Le premier chapitre donne une description générale sur les bases, le principe de la méthode d'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 qui peut être utilisé pour la réalisation ou la conception d'un système d'apprentissage par renforcement cet outil est la logique flou spécialement on utilisant un Système d'Inférence Floue.

Par la suite le chapitre trois est aussi consacrer pour présenter un autre outil qui peut être utilisé dans cette méthode d'apprentissage, cet outil sont les réseaux de neurones artificiels où nous avons présenté leurs fondements mathématique, leurs models,…etc.

Dans le chapitre quatre nous présenterons une méthode pour la conception d'un système d'apprentissage par renforcement, cette méthode est basée sur l'utilisation des réseaux de neurones artificiels de type perceptron multicouches (**PMC**).

Le cinquième chapitre est le fruit de notre travail, où on a proposé un système d'apprentissage par renforcement qui utilise un réseau de neurones artificiel de type **MLP**. A la suite nous présentons quelques résultats de simulation et à la fin nous terminerons notre travail par une conclusion générale.

Chapitre 1 Apprentissage par renforcement

1.1-Introduction :

Ce chapitre est consacré pour la présentation des types d'apprentissage qui existent, ainsi pour la description de la technique d'apprentissage par renforcement et ces éléments essentiels.

1.2-C'est quoi l'apprentissage :

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.

Il existe trois grandes familles de méthodes d'apprentissage: l'apprentissage supervisé, l'apprentissage non supervisé, et l'apprentissage par renforcement.

1.2.1- L'apprentissage supervisé :

C'est l'apprentissage le plus couramment utilisé et il est très bien maîtrisé. Son inconvénient est que l'agent n'est pas immédiatement autonome, puisqu'il a besoin d'un superviseur qui dans un premier temps lui indique la marche à suivre dans des situations qu'il pourra rencontrer.

Si la base d'apprentissage est complète, l'agent saura réagir aux situations auxquelles il sera confronté.

Cependant les situations doivent présenter une certaine constance : si l'agent est confronté à une situation entièrement nouvelle, il sera incapable de s'y adapter car aucun exemple donné par le superviseur n'y correspondra.

1.2.2-L'apprentissage non supervisé :

Contrairement à l'apprentissage supervisé, on ne fournit ici qu'une base d'entrées, et c'est le système qui doit déterminer ses sorties en fonction des similarités détectées entres les différentes entrées (règle d'auto organisation).

1.2.3-L'apprentissage par renforcement :

Imaginant un agent, capable de percevoir et agir, qui va choisir ses actions en fonction de sa perception de son environnement et dont le but va être de maximiser une récompense qu'il obtient en fonction des actions qu'il réalise. **[1]**

Dans ce contexte, l'apprentissage par renforcement est une technique qui permet de trouver par un processus d'***essais et d'erreurs***, l'action optimale à effectuer pour chacune des situations « états» que l'agent va percevoir afin de maximiser ses récompenses. **[2]** C'est une méthode d'apprentissage ***orienté objectif*** qui conduit à un contrôleur optimal pour la tâche spécifiée par les récompenses. Elle est aussi non supervisée car la récompense ne donne pas l'action optimale à réaliser mais simplement une évaluation de la qualité de l'action choisie. **[2]**

1.3- Problème d'apprentissage par renforcement :

1.3.1-Principe :

L'apprentissage par renforcement définit un type d'interaction entre l'agent et l'environnement à partir d'un état ou une situation **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**. Ce signal est utilisé par l'agent pour améliorer sa stratégie « politique », c'est-à-dire la séquence de ses actions, afin de maximiser ses récompenses futures. **[3]**

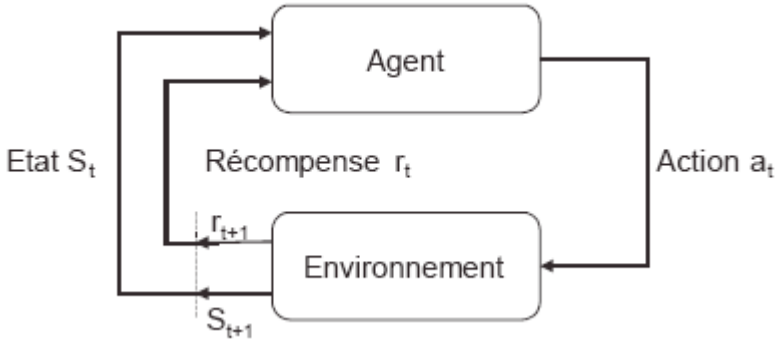


Figure 1.1 : Système d'apprentissage par renforcement

1.3.2-Formalisation :

Pour un agent le problème de l'apprentissage par renforcement est définit par l'utilisation des éléments suivants :

- Un ensemble d'états S correspondant à la perception de l'agent à son environnement.
- Un ensemble d'actions possibles A.
- Une fonction de renforcement R.

Le but de l'agent est de choisir les actions qui lui amènent le plus de récompenses et le moins de punitions.

Ceci nous conduit à donner une formalisation pour ce problème d'apprentissage en utilisant un Processus de Décision Markovien (MDP).

1.3.3- Processus Décisionnels de Markov :

Le Processus Décisionnel de Markov forme le modèle le plus classique des problèmes de décision séquentielle. C'est un processus stochastique, il assigne des récompenses aux transitions des états.

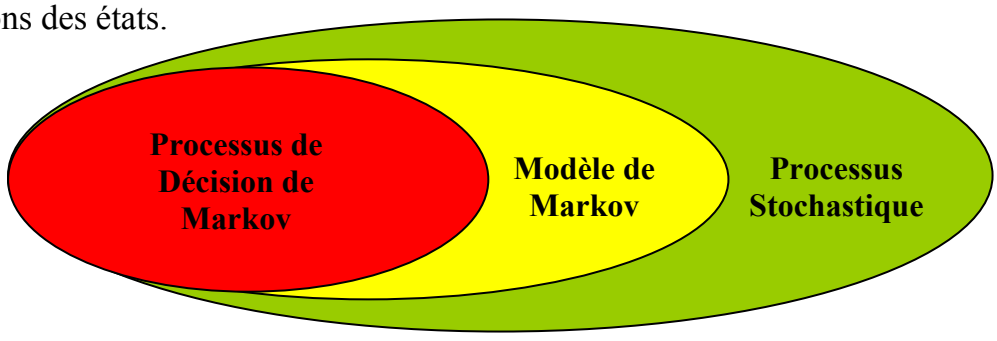


Figure 1.2 : Structure des processus stochastique

Les récompenses et les probabilités de transitions d'un état à un autre sont des fonctions de l'état courant et de l'action courante, et non de la trajectoire passée « états précédents, actions précédentes ».

Un Processus de Décision Markovien « MDP » est un modèle de Markov d'ordre 1 on le définit par :

- S : c'est l'espace d'états.
- A : c'est l'espace d'actions.
- T : c'est une fonction de transition markovienne qui est définit de $S \times A \times S \rightarrow [0 \quad 1]$, T(s, a, s') représente la probabilité d'aller de l'état « **s** » à l'état « **s'** » en effectuent l'action **a**.
- r : c'est une fonction des renforcements Markovienne qui est définit de $S \times A \times S \rightarrow R$. r(s, a, s') représente le renforcement obtenu en effectuant l'action « **a** » et en se retrouvant dans l'état « **s'** ».

Remarque :

On dit que la fonction de Transition est Markovienne car la probabilité de transition entre **s** et **s'**, lorsqu'en effectue l'action **a** ne dépend pas des états précédents et/ou des actions précédemment effectuées. On dit aussi que la fonction de renforcement est Markovienne car le renforcement reçu lors de la transition entre **s** et **s'** lorsque on effectue l'action **a** ne dépend pas des états précédemment visités et/ou des actions précédemment effectuées. **[4]**

1.3.3. a- Politique :

Une politique c'est la stratégie ou la façon qui modélise ou représente le comportement d'un agent dans un instant **t** dont il va choisir une action « **a** » à exécuter dans un état « **s** » donné. Dans le cas général il s'agit d'une fonction déterministe ou aléatoire qui est notée par π. Une politique aléatoire permet de choisir pour un état donné différentes actions suivant une probabilité, elle est définit de l'ensemble des couples état-action dans l'intervalle [0, 1].

On écrit :

π: $S \times A \rightarrow [0 \quad 1]$.

$(s,a) \mapsto \pi(s,a) = \Pr(a_t = a, s_t = s)$.

La politique π nous donne la probabilité de choisir une action donnée.

Une politique déterministe ne pose qu'une action par état elle est définit par :

π: $S \rightarrow A$.

$s \mapsto \pi(s) = a$.

La politique est le cœur du processus de l'apprentissage par renforcement puisqu'elle est suffisante pour déterminer le comportement de l'agent.

1.3.3. b- Fonctions valeur :

C'est une fonction d'estimation où la plupart des problèmes d'apprentissage par renforcement sont basés sur elle. Elle est définit pour un ***état*** ou une paire « ***état***, ***action*** », cette fonction permet à l'agent de s'avoir comment il apprend à choisir les bonnes actions et comment mesurer leurs utilité.

Une fonction valeur est optimale si elle est égale à la somme pondérés des renforcements reçus depuis l'état initiale jusqu'à l'état final en exécutent des actions optimales **[Sutton and Barto, 1998]**.

1.3.3. c- Fonction « Valeur - état » :

La valeur d'un état $s \in S$ en appliquant la politique $\boldsymbol{\pi}$ est notée par $V^{\pi}(s)$, c'est une fonction des états dans les réels R, $V^{\pi} : S \rightarrow R$.Elle est définit comme suit :

$$V^{\pi}(s) = E_{\pi}\{R_t / s_t = s\} \qquad \textbf{1.1}$$

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} / s_t = s\right\} \qquad \textbf{1.2}$$

Avec $0 \leq \gamma < 1$.

Cette fonction est l'espérance (environnement non-déterministe) des renforcements actualisés lorsqu'on part de l'état s (**s_t=s**) et on suit la politique π par la suite.

Si l'environnement est déterministe cette fonction s'écrit comme suit :

$$V^{\pi}(s)=\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} / s_t = s\right\} \qquad \textbf{1.3}$$

Avec $0 \le \gamma < 1$.

De plus cette valeur ne dépend que de l'état s dans le quel on se trouve, car le renforcement dans cet état ne dépend pas des états visités avant d'arriver à l'état s.

1.3.3. d- Fonction « valeur état-action » :

Nous allons donner une notre fonction de valeur qui représente la valeur d'un couple « **état-action** » $(s,a) \in S \times A$ pour une politique π. Cette fonction est notée par $Q^{\pi}(s,a)$, elle est définit de l'ensemble état-action dans les réels, $Q^{\pi} : S \times A \to R$ donc cette fonction ce définit comme suit :

$$Q^{\pi}(s,a) = E_{\pi}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... / s_t = s, a_t = a\right\} \qquad \textbf{1.4}$$

$$Q^{\pi}(s,a) = E_{\pi}\left\{\sum \gamma^{k} r_{t+k+1} / s_t = s, a_t = a\right\} \qquad \textbf{1.5}$$

Avec $0 \le \gamma < 1$.

Cette fonction est l'espérance (environnement non déterministe) de la somme des renforcements actualisés lorsqu'on part de l'état **s** et on exécute l'action **a** en **s** ($\mathbf{s_t}$=**s**) et en suivant la politique π à l'instant **t**.

Si l'environnement est déterministe cette fonction s'écrit comme suit :

$$Q^{\pi}(s,a) = \left\{\sum \gamma^{k} r_{t+k+3} / s_t = s, a_t = a\right\} \qquad \textbf{1.6}$$

Avec $0 \le \gamma < 1$.

1.3.3. e- Fonction de valeur optimale :

La résolution du problème d'apprentissage par renforcement consiste à trouver une politique optimale qui donne un maximum de récompenses à l'agent.

Une politique π est supérieure à une politique π' si : $V^{\pi'}(s) \leq V^{\pi}(s)$ pour les états possibles et on note cela par π'≤π. Il existe au moins une politique qui est meilleure de toutes les autres politique, appelée **politique optimale** qui est noté par π*, cette politique π* à une fonction valeur d'état notée par V^* appelée **fonction de valeur d'état optimale** tel que :

$$V^*(s) = \max_{\pi} \left[V^{\pi}(s) \right] \qquad \textbf{1.7}$$

Cette politique à aussi une **fonction de valeur d'état-action optimale** notée Q^* est défini par :

$$Q^*(s) = \max_{\pi} \left[Q^{\pi}(s,a) \right] \qquad \textbf{1.8}$$

1.4-Systeme d'apprentissage par renforcement :

Dans le paragraphe précèdent nous avons présenté une formalisation du problème d'apprentissage par renforcement. Ceci nous permet maintenant de donner une présentation à ce système d'apprentissage qui est basé sur les éléments suivants :

1.4.1-Agent :

On appelle agent toute entité physique ou virtuelle qui est:

a- Capable d'agir dans un environnement.

b- Capable de percevoir son environnement.

c- Possède des ressources propres.

Ces caractéristiques peuvent être dotées à un agent et il existe d'autres.

1.4.2- Le temps :

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

- Discret ou continu.

- Fini ou infini.

- Déterministe ou aléatoire.

La plupart des études sur l'apprentissage par renforcement utilisent un espace de temps discret.

1.4.3- Les états :

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

-Une situation relationnelle de l'agent par rapport à l'environnement (position).

-Une situation propre à l'environnement (Modification du milieu).

-Une situation interne à l'agent (sa mémoire, ses capteurs…etc.).

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

1.4.4-Les actions :

L'agent choisit une action parmi les actions possible 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 suivante :

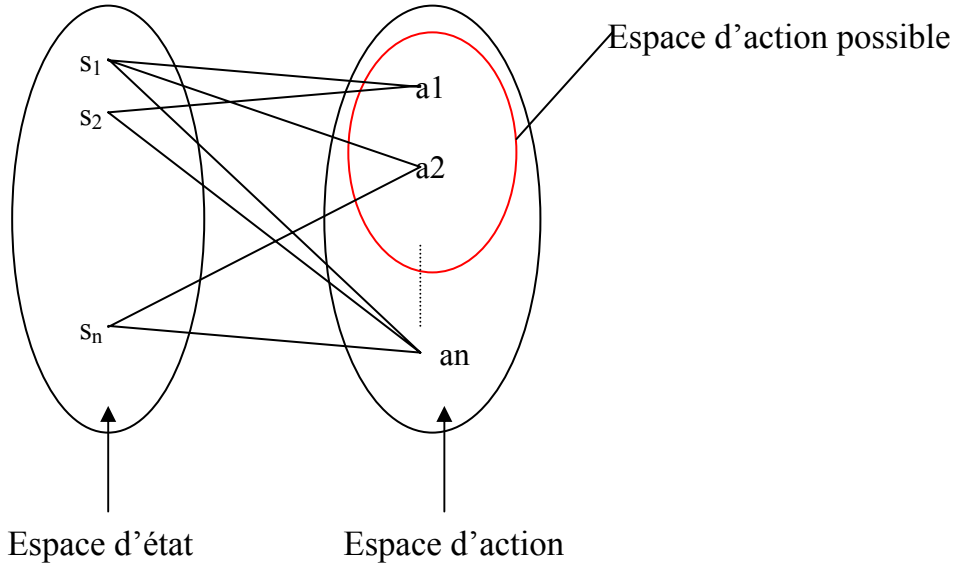


Figure 1.3 : Relation états, actions

1.4.5-Le signal ou la fonction de renforcement :

A chaque instant d'interaction, une valeur de renforcement $\mathbf{r_t}$ est produite, qui est une valeur numérique bornée qui mesure la justesse de la 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 qui sera reçu dans le futur :

$$R_t = r_{t+1} + r_{t+2} + ... + r_T \qquad \textbf{1.9}$$

Où T est l'instant final qui met fin à l'interaction.

Pour éviter la divergence de R_t on utilisera un facteur de pondération γ, tel que : $\gamma \in [0, \quad 1[$

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad \textbf{1.10}$$

De plus ce facteur de pondération détermine la valeur présente d'une récompense future

1.4.6-L'environnement :

Initialement, l'agent est mis dans un espace de travail dynamique appelé environnement. Le but ou l'objectif du système d'apprentissage par renforcement est d'apprendre une projection topologique des situations « états » vers les actions par interactions ***essa***i et ***erreur*** avec cet environnement. Le système d'apprentissage par renforcement doit être capable d'observer parfaitement toutes les informations de l'environnement il est alors possible de choisir une action pertinente basée sur les états réels. Les observations « états ou situations » de l'agent peuvent être les résultats des mesures de capteurs, ainsi les actions peuvent être des tâches à exécuter, un changement de tension ou courant des moteurs, ….etc.

Il existe plusieurs types d'environnements qui sont utilisés pour les méthodes d'apprentissage ; nous citons par exemples :

- **Continu / Discret :** S'il existe un nombre limité de perceptions et d'actions possibles, on parle donc d'un environnement discret.
- **Dynamique / statique :** 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.

- **Déterministe / non déterministe :** On parle d'un environnement déterministe s'il est complètement déterminé par son état courant et par l'action sélectionnée par l'agent.

Donc on peut donner un modèle standard de l'apprentissage par renforcement, on sait q'il existe des interactions entre l'agent et son environnement généralement sous formes de mesures des capteurs. L'agent doit alors choisir une action de telle manière à améliorer les performances de son comportement. Chaque action doit être jugée par un signal de renforcement. On trouve donc trois parties fondamentales qui réalisent un système d'apprentissage par renforcement qui sont : Environnement, Fonction ou signal de renforcement, Fonction de valeur.

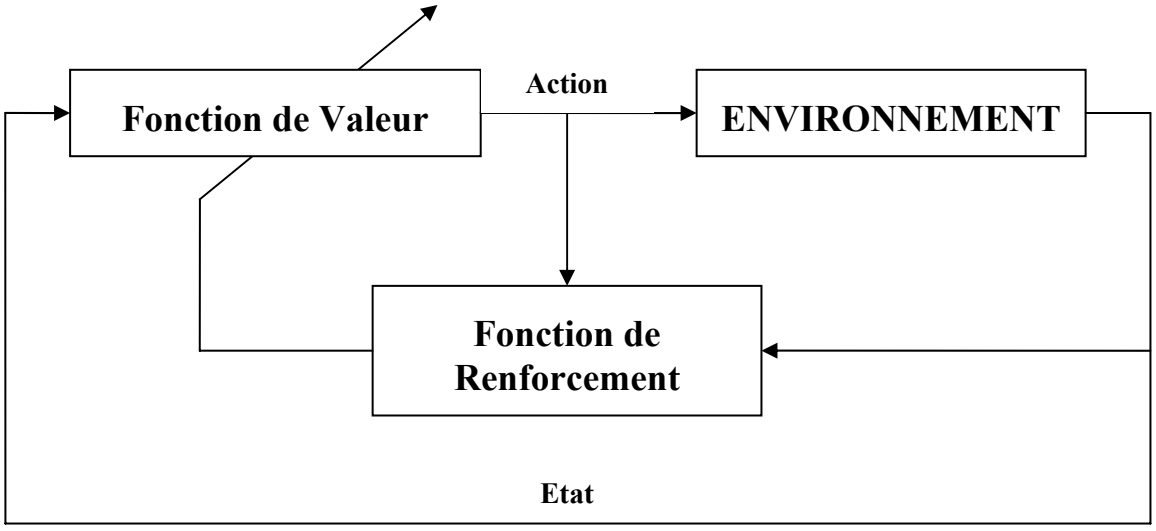


Figure 1.4 : Parties d'un système d'apprentissage par renforcement

1.5-Methodes de résolution :

La résolution du problème d'apprentissage par renforcement est basée sur l'une des méthodes suivantes :

- Méthode de **la programmation dynamique** « **PD** ».
- Méthode de **Monte Carlo** « **MC** ».
- Méthode des **différences temporelles** « **TD** ».

Chaque méthode possède des avantages et des inconvénients.

1.5.1- Méthode de la Programmation Dynamique :

La programmation dynamique est une méthode de résolution des problèmes d'optimisation due essentiellement à R. Bellman [1957]. Elle correspond à un ensemble ou une collection d'algorithmes qui sont basés sur la présence d'un modèle parfait de l'environnement pour calculer une politique / stratégie optimale.

Les algorithmes utilisés en ***Programmation Dynamique*** pour la résolution du problème d'Apprentissage par Renforcement ont une utilisation limités à cause de :

- Leurs besoins d'un modèle parfait pour l'environnement.
- Ils ont un coût de calcul élevé.

Mais ces algorithmes de la programmation dynamique restent avec une grande utilité et importance théorique car ils sont bien développés mathématiquement.

L'idée principale ou générale de la programmation dynamique est de déterminer une politique optimale, ceci est fait par le calcul des fonctions valeurs optimale V^* ou Q^* en utilisant les équations de Bellman. La recherche d'une politique optimale est basée sur deux phases ou étapes suivantes:

1.5.1. a- Evaluation d'une politique :

La première étape de la programmation dynamique est l'estimation de la fonction valeur pour une politique π donnée. Ceci est fait soit en résolvant le système d'équation de Bellman, tel que :

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s,a) \sum_{s' \in S} T(s,a,s') \left[R(s,a,s') + \gamma V^{\pi}(s') \right] \forall s \in S \qquad \textbf{1.11}$$

Cette équation nous donne un système de **s** équation linéaire.

Où la fonction valeur d'état et donnée par :

$$V^{\pi}(S) = E_{\pi} \left\{ r_{t+1} + \gamma \sum \gamma^{k} r_{t+k+2} \quad / s_t = s \right\} \qquad \textbf{1.12}$$

Soit en utilisant un algorithme d'évaluation itérative d'une politique.

1.5.1. b- Amélioration d'une politique :

L'étape suivante de la programmation dynamique consiste à améliorer la politique π. Après avoir calculé V^{π} dans l'étape précédente pour un état donné « s » selon la politique π, serait il mieux de choisir une action $a \neq \pi(a)$?

On sait que :

$$Q^{\pi}(s,a) = E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s') \quad / s_t = s, a_t = a\right\} \qquad \textbf{1.13}$$

$$Q^{\pi}(s,a) = \sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi}(s')\right] \qquad \textbf{1.14}$$

Si on a : $Q^{\pi}(s,a) > V^{\pi}$ il est mieux de basculer vers l'action « a » pour l'état « s », cette opération doit être effectuée pour tout les états, ceci nous conduit d'avoir une nouvelle politique π' au lieu de π.

Avec :

$$\begin{aligned} \pi'(s) &= \arg\max_{a} Q^{\pi}(s,a) \\ \pi'(s) &= \arg\max_{a} E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s') \quad / s_t = s, a_t = a\right\} \\ \pi'(s) &= \arg\max_{a} \sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi}(s')\right] \end{aligned} \qquad \textbf{1.15}$$

Où

argmax : Désigne l'action qui maximise l'expression $\left(T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi}(s')\right]\right)$ pour toutes les actions « a » possibles.

Alors :

$\forall s \in S : Q^{\pi}\left(s, \pi'(s)\right) \geq V^{\pi}(s)$

Si la nouvelle politique π' égale à π ($\forall s \in S : \pi(s) = \pi'(s)$) alors on atteint la politique optimale.

Donc on a :

$$\forall s \in S : V^{\pi}(s) = \arg\max_{a} \sum_{s' \in S} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi}(s')\right] \qquad \textbf{1.16}$$

Alors on peut donner le schéma suivant qui donne un aperçu sur la méthode suivi pour déterminer une politique optimale suivant la programmation dynamique.

$$\pi_0 \xrightarrow{E} V^{\pi 0} \xrightarrow{A} \pi_1 \xrightarrow{E} V^{\pi 1} \xrightarrow{A} \pi_2 \xrightarrow{E} \ldots\ldots\ldots\ldots \xrightarrow{A} \pi^* \xrightarrow{E} V^*$$

Où :

$\xrightarrow{E}$ Evaluation d'une politique.

$\xrightarrow{A}$ Amélioration d'une politique.

1.5.2- Méthode de Monté Carlo :

Les méthodes de ***Monté Carlo*** utilisées pour l'apprentissage par renforcement ont étés identifiés dans les années 90. La propriété de convergence de ces méthodes n'est pas encore claire, leurs utilisations en pratique est peu. Ces méthodes de ***Monté Carlo*** ne nécessitent pas un modèle de l'environnement, elles sont utilisées pour l'apprentissage des fonctions valeurs $V^{\pi}(s)$ et $Q^{\pi}(s,a)$ est pour la découverte de la politique optimale ces méthodes consistent à :

- Utiliser des simulations ou des résultats réels « expériences » pour les interactions de l'agent et de l'environnement.
- Tenir ou enregistrer ces résultats de simulations ou d'expériences.
- Calcul des moyennes de ces résultats de simulations ou d'expériences.

Supposons que nous voulions évaluer une politique π, ceci est réalisé par l'estimation de la fonction valeur d'un état « s » $V^{\pi}(s)$ selon la politique π. On va lancer un ensemble de séquences « état - action - récompense……. » réalisées par l'agent suivant la politique π, les résultats obtenues pour chaque état de ces séquences seront enregistrer, alors il est possible d'estimer $V(s)$ par la moyenne des résultats enregistrer pour chaque état « s » donc on peut écrire :

$$V^{\pi}(s) = \frac{1}{n}\sum_{n} R_t^n \qquad \textbf{1.17}$$

Avec :

$$R_t^n = r_{t+1}^n + \gamma r_{t+2}^n + \gamma^2 r_{t+3}^n + \gamma^3 r_{t+4}^n + \ldots\ldots + \gamma^T r_T^n \qquad \textbf{118}$$

C'est-à-dire la somme pondérée des récompenses après la visite de l'état « s ». Cette méthode s'applique de la même façon pour la fonction $Q(s,a)$. L'utilisation des expériences réalisées par l'agent dans son environnement et l'estimation des fonctions valeurs $V(s)$ et $Q(s,a)$ permettent de déterminer la politique optimale.

1.5.3- Méthode de Différence Temporelles :

Si on voudrait identifier une idée centrale et nouvelle de l'apprentissage par renforcement, c'est sans doute la méthode des ***Différences Temporelles*** qui à été développer par **R. Sutton**.

Les méthodes d'apprentissage par Différences Temporelles sont basées sur l'association des idées des méthodes de la **Programmation Dynamique** et méthodes de **Monté Carlo**.

Les méthodes d'apprentissage par ***Différences Temporelles*** apprennent à partir de l'expérience directe de l'agent sans avoir un modèle de l'environnement comme les méthodes de **Monté Carlo**.

Pour l'estimation de la valeur d'un état, les méthodes d'apprentissage par Différences temporelles ce fait en fonctions des états voisins sans dépendre des états finals comme les méthodes de la Programmation Dynamique.

Les méthodes Différences Temporelles calculent pendant une séquence les nouvelles évaluations à partir des évaluations précédentes.

Soit « **s** » un état non terminal visité à l'instant « t » la mise à jour de la fonction V^π est menée sur la base de ce qui arrive après cette visite. **[3]**

$$V_{t+1}^\pi(s) \leftarrow V^\pi(s) + \alpha\left[r_{t+1} + \gamma V_t^\pi(s') - V_t^\pi(s)\right] \qquad \textbf{1.19}$$

Ainsi l'erreur de prédiction du gain estimé est :

$$r_{t+1} + \gamma V_t^\pi(s') - V_t^\pi(s) \qquad \textbf{1.20}$$

L'algorithme de prédiction des méthodes Différences Temporelles est le suivant :

Initialisation arbitraire de $V^{\pi}(s)$, π est la politique à évoluer.
Répéter (pour chaque épisode)
Initialiser s
Répéter pour chaque itération de l'épisode.
a ← action donnée par π pour l'état s.
Exécuter l'action a, Observer la récompense r et l'état suivant s'.

$$V(s) \leftarrow V(s) + \alpha\left[r_{t+1} + \gamma V(s') - V(s)\right]$$

$$s \leftarrow s'$$

Jusqu'à l'état terminal s.

Tableau 1.1 : L'algorithme de prédiction des méthodes Différences Temporelles.

Dans les méthodes de Différence Temporelles l'évaluation ce fait à un pas de temps et elles n'ont pas besoins d'un modèle de l'environnement. Plusieurs algorithmes sont liés à ces méthodes par exemple : SARSA, TD(λ), R-Learning, Q-Learning……..

Dans le paragraphe suivant nous allons donner une bonne description de l'algorithme

Q-Learning.

1.5.3. a- Q-Learning :

Définition : L'algorithme Q-Learning se classe dans les méthodes de Différence Temporelles comme une méthode de résolution du problème d'apprentissage par renforcement pour un modèle d'environnement inconnu, c'est une méthode qui se base sur l'équation réactualisation :

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q_t(s',a') - Q_t(s,a)\right] \qquad \textbf{1.21}$$

De plus cette méthode est basée sur les fonctions suivantes :

- **Une fonction de sélection :**

A partir de l'état actuel qui est observé par l'agent, une action est choisie et exécutée en se basant sur la connaissance disponible au sein de la mémoire interne (Cette connaissance est stockée sous forme de valeur d'utilité associée à une paire « Etat, Action »).

- **Une fonction de renforcement :**

Après l'exécution de l'action dans le monde réel. La fonction de renforcement utilise la nouvelle situation pour générer la valeur de renforcement. Ce renforcement prend en général une simple valeur +1, -1 ou 0.

- **Une fonction de mise à jour :**

Cette fonction utilise la valeur de renforcement pour ajuster la valeur associée à l'état ou bien à la paire « Etat, Action » qui vient d'être exécutée.

Le ***Q-Learning*** a comme principe l'estimation de la fonction Q^* qui est définie par :

$$Q^*(s,a) = E\left[r_{t+1} + \gamma V^*(s)\right] = E\left[r_{t+1} + \gamma \max_{a'} Q^*(s',a')\right] \qquad \textbf{1.22}$$

En utilisant l'équation de réactualisation on trouve que :

$$Q^{\pi}_{t+1}(s_t,a_t) = (1-\alpha_t)Q^{\pi}_t(s_t,a_t) + \alpha_t \quad \left[r_{t+1} + \gamma \max_{a'}\left(Q^{\pi}_t(s',a')\right)\right] \qquad \textbf{1.23}$$

Où :

r_{t+1} est le renforcement reçu quand on choisie l'action « a » dans l'état « s » pour passer à l'état « s' ».

α_t est un réel positif : $\alpha_t \in]0,1[$.

En principe il faut ***explorer*** aléatoirement l'environnement pendant un grand nombre d'itérations pour que le ***Q-Learning*** converge vers la fonction *Q* optimale, en suite on peut utiliser la politique optimale définie par :

$$\pi'(s) = \arg\max_{a \in A} Q^*(s,a) \qquad \textbf{1.24}$$

- **Politique d'Exploration / Exploitation**

La politique optimale est obtenue en choisissant l'action qui, à chaque état maximise la fonction de valeur Q. Pour avoir une bonne estimation de cette fonction, on doit balayer et évoluer l'ensemble des actions possibles, pour tous les états. C'est ce que l'on appelle la phase d'exploration par rapport à l'exploitation. **[Glo. P.Y 99]**

On note ***PEE(s)*** la Politique d'Exploration / Exploitation qui permet le choix d'une action suivant cette politique dans l'état « s ».

L'algorithme de ***Q Learning*** s'écrit comme suit :

Initialisation de Q(s, a) arbitrairement.

Pour chaque épisode.
Choisir un état de départ « s ».

Pour chaque itération dans l'épisode.
Choisir « a » en fonction Q(s, a) selon la stratégie d'exploration exécuter a.

Observer s et r.

$$Q_{t+1}^{\pi}(s_t, a_t) = (1-\alpha_t) Q_t^{\pi}(s_t, a_t) + \alpha_t \left[r_{t+1} + \gamma \max_{a'} \left(Q_t^{\pi}(s', a') \right) \right]$$

$$s \leftarrow s'$$

Jusqu'à l'état final « s ».

Tableau 1.2 : Algorithme Q-Learning

1.6-Conclusion :

Dans ce chapitre nous avons essayé de donner une description générale pour le problème d'apprentissage par renforcement basée sur la transition d'un état « s » à un état « s' » en exécutant une action « a » qui ce fait suivant une politique π et qui est caractérisée par ces fonctions valeurs V et Q. Un signal de renforcement « r » juge l'action « a » exécutée par l'agent qui doit maximiser ce signal. Les méthodes utilisées pour la résolution de ce problème ont été présentées.

Chapitre 2 Systèmes d'Inférence Floue et le *Q* Learning Floue

2.1-Introduction :

La logique floue a été formalisée par ***A. Zadeh*** dans le milieu des années soixante, la logique floue est une généralisation de la logique booléenne classique. Elle ajoute cependant une fonctionnalité déterminante : la possibilité de calculer un paramètre en disant simplement dans quelle mesure il doit se trouver dans telle ou telle zone de valeur.

Dans ce chapitre on ne parlera pas en détail sur les bases de la logique floue, mais nous allons seulement présenter la structure générale d'un Système d' Inférence Floue (SIF).

2.2- Rappel sur la logique flou :

La logique floue est basée sur les variables floues qui présentent toute une gradation entre la valeur ***« vrai »*** et la valeur ***« faux »*** contrairement aux variables binaires qui sont définies par les états ***« vrai »*** ou ***« faux »*** ; ces variables floues sont aussi appelé des variables linguistique dont les valeurs sont des mots ou des expressions du langage naturel, telle que ***grand, moyen, petit, proche, loin, rapide,*** etc. De plus la logique floue est basée sur les ***systèmes d'inférence floue*** qui représentent les différentes règles linguistiques reliant les variables floues d'entrée du système aux variables floues de sortie de ce système.

Les systèmes d'inférence floue s'expriment sous forme de règles linguistiques de type

Si condition 1 ***et/ou*** condition 2 ***et/ou*** ….**Alors** conclusion. La partie « **Si**… » Est la description de l'état du système qui doit déclencher la règle et la partie « **Alors**… » Sont les actions qui doivent êtres effectuées sur le système.

Ces règles sont très proches au raisonnement humain, ce qui rend en général facile la conception d'un contrôleur flou. La figure (2.1) représente un exemple des différentes parties d'un contrôleur flou tel qu'il est défini par Mamdani **[5]**

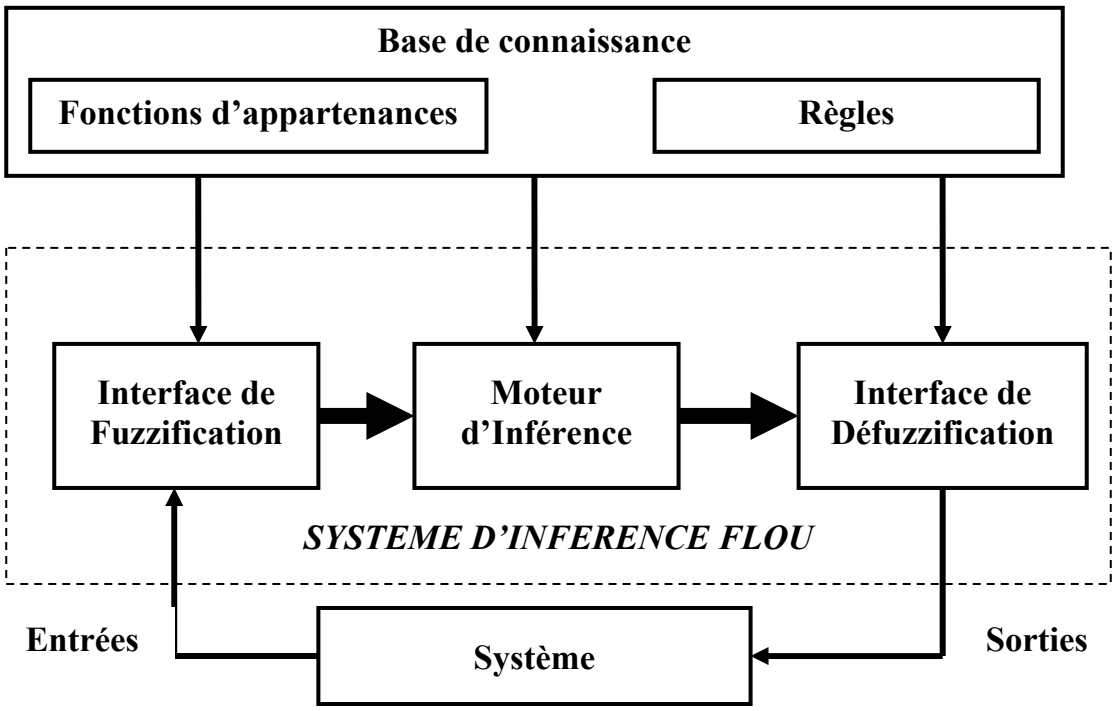


Figure 2.1: Structure générale des systèmes d'inférence flous

2.2.1-Structure générale des systèmes d'inférence flous :

Un système d'inférence flou peut être vu comme un système expert, comme le présente la figure (2.1) un système d'inférence floue peut se décomposer en quatre blocs principaux :

2.2.1. a- Base de connaissance :

Ce bloc est composé :

- Un nombre de sous ensembles flous répartis sur un univers de discours caractérisant les variables linguistiques des entrées et des sorties. **[6]**
- L'univers de discours est considéré comme le domaine de fonctionnement du système, il peut être discret ou continu.
- Les fonctions d'appartenance des entrées et des sorties sont utilisées pour la description des sous-ensembles flous, où elles sont caractérisées par : type, noyau, support et hauteur.

La base de règle constitue un ensemble d'expressions linguistiques structurées autours d'une connaissance d'expert, représentée sous formes de règles ***Si-Alors***. **[6]**

2.2.1. b- Moteur d'Inférence :

En utilisant la base de connaissance et la logique floue, le moteur d'inférence appelée aussi ***Unité de raisonnement*****,** qui représente le cerveau du contrôleur flou, détermine les commandes floues à appliquer au système.

La fonction principale d'un moteur d'inférence est de déterminer la valeur globale de la variable de sortie de contrôle calculée à partir de la contribution de chaque règle. Le moteur d'inférence est constitué de deux fonctions principales qui sont L'inférence floue et l'agrégateur de règles. **[6]**

A partir des entrées floues et les différentes règles activées cette unité génère une conclusion.

2.2.1. c- Interface de Fuzzification :

Le raisonnement flou est basé sur cette première étape, elle réalise une transformation des grandeurs physiques en grandeurs floues. A l'aide des fonctions d'appartenances étalées sur un axe appelé l'univers de discours.

2.2.1. d- Interface de Défuzzification :

Cette interface permet de transformer les grandeurs de la commande floue en grandeurs numériques. Plusieurs méthodes de défuzzification ont été proposées, parmi les plus utilisées, on cite :

- La méthode des maximums.
- La méthode de la moyenne des maximums.
- La méthode de centre de gravité.
- La méthode des hauteurs.

2.3-Type des Systèmes d'Inférence Flou (SIF) :

Après avoir décrit le fonctionnement général du raisonnement flou, il nous faut préciser ce qui définit entièrement un SIF. En premier lieu il faut préciser si le système, de par de

choix des opérateurs, appartient à une famille répandue (SIF de type Mamdani ou Takagi-Sugeno), ensuite la spécification du type de partitions utilisées et enfin l'ensemble des caractéristiques structurelles et paramétriques. **[7]**

Dans ce qui suit nous allons présenter les deux familles de SIF les plus utilisés dans le cadre du contrôle de processus et décrire leur fonctionnement. Le résultat est fourni directement par l'opérateur d'implication, sans avoir besoin de choisir un opérateur pour implémenter la T-norme. Ensuite nous préciserons ce que sont les caractéristiques paramétriques et structurelles d'un SIF.

2.3.1- SIF de type Mamdani :

Les SIF de type Mamdani se caractérisent par le choix de l'opérateur min, que cela soit pour la conjonction dans le calcul des propositions ou pour l'implication T-norme **[7]**

Soit le SIF réduit aux deux règles linguistiques suivantes :

R1 : **Si** distance est proche **e**t vitesse est élevée **alors** freinage est fort.

R2 : ***Si*** la distance est moyenne **et** vitesse est moyenne **alors** freinage est moyen.

Après observation des faits $(\{d\},\{v\})$, Le raisonnement flou se décompose comme suit :

1-Calcul des valeurs de vérité des propositions, soit :

$$\alpha_{R_1}(d,v) = \min(\mu_P(d), \mu_F(v)) \qquad \textbf{2.1}$$

$$\alpha_{R_2}(d,v) = \min(\mu_M(d), \mu_M(v)) \qquad \textbf{2.2}$$

2- Calcul de la contribution des règles par l'implication T-norme :

$$\forall f \in F, \mu_{C_1}(f) = \min(\alpha_{R_1}(d,v), \mu_F(f)) \qquad \textbf{2.3}$$

$$\forall f \in F, \mu_{C_2}(f) = \min(\alpha_{R_2}(d,v), \mu_M(f)) \qquad \textbf{2.4}$$

3- Agrégation des règles pour former la conclusion finale floue C :

$$\forall f \in F, \mu_C(f) = \max(\mu_{C_1}(f), \mu_{C_2}(f)) \qquad \textbf{2.5}$$

Avec :

d : Distance.

v : Vitesse.

f : fonction à approximer.

L'interprétation géométrique de l'inférence sur cette base de règles produite par cette méthode est présentée sur la figure (2.2) généralement l'utilisation de la méthode du centre de gravité permet d'obtenir une conclusion de nature précise à partir du sous-ensemble flou C.

Avec ce type de SIF, l'approximation de la fonction f est effectuée en cherchant la relation R (ensemble des règles R_i de type possible) qui contient la fonction f . **[7]**

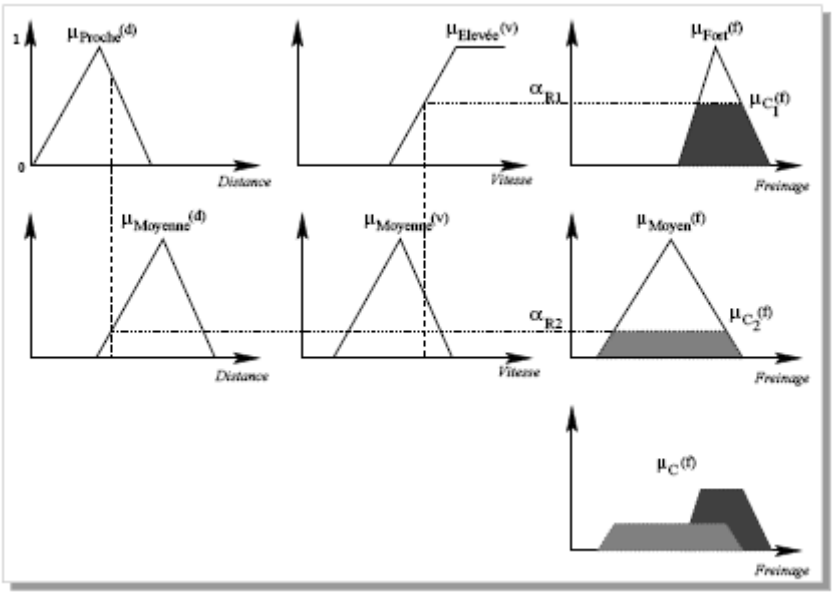


Figure 2.2: Interprétation de l'inférence dans un système de type Mamdani.

2.3.2- SIF de type Takagi-Sugeno :

Les SIF de type Takagi-Sugeno possèdent la particularité de ne pas avoir une conclusion floue. La conclusion est en fait une fonction des entrées **[7].** La base de ***N*** règles pour un vecteur d'entrée ***x*** de dimension ***n*** est donc la forme :

Si x_1 est $X_1^i \ldots$ et x_n est X_n^i alors $y^i = F_i(x)$

Avec :

$$F_i(x) = \sum_{j=1}^{n} a_j^i x_j + b^i \qquad \textbf{2.6}$$

Dans ce cas l'inférence floue est composée des étapes suivantes :

1- Calcul des valeurs de vérité des propositions avec l'opérateur de conjonction implémenté par le produit, soit :

$$\alpha_{R_i}(x) = \prod_{j=1}^{n} \mu_{X_j^i}(x_j) \text{ avec } i = 1.....N . \qquad \textbf{2.7}$$

2- Calcul de la valeur de la conclusion da chaque règle (de nature précise) avec l'implication T-norme type Larsen :

$$y^i = \alpha_{R_i}(x) \times F_i(x) \text{ avec } i = 1.....N \qquad \textbf{2.8}$$

3- Calcul de la conclusion finale non floue y par la somme pondérée des conclusions locales y^i :

$$y = \frac{\sum_{i=1}^{N} y^i}{\sum_{i=1}^{N} \alpha_{R_i}(x)} . \qquad \textbf{2.9}$$

Les SIF de type Takagi-Sugeno sont adaptés à l'approximation de fonctions et la modélisation de système non linéaire.

2.4-Caractéristiques d'un système d'inférence flou :

En tout premier lieu, un Système d'Inférence Flou est caractérisé par son type (Mamdani, Takagi-Sugeno, …) et par les partitions floues qu'il met en œuvre. Cependant, la définition totale d'un SIF passe par la spécification d'un ensemble de caractéristiques dites structurelles et paramétriques.

2.4.1- Caractéristiques structurelles :

Ces caractéristiques spécifient tous les éléments du SIF qui influent sur sa structure. Ces éléments sont constitués par :

Le type de fonction d'appartenance utilisé (Triangle, Trapèze, Sigmoïde…) pour chaque terme linguistique.

- Le nombre de termes linguistiques pour chaque variable.

- Le nombre optimal de règles.

- Les variables participant à ces règles, et les opérateurs de conjonction, de disjonction et implication.

- La technique de défuzzification.

2.4.2- Caractéristiques paramétriques :

Après avoir choisi la structure du SIF, le problème est alors le placement optimal des fonctions d'appartenance d'entrées et de sorties ou des singletons de sorties.

Les caractéristiques paramétriques se situent au plus bas niveau de spécification d'un SIF. Elles représentent en fait l'aspect purement numérique du système flou et définissent les sous-ensembles qui le constituent :

- Les paramètres des fonctions d'appartenance des variables d'entrée (point modal, base, écart type…).
- Les paramètres des fonctions d'appartenance des variables de sortie, ou les paramètres de la fonction pour les SIF de type Takagi-Sugeno.

2.5-L'apprentissage par renforcement de système d'inférence flou :

L'apprentissage par renforcement d'un Système d'Inférence Flou est basé sur leurs caractéristiques spécifiques, certains chercheurs utilisent les SIF de type Takagi-Sugeno a cause de leur large application dans le domaine de la navigation des robots mobiles.

Généralement, la structure de ces SIF est fixée a priori par l'utilisateur, qui doit indiquer les sous ensembles floues qu'il veut manipuler sur chaque entrée pour l'algorithme Fuzzy-Q-Learning (FQL). Les partitions floues sur chaque domaine d'entrée, la T-norme…..etc.

2.5.1- L'algorithme Fuzzy Q-Learning :

Le principe de l'algorithme d'apprentissage, Fuzzy-Q-Learning consiste à choisir pour chaque règle R_i une conclusion parmi un ensemble de conclusions disponibles pour cette règle. Il implémente alors un processus de compétition entre actions. La gestion de ce processus est réalisée en associant à chaque action une qualité dépendant de l'état. Cette qualité détermine alors la probabilité du choix de l'action. **[5]**

Il existe plusieurs extensions du Q-Learning pour les SIF. La première, QFUZ, n'exploite pas les propriétés d'interpolation des SIF et utilise des sorties discrètes. Deux autres extensions permettent des sorties continues : la première peut être utilisée pour de l'extraction de connaissance, la deuxième pour de la fusion de connaissances et à la fin la mise à jour des qualités. **[Glor. P.Y 99]**

2.5.2-Extraction de connaissance:

L'extraction de connaissance est l'extension la plus naturelle du Q-Learning de base. Le principe d'extraction de connaissance pour Le Q Learning Flou consiste à donner pour chaque règle plusieurs conclusions et à associer pour chaque conclusion une fonction de qualité qui sera évaluée.

Les règles sont données par :

Si s_1 est A_1^i ***et***……***et*** s_n est A_n^i ***Alors*** $y = a[i,1]$ avec $q[i,1] = 0$

Ou $y = a[i,2]$ avec $q[i,2] = 0$

…….

Ou $y = a[i,J]$ avec $q[i,J] = 0$

Où $(a[i,J])_{j=1}^{J}$ sont des solutions potentielles dont la qualité est initialisée à zéro**.**

La conclusion de chaque règle au début d'apprentissage est choisie au moyen d'une politique d'Exploration/Exploitation donnée notée par PEE : $i \to PEE(i) \in \{1.....J\}$ Dans ce cas, la sortie inférée est donnée par :

$$a(s) = \frac{\sum_i \alpha_i(s) a[i, PEE(i)]}{\sum_i \alpha_i(s)} \qquad \textbf{2.10}$$

La qualité de cette action correspondante sera alors :

$$Q(s,a) = \frac{\sum_i \alpha_i(s) q[i, PEE(i)]}{\sum_i \alpha_i(s)} \qquad \textbf{2.11}$$

Le tableau (2.1) représente l'algorithme correspondant :

1. Observer l'état s.

2. Pour chaque règle, choisir une conclusion

à l'aide d'une PEE.

3. Calculer la sortie a(s) et

sa qualité correspondante Q(s, a).

4. Appliquer l'action a(s). Soit y le nouvel état.

5. Recevoir le renforcement r.

6. Calculer la mise à jour de Q en utilisant l'équation (2.12).

7. Mettre à jour les paramètres q_{ij} des conclusions utilisées.

Tableau 2.1: Algorithme pour l'extraction des connaissances

Où l'équation de mise à jour de Q est donner par :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r(s_t, a_t) + \gamma \max_{a \in A}(Q(s_{t+1}, a) - Q(s_t, a_t) \right] \qquad \textbf{2.12}$$

2.5.3-Fusion des connaissances :

Cette deuxième extension permet l'utilisation de plusieurs Systèmes d'Inférence flou en compétition où chaque SIF calcule sa propre conclusion pour un vecteur d'entrée donné et la qualité correspondante. La PEE permet alors de choisir, non pas une conclusion possible pour chaque règle, comme dans l'extraction de connaissances, mais une sortie parmi toutes les sorties inférées. **[Glor. P.Y 99]**

La base de règles initiale s'écrit alors:

Si s_1 est A_1^i **et**……**et** s_n est A_n^i Alors $y = a[i,1]$avec $q[i,1] = 0$

Et $y = a[i,2]$avec $q[i,2] = 0$

…….

Et $y = a[i,J]$avec $q[i,J] = 0$

La sortie inférée par le SIF numéro k est donc :

$$a_k(s) = \frac{\sum_i \alpha_i(s) a[i,k]}{\sum_i \alpha_i(s)} \qquad \textbf{2.13}$$

ET la qualité correspondante:

$$Q(s,a_k) = \frac{\sum_{i=1}^{N} \alpha_i(s) \times q[i,k]}{\sum_{i=1}^{N} \alpha_i(s)} \qquad \textbf{2.14}$$

La politique d'exploration/exploitation rend un entier, noté $PEE \in \{1.....J\}$. L'action a(s) qui sera appliquée au système et la qualité correspondante seront :

$$a(s) = a_{PEE}(s) \qquad \textbf{2.15}$$

$$Q(s,a) = Q(s,a_{PEE}) \qquad \textbf{2.16}$$

L'algorithme pour la fusion des connaissances est présenté dans le tableau (2.2) :

1. Observer la situation s.

2. Pour chaque SIF calculer

$a_k(s)$ et Q(s, a_k (s)), k= 1 à J.

3. Choisir une conclusion,

notée a*(s) à l'aide d'une PEE.

4. Appliquer l'action a*(s). Soit y le nouvel état.

5. Recevoir le renforcement r.

6. Calculer la mise à jour de Q en utilisant l'équation (2.12).

7. Mettre à jour les paramètres q_{ij} des conclusions utilisées.

Tableau 2.2 : Algorithme pour la fusion des connaissances

2.5.4-Mise à jour des qualités :

Dans les deux extensions précédentes, l'action choisie, a(s), est appliquée au système et l'environnement renvoie le renforcement r qui sert à la mise à jour de Q(s, a(s)). On appelle a [i, j] une action élémentaire et q [i, j] la qualité élémentaire associée. L'action élémentaire choisie par la PEE est notée a[i, j+].

Soient $\widetilde{Q} = \widetilde{Q}(s,a)$ la nouvelle évaluation de $Q = Q(s,a)$ après avoir reçu le renforcement r et le nouvel état s' . On a:

$$\begin{aligned} &\widetilde{Q}(s,a) = Q(s,a) + \Delta Q \\ &\Delta Q = \alpha\left(r + \gamma \max_a Q(s',a) - Q(s,,a)\right) \end{aligned} \qquad \textbf{2.16}$$

Soit *E* l'erreur quadratique :

$$E = \frac{1}{2}\left|\widetilde{Q} - Q\right|^2 = \frac{1}{2}\Delta Q^2 \qquad \textbf{2.17}$$

L'application d'une méthode de gradient stochastique visant à réduire E conduit à mettre à jour les qualités élémentaires par :

$$\Delta q[i,j] = \begin{cases} \varepsilon \times \Delta Q \times \dfrac{\alpha_i}{\sum_i \alpha_i} & si \quad j = j^+ \\ 0 & \text{sinon} \end{cases} \qquad \textbf{2.18}$$

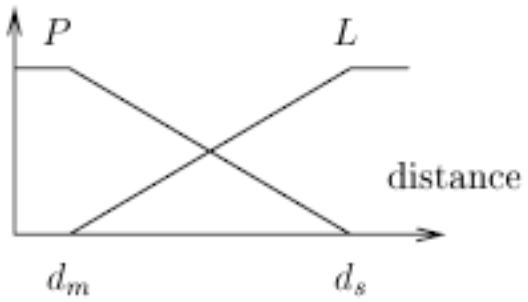
2.6-Exemple d'application :

Comme exemple d'application de l'utilisation d'apprentissage par renforcement des Systèmes d'Inférence Floue, on peut présenter l'exemple proposé par ***P. Y. Glorennec*** et ***L. Jouffe*** de la navigation d'un robot mobile dans un environnement inconnu et qui contient des obstacles, présenté dans l'article **A REINFORCEMENT LEARNING METHOD FOR AN AUTONOMOUS ROBOT**.

2.6.1- Système d'Inférence Floue :

Ces deux auteurs on proposé pour leur simulation un système d'inférence floue de type ***Takagi – Sugeno*** qui est basée sur :

- La distance minimum entre le robot et l'obstacle dans les trois directions, à gauche, en face et à droite, notée par d_m et la distance de sécurité pour la quelle le robot circule à une vitesse élevée notée par d_s , ces distances sont utilisées pour définir deux sous-ensembles flous « ***Proche*** » notée par ***P*** et « ***Loin*** » notée par ***L***. Qui sont présentés sur la figure (2.3) ont indiquant les états possibles
- La vitesse de déplacement du robot notée par v .
- L'orientation du robot noté par ϕ .



$$\begin{aligned} S_1 &: (P,P,P) \\ S_2 &: (P,P,L) \\ S_3 &: (P,L,P) \\ S_4 &: (P,L,L) \\ S_5 &: (L,P,P) \\ S_6 &: (L,P,L) \\ S_7 &: (L,L,P) \\ S_8 &: (L,L,L) \end{aligned}$$

Figure 2.3: Les états possibles

Et les règles suivantes :

Si S_1 ***alors*** v est ZR et $\Delta\phi$ est PG

Si S_2 ***alors*** v est ZR et $\Delta\phi$ est NG

Si S_3 ***alors*** v est $C \times V_{\max}$ et $\Delta\phi$ est ZR

Si S_4 ***alors*** v est V_{max} et $\Delta\phi$ est NP

Si S_5 ***alors*** v est ZR et $\Delta\phi$ est PG

Si S_6 ***alors*** v est ZR et $\Delta\phi$ est PG

Si S_7 ***alors*** v est V_{max} et $\Delta\phi$ est NP

Si S_8 ***alors*** v est V_{max} et $\Delta\phi$ est NM

Avec :

C : est un coefficient de réduction de vitesse dans un couloir.

Et les valeurs symboliques des conclusions sont remplacées par les valeurs v_i et ϕ_i ,qui sont données par :

$$v = \sum_{i=1}^{8} \alpha_i v_i \qquad \textbf{2.19}$$

Et

$$\Delta\phi = \sum_{i=1}^{8} \alpha_i \phi_i \qquad \textbf{2.20}$$

Où :

α_i représente la valeur de vérité de la règle ***i***.

$\Delta\phi$ est la variation dans l'angle d'orientation du robot.

2.6.2-Modèle du robot :

Le robot est modélisé par les équations suivantes :

$$\frac{dv}{dt} = \frac{v^* - v}{\tau} \qquad \textbf{2.21}$$

$$\frac{d\Delta\phi}{dt} = \frac{\Delta\phi^{*} - \Delta\phi}{\tau} \qquad \textbf{2.22}$$

Où :

v^{*} : Vitesse désirée.

$\Delta\phi^{*}$: Changement de direction désiré.

τ : Constante de temps qui représente un retard entre les commandes réelles et effectives.

2.6.3-Fonction de renforcement :

Cette fonction est définie par le signal de renforcement ***r*** où :

$$r = \begin{cases} -1 & si \quad dis\tan ce < \quad d_m \\ 0 & \sin on \end{cases} \qquad \textbf{2.23}$$

Ce signal est utilisé pour déterminer la meilleure interprétation numérique des termes linguistiques utilisés.

2.6.4- Q-Learning floue :

Ces deux auteurs ont choisis ***Q-Learning floue*** en extraction de connaissances, que nous avons présentées précédemment, où ils ont utilisés pour chaque changement de direction $\Delta\phi$ trois interprétations. Par exemple, $\Delta\phi = PB$ peut être interprétée par $\Delta\phi = 35^{\circ}$ ou $\Delta\phi = 40^{\circ}$ ou $\Delta\phi = 45^{\circ}$. La fonction $q[i, j]$ est associée à l'interprétation ***j*** de la règle ***i***, qui devient :

Si S_i **alors** v est v_i et $\Delta\phi$ est ϕ_{i1} avec la qualité $q[i,1]$

Ou $\Delta\phi$ est ϕ_{i2} avec la qualité $q[i,2]$

Ou $\Delta\phi$ est ϕ_{i3} avec la qualité $q[i,3]$

La qualité globale de la sortie du SIF est donnée par :

$$Q(x, \Delta\phi) = \sum_{i=1}^{8} \alpha_i(x) \times q[i, j^{*}(i)] \qquad \textbf{.2.24}$$

Où $j^*(i)$ est le vainqueur local pour la règle i, en fonction d'une politique d'exploration/exploitation donnée.

La figure (2.4) montre le résultat de simulation obtenue, avec $\tau = 0.5$.

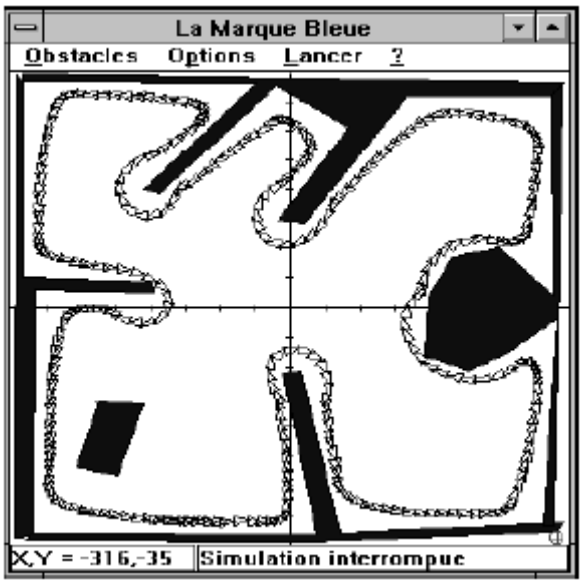


Figure 2.4 : Résultat de simulation pour $\tau = 0.5$

2.7- Conclusion :

Dans ce chapitre nous avons essayé de donner quelques informations sur la logique floue spécialement des informations sur les Systèmes d'Inférence Flou (SIF) tel que leurs types, leurs caractéristiques…etc. aussi l'utilisation de ces systèmes pour un apprentissage par renforcement par la suite nous avons donné en bref une description pour l'algorithme Q Learning Flou (QLF), et à la fin on présenté un exemple d'application de navigation d'un robot mobile en utilisant l'algorithme Fuzzy Q-Learning.

Chapitre 3 Réseaux de neurones artificiels

3.1-Introduction :

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.

3.2-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. **[8]**

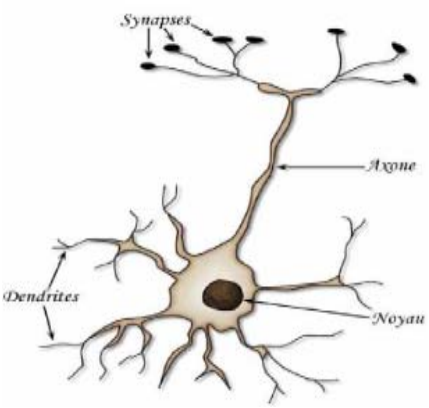


Figure3.1 : Neurone Biologique.

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**. [9]**

3.3-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 :

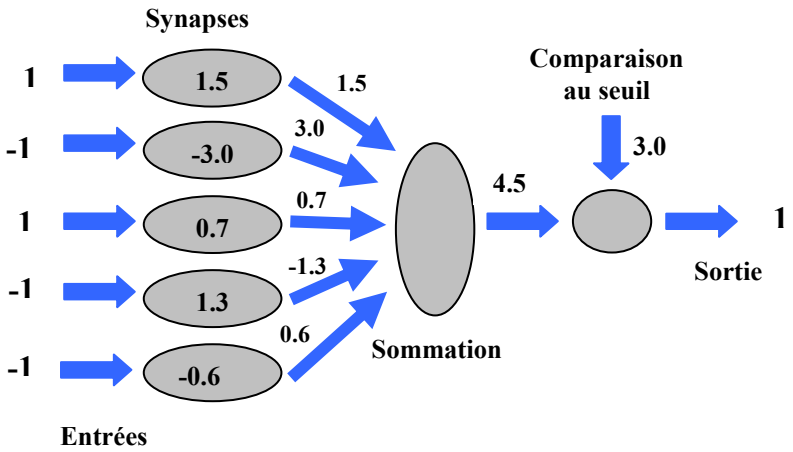


Figure 3.2 : Neurone Formel

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 3.1 : Analogie entre le neurone Biologie et le neurone Formel

3.3.1- Poids de connexion :

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

3.3.2- Les entrées :

Elles peuvent êtres :

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

3.3.3- 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 :

3.3.3. a- Fonction Seuil :

Fonction Heaviside définie par :

$$H(x) = \begin{cases} 1 & si \quad x \geq 0 \\ 0 & si \quad non \end{cases} \qquad .3.1$$

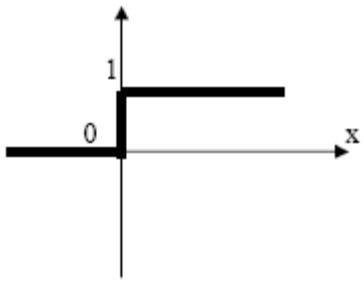


Figure 3.3 : Fonction Heaviside.

Fonction Signe définie par :

$$Sgn(x) = \begin{cases} 1 & si \quad x \geq 0 \\ -1 & \sin on \end{cases} \qquad \textbf{3.2}$$

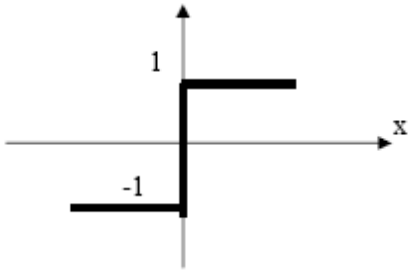


Figure 3.4 : Fonction Signe

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

3.3.3. b- Fonction linéaire :

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

$$F(x) = x \qquad \textbf{3.3}$$

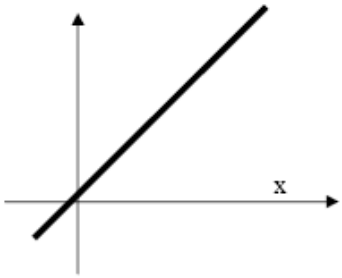


Figure 3.5 : Fonction Linéaire.

3.3.3. c- Fonction Linéaire à seuil :

Cette fonction d'activation est définie comme suit :

$$F(x)=\begin{cases} x & si \quad x \in [u \quad v] \\ v & si \quad x \geq v \\ u & si \quad x \leq u \end{cases} \qquad \textbf{3.4}$$

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.

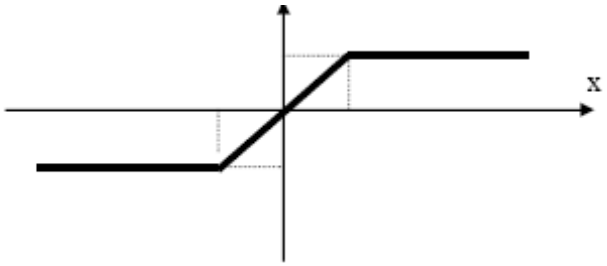


Figure 3.6 : Fonction Linéaire à seuil

3.3.3. 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 :

$$F(x)=\frac{1}{1+e^{-x}} \qquad \textbf{3.5}$$

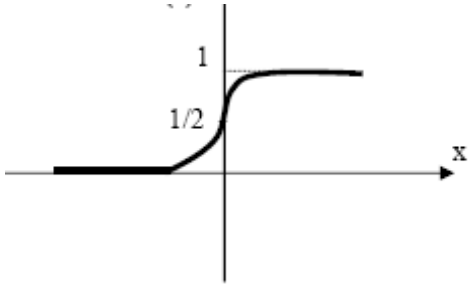


Figure 3.7 : Fonction sigmoïde.

3.3.4- 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.

3.4- 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 x1 à xm. 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 w1 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 :

$$w_1 x_1 + w_2 x_2 + ... + w_m x_m = \sum_{j=1}^{m} w_j x_j \qquad \textbf{3.6}$$

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 x1 à xm est ainsi donnée par :

$$F\left(\sum_{j=1}^{m} w_j x_j - \theta \right) \qquad \textbf{3.7}$$

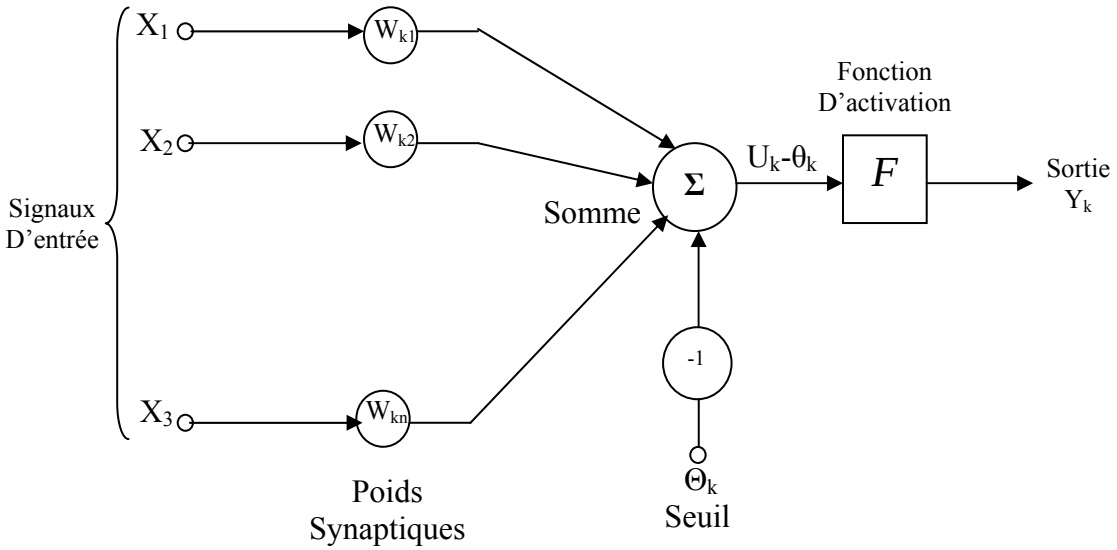


Figure 3.8 : Modèle général d'un neurone.

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.

3.5-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 ».

3.6-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. **[10]**

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

1- Les réseaux proactifs (feed forward).

2- Les réseaux récurrents (feedback).

3.6.1- 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. **[10]**

3.6.1. 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 composées de neurones de calcul transmettant les résultats du traitement au milieu extérieur.

La figure (3.9) 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. **[10]**

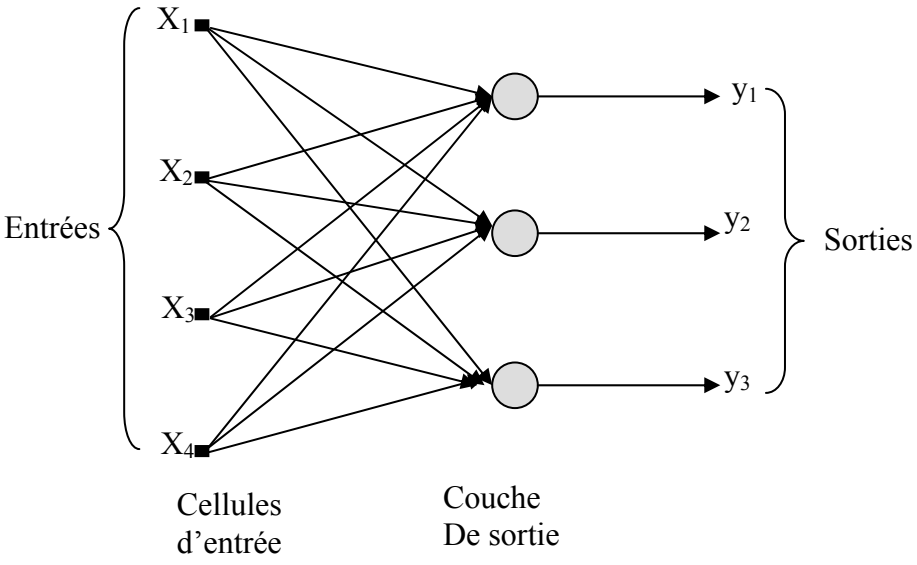


Figure 3.9 : Réseau proactif monocouche (Perceptron)

3.6.1. 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).**[10]**

La figure (3.10) 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.

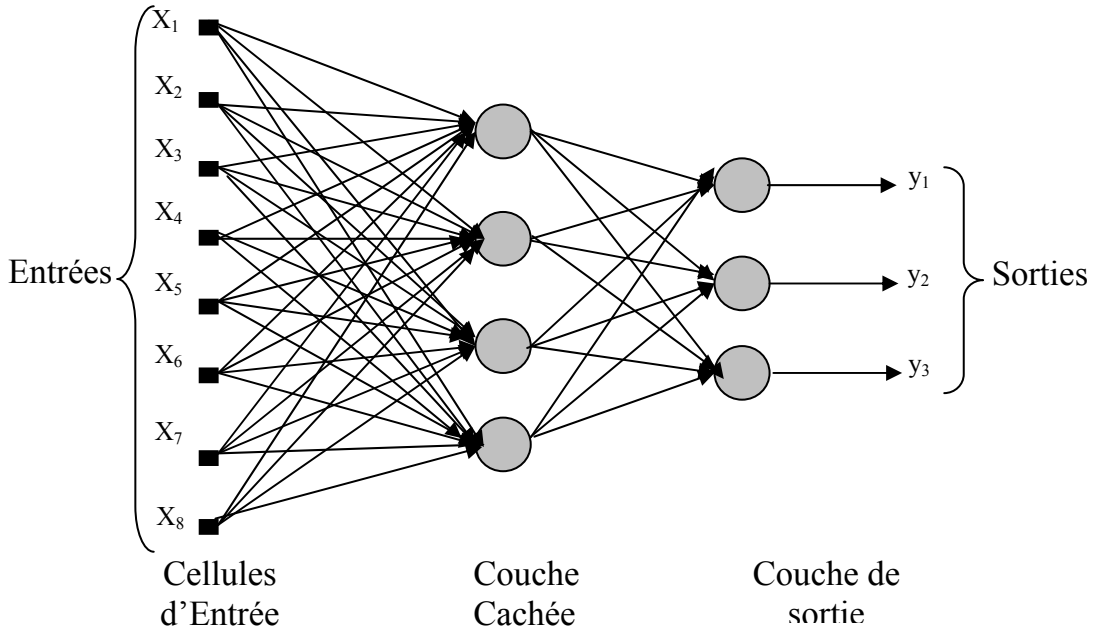


Figure 3.10 : Réseau proactif complètement connecté avec une seule couche cachée

3.6.2- 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. **[10]**

La figure (3.11) 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.

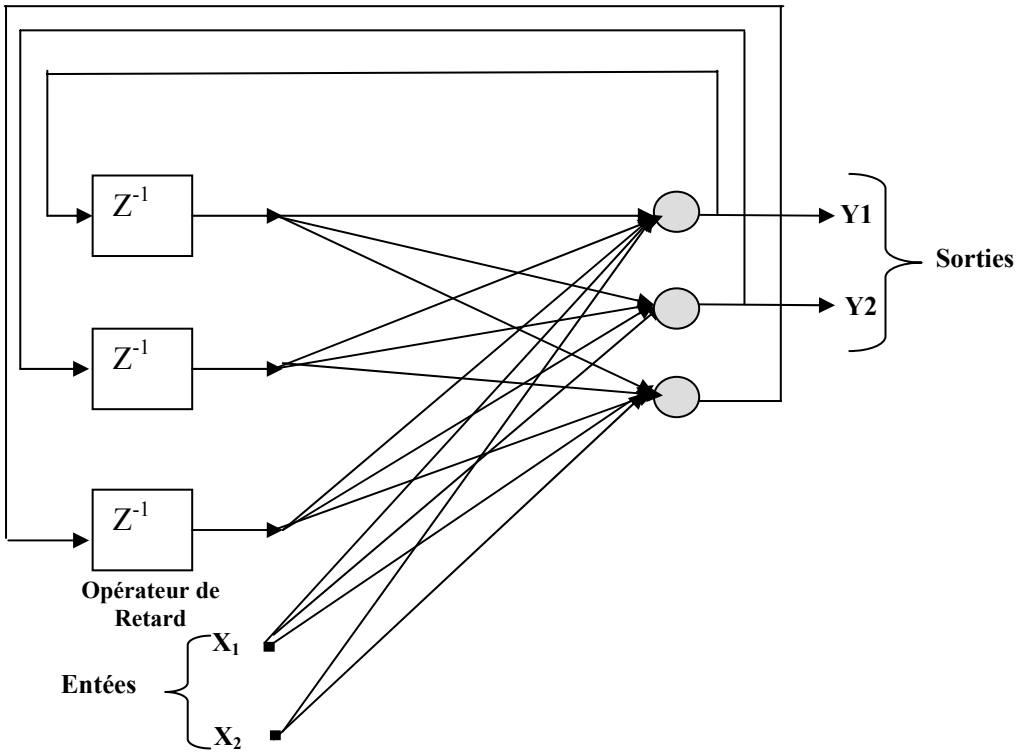


Figure 3.11 : Réseau récurent avec neurones cachés

3.7- Quelques Modèles des Réseaux de Neurones Artificiels :

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

3.7.1- 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. **[11]**

3.7.2- 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. **[11]**

3.7.3- 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. **[11]**

3.7.4- 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 présenter par classe.

3.7.5- Le perceptron multicouches :

3.7.5. a- 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.

3.7.5. b-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.

3.8-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. **[12]**

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. **[13]**

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 sont entrée **j**. au temps **t**, un changement $\Delta w_{ij}(t)$ de poids peut s'exprimer par l'équation suivante :

$$\Delta w_{ij}(t) = w_{ij}(t+1) - w_{ij}(t) \qquad \textbf{3.8}$$

Par conséquent :

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \qquad \textbf{3.9}$$

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 algorithme d'apprentissage du réseau. A titre d'exemple on peut cité : apprentissage par la règle de Hebb, apprentissage par la correction d'erreur.

Dans la suite de ce chapitre, nous allons donner une description pour l'algorithme de rétropropagation des l'erreurs qui est généralement utilisé pour l'apprentissage de perceptron multicouches.

3.9-Algorithme de rétro propagation du gradient:

3.9.1- 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. **[14]**

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.

3.9.2- Algorithme :

3.9.2. 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}.

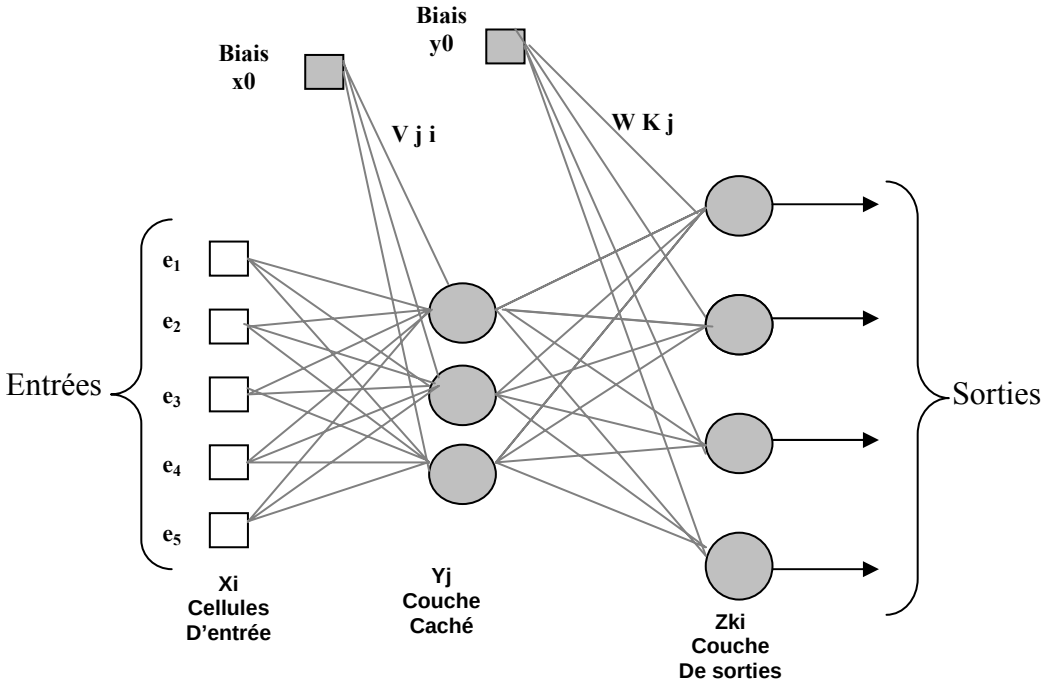


Figure 3.12 : Exemple de réseau MLP à une couche cachée avec 5 entrées, 3 neurones dans la couche cachée, et quatre 4 sorties.

3.9.2. b- Les Etapes de l'algorithme :

- **ETAPE 1** : **Initialisation au hasard ou aléatoire des poids des connexions** v_{ji} et w_{kj}

- **ETAPE 2** : **Propagation des entrées** $x_i = e_i$.

  - On propage vers la couche cachée :

$$y_j = f\left(\sum_{i=1}^{m} x_i v_{ij} + x_0\right) \qquad \textbf{.3.10}$$

  - Puis de la couche cachée vers la couche de sortie :

$$z_k = f\left(\sum_{j=1}^{n} y_j w_{kj} + y_0\right) \qquad \textbf{3.11}$$

Les valeurs x_0 et y_0 sont des Biais, f est la fonction d'activation qu'on a choisie où on a définit notre réseau MLP.

- **ETAPE 3** : **rétropropagation de l'erreur** pour chaque neurone de la couche de sortie On calcule l'erreur, c'est-à-dire la différence entre la sortie désirée s_k et la sortie réelle (obtenu) z_k .

$$E_k = z_h(1 - z_k)(s_k - z_k) \qquad \textbf{3.13}$$

On propage cette erreur sur la couche cachée ; l'erreur de chaque neurone de la couche cachée est donnée par :

$$F_j = y_j(1 - y_j)\sum_{k=1}^{p} w_{kj} E_k \qquad \textbf{3.14}$$

- **ETAPE 4** : **Correction des poids des connexions** Il reste maintenant la modification des poids des connexions et aussi les biais.

  - Entre la couche d'entrée et la couche cachée :

$$\begin{cases} \Delta v_{ji} = \eta y_j F_j \\ \Delta x_0 = \eta F_j \end{cases} \qquad \textbf{.3.15}$$

- Entre la couche cachée et la couche de sortie :

$$\begin{cases} \Delta w_{kj} = \eta z_k E_k \\ \Delta y_0 = \eta E_k \end{cases} \qquad \textbf{3.16}$$

η Etant un paramètre qu'il reste à déterminer.

- **BOUCLER à l'étape 2** jusqu'à un critère d'arrêt à définir.

3.9.3- Sommaire de l'algorithme 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 souche de sortie.***
5. ***Boucler à 2 jusqu'à un critère d'arrêt à définir.***

Tableau 3.2 : Sommaire de l'algorithme Rétropropagation de l'erreur

3.10- Application des réseaux de neurones :

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

- Autopilotage des avions.
- Système de guidage des automobiles.
- Lecture automatique des cheques 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.

3.11- Conclusion.

Dans ce chapitre nous avons essayé de donner en bref une 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 4 Conception d'un système d'apprentissage par renforcement

4.1-Introduction :

Comme nous l'avons présenté dans le chapitre un, le système d'apprentissage par renforcement est basé sur des fonctions ou des blocs principales qui sont, L'environnement, Fonction de renforcement et la fonction valeur.

Dans ce chapitre on va essayer de donner une description pour chaque fonction qui participe à la conception du système d'apprentissage par renforcement.

4.2-Description des fonctions du système d'apprentissage par renforcement :

Le modèle standard du système d'apprentissage par renforcement est généralement basé sur des interactions entre l'agent est son environnement sous forme de mesure des capteurs. Alors l'agent doit choisir une action de telle manière à améliorer les performances de son comportement. Chaque action choisie par l'agent sera jugée par un signal scalaire appelé signal de renforcement.

4.2.1- Environnement :

Ce système d'apprentissage est basé sur l'interaction essai et erreur de l'agent avec son environnement où ce dernier représente l'espace de travail dynamique pour l'agent. L'objectif de ce système d'apprentissage est d'apprendre une projection topologique des situations (états) vers des actions. Une condition nécessaire sur l'environnement considéré est qu'il soit au moins partiellement observable par le système d'apprentissage. Si le système d'apprentissage par renforcement peut observer parfaitement toutes les informations de l'environnement il est alors possible de choisir

une action pertinente basée sur les états réels **[12].** Les observations de l'agent pour son environnement peuvent être les résultats des mesures des capteurs.

4.2.2-Fonction de renforcement :

Cette fonction doit être définie par le concepteur du système d'apprentissage par renforcement, elle est généralement présentée par un scalaire. Si l'agent exécute une bonne action ce moment là il reçoit un renforcement positif appelé **récompense** dans le cas contraire il reçoit un renforcement négatif appelé **punition**.

4.2.3-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 $Q^{\pi}(s,a)$.

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.

4.3- 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 :

$$V(s_t) = V^*(s_t) + e(s_t) \qquad \textbf{4.1}$$

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

$$V^*(s_t) = r(s_t) + \gamma V^*(s_{t+1}) \qquad \textbf{4.2}$$

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 :

$$V(s_t) = r(s_t) + \gamma V(s_{t+1}) \qquad \textbf{4.3}$$

Substituant l'équation (4.1), (4.2) dans l'équation (4.3), on obtient la formule suivante :

$$e(s_t) + V^*(s_t) = r(s_t) + \gamma\left(e(s_{t+1}) + V^*(s_{t+1})\right). \qquad \textbf{4.4}$$

Donc :

$$e(s_t) + V^*(s_t) = r(s_t) + \gamma e(s_{t+1}) + \gamma V^*(s_{t+1}) \qquad \textbf{4.5}$$

L'utilisation des équations (4.2) et (4.5) nous conduit à obtenir l'équation suivante :

$$e(s_t) = \gamma e(s_{t+1}) \qquad \textbf{4.6}$$

Qui représente la relation entre les erreurs d'approximation de la fonction valeur entre deux états (situations) successifs $\mathbf{s_t}$ et $\mathbf{s_{t+1}}$. 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.

4.4-Génération de la fonction valeur :

4.4.1-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. **[12]**

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 (4.3) en balayant tout les états $\mathbf{s_t}$. Pour que l'algorithme converge il faut que l'erreur d'approximation donnée par l'équation suivante :

$$\Delta V(s_t) = V^*(s_t) - V(s_t) \qquad \textbf{4.7}$$

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 (4.1) présente le modèle général de l'algorithme d'apprentissage par renforcement « ***Q Learning*** »

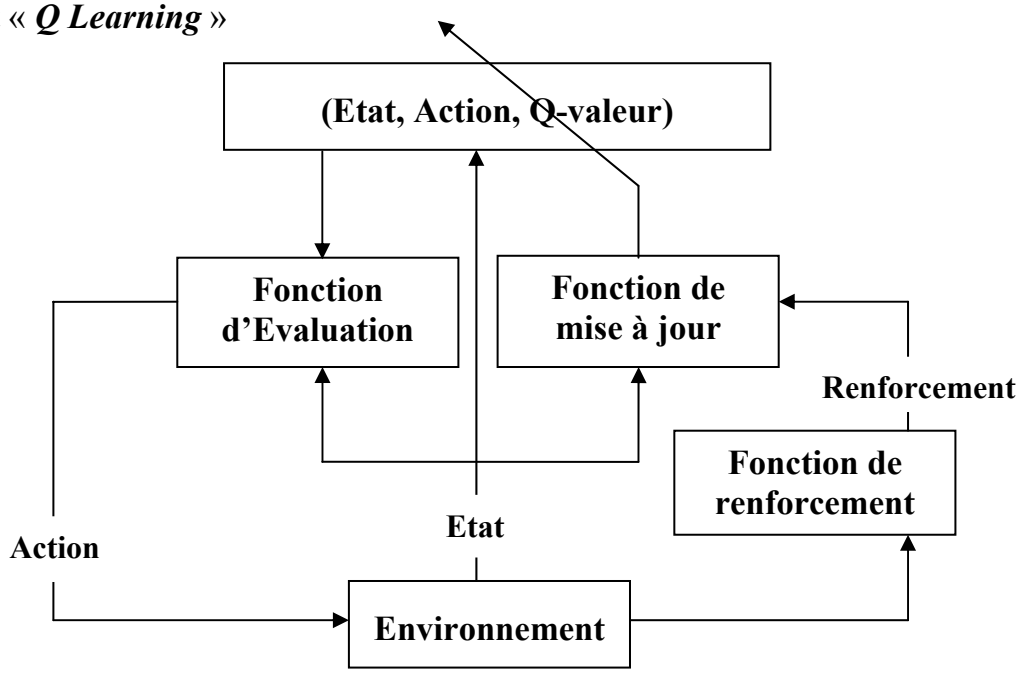


Figure 4.1 : Modèle général pour les algorithmes d'apprentissage par renforcement, (ici le Q-Learning).

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, la figure (4.2) représente le signal de renforcement reçus par l'agent (+1 : bon, - 1 : mauvais, 0 : neutre).

r	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_3}$	$\mathbf{a_4}$	$\mathbf{a_5}$	$\mathbf{a_6}$
$\mathbf{S_1}$					+1	
$\mathbf{S_2}$		+1				
$\mathbf{S_3}$		0	0			
$\mathbf{S_4}$			-1		0	
$\mathbf{S_5}$			-1			
$\mathbf{S_6}$			0			-1

Figure 4.2 : 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. La figure (4.3) représente Les valeurs d'utilité Q calculées grâce à l'application de l'équation (1.21) de mise à jour indiquée dans le chapitre un.

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

Figure 4.3 : Valeurs de Q calculées.

4.4.2- Réseau de Neurones Artificiel :

L'utilisation d'un tableau de consultation pour la génération de la fonction de valeur [Touzet 1997] limite la dimension des problèmes qu'on peut résoudre, dans le monde réel il existe beaucoup de problèmes qui ont un grand espace d'état. Une méthode originale est présentée pour faire une extension de la fonction de valeur pour des espaces continus de grande dimension est d'utiliser un ***réseau de neurones artificiel*** pour estimer la fonction optimale.

Soit $V(s_t, w_t)$ l'approximation de $V^*(s_t)$ avec $w_t = [w_{ij}(t)]$ un vecteur qui contient les poids du réseau. La propagation en avant de l'exemple (état) $\mathbf{s_t}$ dans le réseau de neurones va générer une valeur $V(s_t, w_t)$, qui représente la fonction de valeur.

Pour minimiser l'erreur quadratique entre la sortie désirée (optimale) et la sortie obtenue on utilise l'algorithme de rétropropagation du gradient, cette erreur est considérée comme une fonction des poids des connexions et elle est présentée avec la relation suivante :

$$E(w_t) = \frac{1}{2}\left[V^*(s_t) - V(s_t)\right]^2 \qquad \textbf{4.8}$$

Après chaque passage d'un exemple (état) $\mathbf{s_t}$, une modification de chaque poids des connexions du réseau de neurones est faite suivant la formule :

$$w_t = w_{t-1} + \alpha\left[V^*(s_t) - V(s_t, w_t)\right]\frac{\partial V(s_t, w_t)}{\partial w_t} \qquad \textbf{4.9}$$

Où α est le pas d'apprentissage qui est strictement positif.

La formule (4.9) représente l'algorithme direct de l'apprentissage, ceci peut engendrer des oscillations pour les poids du réseau de neurones qui conduit à une divergence de l'apprentissage. Pour éviter ce problème on doit remplacer l'algorithme d'apprentissage direct par l'algorithme résiduel du gradient **[12]**, où l'ajustement des poids est donné par l'équation suivante :

$$w_t = w_{t-1} + \alpha\left[V^*(s_t) - V(s_t, w_t)\right]\left[\frac{\gamma\partial V(s_{t+1t}, w_t)}{\partial w_t} - \frac{\partial V(s_t, w_t)}{\partial w_t}\right] \qquad \textbf{4.10}$$

Les algorithmes directs sont rapides mais parfois instables par contre les algorithmes résiduels sont lents mais garantissent une bonne stabilité du réseau de neurones.

Soit un réseau de neurones qui est caractérisé par, *n* cellules en entrée et *un* neurone en sortie, avec un nombre quelconque de couches cachées. Chaque neurone est connecté à des neurones dans le sens entrée-sortie mais pas forcement à des neurones situés sur la couche immédiatement suivante.

La figure (4.4) représente un exemple de réseau de neurones pour la génération de la fonction de valeur.

La sortie du neurone ***j*** est donnée par $\boldsymbol{O_j}$, l'entrée du neurone ***i*** est donnée par $\boldsymbol{I_i}$ donc $I_i = \sum_j w_{ij} O_j$. Où l'indice j est porté sur les neurones sur lesquels le neurone i reçoit des connexions, la fonction d'activation d'un neurone est la fonction sigmoïde notée par $\boldsymbol{f}$ et sa dérivée par rapport à l'entrée est notée par $\boldsymbol{f'}$. Ces notations son données sur la figure (4.2).

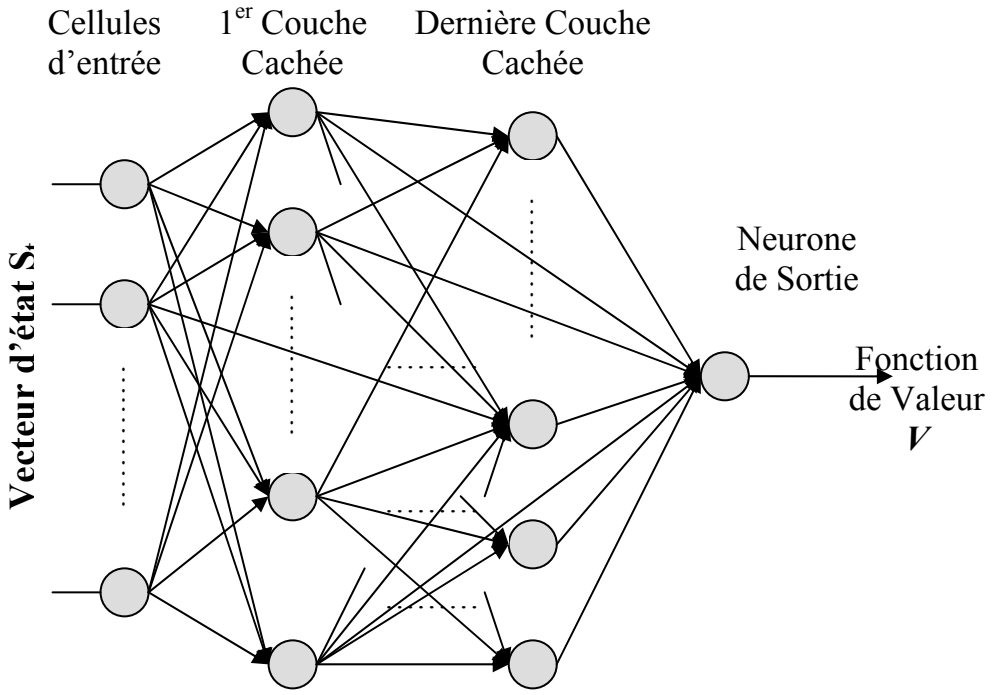


Figure 4.4 : Génération de la fonction de valeur par un réseau de neurone

Soit $V(s_t, w_t)$ qui est la sortie obtenue effectivement à l'issue de la propagation de l'exemple (état) $\mathbf{s_t}$ dans le réseau de neurones. Comme nous avons présenté l'algorithme de rétropropagation dans la chapitre deux, cet algorithme consiste à effectuer une modification des poids w_{ij} du réseau de neurone, après chaque passage d'un état à un autre.

L'ajustement des poids pour une telle situation est faite si le signale de renforcement est une punition, où il faut calculer pour tous les poids w_{ij} le gradient de la fonction $V(s_t, w_t)$.

$$\frac{\partial V}{\partial w_{ij}} = \frac{\partial V}{\partial I_i}\frac{\partial I_i}{\partial w_{ij}} \qquad \textbf{4.11}$$

Où :

$$\frac{\partial I_i}{\partial w_{ij}} = \frac{\partial\left(\sum_p w_{ip} O_p\right)}{\partial w_{ij}} = O_j \qquad \textbf{4.12}$$

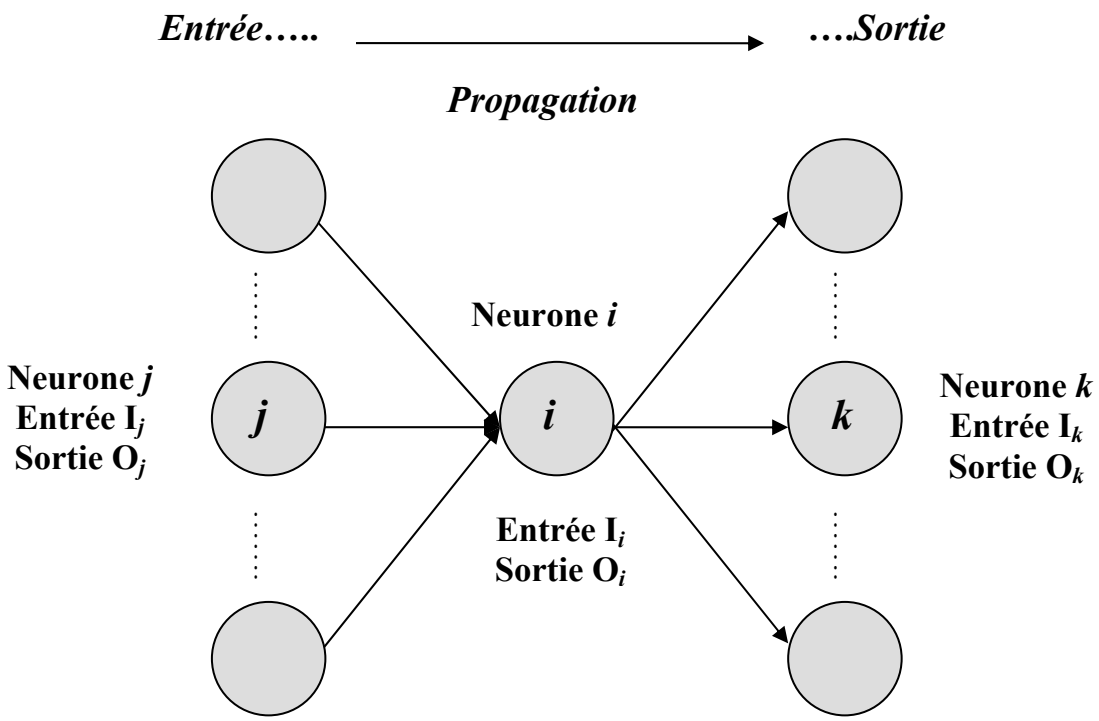


Figure 4.5 : Résumé des notations.

L'indice ***p*** portent sur des neurones appartenant à la couche précédente du neurone ***i*** les sorties O_p de cette couche ne dépendent pas de w_{ij} on obtient donc :

$$\frac{\partial V}{\partial w_{ij}} = \frac{\partial V}{\partial I_i} O_j \qquad \textbf{4.13}$$

On pose $d_i = \frac{\partial V}{\partial I_i}$ alors on aura :

$$\frac{\partial V}{\partial w_{ij}} = d_i O_j \qquad \textbf{4.14}$$

Pour le neurone de sortie ***i*** on a :

$$d_i = f'(I_i) \qquad \textbf{4.15}$$

Pour les neurones des couches cachées, on peut écrire :

$$d_i = \sum_k \frac{\partial V}{\partial I_k} \frac{\partial I_k}{\partial I_i} = \sum_k d_k \frac{\partial I_k}{\partial I_i} \qquad \textbf{4.16}$$

Où l'indice ***k*** porte sur les neurones sur lesquels le neurone ***i*** envoi des connexions.

En effet, les entrées I_k des autres neurones ne dépendent pas de I_i, on a donc :

$$d_i = \sum_k d_k \frac{\partial I_k}{\partial O_i} \frac{\partial O_i}{\partial I_i} \qquad \textbf{4.17}$$

Où :

$$\frac{\partial I_k}{\partial O_i} = \frac{\partial \left(\sum_p w_{kp} O_p \right)}{\partial O_i} = w_{ki} \qquad \textbf{4.18}$$

Et comme l'indice ***p*** est porté sur les neurones reliant le neurone ***k***, donc ces neurones sont situés sur la même couche que les neurones ***i***, ceci nous donne que les O_p ne dépendent pas des O_i (avec $p \neq i$). On sait aussi que $O_i = f(I_i)$, on a finalement :

$$d_i = \sum_k d_k w_{ki} f'(I_i) \qquad \textbf{4.19}$$

Pour un état $\mathbf{s_t}$ à l'instant **t** la règle de modification des poids du réseau de neurones par l'algorithme de rétropropagation est donnée par la formule suivante :

$$w_{ijt}(t) = w_{ij}(t-1) + \alpha \left[V^*(s_t) - V(s_t, w_t) \right] d_i O_i \qquad \textbf{4.20}$$

Où d_i est définit par les équations suivantes:

- Pour le neurone de sortie :

$$d_i = f'(I_i) \qquad \textbf{4.21}$$

- Pour les couches cachées :

$$d_i = \sum_k d_k w_{ki} f'(I_i) \qquad \textbf{4.22}$$

Où l'indice ***k*** est porté sur les neurones vers lesquels le neurone ***i*** envoie une connexion.

4.5-Conclusion :

Dans ce chapitre nous avons donné une description sur les blocs qui constitue un système d'apprentissage par renforcement tout en citant le rôle de chacun. On a évoqué la méthode de Touzet qui donne la possibilité de représenter l'approximation de la fonction de valeur par un tableau de consultation dans lequel chaque case corresponde à une approximation de la fonction de valeur optimale. Toutefois, cette méthode pose des

problèmes de dimension. Pour résoudre ce problème, nous avons adopté une méthode qui repose sur le concept des ***réseaux de neurones artificiel*** afin de pouvoir faire une extension de la fonction de valeur pour des espaces continus de grande dimension et estimer ainsi la fonction optimale.

Chapitre 5 Application et Résultats

5.1-Introduction :

Dans ce qui suit nous allons essayer de donner plus d'informations sur le travail qu'on a effectué, on premier lieu nous avons proposé une architecture pour le système d'apprentissage par renforcement qui est constitué de quelques blocs, ce système est basée sur la perception de l'agent (robot) de son environnement de travail, cette perception représente l'état de l'environnement où elle seras utilisée comme entrée pour le 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.

5.2-Architecture du système Apprentissage par renforcement proposé:

L'objectif de cette proposition est d'avoir un système d'apprentissage qui permet à un robot de ce déplacer dans un environnement qui est totalement inconnu en évitant les obstacles.

L'un des algorithmes utilisé dans le domaine de l'apprentissage par renforcement est le ***Q-Learning*** que nous avons donné une description générale dans le premier chapitre. 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***, et le vecteur d'actions possible. Le choix de l'action que l'agent doit exécuter est faite par une fonction appelée ***fonction de sélection***.

La figure (5.1) représente la structure du système d'apprentissage par renforcement basé sur un réseau de neurone.

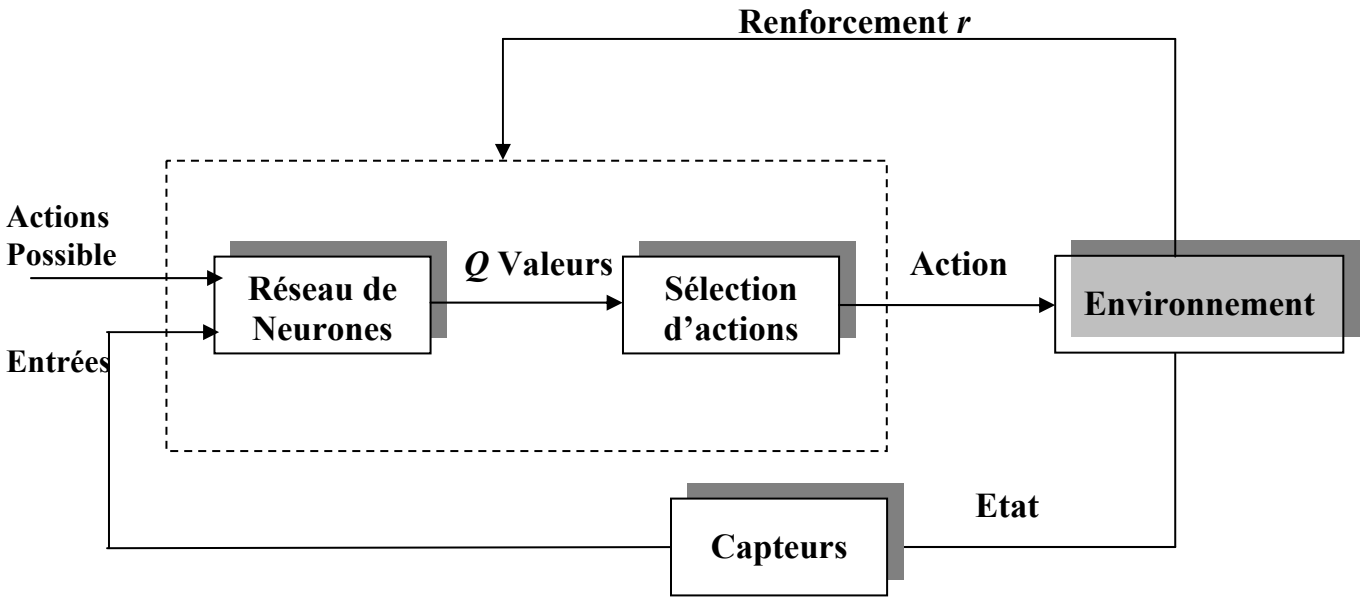


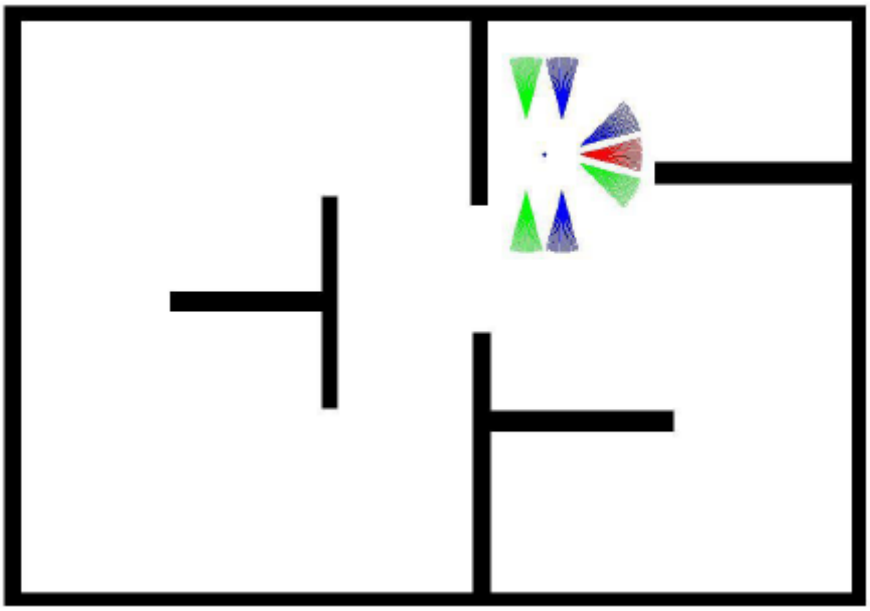
Figure 5.1 : Structure du système d'apprentissage par renforcement

5.2.1- 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 proximité. L'angle d'ouverture de chaque capteur varie entre $-\pi/12$ et $\pi/12$. Le vecteur d'état ***S*** est choisi de manière à avoir des informations sur l'existence d'obstacles sur les trois côtés du robot.

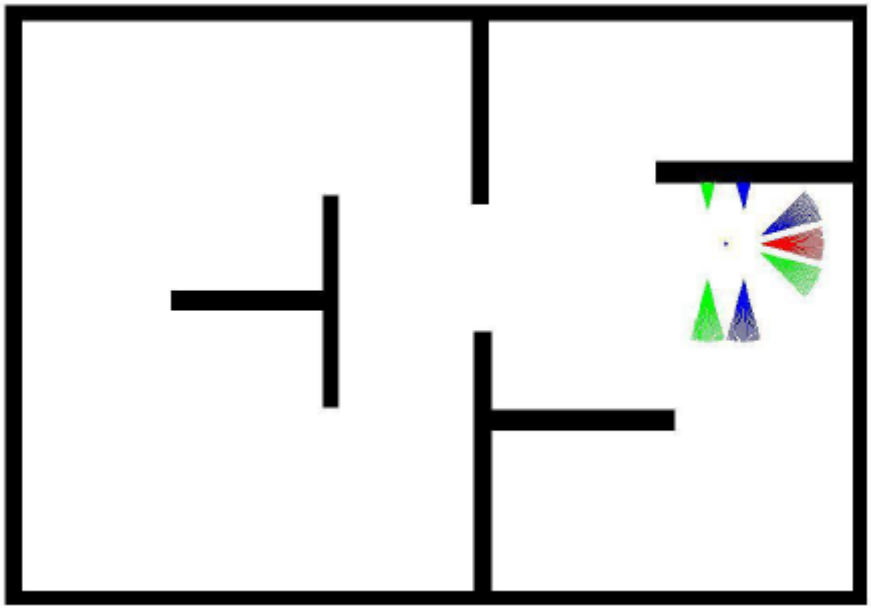
Ce vecteur est composé de sept variables binaires s_i, $i=1,7$. Le choix de ces variables est fait de manière à avoir des informations tout ou rien. C'est-à-dire, si par exemple $s_i=1$ alors il existe un obstacle près du robot, si $s_i=0$ pas d'obstacle près du robot.

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



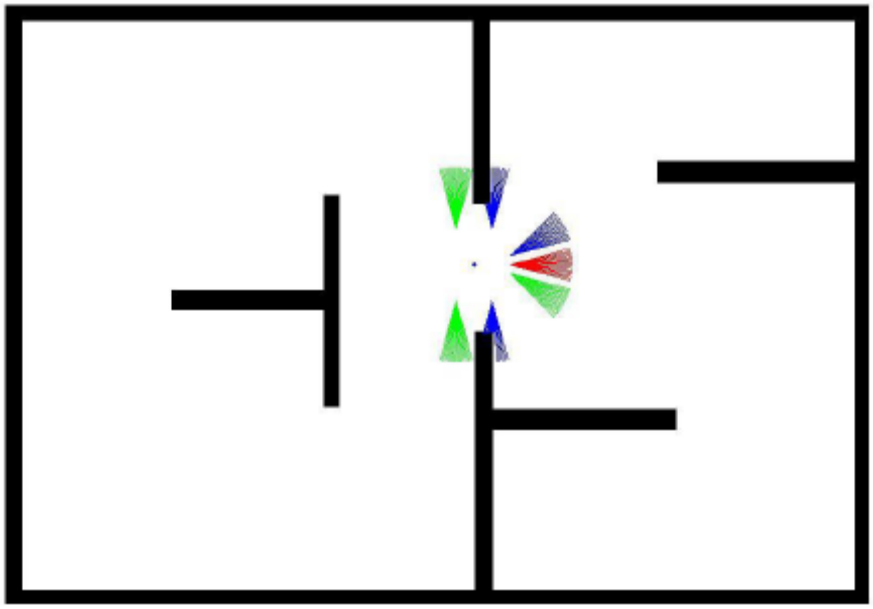
$$S = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

Figure 5.2 : 1er Etat de l'environnement.



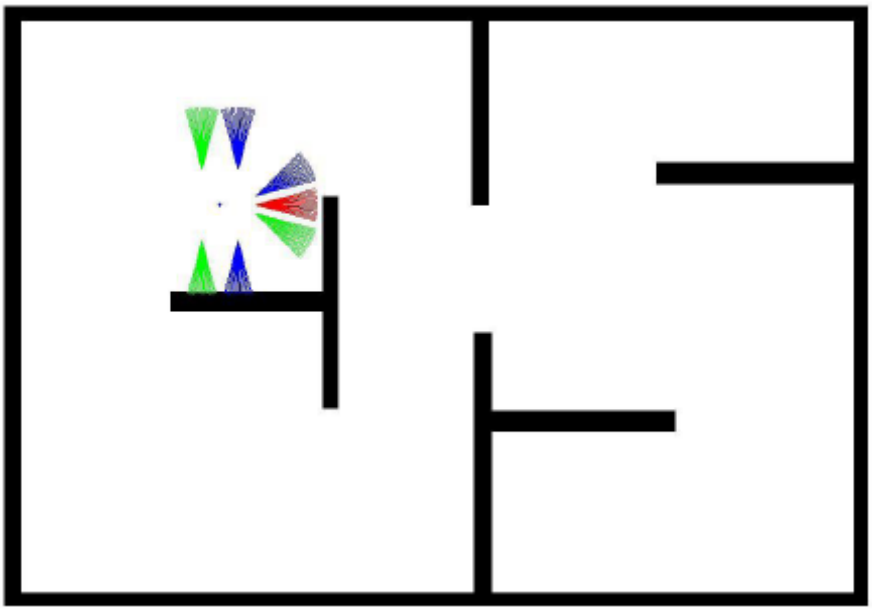
$$S = [1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$$

Figure 5.3 : 2^{em} Etat de l'environnement.



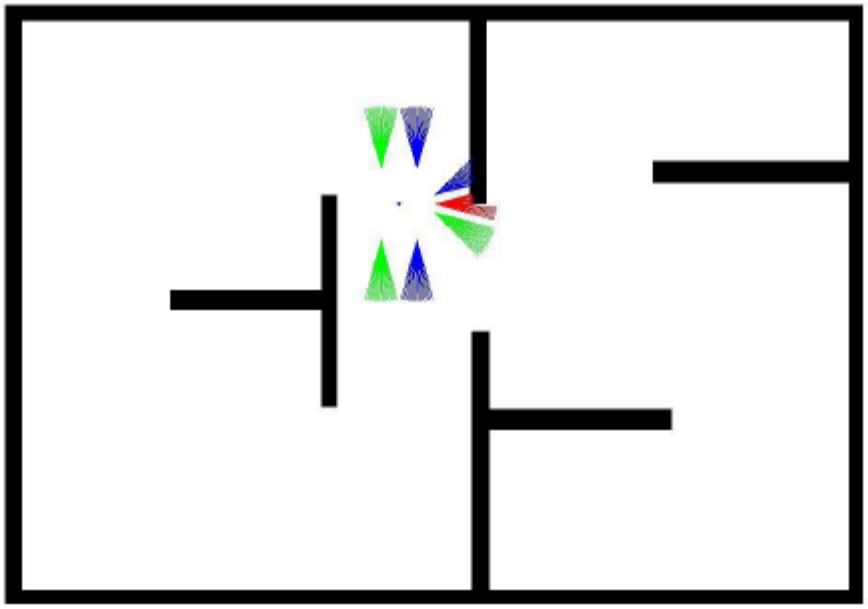
$$S = [0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]$$

Figure 5.4 : 3^{em} Etat de l'environnement.



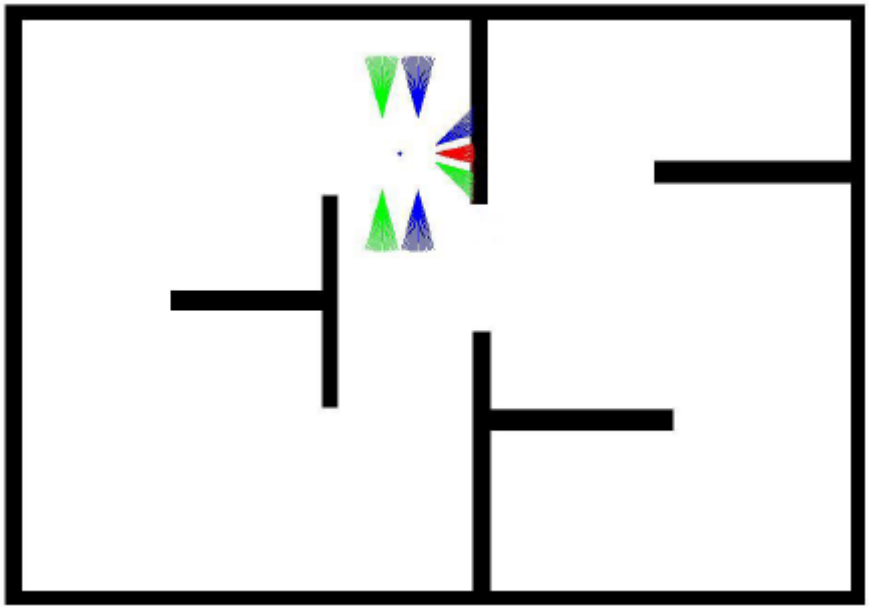
$$S = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1]$$

Figure 5.5 : 4^{em} Etat de l'environnement.



$$S = [0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0]$$

Figure 5.6 : 5^{em} Etat de l'environnement.



$$S = [0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0]$$

Figure 5.7 : 6^{em} Etat de l'environnement.

5.2.2- Fonction du renforcement :

Pour chaque état où ce 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, **+1** lorsque le robot avance tout droit et **0** dans tous les autres cas.

5.2.3- 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 n_c de neurones.

La figure (5.8) représente le schéma du réseau de neurone proposé pour la commande du robot.

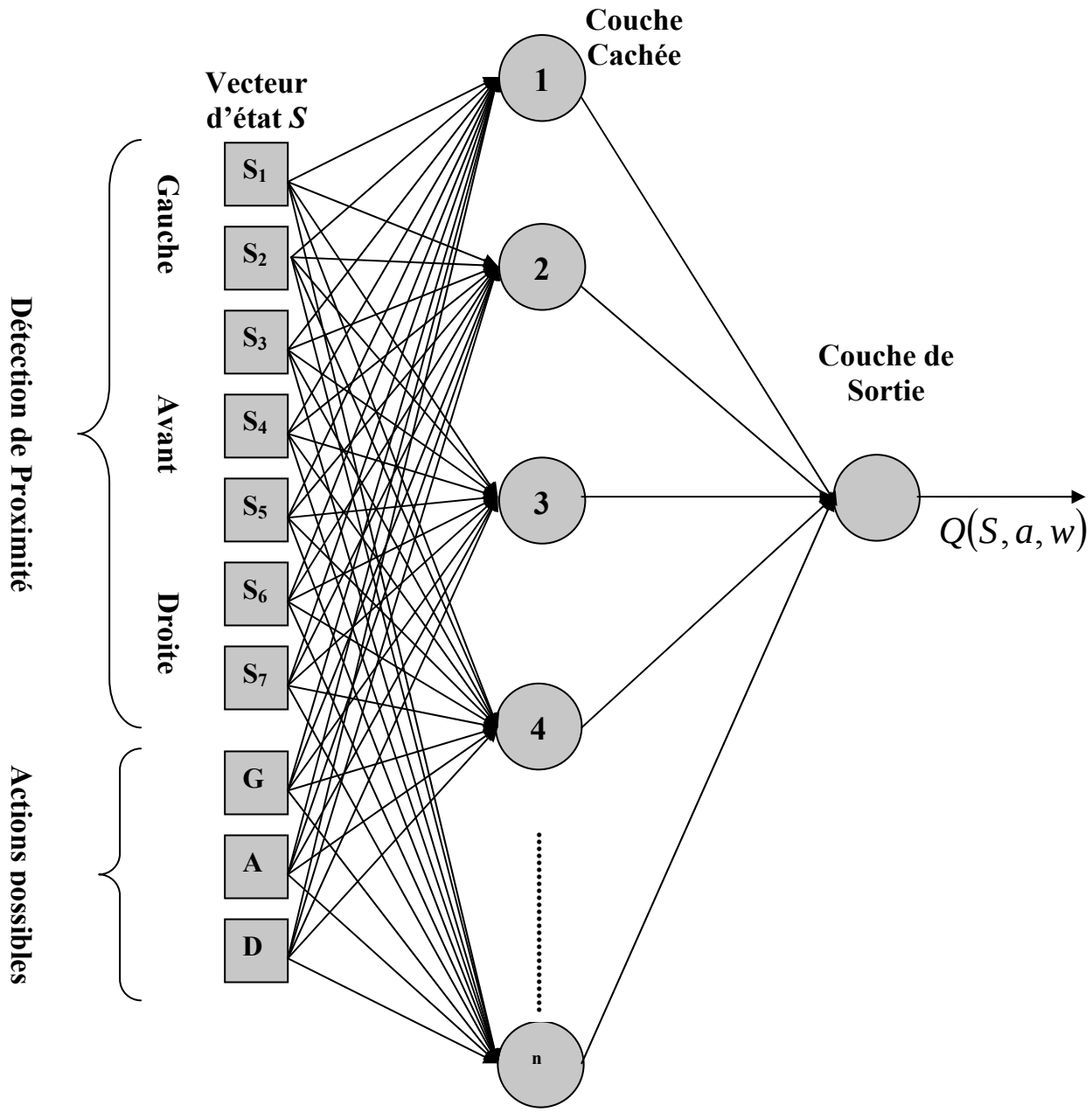
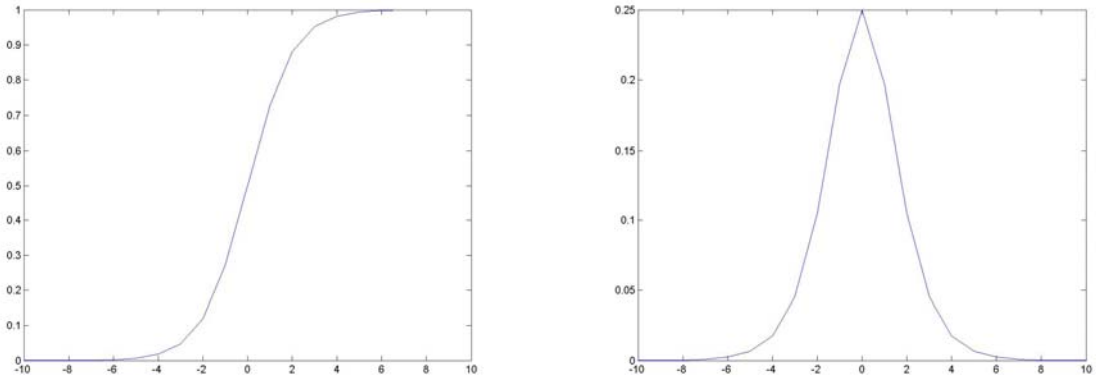


Figure 5.8 : Réseau de neurones utilisé pour la commande du robot.

La fonction d'activation choisie pour les neurones de la couche cachée et la couche de sortie est la fonction sigmoïde qui est définit par :

$$f(x)=\frac{1}{1+e^{-x}} \qquad \textbf{5.1}$$

La représentation graphique de cette fonction et sa dérivée est représentée sur la figure (5.9).



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

Figure 5.9 : Fonction d'activation et sa dérivée

5.2.4-Fonction de sélection d'action :

Le 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 tourner à Gauche.
- $\mathbf{a_2}$: Action Avancer.
- $\mathbf{a_3}$: Action tourner 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 $1-\varepsilon$, soit :

- $p \in [0,1]$, un nombre au hasard.
- Si $p < \varepsilon$ on choisit une action a au hasard (***Exploration***), où $a \in A$ l'ensemble de l'action possible.
- Si $p > \varepsilon$ on choisie $a(S) = Arg \max_{b \in A} Q(S,b,w)$ (***Exploitation***).

$$a(S) = Arg \max_{b \in A} Q(S,b,w) \qquad \mathbf{5.2}$$

5.3-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 en utilisant l'équation de la mise à jour de l'algorithme classique du ***Q Learning*** et aussi l'algorithme de ***rétropropagation*** :

5.3.1- La couche de sortie :

- Pour la matrice des poids :

$$w_2(t) = w_2(t-1) + \alpha\left[\left(r(s_t, a_t) + \gamma a(S)\right) - Q(s_t, a_t, w_t)\right] f(S) \frac{\partial Q}{\partial w_2} \left(a(S) - Q_{target}\right) \quad \textbf{5.3}$$

- Pour le vecteur des biais on a :

$$b_2(t) = b_2(t-1) + \alpha\left[\left(r(s_t, a_t) + \gamma a(S)\right) - Q(s_t, a_t, w_t)\right] f(S) \frac{\partial Q}{\partial w_2} \left(a(S) - Q_{target}\right) \quad \textbf{5.4}$$

Où :

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

$$Q_{target} = r(s_t, a_t) + \gamma \max Q(S, a_t, w) \quad \textbf{5.5}$$

f : Fonction d'activation des neurones de la couche de sortie.

$Q(s_t, a_t, w)$: Fonction de valeur état / action correspondant à l'action exécutée.

S : Etat de l'environnement.

5.3.2-La couche cachée :

- Pour la matrice des poids :

$$w_1(t) = w_1(t-1) + \alpha\left[\left(r(s_t, a_t) + \gamma a(S)\right) - Q(s_t, a_t, w_t)\right] S \frac{\partial f}{\partial w_1} \sum w_2 S \quad \textbf{5.6}$$

- Pour le vecteur des biais on a :

$$b_1(t) = b_1(t-1) + \alpha\left[\left(r(s_t, a_t) + \gamma a(S)\right) - Q(s_t, a_t, w_t)\right] S \frac{\partial f}{\partial w_1} \sum w_2 S \quad \textbf{5.7}$$

Avec :

f : Fonction d'activation des neurones de la couche cachée.

S : Etat de l'environnement.

$Q(s_t, a_t, w)$: Fonction de valeur état / action correspondant à l'action exécutée.

Ces modifications des valeurs pour les matrices des poids et du vecteur des biais est présente si le signal de renforcement est **: -1**ou **0.**

5.4- Modèle du Robot :

La conception d'une commande pour un système dynamique est généralement basée sur sa modélisation. La figure (5.10) 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***.

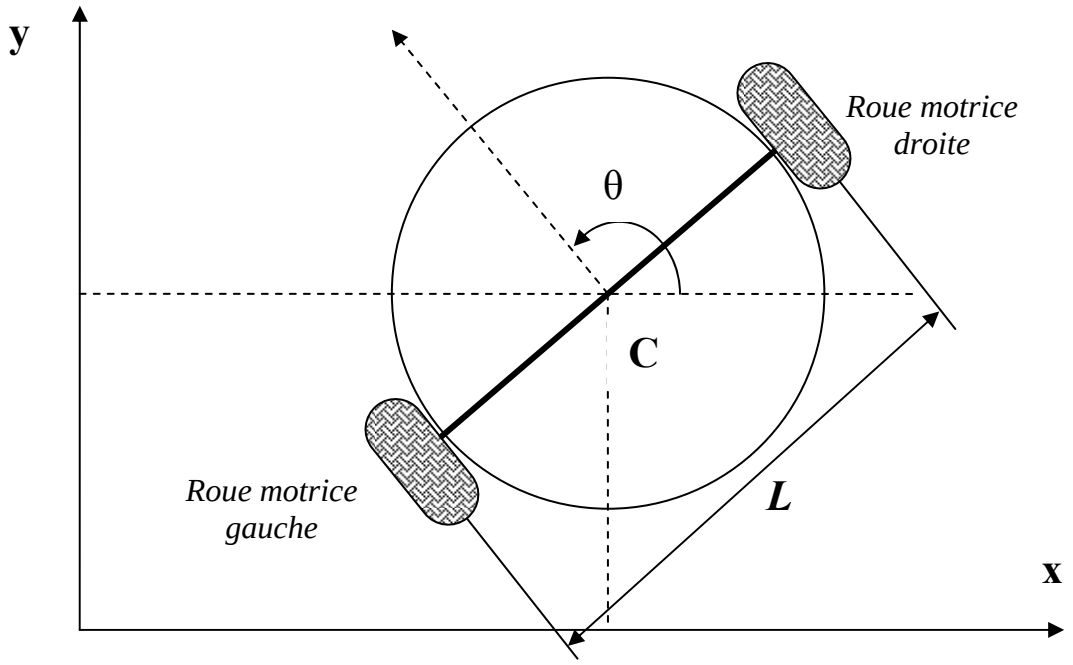


Figure 5.10 : Type du robot utilisé

Ce robot est caractérisé par les équations cinématiques suivantes :

5.4.1-La position du robot :

$$x_r(k+1) = x_r(k) + v(k) * \cos\left(\theta(k) + \frac{\Delta\theta(k)}{2}\right) \quad \textbf{.5.8}$$

$$y_r(k+1) = y_r(k) + v(k) * \sin\left(\theta(k) + \frac{\Delta\theta(k)}{2}\right) \quad \textbf{.5.9}$$

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

5.4.2-L'orientation du robot :

$$\theta(k+1) = \theta(k) + \Delta\theta(k) \quad \textbf{.5.10}$$

Avec :

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

5.4.3-la vitesse linière du robot :

$$v(k) = \frac{1}{2}(\theta_r(k) + \theta_l(k)) \quad \textbf{5.11}$$

Avec :

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

$\theta_l(k)$: La vitesse angulaire de la roue droite du robot.

5.4.4-La variation de thêta :

$$\Delta\theta(k) = \frac{1}{2 * L}(\theta_r(k) - \theta_l(k)) \quad \textbf{5.12}$$

Où :

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

5.5-Algorithme et résultat de simulation :

5.5.1-Algorithme :

1-Initialisation aléatoire des poids W du réseau de neurone ;

2-Donner la position initiale du robot [$\mathbf{(X_r(0), Y_r(0), \theta_r (t))}$];

Pour t=1 jusqu'a *t* itération

3-Lecture de l'état S_t de l'environnement par les sept capteurs (***G, A, D***) ;

4-Pour un état fixe, calcul de la fonction ***Q*** par le réseau de neurone pour les trois actions possible (***TG, AV, TD***);

5-Determination de l'action optimale c'est l'action qui correspond à la plus grande valeur de *Q* (**Action_optimale** $\rightarrow$ **Max (*Q*)**) ;

6-Exécuter l'action optimale avec une probabilité **ε** ou d'une action aléatoire avec une probabilité **1-ε** ;

7-Lecture du nouvel état S_{t+1} de l'environnement par les Sept capteurs (***G, A, D***) ;

8-Renforcement ;

9-Test du renforcement

Si **r=-1** (il y a un obstacle) ou **r=0**

1- Mise à jour des poids et des biais du réseau de neurone avec la formule du ***Q-Learning***; Utilisation des équations : **(5.3), (5.4), (5.6), (5.7)**.

2-Retour à l'état initial.

Fin si

10- $\varepsilon = \varepsilon * 0.99$ pour diminuer ε progressivement ;

Fin Pour

5.5.2-Quelques Résultats de simulation :

Dans cette partie de ce chapitre nous allons présenter quelques résultats obtenus lors de notre simulation, où le pas d'apprentissage $\alpha = 0.9$, le coefficient d'apprentissage $\gamma = 0.9$ et le nombre de neurones dans la couche cachée est 3.

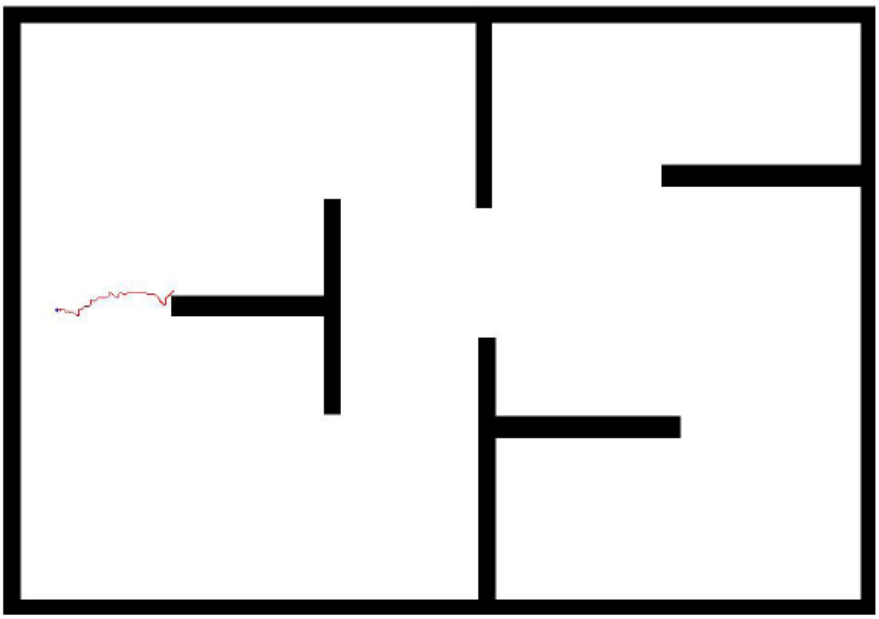


Figure 5.11 : Trajectoire du robot pendant l'apprentissage (1500 itérations).

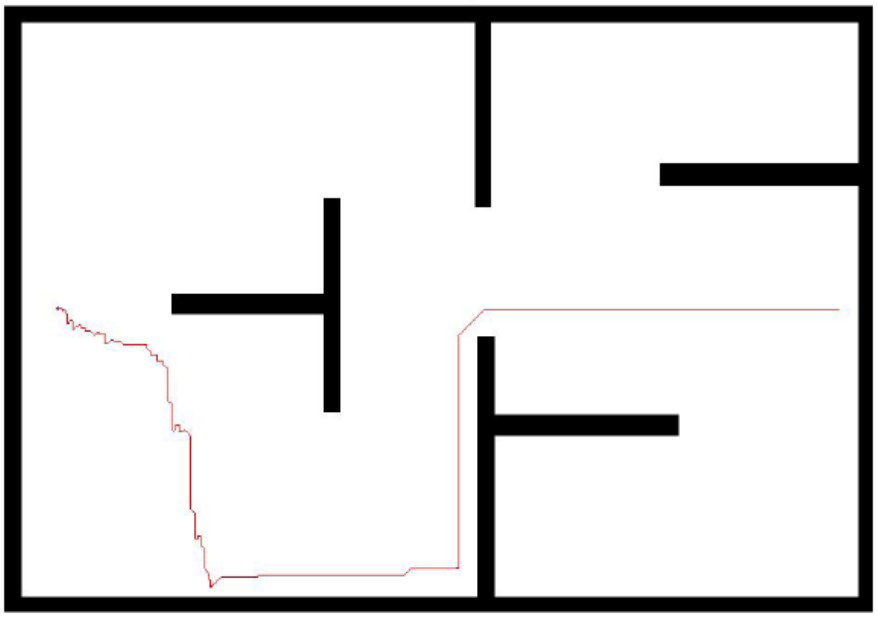


Figure 5.12 : Trajectoire du robot après apprentissage (1500 itérations).

On peut constater sur les figures (5.11) et (5.12) que le déplacement du robot est conforme à notre objectif qui est l'évitement d'obstacles.

Pour un test de notre réseau de neurones nous avons utilisé trois autres environnements où on a changé les conditions initiales, les résultats obtenus sont présentés sur les figures (5.13), (5.14) et (5.15).

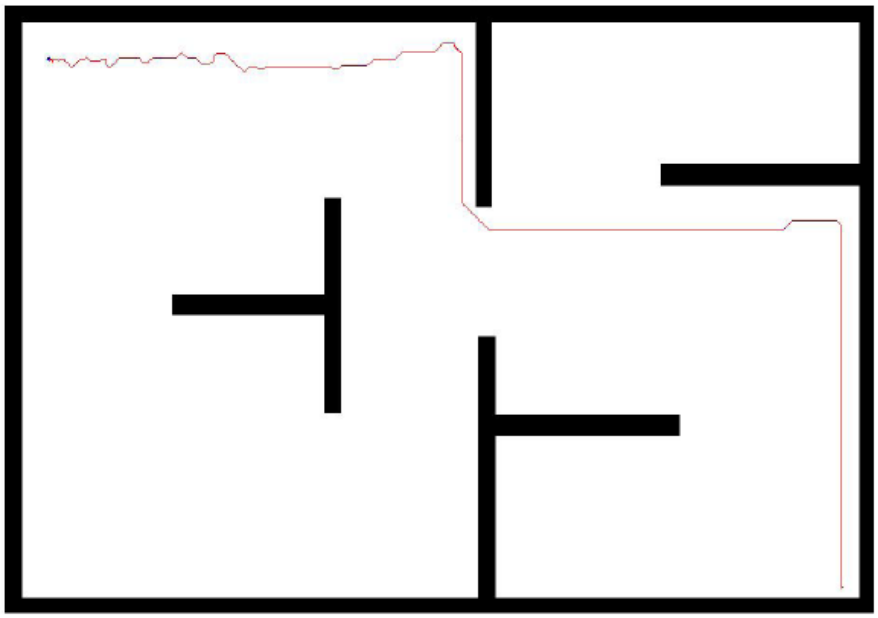


Figure 5.13 : Trajectoire du robot après apprentissage (1500 itérations), avec changement du point de départ.

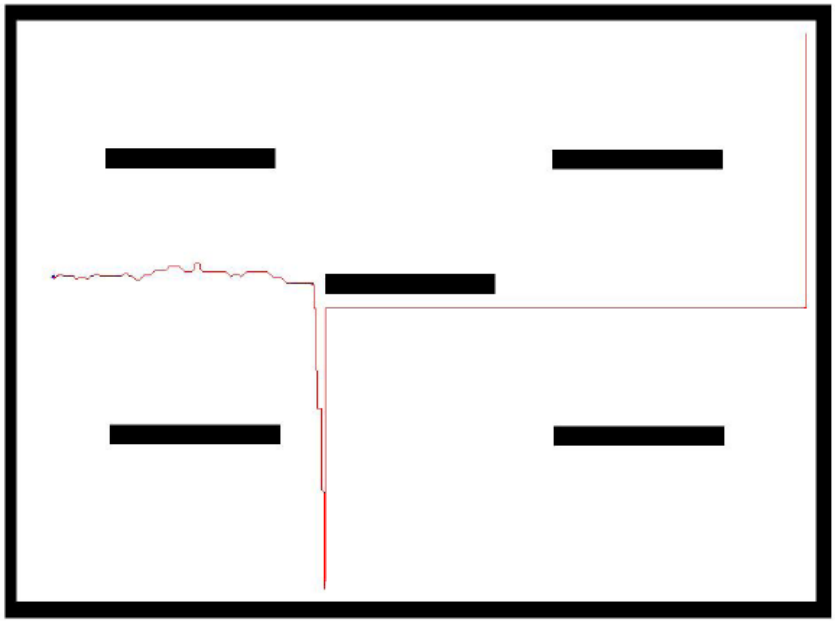


Figure 5.14 : Trajectoire du robot après apprentissage (1500 itérations), avec changement d'environnement.

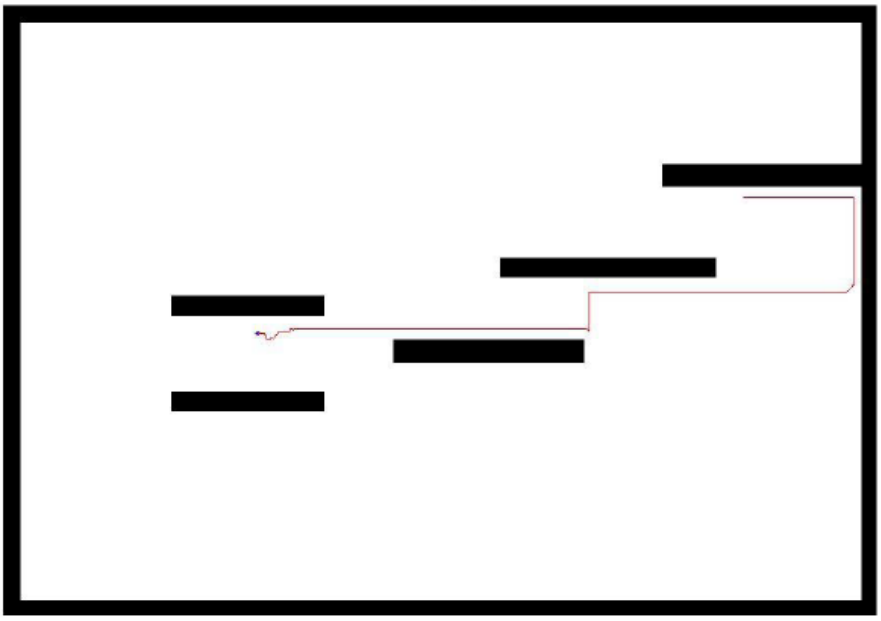


Figure 5.15 : Trajectoire du robot après apprentissage (1500 itérations), avec changement d'environnement et point de départ.

D'après ces figures, on remarque que la trajectoire parcourue par le rebot réalise notre objectif qui est l'évitement d'obstacles, malgré le changement total de l'environnement.

5.6-Conclusion :

L'architecture du système d'apprentissage par renforcement que nous avons proposé est basée sur l'utilisation de l'algorithme ***Q-Learning***. Cette méthode nous a permis d'avoir des résultats acceptables et qui répondent aux objectifs visés.

Conclusion Générale

L'objectif principal de ce travail, est de développer un système d'apprentissage qui permet à un robot mobile d'éviter les obstacles qui peut rencontrer dans son environnement de travail, qui est totalement inconnu. Ainsi on a opte pour la méthode des différences temporelles comme méthode de résolution de ce type d'apprentissage qui s'inscrit dans l'ensemble des méthodes d'apprentissage par renforcement ou beaucoup des travaux pour différentes applications ont été réalises. L'un des algorithmes d'apprentissage qui a suscite beaucoup d'importance est le ***Q-Learning***, cet algorithme est basé sur une fonction appelée fonction valeur « état - action » (***Q(s, a)***).
Les fonctions ***Q*** peuvent être déterminées soit par un Tableau à qui on se réfère à chaque fois qu'on a besoin de stocker ou de lire ces fonctions ou bien déterminées d'une façon systématique et plus convenable par un réseau de neurone artificiel. Pour présenter la fonction ***Q*** en utilisant les réseaux de neurones, il est préférable de procéder soit par un seul réseau ou plusieurs réseaux séparés, chacun avec une seule sortie.

Le développement de ce système d'apprentissage est basé sur, la perception de l'environnement de travail du robot qui présente l'état de cet environnement, les actions possibles et l'architecture du réseau de neurone.

La perception de l'environnement de travail est assurée par des capteurs de proximité ou distance, placées dans sept points différents tout autour du de la structure du robot (Avant, Droite, Gauche), ces capteurs retournent des valeurs binaires qui indiquent l'existence ou l'absence d'obstacles, et comme la méthode d'apprentissage par renforcement, est basée sur l'interaction du robot avec son environnement de travail

nous avons choisie pour cette raison trois actions possibles (Avancez, Tournez à Droite, Tournez à Gauche) qui peuvent êtres effectués par le robot. Ces données sont utilisées comme vecteur d'entrée pour le réseau de neurone artificiel du type ***MLP***, sur quoi l'apprentissage se fait par l'algorithme de rétro propagation.

Le choix ou la sélection de l'action qui est exécuté par le robot mobile, est basé sur la méthode ε-gloutonne (ε-greedy), cette méthode permet l'exploration et l'exploitation de l'environnement du robot, le signal de renforcement est un scalaire qui vaux **-1**, **0**, **+1**.

Les résultats obtenus montrent l'efficacité de l'approche. En effet cette technique a réussi a mettre en œuvre système décisionnel qui permet à un robot de ce déplacer dans un environnement totalement inconnu tout en évitant d'éventuels obstacles dans son milieu de travail.

La méthode utilisée pour choisir le changement des poids est critique à la vitesse d'apprentissage et aussi à la qualité de la solution finale. Jusqu'à présent il n'existe pas d'algorithme optimal (d'après la littérature), et plusieurs méthodes ont été proposé. En utilisant l'apprentissage en ligne, les poids sont mis a jour après présentation de chaque pair d'entrée/sortie.

Comme perspective, on pourra ajouté que l'apprentissage peut se faire dans un environnement qui contient plus d'un robot mobile, et de la on tombe dans la situation des systèmes multi agents

Bibliographie

[1] : Planification, Ordonnancement et Apprentissage par renforcement.

Camille Besse Patin. 2005, D.A.M.A.S. U. Laval.

[2] : Robotique Mobile. David Filliat Année 2004.

[3] : Apprentissage par renforcement et systèmes distribués -Youssef Zennir. Thèse doctorat juillet 2004.

[4] : Exploration guidée et induction de comportement géométrique en Apprentissage par renforcement - Pascal Garcia. Thèse doctorat juillet 2004.

[5] : Apprentissage des systèmes d'inférence floue par la méthode de renforcement génétique- Nemra Abdelkrim Mémoire Magister 2006

[6] : Application de la logique floue à la commande des processus complexes de la manipulation mobile. Abdessemed Foudil Thèse doctorat d'état -Es -Science 2003.

[7] : Apprentissage de système d'inférence floue par des méthodes de renforcement. Jouffe Lionel Thèse doctorat 1997.

[8]- Cours Rafic Younes 2005. « Chapitre 3 RN. Pdf ».

Site WEB : http://www.ryounes.net/cours/

[9]-Apprentissage ; Barra Vincent 2005/2006 I.S.I.M.A

[10]-Etude de la tolérance aux aléas logiques des réseaux de neurones artificiels.

Ammar Assoum thèse doctorat 1997.

[11] : Les réseaux se neurones Rachid LADJADJ 2002/2003 Site WEB:

http://www-igm.univ-mlv.fr/~dr/XPOSE2002/Neurones/index.php?rubrique=Accueil

[12] : Réseau de neurones. Marc Parizeau 2004.U.Laval

[13] : Introduction au connexionnisme. Claude Touzet Juillet 1992

[14] : Les multi layer perceptron (MLP) Clément Châtelain, novembre 2003.

[15] : Apprentissage par renforcement pour la navigation d'un robot mobile dans des environnements inconnus. M. Tarek Madani, Mémoire de stage DEA Robotique 2000. Université Paris XII.

[16] : Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance. BING-OIANG HUANG. Guang-Yicao.Min Guo. Août 2005

[17] : Apprentissage et Contrôle –apprentissage par renforcement- Frédéric Garcia 2004. Département De mathématique appliquée INRA-UBIA.

[18] : Aide pour l'apprentissage par renforcement Fabien Montagne. Laboratoire d'informatique du Littoral. Septembre 2004.

[19] : Apprentissage Automatique Bruno Bouzy. Octobre 2005.

[20] : Les réseaux de neurones Artificiels – Introduction au connexionnisme Cours, exercices et Travaux pratiques- Claude Touzet juillet 1992.

[Glo. P.Y 99] : Algorithmes d'apprentissage pour les systèmes d'inférence floue.

Liste Des Figures

Figure 1.1 : Système d'apprentissage par renforcement 03

Figure 1.2 : Structure des processus stochastique 03

Figure 1.3 : Relation états, actions 08

Figure 1.4 : Parties d'un système d'apprentissage par renforcement 10

Figure 2.1: Structure générale des systèmes d'inférence flous 19

Figure 2.2: Interprétation de l'inférence dans un système de type Mamdani 22

Figure 2.3: Les états possibles 29

Figure 2.4 : Résultat de simulation pour $\tau = 0.5$ 32

Figure 3.1 : Neurone Biologique 33

Figure 3.2 : Neurone Formel 34

Figure 3.3 : Fonction Heaviside 36

Figure 3.4 : Fonction Signe 36

Figure 3.5 : Fonction Linéaire 36

Figure 3.6 : Fonction Linéaire à seuil 37

Figure 3.7 : Fonction sigmoïde 37

Figure 3.8 : Modèle général d'un neurone 38

Figure 3.9 : Réseau proactif monocouche (Perceptron) 40

Figure 3.10 : Réseau proactif complètement connecté avec une seule couche cachée 42

Figure 3.11 : Réseau récurent avec neurones cachés 43

Figure 3.12 : Exemple de réseau MLP à une couche cachée avec 5 entrées, 3 neurones dans la couche cachée, et quatre 4 sorties 47

Figure 4.1 : Modèle général pour les algorithmes d'apprentissage par renforcement, ici le *Q-Learning* 54

Figure 4.2 : Renforcement reçus par l'agent 55

Figure 4.3 : Valeurs de Q calculées 56

Figure 4.4 : Génération de la fonction de valeur par un réseau de neurone 58

Figure 4.5 : Résumé des notations 59

Figure 5.1 : Structure du système d'apprentissage par renforcement 63

Figure 5.2 : 1er Etat d'environnement 64

Figure 5.3 : 2em Etat d'environnement 64

Figure 5.4 : 3em Etat d'environnement 65

Figure 5.5 : 4em Etat d'environnement 65

Figure 5.6 : 5em Etat d'environnement 66

Figure 5.7 : 6em Etat d'environnement 66

Figure 5.8 : Réseau de neurones utilisé pour la commande du robot 68

Figure 5.9 : Fonction d'activation et sa dérivée 69

Figure 5.10 : Type du robot 71

Figure 5.11 : Trajectoire du robot pendant l'apprentissage 74

Figure 5.12 : Trajectoire du robot après apprentissage 74

Figure 5.13 : Trajectoire du robot après apprentissage, avec changement du point de départ 75

Figure 5.14 : Trajectoire du robot après apprentissage, avec changement d'environnement 75

Figure 5.15 : Trajectoire du robot après apprentissage, avec changement d'environnement et point de départ 76

Liste Des Tableaux

Tableau 1.1 : L'algorithme de prédiction des méthodes Différences Temporelles 15

Tableau 1.2 : Algorithme Q-Learning 17

Tableau 2.1: Algorithme pour l'extraction des connaissances 26

Tableau 2.2 : Algorithme pour la fusion des connaissances 28

Tableau 3.1 : Analogie entre le neurone Biologie et le neurone Formel..... 35

Tableau 3.2 : Sommaire de l'algorithme rétropropagation de l'erreur..... 49