PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
- TLEMCEN -

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
- تلمسان -

Mémoire de fin d'études

Pour l'obtention du diplôme d'Ingénieur

Filière : Automatique

Spécialité : Automatique

Présenté par :

Abdelhadi AOUAICHIA et Othmane BOUHALLOUFA

Thème :

Apprentissage par renforcement pour l'ajustement automatique des régulateurs en boucle

Soutenue à huis clos, le 09/09/2020, devant le jury composé de :

Dr. Ghouti ABDELLAOUI,	MCB	ESSA-Tlemcen	Président
Dr. Sidi Mohammed ABDI,	MCB	ESSA-Tlemcen	Directeur de mémoire
Dr. Fouad MALIKI,	MCB	ESSA-Tlemcen	Examinateur 1
Pr. Ahmed TAHOUR,	Prof	ESSA-Tlemcen	Examinateur 2

Année universitaire : 2019/2020

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION
AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
- TLEMCEN -

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
- تلمسان -

Mémoire de fin d'études

Pour l'obtention du diplôme d'Ingénieur

Filière : Automatique

Spécialité : Automatique

Présenté par :

Abdelhadi AOUAICHIA et Othmane BOUHALLOUFA

Thème :

Apprentissage par renforcement pour l'ajustement automatique des régulateurs en boucle

Soutenue à huis clos, le 09/09/2020, devant le jury composé de :

Dr. Ghouti ABDELLAOUI,	MCB	ESSA-Tlemcen	Président
Dr. Sidi Mohammed ABDI,	MCB	ESSA-Tlemcen	Directeur de mémoire
Dr. Fouad MALIKI,	MCB	ESSA-Tlemcen	Examinateur 1
Pr. Ahmed TAHOUR,	Prof	ESSA-Tlemcen	Examinateur 2

Année universitaire : 2019/2020

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

À nos parents pour leur dévouement,

à nos proches pour leurs ténacités,

à tous nos enseignants,

et à tout lecteur de ce mémoire.

Remerciements

Tout d'abord, nous remercions le bon Dieu tout-puissant, de nous avoir donnés la force et l'audace de dépasser toutes les difficultés, et de mener à bien ce travail. Au nom d'ALLAH le clément et le miséricordieux.

Nous tenons à remercier notre encadrant et enseignant monsieur Sidi Mohammed ABDI qui nous a guidés avec cordialité et bienveillance durant le temps de ce mémoire. Il est le premier à nous avoir accordés sa confiance en acceptant notre proposition du thème, bien qu'à ce moment, nous n'avons que peu de connaissances en apprentissage par renforcement.

De même, nous souhaitons remercier les différents membres constituant le jury de mémoire pour avoir accepté de donner de leur temps pour évaluer notre travail.

Enfin, nous voudrons remercier nos familles et nos proches pour leurs soutiens sans failles qu'ils nous ont apportés depuis toujours. Nous remercions également tous nos amis et tous ceux qui ont été impliqués d'une manière ou d'une autre dans la réussite de ce travail.

Table des matières

Table des figures

Chapitre 4 : Applications et résultats **63**

Liste des tableaux

Introduction

> Douter de tout ou tout croire, ce sont deux solutions également commodes qui l'une et l'autre nous dispensent de réfléchir.
>
> Henri Poincaré

1 Motivation

L'idée que nous apprenons en interagissant avec notre environnement est probablement la première qui nous vient à l'esprit lorsque nous réfléchissons à la nature de l'apprentissage. Lorsqu'un enfant joue, agite les bras ou regarde autour de lui, il n'a pas d'enseignant explicite, mais il a une connexion sensorimotrice directe avec son environnement. L'exercice de cette connexion produit une richesse d'informations sur les causes et les effets, sur les conséquences des actions et sur ce qu'il faut faire pour atteindre les objectifs. Tout au long de notre vie, ces interactions sont sans aucun doute une source majeure de connaissances sur notre environnement et sur nous-mêmes. Que nous apprenions à conduire une voiture ou à tenir une conversation, nous sommes très conscients de la façon dont notre environnement réagit à ce que nous faisons, et nous cherchons à influencer ce qui se passe par notre comportement. L'apprentissage par l'interaction est une idée fondamentale qui sous-tend presque toutes les théories de l'apprentissage et de l'intelligence.

Dans ce mémoire, nous explorons la conception d'un algorithme qui est efficace pour résoudre des problèmes d'apprentissage d'intérêt scientifique, en évaluant les résultats par le biais d'analyses mathématiques ou d'expériences informatiques. L'approche que nous explorons, appelée apprentissage par renforcement

(Reinforcement Learning), est plus orienté vers l'apprentissage par essais-erreurs en interagissant avec l'environnement que les autres approches d'apprentissage.

2 Problématique

Dans le domaine de l'automatique, la régulation est l'une des plus importantes techniques de l'ingénieur offrant les méthodes et outils nécessaires à la prise de contrôle d'un système physique, en vue d'en imposer le comportement. Cette prise de contrôle s'effectue le plus souvent par des régulateurs en boucle, qui nécessite un ajustement des paramètres afin de répondre au cahier des charges.

Le problème de la détermination de ces paramètres est connu par la synthèse des régulateurs. Les méthodes de synthèse sont très nombreuses et l'application de la bonne méthode n'est toujours pas triviale. Pour cette raison, cette discipline ne consiste plus à ajuster les paramètres d'un régulateur mais plutôt de mettre en œuvre des méthodes simples, rapides et suffisamment précises.

D'autre part, les algorithmes d'intelligence artificielle ont prouvés des bons résultats dans différents domaines d'application, notamment les algorithmes d'apprentissage par renforcement pour les applications de commande et de régulation autonome.

Par conséquent, la problématique principale de ce mémoire est d'adapter un algorithme d'apprentissage par renforcement pour faire un ajustement des paramètres d'un régulateur en boucle.

L'un des problèmes des méthodes de synthèse classiques que nous cherchons à résoudre, est le respect de cahier des charges, où plusieurs méthodes ne prendre pas en considération les performances exigées par le cahier des charges.

3 Plan de mémoire

Dans le chapitre 1, nous détaillerons les différents éléments consécutifs de l'apprentissage par renforcement, en décrivant les fondements théoriques de cet apprentissage et l'entraînement des réseaux de neurones. Nous analyserons en particulier deux algorithmes, le Q-Learning et le Deep Q-Network.

Pendant le chapitre 2, nous donnerons un petit rappel et des généralités sur les systèmes asservis, et la régulation en boucle fermée par un contrôleur Proportionnel-Intégral-Dérivé.

Lors du chapitre 3, nous expliquerons l'implémentation des deux algorithmes le Q-Learning et le Deep Q-Network en adoptant ce dernier pour le reste de notre mémoire. Par la suite nous verrons plusieurs expériences sur cet algorithme afin d'augmenter encore son efficacité.

Finalement, au fil du chapitre 4, nous appliquerons notre algorithme sur quelques systèmes, en discutant les résultats obtenus.

Nous terminerons par une conclusion de ce que nous avons fait, et les perspectives que nous imaginons pour les futures recherches.

Chapitre 1

Initiation à l'apprentissage par renforcement

Ce chapitre sert principalement à introduire des notions de base et la structure de l'apprentissage par renforcement et certains algorithmes que nous utiliserons dans le troisième chapitre, en particulier, le Q-Learning et le Deep Q Network avec des réseaux de neurones.

1.1 Machine Learning

Machine Learning (ML) ou l'apprentissage machine est un type d'intelligence artificielle où la machine atteint l'intelligence en apprenant à partir des données ou des expériences. Contrairement à la programmation traditionnelle où le programme contient diverses réponses basées sur plusieurs conditions, l'algorithme ML déduit les règles qui génèrent les sorties en observant les exemples d'entrée.

Les connaissances acquises par le ML peuvent se généraliser au-delà des échantillons d'entraînement en interprétant avec succès des données que l'algorithme n'a jamais vues auparavant, alors qu'un programme codé en programmation traditionnelle ne peut effectuer que les réponses incluses dans le code.

« Machine Learning est la science qui consiste à faire agir les ordinateurs sans qu'ils soient explicitement programmés »

1.2 Types de Machine Learning

Un type de Machine Learning désigne généralement un groupe des algorithmes spécifiques qui conviennent à la résolution d'un type de problème particulier.

Essentiellement, les types de ML les plus populaires sont l'apprentissage supervisé, l'apprentissage non supervisé et l'apprentissage par renforcement.

1.2.1 Apprentissage supervisé

Un algorithme d'apprentissage supervisé (supervised learning) est appliqué lorsque nous sommes très clair sur le résultat à atteindre à partir d'un problème, mais que nous ne sommes pas sûr des relations entre les données d'entrée qui affectent la sortie. Nous aimerions que l'algorithme de ML que nous appliquons sur les données perçoive ces relations entre les différents éléments de données afin d'obtenir le résultat souhaité. Les données de ce type d'apprentissage sont dites étiquetées qui signifié qu'elles sont catégorisées et leurs sorties sont connus.

Il existe plusieurs algorithmes d'apprentissage supervisé, parmi les plus connus dans ce domaine : Classification And Regression Trees (CART), Support Vector Machine (SVM), Artificial Neural Networks (ANN) pour les données bien structuré, Convolutional Neural Network (CNN) appliqué pour la reconnaissance des images, et Recurrent Neural Network (RNN) pour le traitement des audois et de la langue naturelle.

1.2.2 Apprentissage non supervisé

La disponibilité de données étiquetées n'est pas très courante et l'étiquetage manuel des données n'est pas non plus toujours possible. C'est là que l'apprentissage non supervisé (unsupervised learning) entre en jeu. Lorsque les données sont fournies en entrée de l'algorithme, il essaie d'identifier les structures et de regrouper les données avec des types d'attributs similaires.

Parmi les nombreux algorithmes qui existent, nous trouvons : Equivalence Class Transformation (ECLAT) et Frequent Pattern Growth (FPG).

1.2.3 Apprentissage par renforcement

L'apprentissage par renforcement est une méthode d'autoformation pour résoudre des problèmes qui nécessitent de prendre des actions dans un environnement. Cette méthode permet aux machines et aux agents logiciels de déterminer automatiquement le comportement idéal afin de maximiser ses performances, en choisissant les meilleures actions basées sur la récompense des comportements souhaités et la punition des comportements non souhaités.

Les algorithmes d'apprentissage par renforcement étudient le comportement d'un agent dans un environnement et apprennent à optimiser ce comportement pour maximiser une récompense.

Il existe quatre algorithmes d'apprentissage par renforcement très populaire, à savoir Q-Learning, State-Action-Reward-State-Action (SARSA), Deep Q Network (DQN) et Deep Deterministic Policy Gradient (DDPG).

1.3 Le réseau de neurones artificiel

Les réseaux de neurones artificiels sont parmi les principaux outils utilisés en Machine Learning, ils sont efficaces pour résoudre les problèmes de reconnaissance des formes, de classification, de regroupement et d'approximation des fonctions.

Un réseau de neurones est constitué d'un ensemble de nœuds, appelés neurones formels, modelés sur le fonctionnement et la constitution d'un vrai neurone biologique, le réseau reçoit des entrées et sur la base de son traitement interne, il génère une sortie. La meilleure sortie est celle qui produit la plus petite erreur entre les valeurs réelles Y et les valeurs prédites par le réseau $\hat{Y}$ sur différents échantillons de l'ensemble de données.

Maintenant, nous allons essayer de comprendre comment un neurone formel est modélisé mathématiquement (figure 1.1).

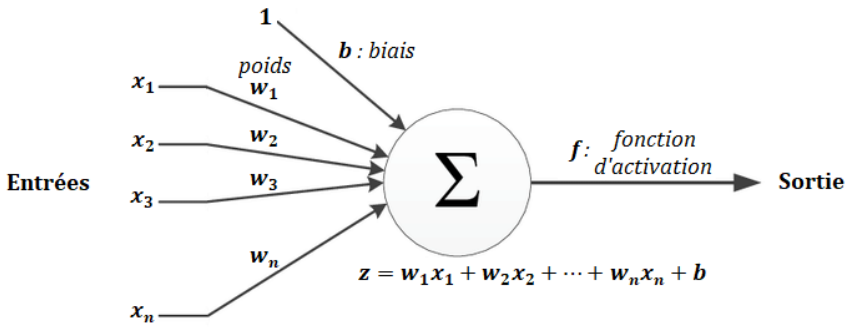


FIGURE 1.1: La structure d'un neurone formel

Les valeurs d'entrée $X = \{x_1, x_2, \ldots, x_n\}$ sont fournies au neurone, ces entrées sont pondérées avec les poids $W = \{w_1, w_2, \ldots, w_n\}$. En plus, le neurone pourrait avoir un biais b .

C'est ce vecteur de poids W que nous devons entraîner ou, en d'autres termes, trouver les valeurs optimales des éléments de poids afin d'obtenir le meilleur résultat.

Le traitement à l'intérieur du neurone est effectué à l'aide d'une fonction appelée « fonction d'activation ». L'entrée de la fonction d'activation f est égale à :

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

La sortie du neurone pourrait être représentée par $f(z)$ .Le biais peut-être considérer comme une entrée égale à 1 pondérée par le poids b .

Une fonction d'activation appropriée doit être choisie pour l'objectif visé par le réseau de neurones.

1.3.1 Fonction d'activation

Parmi les fonctions les plus utilisées comme fonctions d'activation :

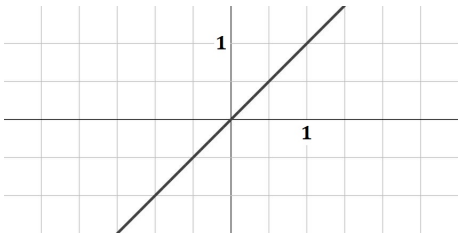
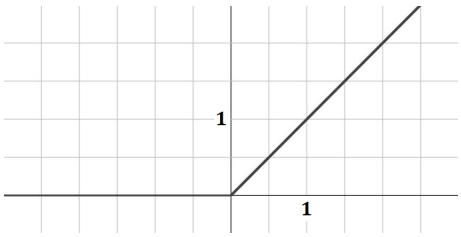
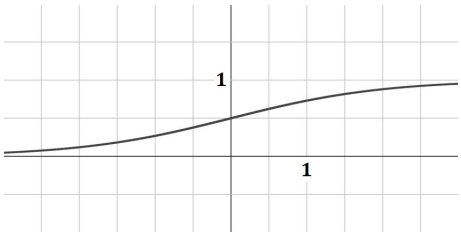
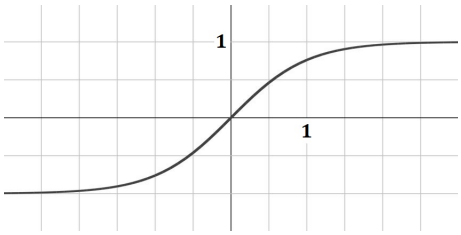
Nom	Graphe	Équation
Identité		$f(z) = z$
Rampe (Rectified Linear Units ReLU)		$f(z) = \begin{cases} 0 & si\ z < 0 \\ z & si\ z \geq 0 \end{cases}$
Sigmoïde		$f(x) = \frac{1}{1+e^{-z}}$
Tangente hyperbolique		$f(z) = \tanh(z)$ $= \frac{2}{1+e^{-2z}} - 1$

Tableau 1.1: Exemple de fonction d'activation

1.3.2 Perceptrons multicouches

Il existe de nombreuses structures de réseaux de neurones. Nous avons choisi de travailler avec des perceptrons multicouches pour des raisons de simplicité et pour leurs bonnes performances, en théorie ils peuvent approximer n'importe quelle fonction avec un nombre de neurones suffisant. Le réseau est organisé en couches où

chaque neurone est connecté à l'ensemble des neurones de la prochaine couche.

Un perceptron multicouche est obtenu en combinant plusieurs neurones sur plusieurs couches. Il s'agit d'un modèle non linéaire à la fois en sortie et en paramètre.

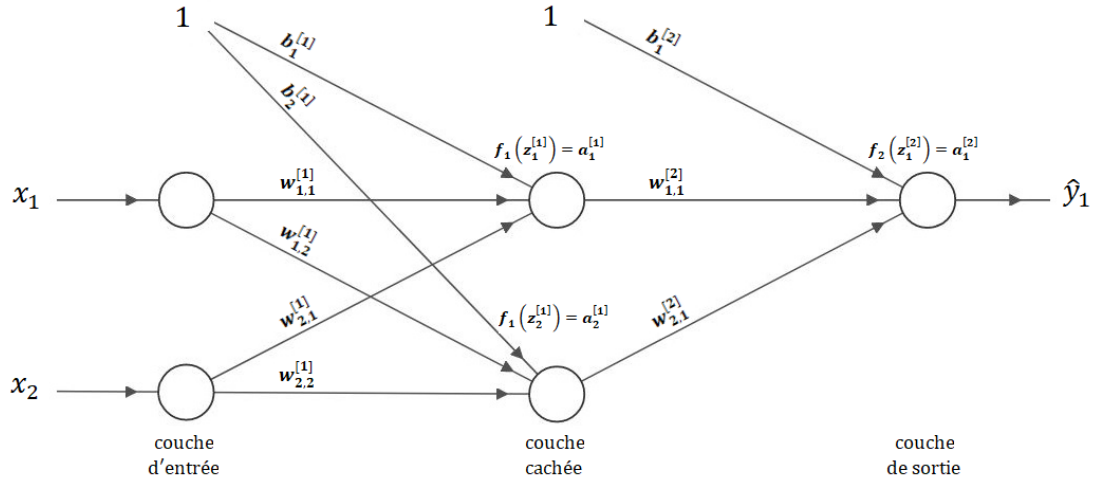


FIGURE 1.2: Un perceptron multicouches

- Les sorties de la couche d'entrée :

$$X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Les sorties de la couche cachée :

$$\begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix} = \begin{bmatrix} f_1\left(z_1^{[1]}\right) \\ f_1\left(z_2^{[1]}\right) \end{bmatrix} = \begin{bmatrix} f_1\left(w_{1,1}^{[1]}x_1 + w_{2,1}^{[1]}x_2 + b_1^{[1]}\right) \\ f_1\left(w_{1,2}^{[1]}x_1 + w_{2,2}^{[1]}x_2 + b_2^{[1]}\right) \end{bmatrix}$$

- La sortie de la couche de sortie :

$$\left[a_1^{[2]}\right] = \left[f_2\left(z_1^{[2]}\right)\right] = \left[f_2\left(w_{1,1}^{[2]}a_1^{[1]} + w_{2,1}^{[2]}a_2^{[1]} + b_1^{[2]}\right)\right]$$

1.3.3 Processus d'approximation

a) Initialisation :

La première étape de l'apprentissage c'est de commencer quelque part. Les réseaux de neurones peuvent commencer de n'importe où, c'est pourquoi une

initialisation aléatoire des poids est une pratique courante.

Le raisonnement est que nous pouvons atteindre le modèle pseudo-idéal par un modèle approprie et un nombre d'itérations suffisant.

b) Propagation en avant (forward-propagation) :

Nous partons des données dont nous disposons, nous les faisons passer par les couches du réseau et nous calculons directement les sorties.

Généralement, l'apprentissage dans un réseau de neurones ne se fait pas par une seule donnée mais plutôt par un nombre d'exemplaire m appelé lot (batch), l'entrée serait donc la matrice X de taille $n \times m$ (n est le nombre de neurones de la couche d'entrées et m est le nombre d'exemplaire pour chaque entrée) :

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix}$$

On note aussi $W^{[1]}$ la matrice des poids de la couche cachée de taille $n_1 \times n$ (n_1 est le nombre de neurones de la première couche cachée), et le vecteur des biais $B^{[1]}$ de taille $n_1 \times 1$:

$$W^{[1]} = \begin{bmatrix} w_{1,1}^{[1]} & w_{2,1}^{[1]} & \dots & w_{n,1}^{[1]} \\ w_{1,2}^{[1]} & w_{2,2}^{[1]} & \dots & w_{n,2}^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,n_1}^{[1]} & w_{2,n_1}^{[1]} & \dots & w_{n,n_1}^{[1]} \end{bmatrix} \quad , \quad B^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{n_1}^{[1]} \end{bmatrix}$$

La matrice des poids de la couche de sortie $W^{[2]}$ est de taille $n_2 \times n_1$ (n_2 est le nombre de neurones de la couche de sortie) avec un vecteur des biais $B^{[2]}$ de taille $n_2 \times 1$.

L'entrée des neurones de la couche cachée est la matrice $Z^{[1]}$ de taille $n_1 \times m$, et la matrice de sortie est $A^{[1]}$ de taille $n_1 \times m$:

$$Z^{[1]} = \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & \dots & z_1^{[1](m)} \\ z_2^{[1](1)} & z_2^{[1](2)} & \dots & z_2^{[1](m)} \\ \vdots & \vdots & \ddots & \vdots \\ z_{n_1}^{[1](1)} & z_{n_1}^{[1](2)} & \dots & z_{n_1}^{[1](m)} \end{bmatrix} = W^{[1]}X + B^{[1]} \quad , \quad A^{[1]} = f_1\left(Z^{[1]}\right)$$

L'entrée de la couche de sortie est la matrice $Z^{[2]}$ de taille $n_2 \times m$, et la matrice de sortie est $\hat{Y} = A^{[2]}$ de taille $n_2 \times m$:

$$Z^{[2]} = W^{[2]}A^{[1]} + B^{[2]} \quad , \quad \hat{Y} = A^{[2]} = f_2\left(Z^{[2]}\right)$$

c) Fonction de perte (loss function) :

Pendant l'apprentissage du réseau, nous devons calculer l'erreur (loss) pour optimiser les poids dans chaque couche. Cette erreur est calculée à l'aide d'une fonction de perte.

La fonction de perte est une mesure de performance du réseau pour atteindre son objectif de produire des résultats proches des valeurs souhaitées.

Nous avons $\hat{Y} = \begin{bmatrix} \hat{Y}^{(1)} & \hat{Y}^{(2)} & \dots & \hat{Y}^{(m)} \end{bmatrix}$ la sortie prédite de notre réseau neuronal, et $Y = \begin{bmatrix} Y^{(1)} & Y^{(2)} & \dots & Y^{(m)} \end{bmatrix}$ la sortie réelle souhaitée.

Parmi les fonctions de perte utilisées pour chaque $\hat{Y}^{(i)}$ (i entre 1 et m) de taille n_2 :

i. L'erreur absolue est la plus simple et intuitive :

$$loss\,(i) = \left|Y^{(i)} - \hat{Y}^{(i)}\right| = \sum_{j=1}^{n_2} \left|y_j^{(i)} - \hat{y}_j^{(i)}\right|$$

ii. La fonction erreur quadratique pour les problèmes de régression :

$$loss\,(i) = \frac{1}{2}\left(Y^{(i)} - \hat{Y}^{(i)}\right)^2 = \frac{1}{2}\sum_{j=1}^{n_2} \left(y_j^{(i)} - \hat{y}_j^{(i)}\right)^2$$

iii. La fonction entropie croisée (CrossEntropy) pour les problèmes de classification :

$$loss\,(i) = -\sum_{j=1}^{n_2} \left[\left(y_j^{(i)} \right) \log \left(\hat{y}_j^{(i)} \right) \; + \left(1 - y_j^{(i)} \right) \log \left(1 - \hat{y}_j^{(i)} \right) \right]$$

iv. La fonction de perte Huber (Huber loss) :

$$loss\,(i) = \sum_{j=1}^{n_2} e_j$$

Avec :

$$e_j = \begin{cases} \frac{1}{2}\left(y_j^{(i)} - \hat{y}_j^{(i)} \right)^2 & si \;\; \left| y_j^{(i)} - \hat{y}_j^{(i)} \right| < 1 \\ \left| y_j^{(i)} - \hat{y}_j^{(i)} \right| - \dfrac{1}{2} & ailleurs \end{cases}$$

On peut aussi utiliser d'autres formes suivant l'objectif du réseau de neurones.

d) Fonction de coût (cost function) :

Nous utilisons une fonction de coût (cost function) pour évaluer les performances du réseau sur des lots de taille m , et pour actualiser les poids et les biais :

$$cost = \frac{1}{m} \sum_{i=1}^{m} loss\,(i)$$

e) Descente de gradient :

Le gradient des poids :

$$\frac{\partial cost}{\partial W^{[k]}} = \frac{\partial cost}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial W^{[k]}}$$

Et le gradient des biais :

$$\frac{\partial cost}{\partial B^{[k]}} = \frac{\partial cost}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial B^{[k]}}$$

L'exemple suivant nous montre comment calculer les gradients des poids et des biais pour un réseau d'une seule couche cachée. Pour $m = 1$, $n_2 = 1$, des fonctions d'activation identités et une fonction de perte erreur quadratique, la fonction de coût est :

$$cost = \frac{1}{1} \sum_{i=1}^{1} loss\,(i) = loss\,(1) = \frac{1}{2} \sum_{j=1}^{1} \left(y_j^{(1)} - \hat{y}_j^{(1)} \right)^2 = \frac{1}{2} \left(y_1^{(1)} - \hat{y}_1^{(1)} \right)^2$$

- Gradient de $W^{[2]}$:

$$\frac{\partial cost}{\partial W^{[2]}} = \frac{\partial loss\,(1)}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial W^{[2]}}$$

Nous avons d'une part :

$$\frac{\partial loss\,(1)}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial Z^{[2]}} = \frac{\partial \frac{1}{2}\left(y_1^{(1)} - \hat{y}_1^{(1)}\right)^2}{\partial \hat{y}_1^{(1)}} \frac{\partial Z^{[2]}}{\partial Z^{[2]}} = -\left(y_1^{(1)} - \hat{y}_1^{(1)}\right)$$

Et d'autre part :

$$\frac{\partial Z^{[2]}}{\partial W^{[2]}} = \frac{\partial \left(W^{[2]} A^{[1]} + B^{[2]}\right)}{\partial W^{[2]}} = \left(A^{[1]}\right)^T$$

Donc :

$$\frac{\partial cost}{\partial W^{[2]}} = -\left(y_1^{(1)} - \hat{y}_1^{(1)}\right)\left(A^{[1]}\right)^T$$

- Gradient de $B^{[2]}$:

$$\frac{\partial cost}{\partial B^{[2]}} = \frac{\partial cost}{\partial \hat{Y}} \frac{\partial \hat{Y}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial B^{[2]}}$$

Nous avons :

$$\frac{\partial Z^{[2]}}{\partial B^{[2]}} = \frac{\partial \left(W^{[2]} A^{[1]} + B^{[2]}\right)}{\partial B^{[2]}} = 1$$

Donc :

$$\frac{\partial cost}{\partial B^{[2]}} = -\left(y_1^{(1)} - \hat{y}_1^{(1)}\right)$$

- Gradient de $W^{[1]}$:

$$\frac{\partial cost}{\partial W^{[1]}} = \left(W^{[2]}\right)^T \left(-y_1^{(1)} + \hat{y}_1^{(1)}\right)\left(X^{[1]}\right)^T$$

- Gradient de $B^{[1]}$:

$$\frac{\partial cost}{\partial B^{[1]}} = \left(W^{[2]}\right)^T \left(-y_1^{(1)} + \hat{y}_1^{(1)}\right)$$

f) Rétro-propagation (back-propagation) :

Nous avons le point de départ des erreurs, qui est la fonction de perte, et nous savons comment calculer le gradient de chaque poids. Donc nous pouvons propager l'erreur de la fin au début.

Un taux d'apprentissage α est introduit comme une constante entre 0 et 1 (généralement très faible), afin de forcer l'actualisation du poids à être très douce et lente (pour éviter les grands pas et les comportements chaotiques).

Nous utilisons la règle suivante pour l'actualisation des poids à chaque itération t , où $t-1$ est l'itération précédente :

$$W_t^{[k]} = W_{t-1}^{[k]} - \alpha \frac{\partial cost}{\partial W^{[k]}}$$

Et pour l'actualisation des biais :

$$B_t^{[k]} = B_{t-1}^{[k]} - \alpha \frac{\partial cost}{\partial B^{[k]}}$$

g) Convergence :

Comme nous mettons à jour les poids avec un petit pas, il faudra plusieurs itérations pour apprendre le réseau et minimiser l'erreur.

Le gradient de la fonction de perte nous assure que la sortie de notre réseau converge vers la valeur réelle $\hat{Y} = Y$.

1.4 Structure d'apprentissage par renforcement

1.4.1 Agent

L'agent est de loin la pièce la plus importante de l'apprentissage par renforcement car il contient l'intelligence nécessaire pour prendre des décisions et recommander l'action optimale dans une situation donnée, il interagit avec son environnement dans lequel il est placé. Ces interactions se produisent de manière séquentielle dans le temps.

1.4.2 Politique (policy)

La politique est une fonction qui met en correspondance un état donné avec les probabilités de sélectionner chaque action possible dans cet état. Nous utiliserons le symbole π pour désigner une politique. Lorsqu'il s'agit de politiques, nous disons formellement qu'un agent « suit une politique ».

Le processus de décision dans le contexte de l'apprentissage par renforcement implique à la politique π (qui aide l'agent), à déterminer la meilleure action à prendre ou la meilleure transition à faire quand il est dans un état actuel spécifique.

1.4.3 La séquence état-action-récompense (state-action-reward)

À chaque étape, l'agent obtient une représentation de l'état de l'environnement, en fonction de cette représentation, l'agent sélectionne une action à exécuter, l'environnement passe alors dans un nouvel état et l'agent reçoit une récompense (ou une pénalité qui est une récompense négative) en conséquence de l'action précédente.

Ce processus de sélection d'une action à partir d'un état donné, de transition vers un nouvel état et de réception d'une récompense se déroule de manière séquentielle, encore et encore, ce qui crée ce que nous appelons une trajectoire qui montre la séquence état-action-récompense.

1.4.4 Notation

Nous avons un ensemble d'états S , un ensemble d'actions A , et un ensemble de récompenses R . Nous supposerons que chacun de ces ensembles comporte un nombre fini d'éléments. À chaque étape $t = 0, 1, 2, \ldots$, l'agent reçoit une représentation de l'état de l'environnement $S_t \in S$. Sur la base de cet état, l'agent sélectionne une action $A_t \in A$. Cela nous donne la paire état-action (S_t, A_t) . Le pas est alors incrémenté jusqu'à l'étape $t+1$ et l'environnement passe à un nouvel état $S_{t+1} \in S$. À ce moment, l'agent reçoit une récompense $R_{t+1} \in R$ pour l'action A_t effectuée à partir de l'état S_t .

Nous pouvons considérer le processus de réception d'une récompense comme une fonction arbitraire f qui met en correspondance les paires état-action et les récompenses. À chaque instant t , nous avons $f(S_t, A_t) = R_{t+1}$.

La trajectoire représentant le processus séquentiel de sélection d'une action à partir d'un état, de transition vers un nouvel état et de réception d'une récompense, elle peut être représentée par :

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

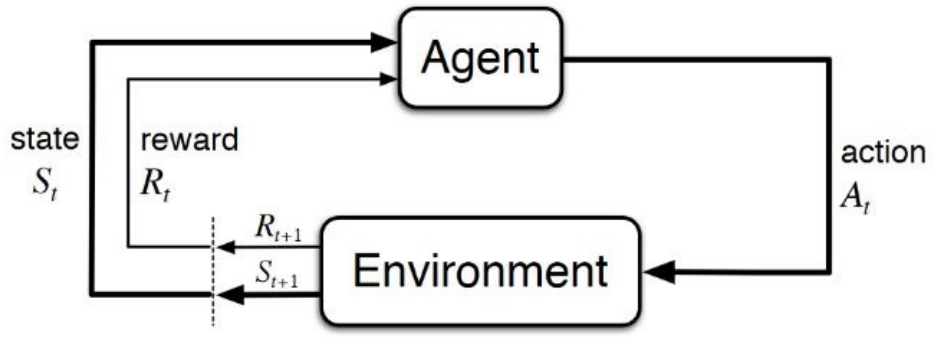


FIGURE 1.3: Structure d'apprentissage par renforcement

i. Au moment t , l'environnement est dans l'état S_t .

ii. L'agent observe l'état actuel et sélectionne l'action A_t .

iii. L'environnement passe à l'état S_{t+1} et accorde à l'agent la récompense R_{t+1} .

iv. Ce processus recommence ensuite pour l'étape suivante $t + 1$.

Notez que $t + 1$ n'est plus dans le futur, mais est maintenant le présent. Lorsque nous traversons la ligne pointillée en bas à gauche, le diagramme montre que l'instant $t + 1$ se transforme en instant t , de sorte que S_{t+1} et R_{t+1} sont maintenant S_t et R_t .

1.5 Processus décisionnel de Markov

Le processus décisionnel de Markov (Markov Decision Processes MDP) est la base de la structuration des problèmes qui sont résolus par « apprentissage par renforcement », c'est un moyen de formaliser la prise de décision séquentielle, il est formé de deux termes, à savoir « Markov » et le « Decision Processes ».

Le terme « Markov » fait référence à la « Propriété Markov » qui est le principe du phénomène de « chaîne de Markov » dont le MDP est une des formes. La propriété de Markov est également appelée la propriété « sans mémoire » pour les processus stochastiques (probabilistes ou incertains en termes plus simples).

Pour un processus qui est passé par plusieurs états, et qui se trouve maintenant dans un état donné spécifique S_t , si ce processus suit la Propriété de Markov, alors la distribution de probabilité conditionnelle de l'état suivant probable S_{t+1} ne dépendrait que de l'état actuel S_t , indépendamment de la séquence d'états de S_0 à

S_t par laquelle le processus est passé pour atteindre son état actuel spécifique :

$$Pr\left[S_{t+1}|S_t\right] = Pr\left[S_{t+1}|S_0, \ldots, S_t\right]$$

Par conséquent, même s'il existe plusieurs façons (séquences) d'atteindre un état particulier, quelle que soit la séquence que le processus adopte, la distribution de probabilité conditionnelle des états suivants à partir de cet état spécifique reste la même.

1.5.1 Objectif du MDP

Le processus décisionnel de Markov fournit une base mathématique pour la modélisation du processus de décision, où les résultats sont en partie sous notre contrôle (affectés par la décision d'action que nous avons prise), et en partie aléatoires (correspondant aux défis de l'estimation et aux incertitudes).

L'objectif du MDP ou d'un agent d'apprentissage par renforcement dans le cadre du MDP est de maximiser la récompense, cela nécessite de trouver une politique qui puisse le faire.

Le processus qui consiste à prendre des mesures conformément à la politique dans chaque état ultérieur, est réduit à une chaîne de Markov.

1.6 La récompense

La récompense est une valeur numérique reçue par l'agent de l'environnement en réponse directe à ses actions. Toutes les actions génèrent des récompenses, qui peuvent être divisées en trois types : les récompenses positives qui mettent en valeur une action souhaitée, les récompenses négatives qui mettent en valeur une action dont l'agent devrait s'écarter, et zéro qui signifie que l'agent n'a rien fait de spécial.

1.6.1 Rendement

Pour que l'agent atteint son but, il doit choisir les meilleurs actions pour maximiser ses récompenses, pour cela non seulement la récompense immédiate est prise en compte, mais aussi les récompenses futures cumulées à chaque instant t .

Pour formaliser ces récompenses cumulatives, nous introduisons le concept de rendement à un moment donné. Pour l'instant, nous pouvons considérer le rendement simplement comme la somme des récompenses futures. Mathématiquement, nous définissons le rendement G à l'étape t comme :

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

Où T est la dernière étape.

Ce concept est super-important car l'objectif de l'agent est de maximiser le rendement, ce qui pousse l'agent à prendre les décisions qu'il prend.

1.6.2 Rendement attendu

Dans certains cas l'état de l'environnement peut-être une fonction continue, et comme chaque état donne une récompense, le rendement risque de prendre des valeurs grandes, pour contrôler l'importance des récompenses futures nous introduisons le concept d'actualisation des récompenses.

L'objectif de l'agent est de maximiser le rendement attendu des récompenses (appelé aussi rendement actualisé).

Pour définir le rendement attendu, nous définissons d'abord le taux d'actualisation γ (discount rate), comme un nombre compris entre 0 et 1 . Le taux d'actualisation sera le taux pour lequel nous calculons les récompenses futures et déterminera la valeur actuelle des récompenses futures. Ainsi, nous définissons le rendement actualisé comme :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Cette définition du rendement attendu permet à notre agent de se soucier davantage de la récompense immédiate que des récompenses futures, puisque les récompenses futures seront plus fortement actualisées. Ainsi, si l'agent tient compte des récompenses qu'il s'attend à recevoir à l'avenir, les récompenses plus immédiates ont plus d'influence lorsqu'il s'agit pour lui de prendre une décision concernant une action particulière.

La relation ci-dessous montre comment les rendements à des moments successifs sont liés les unes aux autres.

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \dots) \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Même si le rendement à l'étape t est la somme d'un nombre infini de termes, le rendement est en fait fini tant que la récompense est non nulle et constante, et $\gamma < 1$.

Par exemple, si la récompense à chaque étape est un nombre constant égal à 1 et que $\gamma < 1$, alors selon le concept de convergence des séries infinies le rendement est :

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

1.6.3 Épisode

Dans notre définition du rendement , nous avons introduit la dernière étape T . Lorsque la notion de la dernière étape prend tout son sens, l'interaction entre l'agent et l'environnement se décompose naturellement en sous-séquences, appelées épisodes, chaque épisode étant un ensemble d'étapes.

L'épisode se termine dans un état terminal T , qui est suivi par la réinitialisation de l'environnement à un état de départ standard ou à un échantillon aléatoire à partir d'une distribution d'états de départ possibles. L'épisode suivant commence alors indépendamment de la façon dont l'épisode précédent s'est terminé.

1.7 La fonction état-valeur (la fonction valeur)

La fonction état-valeur pour la politique π , dénommée V_π , nous indique à quel point un état donné est bon pour un agent qui suit la politique π (la valeur d'un état sous π).

La valeur de l'état s dans le cadre de la politique π est le rendement attendu à

partir de l'état s au moment t et en suivant la politique π par la suite. Mathématiquement, nous définissons $V_\pi(s)$ comme :

$$V_\pi(s) = \mathbb{E}[G_t | S_t = s]$$

Le symbole $\mathbb{E}$ indique l'espérance d'une fonction stochastique, et le symbole | est l'opérateur de conditionnement.

1.8 La fonction action-valeur (la fonction-Q)

De même, la fonction action-valeur pour la politique π , dénommée Q_π , nous indique à quel point il est bon pour l'agent de prendre une action donnée à partir d'un état donné tout en suivant la politique π (la valeur d'une action sous π).

La valeur de l'action a dans le cadre de la politique π est le rendement attendu à partir de l'état s au moment t , en prenant l'action a et en suivant la politique π par la suite.

Mathématiquement, nous définissons $Q_\pi(s, a)$ comme :

$$Q_\pi(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

Par convention, la fonction action-valeur Q_π est appelée fonction-Q, et la sortie de la fonction pour toute paire état-action donnée est appelée valeur-Q.

1.9 Optimalité

L'objectif des algorithmes d'apprentissage par renforcement est de trouver une politique qui rapportera beaucoup à l'agent que toutes les autres politiques s'il suit effectivement cette politique.

1.9.1 Politique optimale

En termes de rendement, une politique π est considérée comme meilleure ou identique à la politique π' si le rendement attendu de π est supérieur ou égal au rendement attendu de π' pour tout les états. En d'autres termes :

$\pi \geq \pi'$ si et seulement si $V_\pi(s) \geq V_{\pi'}(s)$ pour tous $s \in S$.

N'oubliez pas que $V_\pi(s)$ donne le rendement attendu pour commencer dans l'état s et suivre π par la suite. Une politique qui est meilleure ou au moins identique à toutes les autres politiques est appelée une politique optimale.

1.9.2 Fonction état-valeur optimale

La politique optimale a une fonction état-valeur optimale associée. Nous désignons la fonction état-valeur optimale par V_* et la définissons comme :

$$V_*(s) = \max_\pi V_\pi(s)$$

Pour tous $s \in S$. En d'autres termes, V_* donne le plus grand rendement actualisé réalisable par tout politique π pour chaque état.

1.9.3 Fonction action-valeur optimale

De même, la politique optimale a une fonction action-valeur optimale (fonction-Q optimale). Nous la désignons par Q_* et la définissons comme :

$$Q_*(s,a) = \max_\pi Q_\pi(s,a)$$

Pour tous $s \in S$ et $a \in A(s)$. En d'autres termes, Q_* donne le plus grand rendement actualisé réalisable par tout politique π pour chaque paire possible état-action.

1.9.4 La programmation dynamique et l'équation de Bellman

La programmation dynamique est un moyen de simplifier un problème complexe en le décomposant en sous problèmes plus petits, elle vise à résoudre d'abord le problème le plus petit ou le plus simple, puis à utiliser de manière itérative la solution des petits problèmes pour résoudre les problèmes les plus importants jusqu'à ce que les sous solutions se combinent de manière récursive et conduisent à la solution du problème initial dans son ensemble.

Cette solution récursive est l'équation de Bellman qui est une condition nécessaire pour l'optimalité, elle applique le principe d'optimalité de Bellman :

« Une politique optimale a la propriété que quels que soient l'état initial et la décision initiale, les décisions suivantes doivent constituer une politique optimale par rapport à l'état résultant de la première décision ».

Il existe deux grandes approches qui peuvent être appliquées en utilisant la programmation dynamique pour résoudre le MDP à l'aide de l'équation de Bellman, à savoir l'itération de la valeur (value iteration) en calculant la fonction-Q de chaque état et en choisissant les meilleures actions, ou les méthodes d'itération de la politique (policy iteration) en calculant la fonction de valeur et en choisissant la meilleure politique.

1.9.5 Équation d'optimalité de Bellman pour Q_*

La fonction-Q optimale nous donne le rendement maximal dans l'état s en suivant l'action a :

$$\begin{aligned} Q_*(s,a) &= \max_\pi Q_\pi(s,a) \\ &= \max_\pi \mathbb{E}\left[G_t | S_t = s, A_t = a\right] \\ &= \max_\pi \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a\right] \\ &= \max_\pi \mathbb{E}\left[R_{t+1} + \gamma Q_\pi(s',a') | S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[R_{t+1} + \gamma \max_\pi Q_\pi(s',a') \mid S_t = s, A_t = a\right] \end{aligned}$$

Où s' et a' sont l'état et l'action suivants, et $Q_\pi(s',a')$ est la fonction-Q suivante.

En appliquant le principe d'optimalité, la fonction-Q suivante doit être optimale, ce qui nous donne l'équation de Bellman pour Q_* :

$$Q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q_*(s',a') \mid S_t = s, A_t = a\right]$$

Cette équation indique que, pour toute paire état-action (s,a) à l'instant t , en sélectionnant l'action a à partir de l'état s et en suivant la politique optimale, le rendement attendu (également connue comme la valeur-Q de cette paire) sera la récompense attendue que nous obtiendrons en prenant l'action a dans l'état s , qui

est R_{t+1} , plus le maximum du rendement actualisé attendu qui peut être obtenu de toute paire état-action suivante possible (s', a') .

Comme l'agent suit une politique optimale, l'état s' sera l'état à partir duquel la meilleure action suivante a' pourra être effectuée au moment $t + 1$.

Avec Q_* et pour tout état s , un algorithme d'apprentissage par renforcement peut trouver l'action a qui maximise $Q_*(s, a)$.

1.10 Le dilemme exploration/exploitation

Pour former une valeur optimale pour la fonction action-value (fonction-Q), nous pouvons commencer par des valeurs initialisées de façon aléatoire ou une initialisation fixe/heuristique par défaut. Les valeurs de la fonction-Q sont améliorées par rapport aux valeurs par défaut en effectuant des expériences sur les données, scénarios et/ou épisodes d'entraînement.

L'agent essaie de nombreuses actions différentes dans de nombreux états différents, afin d'apprendre toutes les possibilités disponibles et de trouver la trajectoire qui maximisera sa récompense totale, c'est ce que nous appelons « exploration », car l'agent explore son environnement.

En revanche, si l'agent ne fait qu'explorer, il ne maximisera jamais la récompense totale. Il doit également utiliser les informations apprises, c'est ce que nous appelons « exploitation », car l'agent exploite ses connaissances pour maximiser les récompenses qu'il reçoit.

Le compromis entre l'exploration et l'exploitation est l'un des plus grands défis des problèmes d'apprentissage par renforcement, car les deux doivent être équilibrés afin de permettre à l'agent à la fois d'explorer suffisamment l'environnement, mais aussi d'exploiter ce qu'il a appris et de répéter la trajectoire la plus profitable trouvée.

La stratégie que le mécanisme d'apprentissage utilise pour déterminer la prochaine meilleure action à entreprendre sur la base de l'état actuel est la « politique ».

1.10.1 La stratégie Epsilon Greedy (ε-Greedy)

Pour obtenir un équilibre entre l'exploitation et l'exploration, nous utilisons ce que nous appelons la stratégie Epsilon Greedy. Avec cette stratégie, nous définissons un taux d'exploration ε que nous fixons au départ à 1 . Ce taux d'exploration est la probabilité que notre agent explore l'environnement plutôt que de l'exploiter. Avec $\varepsilon = 1$, il est certain à 100 % que l'agent commencera par explorer l'environnement. Au fur et à mesure que l'agent en apprend davantage sur l'environnement, ε se dégradera à un taux que nous fixons de sorte que la probabilité d'exploration devient de moins en moins probable à mesure que l'agent en apprend de plus en plus sur l'environnement. L'agent exploitera davantage l'environnement une fois qu'il aura eu l'occasion de l'explorer et d'en apprendre davantage sur lui.

Pour déterminer si l'agent choisira l'exploration ou l'exploitation à chaque étape, nous générons un nombre aléatoire entre 0 et 1 . Si ce nombre est supérieur à ε , alors l'agent choisira sa prochaine action via l'exploitation. Sinon, son action suivante sera choisie via l'exploration.

1.11 Q-Learning

Q-Learning est un algorithme d'apprentissage par renforcement, considéré comme l'un des plus fondamentaux. Dans sa forme la plus simplifiée, il utilise un tableau appelé tableau-Q pour stocker toutes les valeurs-Q de toutes les paires état-action possibles. Il actualise ce tableau en utilisant l'équation de Bellman, tandis que la sélection des actions est généralement effectuée avec une politique ε-greedy.

1.11.1 Itération de la valeur

L'algorithme Q-Learning actualise les valeurs-Q pour chaque paire état-action en utilisant l'équation de Bellman jusqu'à ce que la fonction-Q converge vers la fonction-Q optimale Q_* . Cette approche est appelée itération des valeurs.

Comme l'agent ne sait rien de l'environnement ou des rendements attendues pour une paire état-action, toutes les valeurs-Q du tableau sont d'abord initialisées à zéro. Au fil du temps, lorsque l'agent effectue plusieurs épisodes, les valeurs-Q produites

pour les paires état-action que l'agent expérimente seront utilisées pour mettre à jour les valeurs-Q stockées dans le tableau-Q.

Les lignes du tableau-Q représentent les états, et les colonnes représentent les actions. Ainsi, les dimensions du tableau sont le nombre d'états par le nombre d'actions.

		Les actions			
		Action 1	Action 2	Action 3	...
Les états	S_0	0	0	0	0
	S_1	0	0	0	0
	S_2	0	0	0	0
	$\vdots$	0	0	0	0

Tableau 1.2: La forme du tableau-Q

Au fur et à mesure que le tableau-Q est actualisé, dans les actions et les épisodes ultérieurs, l'agent peut regarder dans le tableau-Q et baser sa prochaine action sur la valeur-Q la plus élevée pour l'état actuel.

1.11.2 Le taux d'apprentissage :

Le taux d'apprentissage est un nombre compris entre 0 et 1 , qui peut être considéré comme la vitesse à laquelle l'agent abandonne la précédente valeur-Q dans le tableau-Q pour la nouvelle valeur-Q.

Par exemple, supposons que nous avons une valeur-Q pour une paire état-action arbitraire que l'agent a connue dans une étape précédente. Si l'agent se trouve dans la même paire état-action dans une étape ultérieure, une fois qu'il a appris davantage sur l'environnement, la valeur-Q devra être actualisée pour refléter le changement des estimations que l'agent a maintenant pour les rendements futurs.

Nous ne voulons pas simplement remplacer l'ancienne valeur-Q, mais nous utilisons plutôt le taux d'apprentissage comme outil pour déterminer la quantité d'informations que nous conservons sur la valeur-Q précédemment déterminée par rapport à la nouvelle valeur-Q calculée pour la même paire état-action à une étape ultérieure.

Nous indiquerons le taux d'apprentissage avec le symbole α , plus le taux d'apprentissage est élevé, plus l'agent adoptera rapidement la nouvelle valeur-Q, par exemple, si le taux d'apprentissage est de 1 , l'estimation de la valeur-Q pour une paire état-action donnée sera la nouvelle valeur-Q calculée directement et ne tiendra pas compte des valeurs-Q précédentes calculées aux étapes précédentes.

1.11.3 Actualisation de la valeur-Q

Pour mettre à jour la valeur-Q de toute action effectuée à partir de l'état précédent, nous utilisons l'équation d'optimalité de Bellman.

Nous souhaitons que la valeur-Q pour la paire état-action donnée soit aussi proche que possible de la partie droite de l'équation de Bellman afin que la valeur-Q converge finalement vers la valeur optimale Q_* .

La formule de calcul de la nouvelle valeur-Q pour la paire état-action (s, a) à l'étape t est la suivante :

$$Q^{new}(s,a) = (1-\alpha) \underbrace{Q(s,a)}_{\substack{ancienne \\ valeur}} + \alpha \overbrace{\left(R_{t+1} + \gamma \max_{a'} Q_*(s',a') \right)}^{\substack{nouvelle \\ valeur}}$$

Où s' est l'état suivant et a' toute action qui peut être exécuter à partir de cet état.

1.11.4 Algorithme de Q-Learning

Algorithme 1 : Q-Learning

Initialisation : nombre des épisodes, nombre des étapes par épisode T , taux d'actualisation γ , taux d'apprentissage α , taux d'exploration ε;
Créer un tableau-Q de taille nombre d'actions par nombre d'états, et initialisé toutes ses valeurs à zéro;
tant que *Nombre d'épisode < nombre d'épisode maximal* **faire**
    Commencer par l'état initial;
    tant que *Nombre d'étape t < nombre d'étape maximal T* **faire**
        Sélectionnez une action par exploration (action aléatoire) ou exploitation (l'action de la valeur-Q maximale) en suivant la stratégie Epsilon Greedy;
        Exécutez l'action et observez la récompense et l'état suivant;
        Actualiser la valeur-Q : $Q^{new}(s,a) = (1-\alpha)\,Q(s,a) + \alpha Q_*(s,a)$;
        Passer à l'état suivant;
        Actualiser le taux d'exploration ε;
        Sortir de la boucle si le but est atteint;

1.12 Deep Q-Network (DQN)

Une version plus complexe que le Q-Learning est la variante Deep Q-Network (que nous appelons parfois simplement Deep Q-Learning).

Cela nous permettra d'entrer dans le monde de l'apprentissage par renforcement profond (Deep Reinforcement Learning).

1.12.1 Limites du Q-Learning

Une de raison de penser au DQN est la continuité de l'espace d'état ce qui implique un nombre d'états infini.

Le processus itératif de calcul et d'actualisation des valeurs-Q pour chaque paire état-action dans un grand espace d'état devient inefficace et peut-être irréalisable en raison des ressources et du temps nécessaire.

L'avantage de DQN est que l'état de l'agent peut-être une fonction continue, et peut-être défini par plusieurs paramètres.

1.12.2 Deep Neural Network (DNN)

Plutôt que d'utiliser l'itération des valeurs pour calculer directement les valeurs-Q et trouver la fonction-Q optimale, nous utiliserons un réseau de neurones profond (Deep Neural Network DNN) pour estimer les valeurs-Q de chaque paire état-action dans un environnement donné, et à son tour, le réseau se rapprochera de la fonction-Q optimale.

Le DNN est un réseau de neurones artificiel (ANN) avec plusieurs couches cachées, il accepte comme entrée les paramètres $\{p_1, p_2, p_3, \dots\}$ de l'état s_t de l'environnement, et estiment les valeurs-Q pour chaque action qui peut être effectuée à partir de cet état $\{Q(s, a_1), Q(s, a_2), Q(s, a_3), \dots\}$, ce réseau est appelé « réseau de politique » (Policy Network).

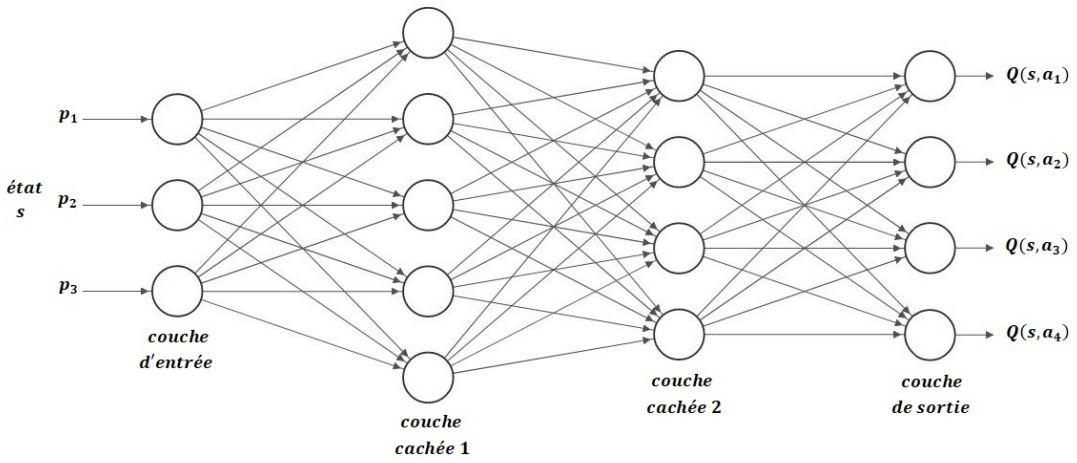


FIGURE 1.4: La structure du réseau de politique

1.12.3 Experience Replay et Replay Memory

Nous définissons une expérience comme un ensemble « état, action, récompense, état suivant » :

$$e_t = (S_t, A_t, R_{t+1}, S_{t+1}) = (s, a, R_{t+1}, s')$$

Cet ensemble contient l'état actuel de l'environnement $S_t = s$, l'action $A_t = a$ prise à partir de cet état, la récompense R_{t+1} donnée à l'agent au temps $t + 1$ en

conséquence de la paire état-action précédente (s, a) , et l'état suivant de l'environnement $S_{t+1} = s'$.

Replay Memory est un ensemble des expériences de taille N, où les dernières N expériences sont stockées, nous utilisons une partie choisie aléatoirement de cet ensemble pour l'apprentissage du réseau DNN (Policy Network). Le fait de stocker les expériences et de choisir une partie pour l'apprentissage est appelé Experience Replay.

Si l'apprentissage de réseau ne se fait qu'à partir des expériences successives telles qu'elles se sont déroulées dans l'environnement, les expériences seraient fortement corrélées et l'apprentissage serait donc inefficace. Le fait de prendre des expériences au hasard à partir de la Replay Memory résoudre ce problème de corrélation.

1.12.4 Acquisition d'expérience

Pour chaque épisode, nous initialisons l'environnement à l'état de départ, et pour chaque étape t de l'épisode, soit nous explorons l'environnement et sélectionnons une action aléatoire, soit nous exploitons l'environnement et sélectionnons l'action qui donne la valeur-Q la plus élevée (cela se fait en passant les paramètres de l'état actuel au réseau de politique et choisir l'action correspondante à $\max_i Q(s, a_i)$).

L'agent exécute alors l'action sélectionnée a et observe la récompense R_{t+1} donnée pour cette action, et nous observons également l'état suivant de l'environnement s' . Ensuite, l'expérience $e_t = (S_t, A_t, R_{t+1}, S_{t+1}) = (s, a, R_{t+1}, s')$ est stockée dans la Replay Memory.

1.12.5 Apprentissage de DNN

Après avoir enregistré des expériences dans la Replay Memory, nous prélevons ensuite une partie aléatoire, et pour chaque expérience, nous passons les paramètres de l'état au réseau de politique comme entrée.

L'entrée se propage ensuite à travers le réseau, en utilisant la même technique de propagation en avant (forward propagation) que celle dont nous avons parlé dans les ANN. Le modèle produit ensuite une valeur-Q estimée pour chaque action possible à partir de l'état d'entrée donné.

Comme le réseau calcule des valeurs-Q et essaie de se rapprocher de la valeur

optimale, la fonction de perte est définie comme l'erreur entre la valeur-Q optimale et la valeur-Q de l'action stocker dans la Replay Memory :

$$loss = Q_*(s,a) - Q(s,a)$$
$$loss = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q_*(s',a')\right] - Q(s,a)$$

Pour calculer la perte nous devons d'abord calculer la valeur-Q cible (target Q-value) : $Q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q_*(s',a')\right]$

Pour cela, nous utilisons un autre réseau de neurones appelé « réseau cible » (Target Network) pour calculer $\max_{a'} Q_*(s',a')$ en passant les paramètres de l'état suivant s' comme entrée et choisir la valeur $Q_*(s',a')$ maximale.

L'utilisation d'un autre réseau est pour une raison importante, si le même réseau de politique est utilisé à chaque étape, lorsque les poids s'actualisent, les valeurs-Q produites s'actualisent, mais les valeurs-Q cibles seront aussi actualisées puisqu'elles sont calculées en utilisant les mêmes poids. Donc, les valeurs-Q sont actualisées à chaque itération pour se rapprocher des valeurs-Q cibles, mais les valeurs-Q cibles évoluent également dans la même direction, et cela crée un problème que le réseau de politique essaye de rattraper lui-même .

Le réseau cible est un clone du réseau de politique. Ses poids sont gelés avec les poids du réseau de politique d'origine, et nous actualisons les poids dans le réseau cible par les nouveaux poids du réseau de politique après un certain nombre d'étapes.

Après avoir calculé la valeur-Q optimale pour notre paire état-action, nous pouvons calculer la perte entre la valeur-Q optimale et la valeur-Q produite par le réseau de politique pour cette paire état-action.

$$loss = \mathbb{E}[R_{t+1} + \gamma \max_{a'} \underbrace{Q_*(s',a')}_{\substack{\textit{calculer par} \\ \textit{le réseau cible}}}] - \overbrace{Q(s,a)}^{\substack{\textit{calculer par} \\ \textit{le réseau de politique}}}$$

Une méthode d'optimisation comme la descente de gradient est alors effectuée pour actualiser les poids dans le réseau afin de minimiser la perte.

Dans ce cas, minimiser la perte signifie que nous cherchons que les valeurs-Q de sortie du réseau de politique pour chaque paire état-action se rapprochent des valeurs-Q cibles données par l'équation de Bellman.

Tout ce que nous avons fait était pour une seule étape. Nous passons ensuite à l'étape suivante de l'épisode et nous répétons ce processus encore et encore jusqu'à la fin de l'épisode. À ce moment-là, nous commençons un nouvel épisode, et nous répétons ce processus jusqu'à ce que nous atteignions le nombre maximal des épisodes.

1.12.6 Algorithme Deep Q-Network

Algorithme 2 : Deep Q-Network

```
Initialisation du réseau de politique avec des poids aléatoires;
Initialisation : nombre des épisodes, nombre des étapes par épisode T , taux
  d'actualisation γ , taux d'exploration ε, la capacité N de Replay Memory;
Clonez le réseau de politique et appelez-le le réseau cible;
tant que Nombre d'épisode < nombre d'épisode maximal faire
    Commencer par l'état initial;
    tant que Nombre d'étape t < nombre d'étape maximal T faire
        Sélectionnez une action (via l'exploration ou l'exploitation) en suivant
          la stratégie Epsilon Greedy;
        Exécutez l'action et observez la récompense et l'état suivant;
        Stocker l'expérience dans la Replay Memory;
        Sélectionner une partie de la Replay Memory pour l'apprentissage;
        Passer les paramètres des états au réseau de politique;
        Calculer la perte entre les valeurs-Q de sortie et les valeurs-Q cibles
          (passer les paramètres de l'état suivant au réseau cible);
        La descente de gradient pour actualiser les poids dans le réseau de
          politique pour minimiser les pertes;
        Après un certain nombre d'étapes, les poids dans le réseau cible sont
          actualisés par les poids du réseau de politique;
        Actualiser le taux d'exploration ε;
        Sortir de la boucle si le but est atteint;
```

1.13 Conclusion

Au cours de ce chapitre, nous avons discuté la conception de base de l'apprentissage par renforcement, où l'agent interagit avec l'environnement pour le modifier et reçoit des récompenses (ou des pénalités). Ayant une compréhension de ce que sont les récompenses et le rendement, l'objectif de l'agent d'apprentissage par le renforcement est de maximiser le rendement attendu en trouvant le comportement (les actions) optimal. Ce processus est décrit comme un processus décisionnel de Markov qui peut être résolu en utilisant l'équation de Bellman sur la fonction-Q.

Par la suite, nous avons abordé deux algorithmes d'apprentissage par renforcement utilisés dans le chapitre 3, le Q-Learning qui utilise une tableau-Q pour trouver les meilleures actions dans chaque état, et le DQN qui utilise un réseau de neurones dont nous avons expliqué le fonctionnement.

Chapitre 2

Systèmes et régulation

Ce chapitre présente des généralités d'une part sur les systèmes asservis et leur comportement, et d'autre part sur les régulateurs Proportionnel-Intégral-Dérivé.

2.1 Systèmes linéaires

Afin d'étudier les caractéristiques statiques et dynamiques d'un procédé, il est nécessaire de représenter les fonctions de ses éléments constitutifs. Pour exprimer les relations entre les grandeurs incidentes et grandeurs à maîtriser, il est alors pratique d'utiliser les termes de grandeurs d'entrées et de sorties. Tout procédé étudié est alors représenté par un système comportant une ou plusieurs entrées et une ou plusieurs sorties en fonction du temps.

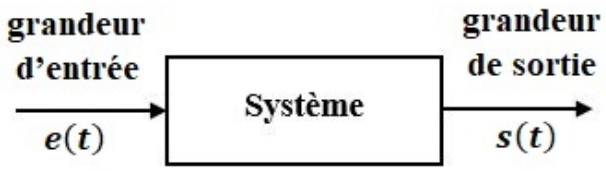


FIGURE 2.1: Système à une entrée et une sortie

Un système linéaire peut être décrit par une équation différentielle entre l'entrée $e(t)$ et la sortie $s(t)$:

$$a_n \frac{d^n s}{dt^n}(t) + \cdots + a_1 \frac{d\ s}{dt}(t) + s(t) = b_m \frac{d^m e}{dt^m}(t) + \cdots + b_1 \frac{d\ e}{dt}(t) + b_0 e(t)$$

Où les coefficients a_i et b_j sont des constants.

Le système est dit d'ordre n d'après le degré de la dérivée d'ordre le plus élevé sur $s(t)$. L'équation différentielle décrit le comportement du régime dynamique du système, mais aussi du régime permanent ou statique.

Si nous appliquons la transformation de Laplace aux deux membres de cette équation, tout en supposant nulles les différentes conditions initiales, il vient :

$$a_n p^n S(p) + \cdots + a_1 p S(p) + S(p) = b_m p^m E(p) + \cdots + b_1 p E(p) + b_0 E(p)$$

Soit : $[a_n p^n + \cdots + a_1 p + 1] S(p) = [b_m p^m + \cdots + b_1 p + b_0] E(p)$

D'où :

$$\frac{S(p)}{E(p)} = \frac{b_m p^m + \cdots + b_1 p + b_0}{a_n p^n + \cdots + a_1 p + 1}$$

Cette fraction rationnelle de deux polynômes de la variable complexe p est appelée fonction de transfert du système et communément notée :

$$G(p) = \frac{S(p)}{E(p)}$$

2.1.1 Système du premier ordre

Un système est dit du premier ordre lorsque la relation entre la grandeur d'entrée et la grandeur de sortie peut s'écrire sous la forme d'une équation différentielle du premier ordre telle que :

$$\tau \frac{ds}{dt}(t) + s(t) = G_s e(t)$$

Où τ est la constante de temps du système, et G_s est le gain statique du système.

En considérant les conditions initiales nulles, la transformée de Laplace de cette équation conduit directement à la fonction de transfert d'un système du premier ordre :

$$G(p) = \frac{S(p)}{E(p)} = \frac{G_s}{\tau p + 1}$$

2.1.2 Système du second ordre

Un système est dit du second ordre lorsque la relation entre la grandeur d'entrée et la grandeur de sortie peut s'écrire sous la forme d'une équation différentielle du

second ordre telle que :

$$\frac{1}{{\omega_0}^2}\frac{d^2s}{dt^2}(t)+\frac{2\xi}{\omega_0}\frac{d\ s}{dt}(t)+s(t)=G_s\ e(t)$$

Où ω_0 est la pulsation propre du système non amorti, G_s est le gain statique du système et ξ est le coefficient d'amortissement.

En considérant les conditions initiales nulles, la transformée de Laplace de cette équation est la fonction de transfert d'un système de second ordre :

$$G(p)=\frac{S(p)}{E(p)}=\frac{G_s}{\frac{1}{{\omega_0}^2}p^2+\frac{2\xi}{\omega_0}p+1}$$

La réponse indicielle de ce système lorsqu'un échelon d'amplitude A est appliqué à l'entrée, soit $e(t)=\ A\times u(t)$. La transformée de Laplace de l'échelon $A\times u(t)$ est $E(p)=\dfrac{A}{p}$.

$$S(p)=E(p)\,G(p)=\frac{AG_s}{p\,(1+\tau_1 p)\,(1+\tau_2 p)}$$

Le signal de sortie dépend du coefficient ξ , trois cas sont à étudier :

a) Premier cas : régime apériodique

Pour $\xi>1$, la réponse indicielle $s(t)$, déterminée par la transformées inverse de Laplace, est :

$$s(t)=AG_s\left[1-\frac{1}{\tau_1-\tau_2}\left(\tau_1\exp\left(-\frac{t}{\tau_1}\right)\ -\tau_2\exp\left(-\frac{t}{\tau_2}\right)\ \right)\right]$$

La réponse apériodique ne présente aucun dépassement par rapport à sa valeur finale (courbe 1 – figure 2.2).

b) Deuxième cas : régime critique

Pour $\xi=1$, la réponse indicielle $s(t)$ est :

$$s(t)=AG_s\left[1-\frac{\tau+t}{\tau}\exp\left(-\frac{t}{\tau}\right)\ \right]$$

C'est la réponse la plus rapide des réponses apériodiques, elle est appelée réponse à amortissement critique (courbe 2 – figure 2.2).

c) Troisième cas : régime pseudopériodique

Pour $\xi < 1$, on a :

$$s(t) = AG_s \left[1 - \frac{1}{\sqrt{1-\xi^2}} \exp(-\xi\omega_0 t) \sin\left(\omega_0 t \sqrt{1-\xi^2} + \tan^{-1}\left(\frac{\sqrt{1-\xi^2}}{\xi} \right) \right) \right]$$

Les courbes 3 , 4 et 5 de la figure 2.2 représentent les réponses indicielles pour des valeurs de ξ de plus en plus petites. Nous remarquons que plus le coefficient ξ est petit plus les amplitudes des oscillations sont grandes.

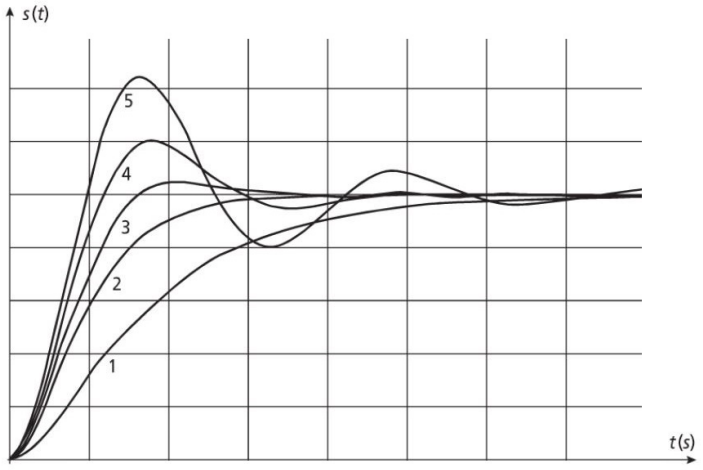


FIGURE 2.2: Réponses indicielles unitaires d'un système de deuxième ordre

2.2 Régulation en boucle fermée

Dans ce type de régulation, la correction s'effectue sur un écart ε entre la mesure et la consigne. Le rôle de la boucle fermée est d'annuler l'écart en utilisant un régulateur approprié.

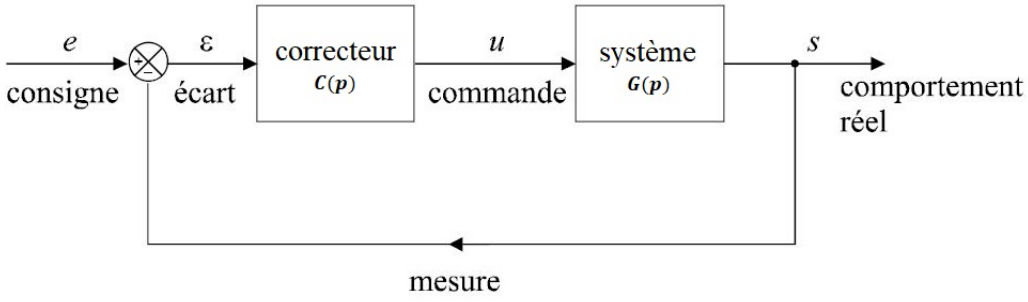


FIGURE 2.3: Une boucle de régulation

2.2.1 Fonction de transfert de la boucle de régulation

On peut écrire :

$$S(p) = G(p)\,C(p)\,\varepsilon(p)$$

Or :

$$\varepsilon(p) = E(p) - S(p)$$

D'où :

$$\begin{aligned} S(p) &= G(p)\,C(p)\,[E(p) - S(p)] \\ S(p) + G(p)\,C(p)\,S(p) &= G(p)\,C(p)\,E(p) \\ [1 + G(p)\,C(p)]\,S(p) &= G(p)\,C(p)\,E(p) \end{aligned}$$

On obtient la fonction de transfert en boucle fermée :

$$F(p) = \frac{S(p)}{E(p)} = \frac{G(p)\,C(p)}{1 + G(p)\,C(p)}$$

2.3 Les qualités d'une régulation

L'objectif d'une régulation ou d'un asservissement est d'assurer le fonctionnement d'un procédé selon des critères prédéfinis par un cahier des charges. Le cahier des charges définit des critères qualitatifs à imposer qui sont traduits par des critères quantitatifs. Les qualités exigées les plus rencontrées industriellement sont la stabilité, la précision et la rapidité :

2.3.1 Stabi1ité

La qualité essentielle et exigée à tout prix pour un système régulé est la stabilité. En effet, un système instable se caractérise soit par des oscillations d'amplitude de plus en plus grande de la grandeur observée (courbe 1 – figure 2.4), soit par une croissance irréversible négative ou positive de la grandeur observée (courbe 2 – figure 2.4). Dans les deux cas, l'objectif de la régulation n'est bien entendu pas atteint, mais surtout il y a risque de détérioration physique du procédé et donc d'insécurité.

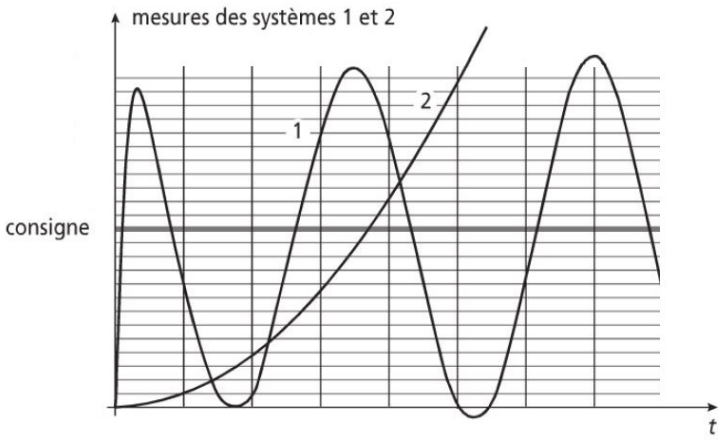


FIGURE 2.4: Deux systèmes instables

Dans une approche simplifiée, un système est considéré comme stable si, pour une variation d'amplitude finie de la consigne ou d'une perturbation, la mesure de la grandeur à maîtriser se stabilise à une valeur finie. Plus le régime transitoire d'un système soumis à une telle variation est amorti plus il est stable. Le degré de stabilité est alors caractérisé par l'amortissement de ce régime transitoire.

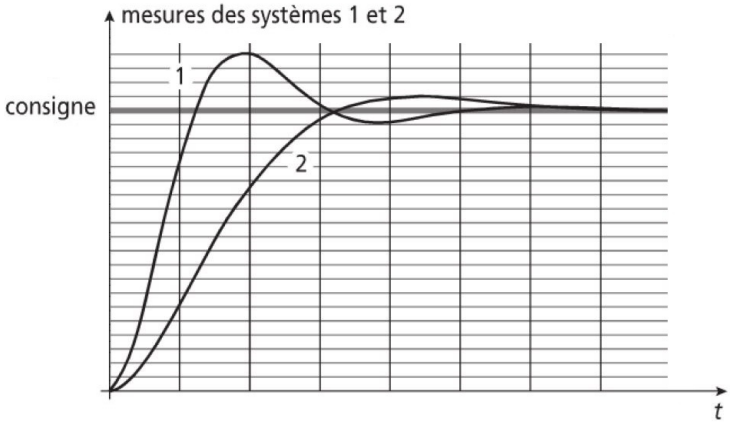


FIGURE 2.5: Deux systèmes stables

2.3.2 Précision

Il est naturel d'évaluer la précision d'un système régulé en comparant l'objectif atteint par rapport à celui exigé. La précision d'un système régulé se mesure donc à l'écart (erreur) entre la consigne demandée et la mesure en régime permanent. Nous parlons alors de précision statique. Plus l'écart statique est petit, plus le système est précis. L'évaluation de la précision statique s'effectue en réalisant une variation

rapide de consigne en amplitude et en mesurant la variation d'amplitude finalement obtenue de la mesure.

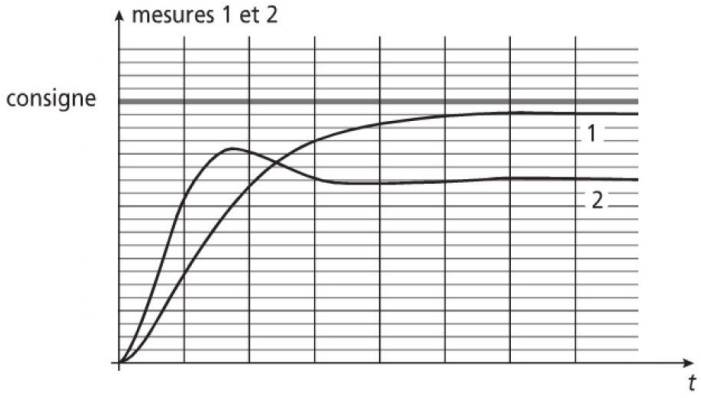


FIGURE 2.6: La précision statique

La précision dynamique est aussi importante lors la régulation des systèmes. Elle s'évaluera généralement par le dépassement maximal $D\%$ que la mesure peut prendre par rapport à la valeur finale.

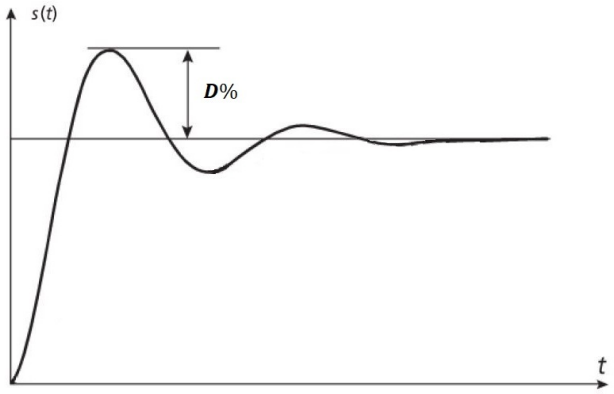


FIGURE 2.7: Le dépassement

2.3.3 Rapidité

La rapidité d'un système régulé s'évalue par le temps que met la mesure à entrer dans une zone à $\pm 5\%$ de sa variation finale (soit entre 95% et 105 %). Ce temps s'appelle le temps de réponse t_r à 5% (figure 2.8). Le système régulé est d'autant plus rapide que le temps de réponse à 5% est court.

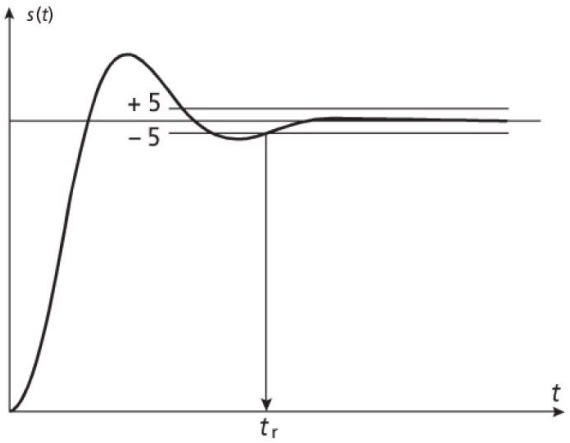


FIGURE 2.8: Temps de réponse à 5%

2.4 Actions élémentaires de régulation

2.4.1 Action proportionnelle

Fonction de transfert :

$$\frac{u\,(p)}{\varepsilon\,(p)} = K_p$$

Où K_p est un gain ou amplification sans unité.

C'est une action franche, elle permet d'agir instantanément sur le procédé lors d'un changement de consigne ou lors d'une perturbation. L'augmentation du gain améliore la précision statique et la rapidité, mais dégrade la stabilité et donc la précision dynamique (premier dépassement). La bande passante d'un procédé en chaîne ouverte est augmentée en chaîne fermée lorsque le gain du régulateur est supérieur à 1 .

2.4.2 Action intégrale

Fonction de transfert :

$$\frac{u\,(p)}{\varepsilon\,(p)} = \frac{1}{T_i p} = \frac{K_i}{p}$$

Où T_i est la constante de temps d'action intégrale. Cette constante est souvent exprimée en minutes. Et K_i est le coefficient d'action intégrale exprimé en $\min^{-1}$:

$$K_i = \frac{1}{T_i p}$$

L'action intégrale améliore la précision statique puisqu'elle agit tant que l'écart persiste et cela progressivement, mais elle dégrade la stabilité à cause de son déphasage constant de $-90°$. Associée à l'action proportionnelle elle agit essentiellement sur les fréquences basses du procédé c'est-à-dire pour celles inférieures à $\frac{10}{T_i}$.

2.4.3 Action dérivée

Fonction de transfert :

$$\frac{u(p)}{\varepsilon(p)} = T_d p = K_d p$$

Où T_d est la constante de temps d'action dérivée, souvent exprimée en secondes.

L'action dérivée n'agit pas sur la précision statique mais elle améliore la stabilité et donc la précision dynamique (premier dépassement). Associée à l'action proportionnelle elle agit sur 1es hautes fréquences du procédé c'est-à-dire pour celle supérieures à $\frac{10}{T_d}$.

L'action dérivée amplifie les bruits et les parasites des signaux, elle dégrade donc le rapport bruit sur signal. C'est pourquoi l'action dérivée peut être associée à un filtre, ou encore s'exercer seulement sur la mesure et non pas sur l'écart.

2.5 Régulateur Proportionnel-Intégral-Dérivé (PID)

Un régulateur proportionnel-intégral-dérivé (régulateur PID) est un mécanisme de boucle de régulation utilisant un feedback qui est largement utilisé dans les systèmes de contrôle industriels et dans diverses autres applications nécessitant une régulation modulée en continu. Un régulateur PID calcule en permanence l'écart ε et applique une correction basée sur les actions proportionnel, intégral et dérivé (désignés respectivement par P, I et D).

Les actions élémentaires d'un régulateur peuvent être associées de plusieurs façons, nous parlons de la structure du régulateur :

a) Structure parallèle :

$$C(p) = \frac{u(p)}{\varepsilon(p)} = K_p + \frac{K_i}{p} + K_d p$$

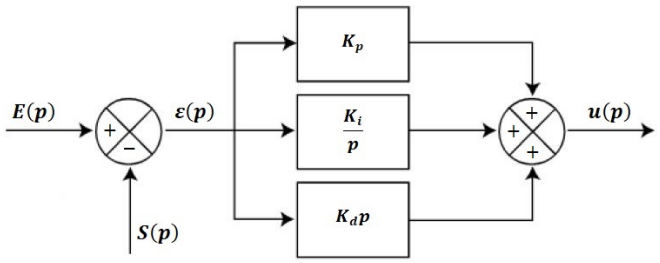


FIGURE 2.9: Correcteur PID à structure parallèle

b) Structure série :

$$C(p) = \frac{u(p)}{\varepsilon(p)} = K_p\left(1 + \frac{K_i}{p}\right)(1 + K_d p)$$

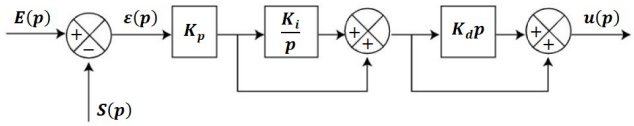


FIGURE 2.10: Correcteur PID à structure série

c) Structure mixte :

$$C(p) = \frac{u(p)}{\varepsilon(p)} = K_p\left(1 + \frac{K_i}{p} + K_d p\right)$$

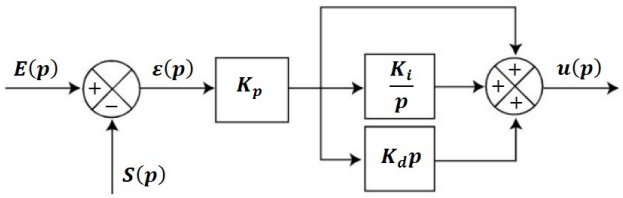


FIGURE 2.11: Correcteur PID à structure mixte

2.6 L'effet des actions du PID

Le tableau suivant montre l'effet de chaque action sur les caractéristique du système :

	Stabilité	Erreur statique ε	Dépassement $D\%$	Temps de réponse t_r
Augmenter K_p	Dégrader	Diminuer	Augmenter	Effet mineur
Augmenter K_i	Dégrader	Éliminer	Augmenter	Augmenter
Augmenter K_d	Améliorer si K_d petit	Pas d'effet	Diminuer	Diminuer

Tableau 2.1: Effet des actions élémentaires

2.7 Réglage pratique de Ziegler-Nichols (méthode du pompage)

Cette méthode est la plus connue des méthodes pratiques de réglage des boucles de régulation. L'avantage de cette méthode est qu'il n'y a pas besoin de connaître la fonction de transfert du procédé et que le réglage se fait directement sur le site en chaîne fermée. Après avoir porté la mesure près du point de consigne désiré (manuellement ou automatiquement avec des valeurs « neutres » des paramètres PID), le correcteur est réglé en action proportionnelle seule. Le gain K_p est alors augmenté progressivement jusqu'à obtention du pompage (oscillation) sans que l'organe réglant ne soit jamais en saturation. Le gain critique K_{pc} du régulateur est le plus petit gain qui permet l'entretien des oscillations. La période d'oscillation T_{osc} est mesurée sur l'enregistrement.

Les réglages des paramètres en fonction de la structure du régulateur utilisé, sont donnés dans le tableau suivant. Ces réglages conduisent à un réglage assez dur, c'est-à-dire à une réponse indicielle d'asservissement où le dépassement $D\%$ est d'environ de 30% .

	K_p	K_i	K_d
PID parallèle	$\frac{K_{pc}}{1.7}$	$\frac{K_{pc}}{0.85T_{osc}}$	$\frac{13.3}{K_{pc}T_{osc}}$
PID série	$\frac{K_{pc}}{3.3}$	$\frac{4}{T_{osc}}$	$\frac{4}{T_{osc}}$
PID mixte	$\frac{K_{pc}}{1.7}$	$\frac{2}{T_{osc}}$	$\frac{8}{T_{osc}}$

Tableau 2.2: Réglages par Ziegler-Nichols

Ces valeurs peuvent ne pas convenir au cahier des charges, le premier dépassement $D\%$ pouvant être trop important, il faut alors légèrement modifier ces réglages.

2.8 Conclusion

Dans ce chapitre, nous avons donné une idée générale sur les systèmes asservis et le contrôle PID. Les contrôleurs PID permettent une régulation optimale en combinant les avantages de chaque action : l'action proportionnelle qui réagit à l'apparition d'un écart de réglage, l'action intégrale qui élimine l'erreur statique et l'action dérivée qui s'oppose aux variations de la grandeur réglée et stabilise la boucle de régulation. Ce type de correcteur est le plus utilisé dans les milieux industriels.

Chapitre 3

Implémentation

Nous venons de présenter, dans les chapitres précédents, l'algorithme d'apprentissage par renforcement, puis des généralités sur les systèmes et méthodes de régulation par PID. Lors de ce chapitre, nous allons identifier les différents éléments de l'apprentissage par renforcement pour ajuster les paramètres d'un régulateur PID, et nous allons présenter les différentes étapes de développement et les résultats obtenus.

3.1 Ajustement des paramètres PID par apprentissage par renforcement

Le but de notre mémoire est d'ajuster automatiquement les paramètres d'un régulateur en boucle. Pour cette raison, nous avons choisi un type de régulateurs relativement simple à savoir le régulateur Proportionnel-Intégral-Dérivé (PID), avec trois paramètres à régler K_p, K_i et K_d .

Dans le chapitre précédent, nous avons présenté la méthode de régulation Ziegler-Nichols, qui est l'une des méthodes pratiques les plus utilisées, mais qui présente parfois des inconvénients lors de son application. En effet, certains systèmes peuvent devenir instables à cause de l'augmentation de gain K_p pour trouver le gain critique K_{pc} , et souvent les paramètres PID obtenus par cette méthode ne satisfont pas le cahier des charges, et nécessitent un ajustement.

Dans ce mémoire, notre approche consiste à utiliser un algorithme d'autoapprentissage pour trouver les trois paramètres du régulateur PID, à savoir

l'apprentissage par renforcement (Reinforcement Learning). Cet algorithme est souvent utilisé dans la robotique industrielle et les manipulations robotiques, la commande des systèmes complexes et les problèmes de prise de décision.

Pour implémenter cet algorithme pour l'objectif mentionné nous devons définir l'environnement et ses états, les actions possibles et leurs récompenses et la politique à suivre par notre agent.

3.2 Développement

Dans cette section, nous allons présenter les différentes phases de la réalisation de notre projet.

3.2.1 Prototype

Nous avons commencé notre développement en étudiant les algorithmes d'apprentissage par renforcement et leurs applications. Ensuite, pour avoir une idée sur le comportement d'apprentissage, nous avons essayé d'appliquer l'un des plus simples algorithmes : le Q-Learning, qui permet d'avoir un seul paramètre pour décrire l'état de l'environnement.

Pour mettre l'algorithme en œuvre, il est nécessaire de comprendre ses différents éléments présentés dans la figure ci-dessous.

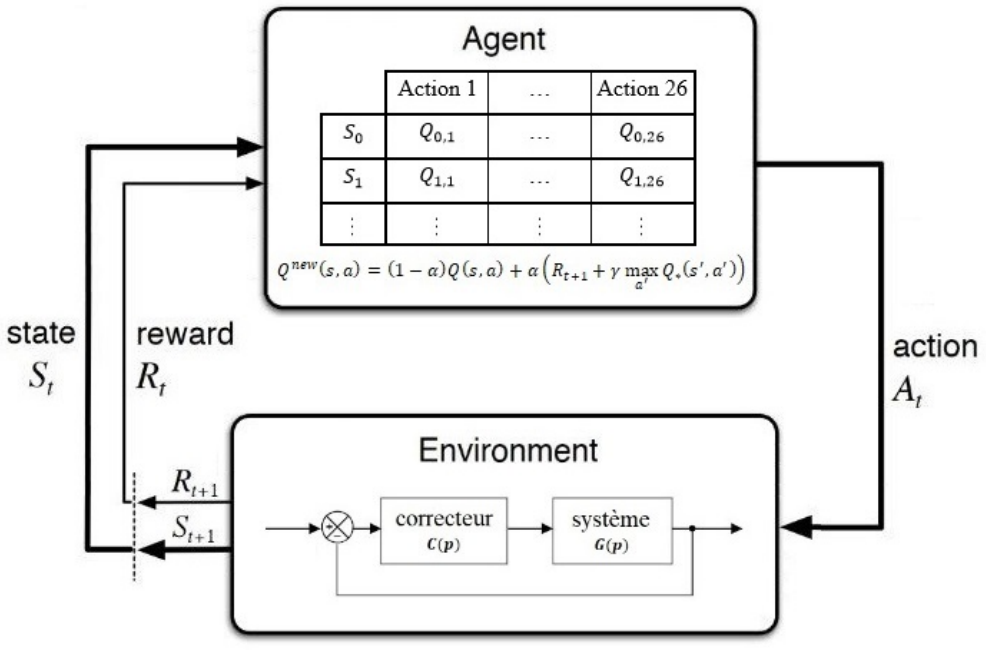


FIGURE 3.1: Les éléments du Q-Learning

L'environnement dans notre cas est la boucle de régulation d'un système $G(p)$, et à chaque étape nous passons des paramètres (K_p, K_i et K_d) au régulateur PID (correcteur $C(p)$). Cet algorithme permet de décrire l'état de l'environnement par un seul paramètre, pour lequel nous avons utilisé le dépassement de la réponse indicielle de la boucle fermée. Pour simplifier et réduire le nombre d'états (les valeurs de dépassement), nous prenons des valeurs arrondies à l'unité.

La raison derrière le choix de cet état est que le dépassement est lié aux caractéristiques de la réponse du système réglé, il représente la précision dynamique et indique la stabilité et la rapidité de notre système.

Avant la première étape $t = 0$, les paramètres de régulateur sont initialisées par des valeurs relativement petites.

Les valeurs des paramètres K_p, K_i et K_d passées au régulateur proviennent de l'action A_t de l'agent. Ils sont initialisés à l'étape $t = 0$ par des valeurs relativement petites, et l'action de l'agent consiste à modifier légèrement ces paramètres. Nous choisissons de modifier les trois paramètres simultanément, en ajoutant/soustrayant une petite valeur à chaque paramètre (une valeur de 0.1) ou en le laissant constant. Ce choix nous donne 26 actions possibles (trois modifications pour chacun des trois paramètres, sans le cas de laisser tous les paramètres inchangés : $3^3 - 1 = 26$).

Dans chaque étape, la sélection de l'action à exécuter est faite selon la stratégie Epsilon Greedy. Ensuite, l'environnement calcule le dépassement de l'étape suivante et passe cet état S_{t+1} à l'agent avec une récompense R_{t+1} .

La récompense est une fonction qui différencie les états les uns des autres. Pour ce prototype, la récompense était la fonction suivante :

$$R_{t+1} = \begin{cases} 1 & si\ S_{t+1} < S_t \\ -1 & si\ S_{t+1} = S_t \\ -2 & si\ S_{t+1} > S_t \end{cases}$$

Cette récompense signifie que nous encourageons le dépassement à être de plus en plus petit, et le but de l'apprentissage est d'arriver à des paramètres de PID qui assure un dépassement de 15% .

Dans le cas de Q-Learning l'agent dispose d'un tableau-Q, et utilise la récompense et l'état suivant pour actualiser ce tableau par l'équation $Q^{new}(s, a)$.

Dans cette étape, l'état suivant devient l'état actuel et le processus est répété en choisissant une nouvelle action suivante. Après certaines étapes, soit l'objectif est atteint ou le nombre limite d'étapes est atteint. Dans les deux cas, l'épisode est terminé et l'épisode suivant commence par l'actualisation de taux d'exploration et l'initialisation de l'environnement.

Après de nombreux épisodes d'entraînement, l'agent apprend à minimiser le dépassement à la valeur définie. Nous pouvons dire que l'objectif est atteint si les derniers épisodes convergent presque vers les mêmes valeurs (paramètres PID et nombre d'étapes).

Ce comportement que l'agent a appris ne tient pas compte des autres caractéristiques de la réponse du système, notamment l'erreur statique et le temps de réponse à 2% , qui représentent respectivement la précision et la rapidité du système.

Il existe d'autres algorithmes d'apprentissage par renforcement qui prend en compte la diversité des états et la continuité de leurs espaces de définition, celui que nous avons utilisé est le Deep Q-Network (DQN).

3.2.2 Deep Q-Network (DQN)

L'avantage de cet algorithme est qu'il permet d'utiliser plusieurs paramètres pour décrire l'état de l'environnement. Le DQN utilise un réseau neuronal appelé réseau de politique pour remplacer le tableau-Q dans le cas du Q-Learning, et utilise un autre réseau appelé réseau cible pour mettre à jour les poids et les biais du réseau de politique. Le réseau cible reprend les poids et les biais du réseau de politique après un certain nombre d'étapes.

Dans ce cas, l'environnement reste inchangé, mais son état est décrit par plusieurs paramètres de la réponse indicielle. Nous passons l'action de l'agent à l'environnement pour trouver les nouveaux états et les récompenses qui seront placés dans la Replay Memory. Cette action est toujours une modification des paramètres PID et elle est sélectionnée selon la stratégie Epsilon Greedy.

L'apprentissage du réseau de politique se fait par lot, comme expliqué dans la section « Le réseau neuronal artificiel ». L'épisode se termine en arrivant à l'objectif d'apprentissage (régulation des paramètres de l'état) ou par d'autres critères d'arrêt.

Notre objectif est de réguler un système en ajustant les paramètres suivants : le dépassement $D\%$ pour assurer la précision dynamique et la stabilité du système, le temps de réponse à 2% tr pour assurer la rapidité et l'erreur ε pour assurer la précision statique.

Les performances de l'entraînement peuvent être évaluées par les paramètres de régulateur K_p, K_i et K_d et le nombre des étapes dans chaque épisode. La convergence de ces paramètres montre que l'entraînement est efficace et bon.

Il est clair que les deux algorithmes mentionnés précédemment n'ont pas besoin de la vraie fonction de transfert du système original, mais seulement des paramètres de la réponse indicielle. Donc c'est possible de les appliquer sur un système réel sans connaitre sa fonction de transfert, ou d'entraîner l'algorithme sur un modèle proche et continuer l'entraînement sur le vrai système.

Dans la section « implémentation de l'algorithme DQN » nous allons discuter les différents choix des états, les choix des actions et de tous les autres paramètres utilisés dans l'algorithme DQN.

3.3 Environnement de travail

3.3.1 Environnement matériel

Dans la plupart des expériences, nous avons utilisé la configuration matérielle suivante :

- Système d'exploitation : Windows 10 professionnel
- CPU : Intel $i7$, 3.5 GHz
- RAM : 12 GO, 2400 MHz

3.3.2 Environnement logiciel

a) Langage de programmation et librairie utilisées :

Le développement de notre algorithme était fait par le langage de programmation Python, approprie pour les applications de Machine Learning et d'intelligence artificielle. Nous avons utilisé la version 3.6 à cause de son adaptation avec le logiciel Matlab.

Parmi les librairies que nous avons utilisées : Torch pour l'apprentissage par renforcement, NumPy pour les calculs matriciels et Matlab Engine pour faire un lien entre Python et Matlab.

b) Environnement de développement :

Nous avons utilisé l'environnement Spyder dans le logiciel Anaconda pour sa flexibilité et son support des outils d'inspection des données.

c) Matlab 2019b :

Matlab est un environnement très puissant que nous avons utilisé pour manipuler les fonctions de transfert, et faire des tests sur les différents systèmes à régler.

3.4 Implémentation de l'algorithme DQN

3.4.1 L'environnement

Comme nous avons précisé précédemment, l'environnement est la boucle de régulation d'un système de deuxième ordre précis. La régulation se fait donc sur ce système seulement, et les paramètres de régulateur sont initialisés par $K_p = 0.02$, $K_i = 0$ et $K_d = 0$.

a) Expérience :

Nous avons effectué plusieurs essais sur cet algorithme. Pour un système de deuxième ordre avec la fonction de transfert suivante :

$$sys5(p) = \frac{2}{p^2 + 2.4p + 0.111}$$

Sa réponse pour un échelon unitaire (figure 3.2) :

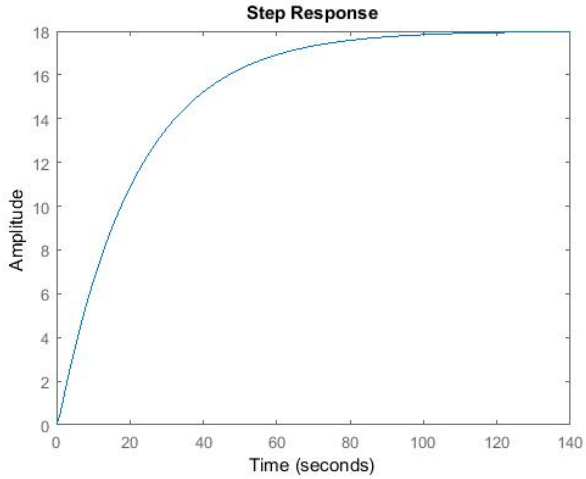


FIGURE 3.2: Réponse indicielle du système 5

Le but de la régulation est d'arriver à une réponse plus rapide (tr à 2% entre 0.2 et 4 secondes) avec un dépassement faible (D% inférieure ou égale 8%) et une erreur statique presque nulle (ε inférieur à 5% ou 0.05 pour un échelon unitaire).

En utilisant le modèle obtenu par l'entraînement sur le système 5 , nous pouvant faire la régulation de ce système en obtenant le résultat suivant :

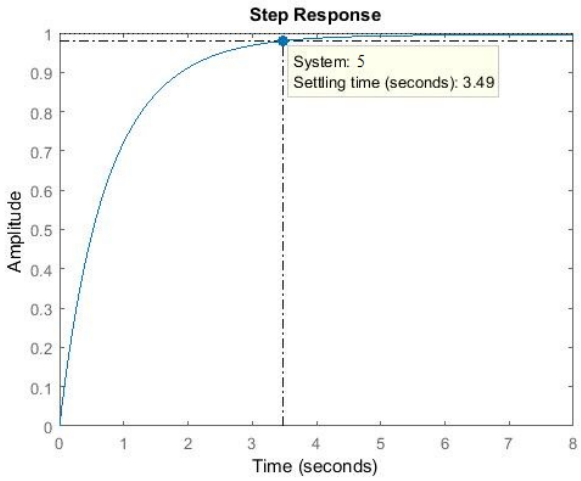


FIGURE 3.3: Réponse indicielle du système 5 après régulation

Le tableau suivant 3.1 présente les résultats de l'entraînement de ce système avec deux essais pour montrer la reproductibilité des résultats.

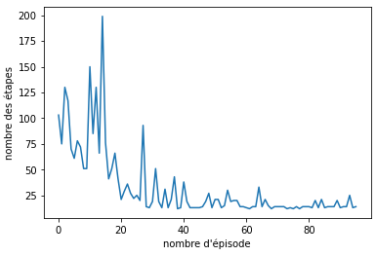
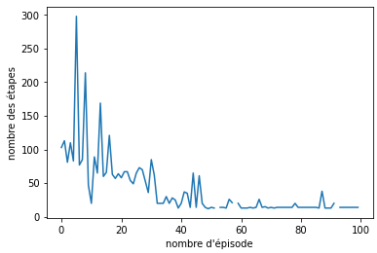
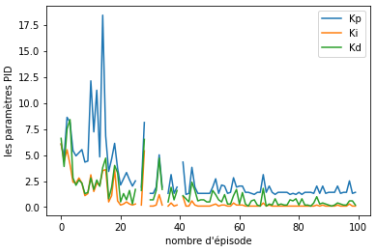
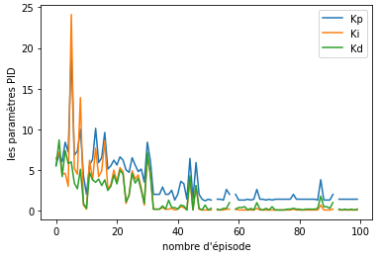
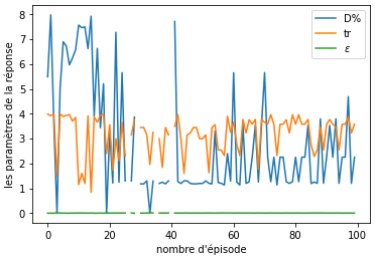
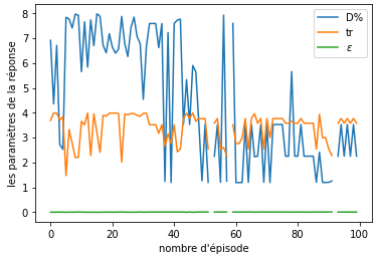
	Essai 1	Essai 2
Nombre des étapes après chaque épisode		
Les paramètres du correcteur après chaque épisode		
Les performances après chaque épisode		

Tableau 3.1: Les résultats de l'entraînement sur le système 5

b) Résultats :

- Nous pouvons constater que le nombre des étapes nécessaires dans chaque épisode diminue et converge. Ce nombre est élevé dans les premiers épisodes car l'agent cherche à obtenir des informations sur son environnement. Au fil du temps, l'agent utilise ces informations pour trouver les meilleurs paramètres PID afin d'atteindre son objectif.

- Les performances ou les paramètres de la réponse indicielle en boucle fermée sont dans les limites indiquées précédemment ($D\% \leq 8\%$, $0.2 < tr \leq 4$ et $\varepsilon \leq 0.05$) et leurs variations importantes sont causées par les petites variations des paramètres PID. Mais cela n'indique pas que la régulation est mauvaise.

- Lors des derniers épisodes de l'entraînement, les petites variations du nombre d'étapes et des paramètres du correcteur ne sont pas très gênantes car l'algorithme

adoptera le comportement du dernier épisode, ce qui signifie que ces chiffres seront constants pendant le test.

- Il existe parfois dans les graphes des discontinuités causées par un critère d'arrêt de l'épisode. Par exemple l'épisode se termine si le nombre des étapes est très élevé, ce critère prévient un mauvais entraînement et arrête l'épisode même si celui-ci peut atteindre le but souhaité.

Le tableau suivant montre que l'algorithme se comporte de la même manière lors de l'entraînement des autres systèmes.

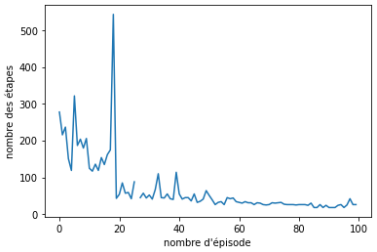
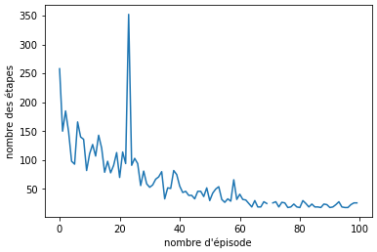
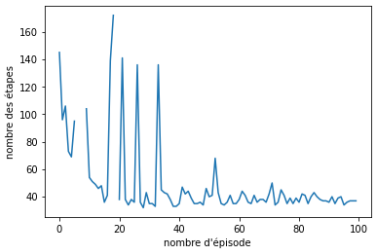
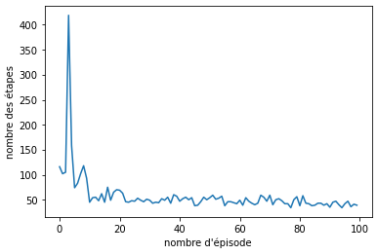
	Essai 1	Essai 2
Systèmes n° 3		
Systèmes n° 18		

Tableau 3.2: Les résultats de l'entraînement sur le système 18 et le système 3

- Le nombre des étapes converge pour les deux systèmes et dans chaque essai, ce résultat est très important et indique que cet algorithme peut effectivement atteindre l'objectif de régulation de tous les systèmes qui peuvent être contrôlés par PID.

L'inconvénient de cette méthode est que l'entraînement se répète pour chaque système à chaque fois que celui-ci est changé. Pour résoudre ce problème, nous avons développé notre algorithme afin d'entraîner sur plusieurs systèmes simultanément. Cette solution permet à l'algorithme de converger sur des systèmes hors l'ensemble de l'entraînement.

L'entraînement se fait sur plusieurs systèmes de deuxième ordre des types suivants :

- Deuxième ordre avec deux pôles et un zéro :

$$G(p) = \frac{a_1 p + a_0}{p^2 + b_1 p + b_0}$$

- Deuxième ordre avec deux pôles :

$$G(p) = \frac{a_0}{p^2 + b_1 p + b_0}$$

- Deuxième ordre avec intégrateur :

$$G(p) = \frac{a_0}{p(p + b_1)}$$

Avec $a_0 \neq 0$ et $b_0 \neq 0$.

La permutation entre les systèmes durant l'entraînement se fait aléatoirement après chaque épisode, cette permutation élimine le problème de corrélation.

3.4.2 Les états

Nous avons utilisé un script sur Matlab pour créer la boucle de régulation, et calculer la réponse indicielle et ses paramètres avec la commande « stepinfo ». Ce script reçoit les paramètres K_p, K_i et K_d du contrôleur PID, et retourne l'état de l'environnement

Au début de nos expériences, nous avons choisi de décrire l'environnement par les paramètres à contrôler : le dépassement, le temps de réponse à 2% et l'erreur statique. Cette représentation a permis de différencier les états de l'environnement mais seulement pour un seul système. Maintenant que notre algorithme utilise plusieurs systèmes pour l'entraînement, la définition de l'état doit être plus précise.

a) Expérience :

Pour augmenter le nombre des paramètres de l'état, nous pouvons ajouter un historique des trois paramètres $D\%$, tr et ε , ou utiliser les paramètres du PID.

L'entraînement était effectué sur trois systèmes en même temps, le tableau

suivant présente le nombre des étapes pour chaque système, en comparant trois expériences parmi de nombreux autres essais effectués.

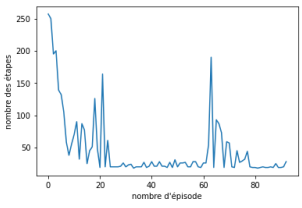
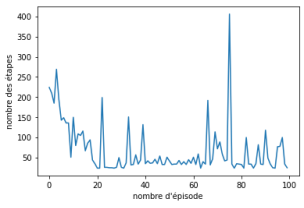
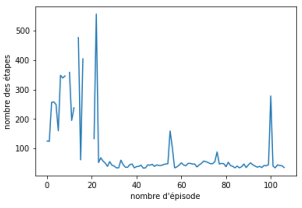
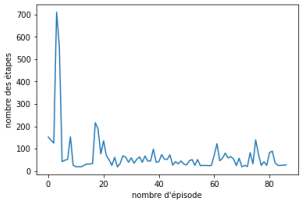
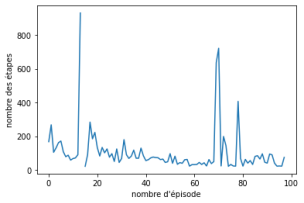
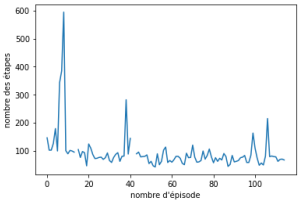
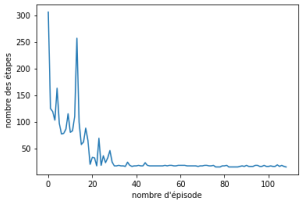
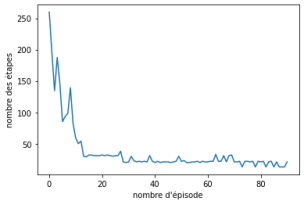
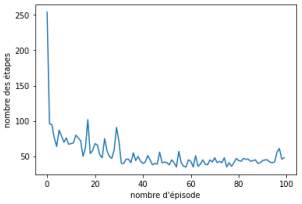
État	Système 3	Système 5	Système 18
$D\%,\ tr,\ \varepsilon$ sans historique			
$D\%,\ tr, \varepsilon$ sans historique avec $K_p,\ K_i,\ K_d$			
$D\%,\ tr, \varepsilon$ avec historique et avec $K_p,\ K_i,\ K_d$			

Tableau 3.3: L'influence de l'état sur l'entraînement

b) Résultats :

- Dans la première expérience, nous pouvons voir que le nombre des étapes présente des variations importantes dans les derniers épisodes, nous pouvons en dire autant de la deuxième expérience. Dans les deux cas, l'entraînement est inefficace et il ne converge pas pour tous les systèmes.

- Il est clair que la troisième expérience donne les meilleurs résultats, le nombre des étapes pour chaque système converge rapidement et les variations sont mineures.

- Les résultats de nos expériences montrent que la meilleure solution était d'augmenter les paramètres de l'état par les paramètres PID, et d'ajouter un historique des trois dernières valeurs de chaque paramètre $D\%$, tr et ε .

3.4.3 Les actions

L'action effectuée par l'agent est une modification des paramètres K_p, K_i et K_d . Nous avons eu le choix entre modifier les trois paramètres simultanément ou indépendamment, en utilisant une combinaison des trois modifications : augmenter le paramètre par un pas de 0.1 ($\nearrow$), diminuer par un pas de 0.1 ($\searrow$) ou sans changement ($\rightarrow$).

a) Expérience :

Le tableau suivant contient certaines expériences avec différents types d'actions, où les paramètres de régulateur sont initialisés à $K_p = 0.02$, $K_i = 0$ et $K_d = 0$, nous avons utilisé 24 systèmes pour l'entraînement, et tous les autres paramètres sont fixés.

La comparaison est faite par le nombre d'épisodes lorsque l'objectif est atteint parmi les 500 épisodes effectués.

		Convergence sur 500 épisodes			
La modification	Exemples de l'action	Essai 1	Essai 2	Essai 3	Moyenne
Simultanément ($\nearrow, \searrow, \rightarrow$) $3^3 - 1 = 26$ *actions*	$K_p \nearrow,\ K_i \searrow, K_d \rightarrow$ $K_p \rightarrow,\ K_i \nearrow, K_d \nearrow$	384	368	340	72.8%
Simultanément ($\nearrow, \rightarrow$) $2^3 - 1 = 7$ *actions*	$K_p \rightarrow,\ K_i \rightarrow, K_d \nearrow$ $K_p \rightarrow,\ K_i \nearrow, K_d \nearrow$	499	490	487	98.4%
Indépendamment ($\nearrow, \searrow$) $3 \times 2^1 = 6$ *actions*	$K_p \nearrow$ $K_d \searrow$	171	140	187	33.2%
Indépendamment ($\nearrow$) $3 \times 1^1 = 3$ *actions*	$K_p \nearrow$ $K_i \nearrow$	412	391	375	78.5%

Tableau 3.4: L'influence de l'action sur l'entraînement

b) Résultat :

- L'utilisation de la modification « diminuer par un pas de $0,1$ ($\searrow$) » donne de mauvais résultats, et lors de plusieurs expériences, les derniers épisodes ne convergent pas. La cause de ce comportement est que chaque paramètre du PID peut prendre des valeurs négatives, ce qui implique un grand nombre d'étapes pour atteindre l'objectif. L'entraînement prend beaucoup de temps et peut ne pas converger.

- Le fait de modifier les trois paramètres simultanément diminue le nombre des étapes nécessaire dans chaque épisode, et offre plus de possibilités de changer chaque paramètre.

- Les expériences montrent que le choix de sept actions est la bonne solution, où presque tous les épisodes atteignent le but et l'entraînement converge toujours.

3.4.4 La récompense

La récompense est la fonction qui guide l'entraînement en fonction des paramètres de l'état. Nous utilisons les paramètres $D\%$, tr et ε les plus récents sans historique pour déterminer la récompense.

Notre choix de cette fonction est basé sur l'objectif de l'entraînement ($D\% \leq 8\%$, $0.2 < tr \leq 4$ et $\varepsilon \leq 0.05$). La récompense est maximale si le temps de réponse est proche de 2 secondes, et elle est plus importante entre 2 et 4 qu'entre $0,2$ et 2 car nous préférons que la réponse ne soit pas trop rapide. Nous cherchons à ce que l'erreur statique et le dépassement soient nuls.

La figure suivante est une présentation en quatre dimensions de la fonction de récompense utilisé pour l'entraînement.

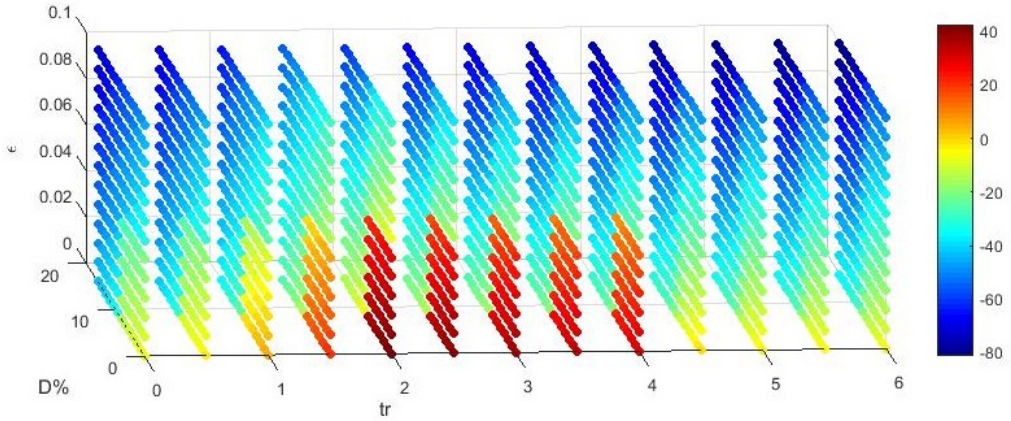


FIGURE 3.4: La récompense en fonction de $D\%$, tr et ε

3.4.5 La politique

L'agent utilise la politique pour trouver une relation entre l'état et l'action à prendre, une telle relation est caractérisée par le réseau de politique.

Nous avons utilisé un réseau de neurones profond de trois couches cachées. La couche d'entrée prendre la taille des paramètres de l'état ($D\%$, tr et ε avec trois historiques et K_p, K_i et K_d ce qui donne 12 entrées), la première couche cachée contient 24 neurones, la deuxième contient 48 neurones, la troisième contient 32 neurones et la couche de sortie contient les valeurs-Q de chaque action possible (sept sorties correspondant aux sept actions). La fonction d'activation de chaque couche cachée est la fonction ReLU.

L'ensemble « état, action, récompense, état suivant » forme une expérience qui serait stockée dans la Replay Memory, et l'apprentissage de réseau se fait par lot de taille 32 choisi aléatoirement de cette mémoire.

La fonction de perte est calculée par l'équation suivante :

$$loss = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q_*\left(s', a'\right)\right] - Q\left(s, a\right)$$

Avec un taux d'actualisation $\gamma = 0.999$.

Comme nous avons déjà expliqué, le terme $\max_{a'} Q_*\left(s', a'\right)$ est calculé par un réseau cible, ce réseau prend les valeurs des poids et des biais de réseau de politique chaque 3000 étapes.

L'actualisation des poids et des biais du réseau de politique se fait par l'algorithme d'optimisation Adam.

• L'optimisateur d'Adam est un algorithme spécialisé de descente de gradient, il utilise deux taux d'apprentissage adaptatifs.

Le gradient est calculé par la formule suivante :

$$\nabla_{W^{[k]}} = \frac{\partial cost}{\partial W^{[k]}}$$

Le premier taux d'apprentissage utilise le gradient, et le deuxième taux utilise le carré de gradient :

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{W^{[k]}}$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_{W^{[k]}})^2$$

Où β_1 et β_2 sont deux termes de décroissance, également appelés taux de décroissance exponentielle, par défaut : $\beta_1 = 0.9$ et $\beta_2 = 0.999$.

Les deux taux d'apprentissage sont corrigés comme suit :

$$\hat{m}_t = \frac{m_t}{1 - {\beta_1}^t}$$
$$\hat{v}_t = \frac{v_t}{1 - {\beta_2}^t}$$

Finalement :

$$W_t^{[k]} = W_{t-1}^{[k]} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$$

Epsilon ε est juste un petit terme qui empêche la division par zéro, généralement $\varepsilon = 10^{-8}$, et $\alpha = 0.001$.

3.4.6 Paramètres de l'apprentissage

Nous avons essayé dans chacune de nos expériences d'estimer et d'interpréter l'impact de certains paramètres sur les performances de l'algorithme proposé. Parmi les paramètres que nous avons testés, ceux qui nous ont semblé les plus importants sont le changement de taux d'exploration ε, le nombre des épisodes et le nombre des étapes par épisode.

- La stratégie Epsilon Greedy utilise un taux d'exploration ε qui change suivant la fonction exponentielle suivante :

$$\varepsilon = 0.01 + (1 - 0.01)\exp(-\acute{e}tape \times \varepsilon_{decay})$$

Où ε_{decay} est le taux de changement de ε , 1 est la valeur initiale et 0.01 est la valeur finale.

Cette stratégie permet de sélectionner l'action suivante soit par exploration (action aléatoire), soit par exploitation (action de la valeur Q maximale), et le taux ε_{decay} est super-important pour maintenir un équilibre entre les deux. Pour l'entraînement sur 24 systèmes, le taux utilisé est $\varepsilon_{decay} = 0.00026$.

- Le nombre des épisodes :

Il est important de choisir le bon nombre d'épisodes afin que l'entraînement soit bon, un petit nombre et l'entraînement n'est pas encore terminé, ou un grand nombre et nous risquons un problème de sur-apprentissage.

- Le nombre des étapes :

Durant l'entraînement sur plusieurs systèmes, chacun prend un nombre différent des étapes. Si nous ne limitons pas ce nombre, nous risquons que l'entraînement ne converge pas lorsqu'un système effectue un grand nombre d'étapes en un seul épisode.

3.5 Conclusion

Dans ce chapitre, nous avons présenté l'algorithme d'apprentissage par renforcement pour ajuster les paramètres d'un régulateur PID. L'objectif de notre mémoire était de faire la régulation sur un seul système, mais avec plus de développement, nous avons pu faire la régulation de plusieurs systèmes par un entraînement sur quelques-uns.

L'algorithme d'apprentissage par renforcement qui s'entraîne sur un seul système donne de très bons résultats. Durant les tests, nous avons constaté que même si les paramètres de la fonction de transfert changent légèrement, le modèle suit les changements et prédit des bons paramètres de régulation.

Chapitre 4

Applications et résultats

Dans ce chapitre, nous allons faire quelques expériences sur le modèle entraîné, nous avons choisi plusieurs systèmes et discuté des résultats obtenus.

4.1 Applications

A ce stade-là, l'algorithme d'apprentissage par renforcement est finalement prêt pour l'application. Par la suite nous allons monter les avantages de notre algorithme avec des tests sur plusieurs systèmes.

4.1.1 Application 1

Nous allons utiliser un modèle d'un bras robotique pour faire une régulation en boucle fermée. Le modèle utilisé est le suivant :

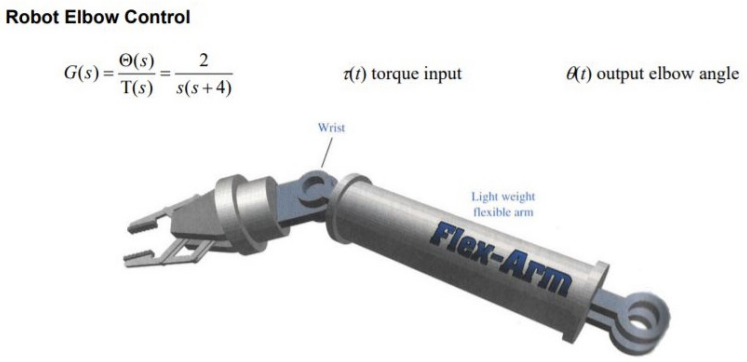


FIGURE 4.1: Le modèle d'un bras robotique

La figure suivante 4.2 est la réponse du système en boucle ouverte :

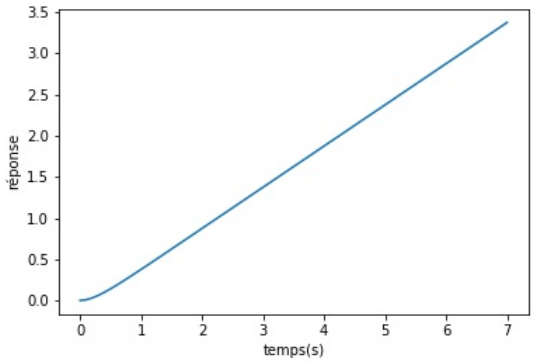


FIGURE 4.2: La réponse du bras robotique en boucle ouverte

En appliquant le modèle résultant de l'entraînement de notre algorithme, nous arrivons à la régulation suivante :

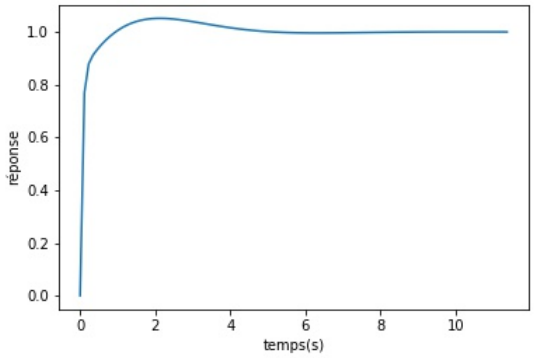


FIGURE 4.3: La réponse du bras robotique après régulation

Les paramètres de régulateur obtenus par l'algorithme sont : $K_p = 12.02$, $K_i = 8.8$ et $K_d = 8$. La figure suivante 4.4 montre la variation des performances (les paramètres $D\%$, tr et ε) durant l'application :

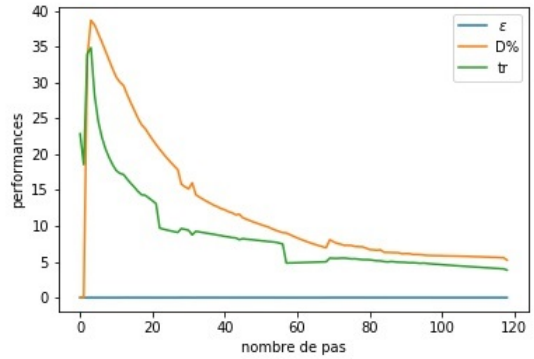


FIGURE 4.4: Les performances du bras robotique durant la régulation

Nous pouvons constater qu'après chaque étape, les performances s'améliorent, jusqu'à ce que l'objectif soit atteint.

4.1.2 Application 2

Nous allons appliquer notre modèle sur la fonction de transfert d'orientation d'une station spatiale :

$$G_1(p) = \frac{20}{p^2 + 20p + 100}$$

La réponse indicielle en boucle ouverte du système $G_1(p)$:

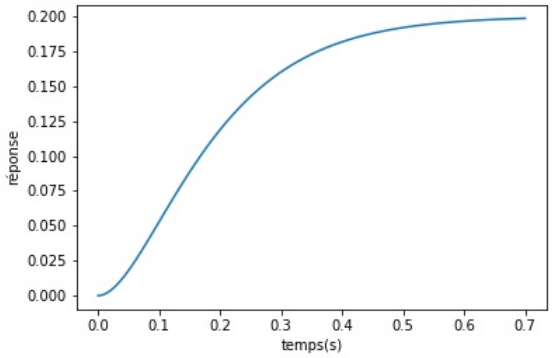


FIGURE 4.5: La réponse du système $G_1(p)$ en boucle ouverte

Après la régulation, nous obtenant la réponse suivante :

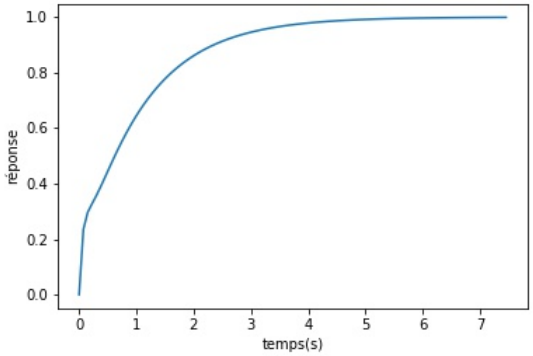


FIGURE 4.6: La réponse du système $G_1(p)$ après régulation

Et les paramètres du régulateur sont : $K_p = 1.82$, $K_i = 5.3$ et $K_d = 0.3$.

4.1.3 Application 3

Le modèle suivant représente la fonction de transfert d'un système SkyCam :

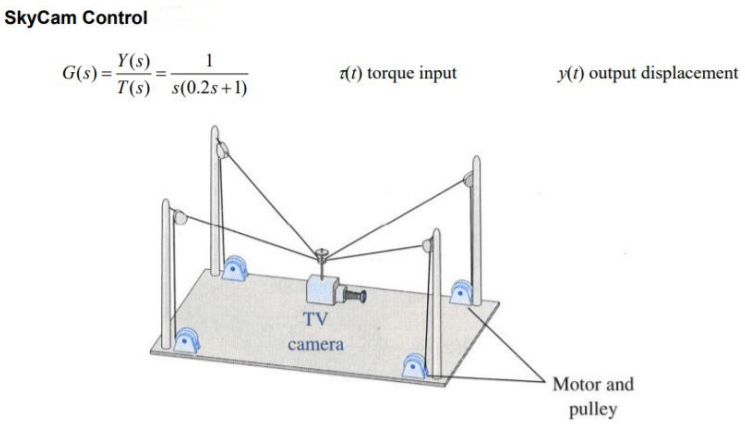


FIGURE 4.7: Le modèle d'un système SkyCam

Sa réponse indicielle en boucle ouverte :

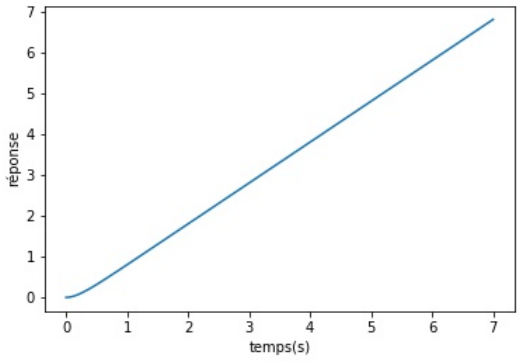


FIGURE 4.8: La réponse du système SkyCam en boucle ouverte

Après la régulation, la réponse du système devient :

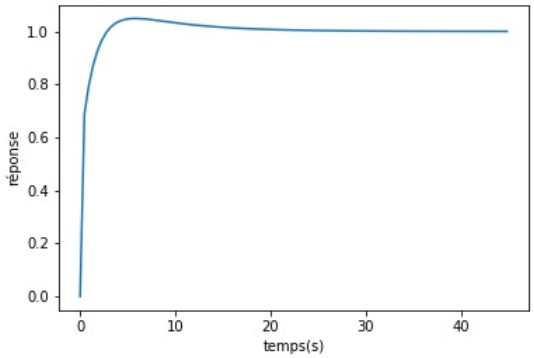


FIGURE 4.9: La réponse du système SkyCam après régulation

Les paramètres du régulateur sont : $K_p = 1.62$, $K_i = 0.2$ et $K_d = 1.2$.

4.1.4 Application 4

Dans cette application, nous allons utiliser la fonction de transfert suivante :

$$G_2(p) = \frac{p+8}{p^2+4p+0.004}$$

Sa réponse indicielle en boucle ouverte :

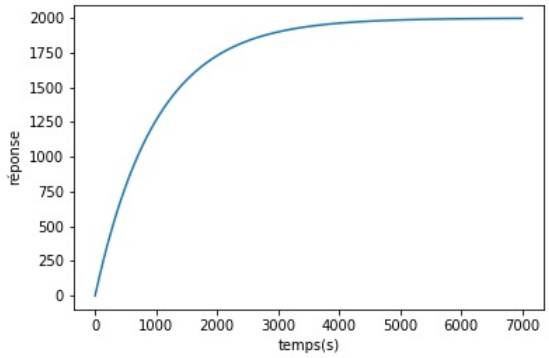


FIGURE 4.10: La réponse du système $G_2(p)$ en boucle ouverte

Après la régulation, la réponse du système devient :

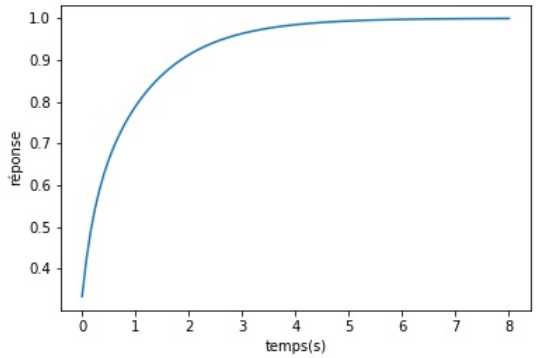


FIGURE 4.11: La réponse du système $G_2(p)$ après régulation

Les paramètres du régulateur sont : $K_p = 0.82$, $K_i = 0.0$ et $K_d = 0.5$.

4.2 Discussion des résultats

Les applications sur des modèles de plusieurs systèmes nous montrent que la régulation est bonne. Le modèle obtenu par l'entraînement permet d'ajuster les

paramètres d'un régulateur PID, et les paramètres obtenus ne se ressemblent pas, ce qui signifie que le modèle peut répondre aux différentes exigences des systèmes.

Pour avoir une régulation plus précise, il suffit de modifier le critère d'arrêt. Mais si nous souhaitons faire la régulation pour des systèmes plus rapides ou des systèmes qui nécessitent des petits paramètres PID, nous pouvons faire un nouvel entraînement en changeant par exemple la fonction de récompense, les systèmes de l'entraînement et le pas de changement des paramètres PID.

Il est important de noter que si le système ne peut pas être contrôler par un régulateur PID, notre algorithme ne donne pas des bonnes résultats.

4.3 Conclusion

Le développement de cet algorithme pour l'adapter sur plusieurs systèmes n'était pas facile, de nombreux essais étaient effectués sur ces différents éléments. Le résultat final est un algorithme qui permet de trouver les paramètres du PID pour un grand nombre des systèmes sans connaître leurs fonctions de transfert.

Conclusion générale

L'apprentissage par renforcement est un outil très performant dans les applications des prises de décision et la régulation des systèmes complexes. Nous avons appliqué avec succès l'algorithme Deep Q-Network de l'apprentissage par renforcement pour l'ajustement des paramètres d'un régulateur Proportionnel-Intégral-Dérivé.

L'algorithme développé durant ce mémoire permet la régulation de plusieurs systèmes en respectant le cahier des charges. En plus, ce n'est pas nécessaire de connaître la fonction de transfert du système à régler, mais seulement des paramètres de sa réponse indicielle. Nous pouvant aussi entraîner le modèle sur d'autre type de système avec des performances différentes.

Nous avons choisi le régulateur Proportionnel-Intégral-Dérivé pour les besoins de ce mémoire, mais il est toujours possible d'utiliser un autre type de régulateur et de faire les ajustements et les essais nécessaires sur l'algorithme.

Il existe d'autre algorithme d'apprentissage par renforcement que nous n'avons pas pu entamé. Parmi eux, Deep Deterministic Policy Gradient (DDPG) qui support la continuité de l'espace de définition des actions, un autre algorithme s'appelle Multi-Agent Reinforcement Learning qui utilise plusieurs agents dans l'environnement.

Durant le développement de n'importe quel algorithme qui utilise des réseaux de neurones, Il est important de savoir modifier ses paramètres et d'avoir des critères de comparaison entre les résultats obtenus.

La porte de la recherche sur les applications de l'intelligence artificielle est toujours ouverte à des nouveaux algorithmes et propositions. Nous pensons que les ingénieurs doivent avoir plus de connaissances dans ce domaine et doivent devenir des leaders dans ce type de recherche.

Bibliographie

[1] Arulkumaran, K., Deisenroth, M. P., Brundage, M., et Bharath, A. A. A Brief Survey of Deep Reinforcement Learning. 2017.

[2] Brownlee, J. Gentle introduction to the adam optimization algorithm for deep learning. (Consulté le 12/04/2020)
URL : https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning.

[3] Brownlee, J. Loss and loss functions for training deep learning neural networks. (Consulté le 28/03/2020)
URL : https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

[4] Bushaev, V. Adam — latest trends in deep learning optimization. (Consulté le 12/04/2020)
URL : https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

[5] Chinnamgari, S. K. *R Machine Learning Projects*. Packt Publishing, 2019.

[6] DeepAI. Machine learning. (Consulté le 28/03/2020)
URL : https://deepai.org/machine-learning-glossary-and-terms/machine-learning.

[7] DeepLizard. Neural network programming - deep learning with pytorch. (Consulté le 05/04/2020)
URL : https://deeplizard.com/learn/playlist/PLZbbT5o_s2xrfNyHZsM6ufI0iZENK9xgG.

[8] DeepLizard. Reinforcement learning - goal oriented intelligence. (Consulté le 28/03/2020)
URL : https://deeplizard.com/learn/playlist/PLZbbT5o_s2xoWNVdDudn51XM8lOuZ_Njv.

[9] Granjon, Y. *Automatique - Systèmes linéaires, non linéaires, á temps continu, á temps discret, représentation d'état Cours et exercices corrigés*. Dunod, Paris, 2010.

[10] HANSEN, C. Optimizers explained - adam, momentum and stochastic gradient descent. (Consulté le 12/04/2020)
URL : https://mlfromscratch.com/optimizers-explained.

[11] Khan, R. Nothing but numpy : Understanding and creating neural networks with computational graphs from scratch. (Consulté le 11/04/2020)
URL : https://www.kdnuggets.com/2019/08/numpy-neural-networks-computational-graphs.html.

[12] L'encyclopédie libre Wikipédia. Artificial neural network. (Consulté le 05/04/2020)
URL : https://en.wikipedia.org/wiki/Artificial_neural_network.

[13] L'encyclopédie libre Wikipédia. Machine learning. (Consulté le 28/03/2020)
URL : https://en.wikipedia.org/wiki/Machine_learning.

[14] L'encyclopédie libre Wikipédia. Reinforcement learning. (Consulté le 28/03/2020)
URL : https://en.wikipedia.org/wiki/Reinforcement_learning.

[15] MOAWAD, A. Neural networks and back-propagation explained in a simple way. (Consulté le 05/04/2020)
URL : https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e.

[16] Moni, R. Reinforcement learning algorithms — an intuitive overview. (Consulté le 28/03/2020)

URL : https://medium.com/@robertmoni_66330/reinforcement-learning-algorithms-an-intuitive-overview-of-existing-algorithms-c2095902867a.

[17] Nordlander, T. E. AI Surveying : Artificial Intelligence in Business. 2001.

[18] Ogata, K. *Modern control engineering*. Prentice-Hall, Boston, 2010.

[19] Prouvost, P. *Automatique - Contrôle et régulation Cours, exercices et problèmes corrigés*. Dunod, Paris, 2010.

[20] Sewak, M. *Deep Reinforcement Learning*. Springer Singapore, 2019.

[21] Simeone, O. A Brief Introduction to Machine Learning for Engineers. 2017.

[22] Sutton, R. S., et Barto, A. G. *Reinforcement learning : An introduction*. MIT press, 2018.

[23] Szepesvári, C. *Algorithms for reinforcement learning*. Citeseer, 2009.

[24] Zimmer, M. *Apprentissage par renforcement développemental*. Thèse de doctorat, Ecole doctorale IAEM Lorraine, 2018.

[25] Zychlinski, S. The complete reinforcement learning dictionary. (Consulté le 28/03/2020)
URL : https://towardsdatascience.com/the-complete-reinforcement-learning-dictionary-e16230b7d24e.

ملخص

التعلم التعزيزي هو نوع من تعلم الآلة يسمح للوكيل بتعلم سلوك لم يحدده الإنسان من قبل. حيث تقوم الآلة باكتشاف البيئة ونتائج أفعالها من خلال التفاعل معها.

هذه المذكرة تركز على تطبيق احدى خوارزميات التعلم التعزيزي، وهي Deep Q-Network ، من أجل التعديل الذاتي لمعاملات PID controller . هذه الخوارزمية تستعمل شبكات عصبية لتحديد سياستها، بيئتها هي control loop ، يتم تحديد حالتها حسب استجابتها وأفعالها هي تغيير معاملات PID .

هذه الخوارزمية تأخذ بعين الاعتبار المواصفات المطلوبة في دفتر الشروط، وتقوم بالتحكم دون ضرورة معرفة transfer function للنظام المراد التحكم فيه.

الكلمات المفاحية: تعديل معاملات PID ، التعلم التعزيزي، Deep Q-Network ، تعلم الآلة.

Abstract

Reinforcement learning is a kind of machine learning that allows an agent to learn a behavior that has never been previously defined by humans. The agent discovers the environment and the different consequences of his actions through interactions with it.

This thesis focuses on the application of one of the reinforcement learning algorithms, namely Deep Q-Network, to tune automatically the parameters of a Proportional-Integral-Derivative controller. This algorithm uses neural networks to define its policy, its environment is the control loop, its state is defined by the performances of the response and its actions consist in adjusting the parameters of the PID controller.

This algorithm takes into account the required specifications, and performs the control without necessarily knowing the transfer function of the system to be controlled.

Keywords: PID tuning, reinforcement learning, Deep Q-Network, machine learning.

Résumé

L'apprentissage par renforcement est un type d'apprentissage machine qui permet à un agent d'apprendre un comportement qui n'a jamais été préalablement défini par l'homme. L'agent découvre l'environnement et les différentes conséquences de ses actions à travers des interactions avec celui-ci.

Ce mémoire s'intéresse à l'application de l'un des algorithmes d'apprentissage par renforcement, à savoir Deep Q-Network, pour faire un ajustement automatique des paramètres d'un régulateur Proportionnel-Intégral-Dérivé. Cet algorithme utilise des réseaux de neurones pour définir sa politique, son environnement est la boucle de régulation, son état est défini par les performances de la réponse indicielle et ses actions consistent à modifier les paramètres du régulateur PID.

Cet algorithme prend en compte les exigences du cahier des charges, et fait la régulation sans nécessairement savoir la fonction de transfert du système à régler.

Mots-clés: régulation du PID, apprentissage par renforcement, Deep Q-Network, apprentissage machine.