

---

UNIVERSITÉ CADI AYYAD  
FACULTÉ DES SCIENCES ET TECHNIQUES MARRAKECH

*Mémoire de Projet de Fin d'Études*

---

**Deep Reinforcement Learning Appliqué aux stratégies  
d'investissements - Cas du marché des actions Marocain**

---

Présenté par : ELJAMIY Mohamed

En vue de l'obtention du diplôme d'Ingénieur d'Etat  
en Finance et Actuariat

Encadré par :

Mr DAAFI Boubker (FST) et Mr TARMOUTI ANAS (Upline Group)

Soutenu le : 22 juin 2020

Devant le jury composé de :

Mme. AKDIM Khadija	Faculté des Sciences et Techniques
Mr. DAAFI Boubker	Faculté des Sciences et Techniques
Mr. EL BOUKFAOUI Youssef	Faculté des Sciences et Techniques
Mr TARMOUTI ANAS	Upline Group
Mr. ZAHID Mehdi	Faculté des Sciences et Techniques

Année universitaire 2019/2020

# Dedicace

**À ma très chère mère Safia , À mon très cher père Lahoucine ,**

Autant de phrases ne sauraient montrer le degré d'amour et d'affection que j'éprouve pour vous ne sauraient exprimer ma gratitude et ma reconnaissance. Vous m'avez comblé avec votre tendresse et affection tout au long de mon parcours. Vous n'avez cessé de me soutenir et de m'encourager durant toutes les années de mes études, vous avez toujours été présents à mes cotés pour me consoler quand il fallait. En ce jour mémorable, pour moi ainsi que pour vous, reçoit ce travail en signe de ma vive reconnaissance et mon profond estime. Puisse le tout puissant vous donner santé, bonheur et longue vie afin que je puisse vous combler à mon tour.

**À ma très chère soeur Youssra , À mon cher frère Aziz,**

En témoignage de l'attachement, de l'amour et de l'affection que je porte pour vous. Malgré la distance, vous êtes toujours dans mon cœur. Je vous dédie ce travail avec tous mes vœux de bonheur, de santé et de réussite.

**À mes amis : Hassan, Oualid, Mohammed, Adil, Ali,**

En souvenir de notre sincère et profonde amitié et des moments agréables que nous avons passés ensemble. Veuillez trouver dans ce travail l'expression de mon respect le plus profond et mon affection la plus sincère.

**À mes collègues de la classe**

A vous ce modeste travail.

# Remerciements

En guise de reconnaissance, il me paraît opportun de commencer ce rapport par l'expression de mes sincères remerciements à toutes les personnes dont l'intervention, de près ou de loin au cours de ce projet, a favorisé son aboutissement.

Ainsi, je remercie principalement mon encadrant interne Mr.DAAFI Boubker , pour les conseils qu'ils m'a prodigué, son judicieux encadrement ainsi que son assistance pour la rédaction du rapport. Un grand merci également aux membres du jury qui ont accepté de juger mon travail.

Toutes mes expressions de reconnaissance vont envers mon encadrant Mr. TARMOUTI Anass qui n'a pas manqué de me préparer les conditions favorables au bon déroulement du projet. Je le remercie pour sa disponibilité, ses conseils et ses orientations qui m'ont aidé à réaliser ce travail. Mes remerciements vont également à Mr. CHARAFE-EDDINE Youssef et à toute l'équipe d'UCM qui ont rendu le milieu de travail plus aimable en faisant preuve d'un grand sens d'hospitalité.

Que le corps professoral et administratif de La FST MARRAKECH trouve ici mes vifs remerciements, pour tout le travail effectué durant mes études. Enfin, je souhaite que mon travail soit à la hauteur des attentes de toutes ces personnes.

# Résumé

Ce rapport a été effectué dans le cadre de mon projet de fin d'études pour l'obtention du diplôme d'ingénieur d'état en finance et actuariat de la FST Marrakech , effectué au sein du groupe Upline.

Cette mémoire propose une méthode d'application de l'apprentissage par renforcement, adaptée à la modélisation et à l'apprentissage de différents types d'interactions en situation réelle, au problème de la prise de décision sur des positions d'achat et de vente sur le marché des actions Marocain.

En effet, Ce problème est considéré comme un processus de Markov qui peut être optimisé par un algorithme basé sur l'apprentissage par renforcement (**DQL& DDQL**). Un algorithme d'apprentissage par renforcement qui ne s'appuie que sur les expériences, est adopté et une approximation des fonctions de valeurs par réseau de neurones artificiels est effectuée pour apprendre les valeurs des états à chaque moment donné.

## **Mots clés :**

- Apprentissage par renforcement ;
- Markov ;
- Fonction de valeur ;
- Réseau de neurones artificiels.

# Abstract

This report was conducted as part of my final project studies for graduation to engineering Diploma in finance and actuarial at FST Marrakerch, performed in the Upline Group.

This memory provides a method of applying reinforcement learning, adapted to the modeling and learning of different types of interactions in real situations to the problem of making decisions on buying and selling positions in Moroccan equity market.

This problem is considered as a Markov process which can be optimized by an algorithm based on reinforcement learning (**DQL & DDQL**). A reinforcement learning algorithm based only on experiments is adopted and an approximation of the value functions by artificial neural network is performed to learn the values of the states at each given moment.

## **Keywords :**

- Reinforcement learning ;
- Markov ;
- Value function ;
- Artificial neural network

# Liste des abréviations

Acronyme	Signification
UCM	Upline Capital Management
OPCVM	Organisme de placement collectif en valeurs mobilières
SICAV	Sociétés d'Investissement à Capital Variable
FCP	Fonds Communs de Placement
AMMC	Autorité marocaine du marché des capitaux
ML	Machine learning
RL	Reinforcement Learning
MDP	Markov decision process
DP	Dynamic programming
MC	Monte-Carlo
TD	Temporal Difference
DQN	Deep Q Networks
DDQN	Double Deep Q Networks

# Table des matières

Liste des abréviations	6
Introduction Générale	11
<b>I Contexte général du projet</b>	<b>12</b>
<b>1 Présentation de l'organisme d'accueil</b>	<b>13</b>
1.1 Upline Group : Présentation . . . . .	13
1.1.1 Upline Capital Management : Présentation . . . . .	14
1.1.2 Les différentes gammes d'OPCVM proposé par UCM . . . . .	15
1.1.3 Organigramme d'UCM . . . . .	16
<b>2 Cadre général du projet</b>	<b>17</b>
2.1 OPCVM : Organisme de Placement Collectif en Valeurs Mobilières . . . . .	17
2.1.1 Définitions & types . . . . .	17
2.1.2 Les Marchés sur lesquels investissent les OPCVM . . . . .	18
2.1.3 Les intervenants dans la vie d'un OPCVM . . . . .	19
2.1.4 Fonctionnement des OPCVM . . . . .	20
2.2 Marché des actions . . . . .	22
2.2.1 Définition d'une action . . . . .	22
2.2.2 Négociation et passation des ordres : Achat et vente . . . . .	23
2.2.3 La structure du marché des actions . . . . .	25
2.3 Généralités sur les techniques de ML et ses applications dans la finance . . . . .	26
2.3.1 Apprentissage supervisé . . . . .	27

2.3.2	Apprentissage non supervisé . . . . .	28
2.3.3	Apprentissage par renforcement-RL . . . . .	28
2.3.4	ML et ses exemples d'aplications en finance . . . . .	29
2.4	Les fondamentaux du Deep Learning . . . . .	30
2.4.1	Les neurones artificiels . . . . .	31
2.4.2	Les ANN's . . . . .	32
2.4.3	Les fonctions d'activation . . . . .	35
2.5	Description du projet . . . . .	38
<b>II</b>	<b>Focus sur RL : Théorie vers Application</b>	<b>39</b>
<b>3</b>	<b>Les bases de RL</b>	<b>40</b>
3.1	RL - Algorithme . . . . .	41
3.2	Les éléments de RL . . . . .	42
3.3	Les Processus décisionnels de Markov - MDP . . . . .	46
3.3.1	Retours et récompenses . . . . .	47
3.3.2	Politiques et fonctions de valeur . . . . .	48
3.4	L'équation de Bellman et l'optimalité . . . . .	51
3.5	Résolution de l'équation d'optimalité de Bellman . . . . .	53
3.5.1	Q-Learning . . . . .	53
3.5.2	Q-Learning avec Deep Neural Networks : DQN . . . . .	56
3.5.3	Q-Learning avec Deep Neural Networks : DDQN . . . . .	61
<b>4</b>	<b>Implémentation d'un framework de Reinforcement Learning sous PYTHON</b>	<b>63</b>
4.1	L'environnement . . . . .	63
4.1.1	Identification d'un état . . . . .	64
4.2	L'agent . . . . .	64
4.2.1	Actions possibles . . . . .	66
4.2.2	Reward . . . . .	66



## TABLE DES MATIÈRES

---

4.3	Exploration/Exploitation . . . . .	67
4.4	Déroulement du programme . . . . .	67
4.4.1	DQN . . . . .	70
4.4.2	DDQN . . . . .	73
	<b>Conclusion générale et perspectives</b>	<b>75</b>
	<b>A Les fonctions d'erreurs</b>	<b>77</b>
	<b>B Code source - Class Agent</b>	<b>81</b>

# Table des figures

Figure 1.1	Les différentes départements du groupe Upline . . . . .	14
Figure 1.2	Organigramme d'UCM . . . . .	16
Figure 2.1	le fonctionnement d'un OPCVM [1] . . . . .	21
Figure 2.2	Différentes types de ML . . . . .	27
Figure 2.3	Principe de RL . . . . .	29
Figure 2.4	Les caractéristiques d'un neurone . . . . .	31
Figure 2.5	Exemple d'un ANN . . . . .	33
Figure 3.1	RL - Algorithme . . . . .	41
Figure 3.2	Diagramme de $v_\pi$ . . . . .	50
Figure 3.3	Diagramme de $v_\pi$ et $q_\pi$ . . . . .	51
Figure 3.4	Différence entre Q-Learning et Deep Q-Learning . . . . .	56
Figure 3.5	Replay memory . . . . .	60
Figure 4.1	Récupuration du vecteur des états à partir de notre data . . . . .	64
Figure 4.2	Objet Agent . . . . .	65
Figure 4.3	La structure du projet . . . . .	68
Figure 4.4	Le training de notre data avec un <code>ep_count = 50</code> . . . . .	70
Figure 4.5	Les positions présent par l'agent pour la stratégie dqn . . . . .	72

# Introduction Générale

Les techniques de machine learning appliquées aux stratégies d'investissements ont été un sujet largement étudié et une variété de méthodes ont été proposées afin d'arriver à des solutions pour y remédier. Récemment, de nombreuses techniques à l'aide de l'apprentissage automatique sont utilisées, Mais ces techniques ont une limitation dans la mesure où ils sont principalement basés sur un apprentissage supervisé qui n'est pas si approprié pour des problèmes d'apprentissage avec des objectifs à long terme et des récompenses retardées.

Dans ce projet, nous proposons une méthode basée sur le RL profond pour les techniques d'investissements sur le marché des actions Marocain. La principale contribution de ce projet est de proposer un agent capable de détecter les positions optimales d'achat et de vente sur le marché des actions en tenant compte le sens de maximisation des récompenses.

Ce rapport se décompose en deux parties, La première est dédiée à une présentation plus générale du contexte général du projet sur laquelle une présentation de l'organisme d'accueil, quelques notions de bases sur les OPCVM et le marché des actions et les différents types de machine learning sont présentés. La deuxième est consacrée à une étude théorique sur les techniques de RL, commençant par les bases et finissant par les techniques de résolution, suivie par étude pratique sur le marché actions Marocain.

# Première partie

## Contexte général du projet

# Chapitre 1

## Présentation de l'organisme d'accueil

Upline Group est la banque d'affaire qui regroupe et développe l'ensemble des métiers de la banque d'investissement du Groupe Banque Populaire.

Mon présente travail s'est effectué au sein de **Upline Capital Management** (L'équipe d'Asset Management du groupe Upline) l'un des principaux acteurs nationaux de l'industrie de la gestion d'actifs au Maroc.

### 1.1 Upline Group : Présentation

Fondée en 1992, Un Véritable acteur de référence et partenaire privilégié des grandes entreprises et institutions, Upline Group, a adapté son organisation en plusieurs lignes métiers : conseil ingénierie financière, gestion d'actifs, intermédiation boursière, bourse en ligne, capital investissement et courtage en assurance.

Fort de son organisation en branches et en filiales spécialisées par activité stratégique, Upline Groupe dispose d'un large réseau relationnel de clientèle privée et institutionnelle, aussi bien au niveau national qu'international.

Les équipes d'Upline Group sont des spécialistes métiers qui disposent d'une expertise importante ainsi que de solides connaissances du marché financier les plaçant, ainsi, aux meilleurs niveaux et standards internationaux. Parce que chaque opération est unique, Upline Group

s'adapte aux besoins spécifiques de chaque entreprise et à ses caractéristiques afin de mieux répondre à ses attentes avec toujours le même souci de qualité.

Upline Group s'articule autour de plusieurs pôles d'activités que sont le **Corporate Finance**, l'**Asset Management**, l'**intermédiation boursière**, la **bourse en ligne**, le **Private Equity** et le **courtage en assurance**.

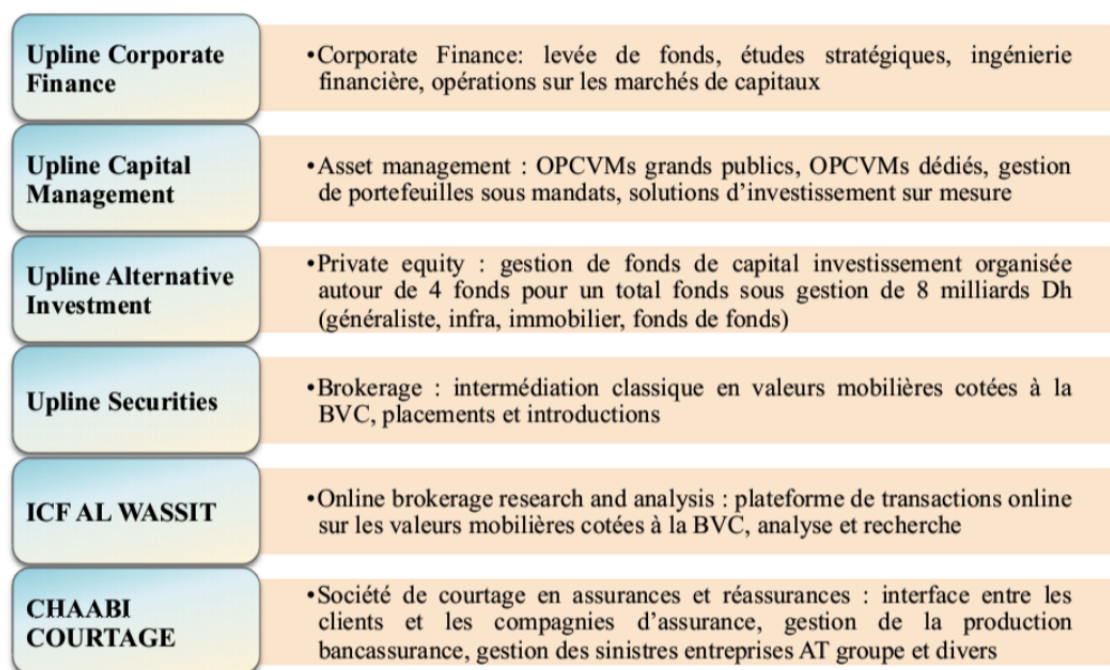


FIGURE 1.1 – Les différents départements du groupe Upline

### 1.1.1 Upline Capital Management : Présentation

Fondée en 1999, UCM est une société de gestion d'actifs, soutenue et détenue à 100 par le Groupe Banque Populaire, par le biais de la Banque d'Affaires Upline Group.

Cet adossement permet de proposer aux clients une transversalité de la prestation bancaire et des synergies à tous les niveaux ; synergies caractérisées par une approche rationnelle (maîtrise des coûts) et fiable (traçabilité et réactivité) du périmètre Client.

Le modèle multi-expert d'UCM repose sur des équipes de spécialistes de l'investissement gérant des produits intégrant une ou plusieurs classes d'actifs ainsi qu'une équipe Structuration

et Innovation qui axe ses efforts sur des solutions d'investissement nouvelles. UCM gère pour le compte de la clientèle grand public, morale ou physique, différentes gammes d'OPCVM offrant des solutions de placement différenciées selon les objectifs, les contraintes, le niveau de risque acceptable et l'horizon de placement de cette clientèle.

### 1.1.2 Les différentes gammes d'OPCVM proposé par UCM

**Monétaires** : Investis dans des instruments monétaires présentant un très faible niveau de risque. Nos fonds phare sur cette classe d'actifs couvrent l'ensemble de l'univers obligataire et offrent différents degrés de performance / risque en fonction des besoins des investisseurs.

**Obligataires Court Terme** : Offrant l'accès à tous les marchés des valeurs à taux fixe, ces OPCVM sont destinés à la rémunération des excédents stables de trésorerie sur une durée comprise en 6 à 12 mois. Les performances sont plus élevées que dans le cadre de la gestion monétaire, il existe cependant un risque de moins-value en cas de hausse des taux.

**Obligataires Moyen Long Terme** : La gestion obligataire se caractérise par des placements à moyen et long termes. L'investissement se fait sur des obligations de différentes catégories : Etat, émetteurs privés, différents types d'obligations. L'objectif est une valorisation du capital investi - à risque maîtrisé - à moyen long terme (3 à 7 ans). Il existe également sur cette classe d'actifs un risque de moins-value plus ou moins important en cas de hausse marquée des taux et de non-respect de l'horizon d'investissement recommandé.

**Diversifiés** : Une formule équilibrée de placements en actions et obligations. Les fonds diversifiés recouvrent plusieurs catégories (prudents, équilibrés, offensifs).

**Actions** : Les fonds actions sont, comme leur nom l'indique, des supports investis principalement en actions. Faisant partie des instruments financiers les plus rémunérateurs mais aussi les plus risqués, ils restent particulièrement sollicités par les investisseurs, du fait notamment des nombreuses stratégies d'investissement qu'ils peuvent suivre.

**Contractuels** : Les fonds Contractuels ont pour objectif d'atteindre, à l'expiration d'une période déterminée, un montant final exprimé en termes de performance et/ou de garantie du montant investi par le client. En contrepartie, ce dernier doit alors s'engager à respecter certaines conditions pouvant porter sur une durée fixe de placement ou encore un montant déterminé à placer.

### 1.1.3 Organigramme d'UCM

A l'instar des grandes maisons de gestion, l'organisation d'UCM se compose des métiers directement liés à la gestion financière des fonds et mandats de gestion : les Front, Middle et Back Offices avec en complément, ceux liés au Contrôle Interne et Conformité ainsi que la Commercialisation et Services Client.

Parallèlement, les fonctions Support (administration et finances, contrôle de gestion, audit interne et contrôle, communication et marketing, ressources humaines, juridique, systèmes d'information) regroupées au sein d'Upline Group accompagnent au quotidien UCM dans ces domaines, lui permettant ainsi de se focaliser entièrement sur la gestion financière des supports de placement destinées à la clientèle.

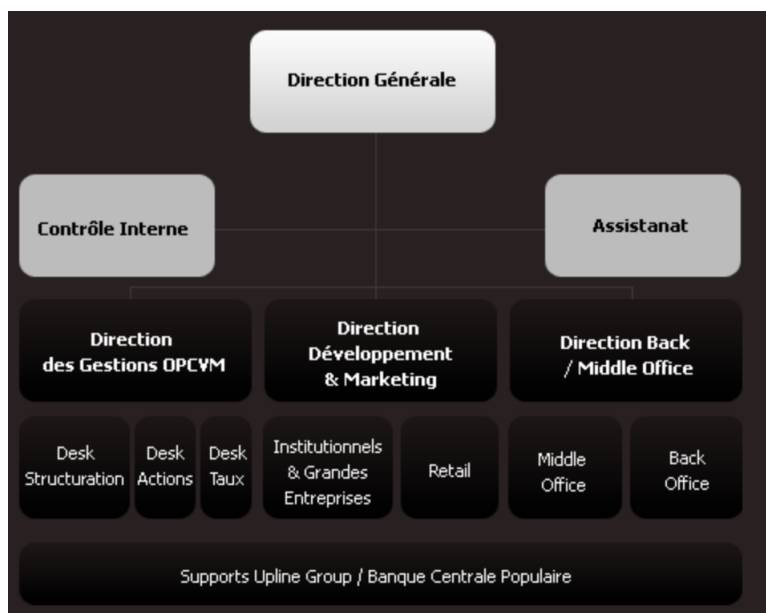


FIGURE 1.2 – Organigramme d'UCM



## Cadre général du projet

### 2.1 OPCVM : Organisme de Placement Collectif en Valeurs Mobilières

#### 2.1.1 Définitions & types

Un Organisme de Placement Collectif en Valeurs Mobilières, ou OPCVM, est un fonds d'investissement permettant à des investisseurs de toutes natures (personnes physiques, entreprises, etc.) d'avoir accès au marché des capitaux dans son ensemble (marché des actions, marché des obligations, marché monétaire, etc.).

En investissant dans un OPCVM, tout investisseur confie son épargne\* à des professionnels du marché des capitaux (les sociétés de gestion) qui vont la mettre en commun avec celle d'autres investisseurs et placer l'intégralité des sommes collectées sur le marché des capitaux en achetant des actions d'entreprises, des bons du Trésor, des obligations d'entreprises ainsi que d'autres instruments financiers disponibles au Maroc comme à l'étranger.

Les OPCVM peuvent prendre deux formes juridiques, les SICAV (Sociétés d'Investissement à Capital Variable) et les FCP (Fonds Communs de Placement) :

- **LA SICAV** : est une société anonyme à capital variable qui émet des actions en échange des sommes qui lui sont confiées et qu'elle a pour mission d'investir sur le marché des capitaux. Les investisseurs en SICAV sont donc des actionnaires de cette dernière et peuvent à ce titre s'exprimer sur la gestion de la SICAV au cours des assemblées générales. La SICAV peut assurer elle-même la gestion de ses investissements ou bien déléguer cette fonction à une société de gestion.
- **LE FCP** : une copropriété d'instruments financiers qui émet des parts (au lieu d'actions). En investissant dans un FCP, les investisseurs deviennent détenteurs de parts du FCP et membres d'une copropriété. La gestion du FCP est toujours assurée par une société de gestion.

### 2.1.2 Les Marchés sur lesquels investissent les OPCVM

Les OPCVM investissent sur l'ensemble des segments du marché des capitaux, dont les principaux sont :

- **Le marché des actions cotées** : marché sur lequel s'échangent les actions (ou titres de propriété) d'entreprises. Ce marché, organisé par une bourse des valeurs qui en assure le bon fonctionnement (au Maroc, il s'agit de la Bourse de Casablanca), permet aux OPCVM de devenir actionnaires des entreprises dont les titres y sont cotés et espérer un gain sous forme de plus-value\* (lorsque le cours des actions détenues augmente) ou de dividendes (qui correspondent à la part des bénéfices de l'entreprise versée à ses actionnaires) ;
- **Le marché obligataire** : marché permettant à l'Etat ou aux entreprises de lever des fonds par dettes. En faisant l'acquisition de ces obligations, les OPCVM escomptent un gain sous forme de plus-value (en cas d'évolution favorable du prix des obligations détenues) ou de coupons versés par les emprunteurs ;
- **Le marché monétaire** : marché de l'argent à court terme, le marché monétaire permet

aux différents intervenants d'emprunter ou d'investir leurs avoirs pour des durées courtes (voire très courtes de l'ordre d'une journée). Sur ce marché, le rendement espéré prend la forme de plus-values et d'intérêts.

### 2.1.3 Les intervenants dans la vie d'un OPCVM

Plusieurs acteurs, ayant chacun un rôle précis, interviennent pour assurer la bonne marche d'un OPCVM :

- A l'initiative de la création de l'OPCVM, sa **société de gestion** en assure la gestion administrative, comptable et financière. Composée de professionnels du marché des capitaux, elle gère l'OPCVM dans l'intérêt exclusif des investisseurs de ce dernier et se doit de disposer à tout moment des moyens humains, financiers, organisationnels et techniques nécessaires à l'exercice de son activité ;
- Le **dépositaire**, généralement un établissement bancaire, est chargé :
  - de la bonne conservation des actifs de l'OPCVM ;
  - de la gestion du passif de ce dernier (à savoir, l'exécution des ordres de souscription dans l'OPCVM ou de sortie de l'OPCVM) ;
  - le contrôle de la régularité des ordres de gestion qu'il reçoit de la société de gestion de l'OPCVM.
- Le **commissaire aux comptes (CAC)**, choisi par la société de gestion parmi les experts comptables inscrit à l'Ordre des Experts Comptables, a pour mission de contrôler la régularité et la sincérité des comptes de l'OPCVM et de procéder à la certification de ses états comptables semestriels et annuels ;
- Le **réseau de commercialisation** assure la promotion de l'OPCVM ainsi que la réception des ordres de souscription ou de rachat des parts ou actions d'OPCVM de la part des investisseurs. Souvent effectuée par la société de gestion elle-même, la commercialisation d'un OPCVM peut également être confiée à un établissement bancaire et son réseau d'agences, une compagnie d'assurances, un réseau de distribution spécialisé, etc.
- L'**AMMC** autorise la création des OPCVM en agréant leur projet de règlement de ges-

tion ou de statuts, elle permet leur commercialisation en visant leur note d'information et, enfin, elle assure le suivi et le contrôle permanents des OPCVM pendant toute leur durée de vie

### 2.1.4 Fonctionnement des OPCVM

- **Opération de souscription (Achat d'actions ou de parts d'OPCVM) :**

A la réception d'une souscription de la part d'un investisseur, l'OPCVM émet en contrepartie des actions ou des parts, selon sa nature juridique, au profit de l'investisseur.

Les sommes récoltées à l'occasion de cette souscription sont alors investies sur le marché des capitaux en fonction de la catégorie de l'OPCVM et de la stratégie de gestion que sa société de gestion met en œuvre.

- **Opération de rachat (vente d'actions ou de parts d'OPCVM) :**

Lorsqu'un investisseur déjà présent dans un OPCVM souhaite se défaire des actions ou parts d'OPCVM qu'il détient, il émet un ordre de rachat via lequel l'OPCVM va se porter acquéreur desdites actions ou parts pour les détruire et livrer en échange à l'investisseur la contre-valeur en espèces de ces actions ou parts.

Afin de pouvoir honorer la demande de rachat qu'il a reçue, l'OPCVM va céder sur le marché des capitaux la quantité nécessaire d'instruments financiers (actions, obligations, etc.).

- **Prix de souscription (achat) et de rachat (vente) :**

Les opérations d'achat ou de vente d'actions ou de parts d'OPCVM par les investisseurs sont effectuées à un prix inconnu avant leur exécution.

Ce prix, appelé « Valeur Liquidative » (ou VL), est la valeur d'une action ou d'une part d'OPCVM. Il est obtenu en divisant le montant total des placements de l'OPCVM (valorisés à leur valeur de marché) diminué des dettes de l'OPCVM, par le nombre de parts ou d'actions émises par l'OPCVM. Il est calculé par la société de gestion à une fréquence au moins hebdomadaire (chaque vendredi).

Ce prix peut, à l'achat, être majoré de frais d'entrée (commissions de souscription) et, à la vente, minoré de frais de sortie (commissions de rachat).

La valeur liquidative, ainsi que le prix de souscription et de rachat, sont affichés dans les locaux de la société de gestion et du réseau de commercialisation. Ils font également l'objet d'une publication, au moins une fois par semaine, dans un journal d'annonces légales.

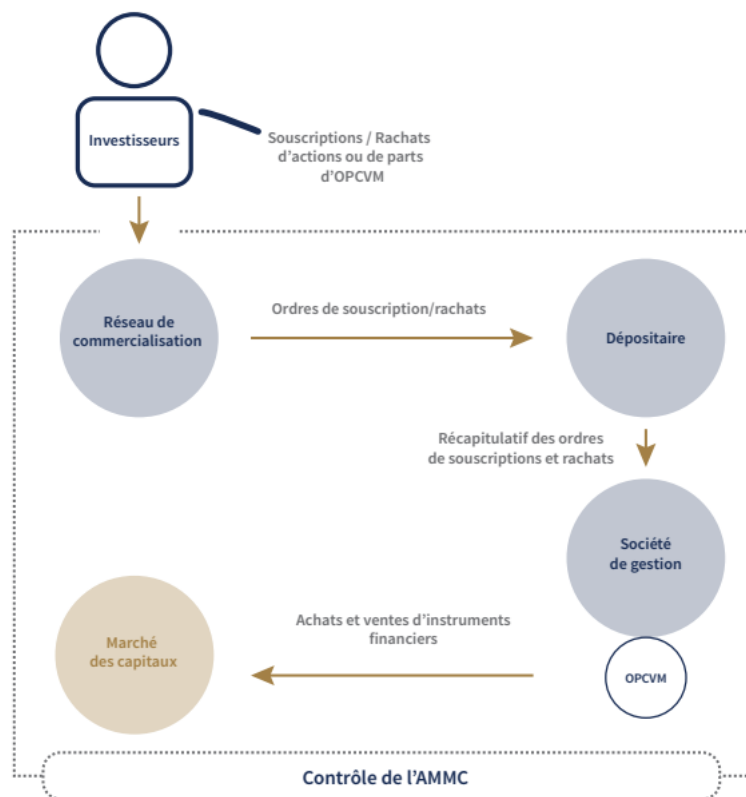


FIGURE 2.1 – le fonctionnement d'un OPCVM [1]

## 2.2 Marché des actions

### 2.2.1 Définition d'une action

Une Action est un titre de propriété qui correspond à une part du capital d'une entreprise. Son détenteur, appelé **actionnaire**, devient propriétaire d'une petite partie de l'entreprise. L'action donne à son détenteur plusieurs droits en contrepartie du capital investi dans la société. Ces droits sont divers :

- **Le droit au dividende** : chaque personne qui dispose d'une action bénéficie d'un dividende. Le montant des dividendes est voté par l'assemblée générale. Les dividendes ne sont qu'une partie du bénéfice net de l'entreprise qui est distribué aux actionnaires.
- **Le droit à l'information** : Tout actionnaire a droit à l'information sur les résultats et la vie de l'entreprise. Il ne doit en aucun cas se faire refuser la possibilité de consulter les documents financiers de la société et toute autre information jugée non confidentielle.
- **Le droit de vote** : Vous pouvez voter en assemblée générale concernant les décisions prises par l'entreprise. Le droit de vote est proportionnel au nombre d'actions que vous détenez.
- **Le droit à une partie de l'actif net de l'entreprise** : Si l'entreprise dans laquelle vous avez investi en actions se trouve en faillite, vous avez droit à une partie de son patrimoine après que ses dettes soient honorées, et ce selon en fonction du nombre d'actions que vous détenez.

Les actions sont émises, échangées et négociées sur un marché organisé, la Bourse de Casablanca. La Bourse constitue un marché d'offre et de demande de titres. Les actions peuvent être cotées en bourse si l'entreprise souhaite se financer par appel public à l'épargne. Les entreprises introduisent leurs actions en bourse pour lever des fonds destinés au financement de leur croissance et projets de développement.

### 2.2.2 Négociation et passation des ordres : Achat et vente

Un ordre de bourse désigne l'instruction donnée par le client à son dépositaire ou à son intermédiaire d'acheter ou de vendre pour lui des valeurs mobilières. Pour son exécution, l'ordre doit comporter une série de mentions obligatoires. L'ordre de bourse doit être matérialisé par une fiche dûment remplie et signée par le client.

Les ordres de bourse peuvent être transmis par tous moyens à la convenance du client et de la société de bourse ou du collecteur d'ordres, notamment par lettre, téléphone ou télécopie.

On cite ci-dessous les différents types des ordres :

#### **A cours limité :**

L'ordre à cours limité est libellé avec un prix fixé. Le prix correspond au maximum lorsqu'il s'agit d'un ordre d'achat, et au minimum s'agissant d'un ordre de vente. Les ordres à cours limité ne sont pas exécutés tant que les cours du marché ne sont pas inférieurs ou égal à leur limite d'achat ou supérieur ou égal à leur limite pour la vente. Ce type d'ordres permet de maîtriser le prix d'exécution.

#### **A la meilleure limite (au prix du marché) :**

L'ordre à la meilleure limite est un ordre de bourse formulé sans indication de prix. Il sera exécuté en fonction des conditions de marché. Si l'ordre est introduit avant l'ouverture de la valeur, il sera servi au cours d'ouverture du jour, si la priorité de l'ordre et les quantités disponibles le permettent. Si l'ordre est introduit pendant la séance de cotation en continu, il sera exécuté au cours de la meilleure limite (et uniquement à cette limite) de l'ordre en sens inverse présent sur la feuille de marché au moment de son introduction.

#### **Ordre à tout prix (au marché) :**

C'est un ordre qui n'est assorti d'aucune indication de prix et destiné à être exécuté à tout prix en fonction de sa quantité et des ordres présents sur la feuille de marché. L'ordre au marché est prioritaire sur tous les autres types d'ordres.

Son exécution dépend du mode de cotation de la valeur concernée et de la phase de cotation

pendant laquelle il est introduit. Si l'ordre est introduit avant l'ouverture de la valeur il sera exécuté au cours d'ouverture, prioritairement à tous les autres types d'ordres, en fonction des quantités disponibles.

Si l'ordre au marché est introduit pendant la séance de cotation en continu (valeurs cotées en continu uniquement) il sera exécuté face à autant de limites de cours présents sur la feuille de marché que sa quantité l'exige et, le cas échéant, s'affiche pour la quantité non exécutée sans limite de cours.

### **Les ordres à seuil de déclenchement :**

C'est un ordre qui Permet de définir un niveau de prix à partir duquel vous déclenchez un achat ou une vente.

### **Les ordres à plage de déclenchement :**

Fonctionne comme les ordres à seuil de déclenchement, sauf que l'opération d'achat ou de vente n'est déclenchée que dans la fourchette de cours indiquée.

Les ordres de bourse sont traités par un système de cotation électronique qui assure la confrontation des ordres selon des règles de priorité et des algorithmes de fixation de cours spécifiques. Quelques principes à retenir :

- **Modes de cotation :**

Toutes les valeurs de la cote ne sont pas cotées de la même manière.

Selon la liquidité historique de chaque valeur, la Bourse de Casablanca définit le mode de cotation de la valeur. Trois modes de cotation sont possibles : le fixing, et le continu.

- **Règles de cotation :**

Selon leur liquidité, les titres sont cotés en continu de 10h00 à 15h30 ou au fixing. La cotation en continu s'applique aux valeurs les plus liquides. La cotation au fixing est réservée aux valeurs de liquidité faible ou moyenne. Deux phases importantes sont à retenir :



- **La préouverture :** A partir de 09h00 les ordres qui arrivent dans le système de cotation de la bourse sont centralisés sans une feuille de marché sans qu'aucune transaction n'ait lieu. Pendant cette phase il est possible aussi de modifier ou d'annuler un ordre existant sur la feuille de marché.
- **L'ouverture :** A l'heure précise fixée par la bourse pour chaque groupe de cotation, il est procédé à une confrontation générale des ordres pour fixer un cours d'exécution. Ce cours résulte d'un algorithme qui, sur la base des ordres présents, détermine le cours qui permet de satisfaire le maximum de quantités.

Ces deux phases sont un passage obligé pour toutes les valeurs de la cote. Après l'ouverture, le traitement réservé aux valeurs dépend de leur mode de cotation.

Pour les valeurs du fixing le fixage de cours à l'ouverture est effectué une fois (fixing), ou renouvelé (plusieurs fixings par jour). Pour les valeurs du continu, après l'ouverture, une phase de cotation en continu permet l'exécution instantanée dès que deux ordres en sens inverse sont compatibles.

### 2.2.3 La structure du marché des actions

La Bourse de Casablanca est un marché dirigé par les ordres par opposition aux marchés dirigés par les prix. Ces ordres passent soit sur le Marché Central, soit sur celui de Blocs.

La Bourse de Casablanca est structurée autour de deux marchés, le marché central et le marché de blocs.

- **Le Marché Central :**

Le marché central joue un rôle pilote dans le fonctionnement du marché boursier dans la mesure où le prix des actions est déterminé sur le marché central suite à la confrontation des ordres d'achat et de vente. En effet, l'ensemble des ordres de bourse sont centralisés dans un carnet d'ordres unique qui permet de confronter l'offre et la demande et d'établir un prix d'équilibre.

### • Le Marché de blocs :

Les ordres de taille importante émanant souvent des investisseurs institutionnels et qui risquent d'avoir un impact significatif sur le cours transitent par le marché de blocs.

Le marché de blocs est un marché de gré à gré, où sont négociés des blocs de titres dans les conditions de cours issues du marché central. Les opérations sur le marché de blocs doivent respecter, sauf cas très particuliers, les conditions suivantes :

- porter sur un nombre de titres au moins égal à la Taille Minimum de Blocs(TMB), définie par la Bourse de Casablanca ;
- Etre conclues à un cours inclus dans la fourchette des prix issue de la feuille du marché central.

## 2.3 Généralités sur les techniques de ML et ses applications dans la finance

Le machine learning dit apprentissage automatique est un phénomène démontrant comment des algorithmes peuvent “apprendre” en étudiant des exemples.

L'idée est alors de collecter un maximum de données, de les analyser et de trouver un lien permettant de créer une règle et donc de réaliser des prédictions.

Il existe différentes manières de fonctionner, et nous allons maintenant examiner trois types d'apprentissage différents : **l'apprentissage supervisé, non supervisé et le renforcement.**

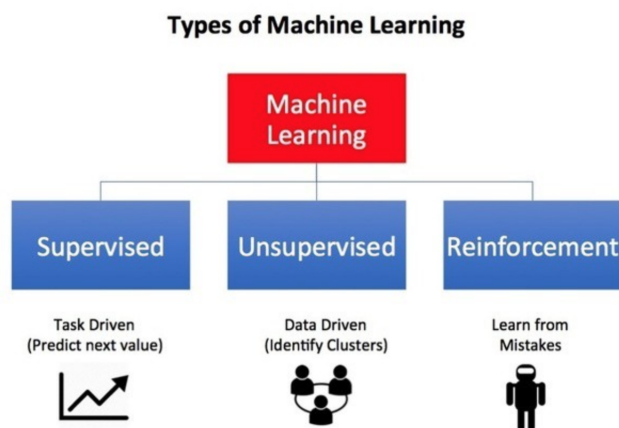


FIGURE 2.2 – Différentes types de ML

### 2.3.1 Apprentissage supervisé

Nous pouvons penser à l'apprentissage supervisé avec le concept d'approximation de fonction, où fondamentalement nous formons un algorithme et à la fin du processus nous choisissons la fonction qui décrit le mieux les données d'entrée, celle qui pour un  $X$  donné fait la meilleure estimation de  $Y$ .

nous alimentons l'ordinateur avec des données de formation contenant les **entrées / prédicteurs** et nous lui montrons les bonnes réponses (sorties) et à partir des données, l'ordinateur devrait être en mesure d'apprendre les modèles.

Les algorithmes d'apprentissage supervisé tentent de modéliser les relations et les dépendances entre la sortie de prédiction cible et les entités en entrée de sorte que nous pouvons prédire les valeurs de sortie pour les nouvelles données en fonction des relations qu'il a apprises des ensembles de données précédents.

Les principaux types de problèmes d'apprentissage supervisé comprennent les problèmes de **régression et de classification**.

### 2.3.2 Apprentissage non supervisé

Apprentissage non supervisé est la famille des algorithmes d'apprentissage automatique qui sont principalement utilisés dans la détection de modèles et la modélisation descriptive . Cependant, il n'y a pas ici de catégories ou d'étiquettes de sortie sur la base desquelles l'algorithme peut essayer de modéliser les relations.

Ces algorithmes essaient d'utiliser des techniques sur les données d'entrée pour extraire des règles , détecter des modèles et résumer et regrouper les points de données qui aident à obtenir des informations significatives et à mieux décrire les données aux utilisateurs.

En revanche, il n'y a aucun enseignant du tout, en fait l'ordinateur peut être en mesure de vous enseigner de nouvelles choses après avoir appris des modèles dans les données, ces algorithmes sont particulièrement utiles dans les cas où l'expert humain ne sait pas quoi rechercher dans les données.

Les principaux types d'algorithmes d'apprentissage non supervisés comprennent **les algorithmes de clustering et les algorithmes d'apprentissage des règles d'association.**

### 2.3.3 Apprentissage par renforcement-RL

La méthode vise à utiliser les observations recueillies lors de l'interaction avec l'environnement pour prendre des mesures qui maximiseraient la récompense ou minimiseraient le risque. L'algorithme d'apprentissage par renforcement (appelé l'agent) apprend continuellement de l'environnement de manière itérative. Dans le processus, l'agent apprend de ses expériences de l'environnement jusqu'à ce qu'il explore la gamme complète des états possibles.

RL est un type d'apprentissage machine , et ainsi également une branche de l'intelligence artificielle . Il permet aux machines et agents logiciels de déterminer automatiquement le comportement idéal dans un contexte spécifique, afin de maximiser ses performances. Un simple

retour de récompense est nécessaire pour que l'agent apprenne son comportement ; c'est ce qu'on appelle le signal de renforcement.

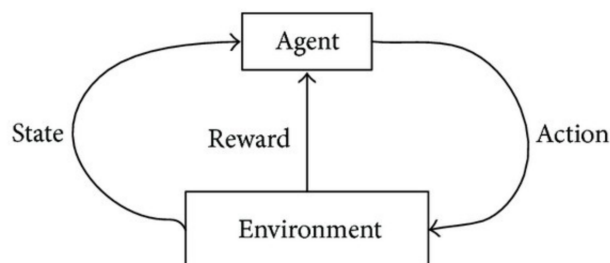


FIGURE 2.3 – Principe de RL

Il existe de nombreux algorithmes différents pour résoudre ce problème. En fait, l'apprentissage par renforcement est défini par un type spécifique de problème, et toutes ses solutions sont classées comme des algorithmes d'apprentissage par renforcement. Dans le problème, un agent est censé décider de la meilleure action à sélectionner en fonction de son état actuel. Lorsque cette étape est répétée, le problème est appelé **processus de décision de Markov**.

### 2.3.4 ML et ses exemples d'applications en finance

Il est possible d'utiliser différents critères pour classer les types d'algorithmes d'apprentissage automatique, mais je pense que l'utilisation de la tâche d'apprentissage est excellente pour visualiser la vue d'ensemble du ML et je crois qu'en fonction de votre problème et des données que vous avez en main, vous pouvez facilement décider si vous utiliserez un apprentissage supervisé, non supervisé ou par renforcement.

Je donnerai plus d'exemples sur chaque type d'algorithmes d'apprentissage automatique comme montré ci-dessous :

	Types	Applications	Exemples d'applications
<b>Perceptions tasks</b>	Supervised Learning	Régression	<ul style="list-style-type: none"> <li>• La prévision des gains ;</li> <li>• La prévision des pertes de crédit ;</li> <li>• Trading algorithmique ...</li> </ul>
		Classification	<ul style="list-style-type: none"> <li>• La prévision des notations ;</li> <li>• La modélisation de défaut ;</li> <li>• Les fraude de carte de crédit ;</li> <li>• La lutte contre le blanchiment d'argent...</li> </ul>
	Unsupervised Learning	Clustering	<ul style="list-style-type: none"> <li>• La segmentation de la clientèle ;</li> <li>• La segmentation de stock...</li> </ul>
		Representation learning	<ul style="list-style-type: none"> <li>• La modélisation des facteurs de bruits ;</li> <li>• Détection de changement de régime ...</li> </ul>
<b>Actions tasks</b>	Reinforcement learning	Optimization of strategy for a task	<ul style="list-style-type: none"> <li>• Les stratégies de trading ;</li> <li>• Asset management...</li> </ul>
		IRL: Learn objectives from behavior	<ul style="list-style-type: none"> <li>• Reverse engineering du comportement des consommateurs ;</li> <li>• Les stratégies de trading ...</li> </ul>

## 2.4 Les fondamentaux du Deep Learning

Dans les prochains chapitres, nous en apprendrons davantage sur Deep reinforcement learning (DRL), qui est une combinaison entre Deep Learning et de RL. DRL a créé beaucoup de discussions autour de la communauté RL et a un impact sérieux sur la résolution de nombreuses tâches RL [4].

Pour comprendre DRL, nous devons avoir une base solide dans Deep Learning. DL est en fait un sous-ensemble de machine learning et il s'agit des réseaux de neurones. DL existe depuis une décennie, mais la raison pour laquelle il est si populaire en ce moment est en raison des avancées informatiques et de la disponibilité d'un énorme volume de données. Avec cet énorme volume de données, les algorithmes de Deep learning surpasseront tous les algorithmes classiques de machine learning.

### 2.4.1 Les neurones artificiels

Avant de comprendre l'ANN, commençons par comprendre ce que sont les neurones et comment fonctionnent réellement les neurones de notre cerveau. Un neurone peut être défini comme l'unité de calcul de base du cerveau humain. Notre cerveau contient environ 100 milliards de neurones. Chaque neurone est relié par des synapses.

Les neurones reçoivent des informations de l'environnement externe, des organes sensoriels ou des autres neurones par le biais d'une structure en forme de branche appelée dendrites, comme le montre le schéma suivant. Ces apports sont renforcés ou affaiblis, c'est-à-dire qu'ils sont pondérés en fonction de leur importance, puis ils sont additionnés dans le soma (corps cellulaire). Ensuite, à partir du corps cellulaire, ces entrées additionnées sont traitées et se déplacent à travers les axones et sont envoyés vers les autres neurones. Le système de base Le neurone biologique est représenté dans le diagramme suivant :

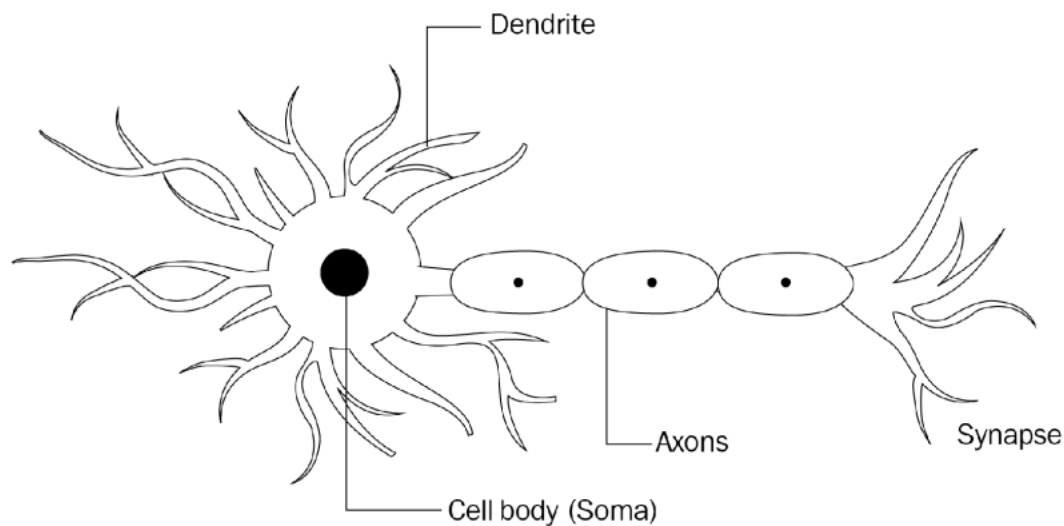
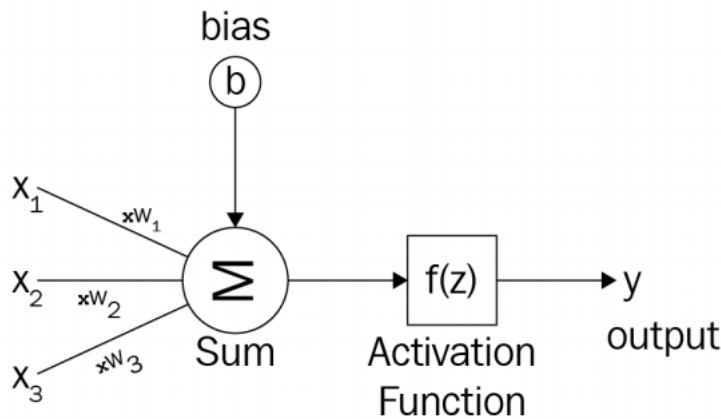


FIGURE 2.4 – Les caractéristiques d'un neurone

Maintenant, **comment fonctionnent les neurones artificiels ?**. Supposons que nous ayons trois entrées,  $x_1$ ,  $x_2$  et  $x_3$ , pour prédire la sortie  $y$ . Ces entrées sont multipliées par des poids,  $w_1$ ,  $w_2$  et  $w_3$ , et sont additionnées, c'est-à-dire  $x_1.w_1 + x_2.w_2 + x_3.w_3$ . **Mais pourquoi multiplions-nous ces entrées par des poids ?** Parce que toutes les entrées n'ont pas la même importance dans le calcul de la sortie  $y$ . Disons que  $x_2$  est plus important dans le calcul de la sortie par rapport à deux autres entrées. Ensuite, nous attribuons une valeur élevée à  $w_2$  plutôt que pour les deux autres poids.

Après avoir multiplié les entrées par les poids, on les additionne et on ajoute une valeur appelée biais  $b$ . Donc,  $z = (x_1.w_1 + x_2.w_2 + x_3.w_3) + b$ , c'est-à-dire :

$$z = \sum(inputs.poids) + biais$$



Dans les neurones, nous prenons l'entrée  $x$ , multiplions l'entrée par les poids  $w$ , et ajoutons le biais  $b$  avant d'appliquer la fonction d'activation  $f(z)$  à ce résultat et de prédire la sortie  $y$ .

### 2.4.2 Les ANN's

Un seul neurone isolé ne peut pas accomplir des tâches complexes, c'est pourquoi notre cerveau compte des milliards de neurones, organisés en couches, formant un réseau. De même, les



neurones artificiels sont organisés en couches. Chaque couche est connectée de telle manière que l'information passe d'une couche à l'autre. Un ANN typique se compose des couches suivantes :

- La couche d'entrée ;
- La couche cachée ;
- La couche de sortie.

Chaque couche comporte une collection de neurones, et les neurones d'une couche interagissent avec tous les neurones des autres couches. Cependant, les neurones d'une même couche n'interagissent pas entre eux. Un ANN typique est illustré dans le diagramme suivant :

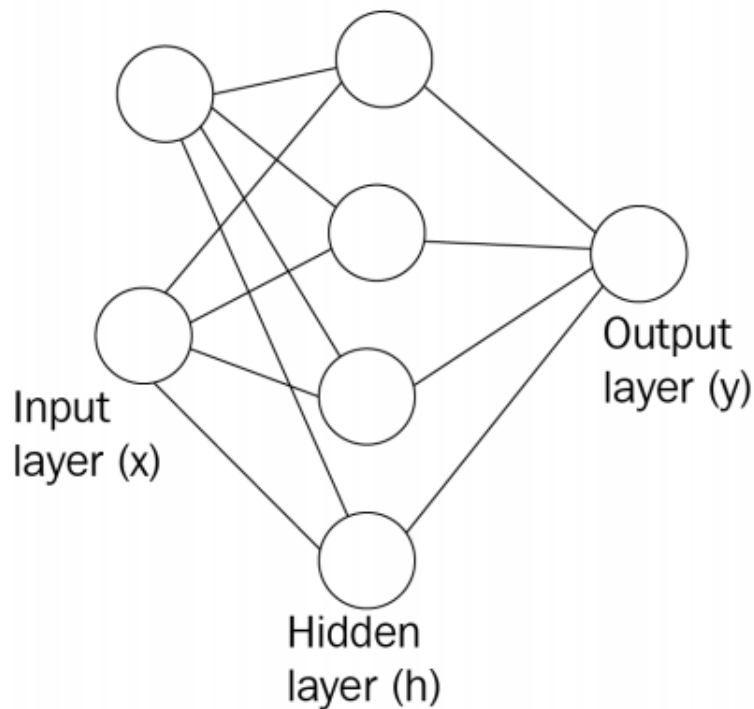


FIGURE 2.5 – Exemple d'un ANN

### - La couche d'entrée :

La couche d'entrée est l'endroit où nous alimentons le réseau. Le nombre de neurones dans la couche d'entrée est le nombre d'entrées que nous alimentons au réseau. Chaque entrée aura une certaine influence sur la prédiction de la sortie et celle-ci sera multipliée par des poids, tandis qu'un biais sera ajouté et transmis à la couche suivante.

### - La couche cachée :

Toute couche située entre la couche d'entrée et la couche de sortie est appelée couche cachée. Elle traite l'entrée reçue de la couche d'entrée. La couche cachée est responsable de la dérivation de relations complexes entre l'entrée et la sortie. En d'autres termes, la couche cachée identifie le modèle dans l'ensemble de données. Il peut y avoir un nombre quelconque de couches cachées, mais nous devons choisir un nombre de couches cachées en fonction de notre problème. Pour un problème très simple, nous pouvons utiliser une seule couche cachée, mais tout en effectuant des tâches complexes comme la reconnaissance d'images, nous utilisons plusieurs couches cachées, chaque couche étant responsable de l'extraction des caractéristiques importantes de l'image afin que nous puissions reconnaître facilement l'image. Lorsque nous utilisons de nombreuses couches cachées, le réseau est appelé un réseau neuronal profond.

### - La couche de sortie :

Après avoir traité l'entrée, la couche cachée envoie son résultat à la sortie couche. Comme son nom l'indique, la couche de sortie émet la sortie. Le nombre de neurones dans la couche de sortie est liée au type de problème que nous voulons résoudre. S'il s'agit d'une classification binaire, alors le nombre de neurones de la couche de sortie nous indiquent à quelle classe appartient l'entrée. Si elle est une classification multi-classes, disons, avec cinq classes, et si nous voulons obtenir la probabilité que chaque classe soit une sortie, puis le nombre de neurones dans la couche de sortie est de cinq, chacune émettant la probabilité. S'il s'agit d'un problème de régression, alors nous avons un neurone dans la couche de sortie.

### 2.4.3 Les fonctions d'activation

La fonction d'activation est importante pour qu'un ANN apprenne et donne un sens à quelque chose de vraiment compliqué. Leur objectif principal est de convertir un signal d'entrée d'un nœud d'un ANN en un signal de sortie. Ce signal de sortie est utilisé comme entrée dans la couche suivante de la pile.

**Définition 2.4.1.** *La fonction d'activation décide si un neurone doit être activé ou non en calculant la somme pondérée et en y ajoutant un biais supplémentaire. Le motif est d'introduire la non-linéarité dans la sortie d'un neurone.*

Si nous n'appliquons pas de fonction d'activation, le signal de sortie serait simplement une fonction linéaire (polynôme à un degré). Maintenant, une fonction linéaire est facile à résoudre, mais leur complexité est limitée, ils ont moins de pouvoir. Sans fonction d'activation, notre modèle ne peut pas apprendre et modéliser des données complexes telles que des images, des vidéos, de l'audio, de la parole, etc.

Les fonctions non linéaires sont celles qui ont un degré supérieur à un et qui ont une courbure. Nous avons maintenant besoin d'un réseau de neurones pour apprendre et représenter presque tout et n'importe quelle fonction complexe arbitraire qui mappe une entrée sur une sortie.

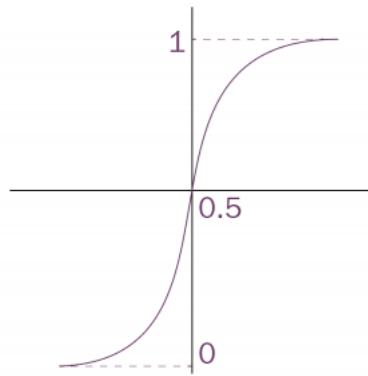
Le réseau de neurones est considéré comme des «approximateurs de fonctions universelles». Cela signifie qu'ils peuvent apprendre et calculer n'importe quelle fonction. Il existe différents types de fonctions d'activation, comme suit :

**- La fonction sigmoïde :**

La fonction sigmoïde est l'une des fonctions d'activation les plus couramment utilisées. Elle se situe entre 0 et 1. La fonction sigmoïde peut être définie comme :

$$f(z) = \frac{1}{1 + e^{-z}}$$

Lorsque nous appliquons cette fonction à  $z$ , les valeurs seront mises à l'échelle dans la plage de 0 à 1. C'est ce qu'on appelle une fonction logistique. Elle est en forme de "S", comme le montre le schéma suivant :

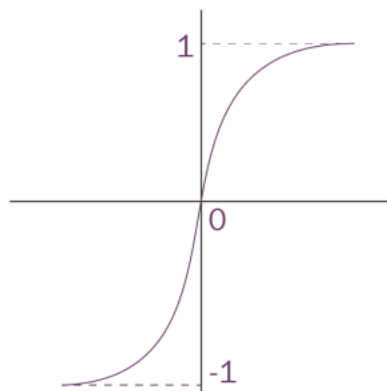


### - La fonction tangente hyperbolique :

Contrairement à la fonction sigmoïde, la fonction tangente hyperbolique échelonne la valeur entre -1 et +1. La fonction tangente hyperbolique peut être définie comme :

$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

Lorsque nous appliquons cette fonction à  $z$ , les valeurs seront mises à l'échelle dans la fourchette de -1 à +1. Elle est également en forme de s mais centrée sur zéro, comme le montre le schéma suivant :

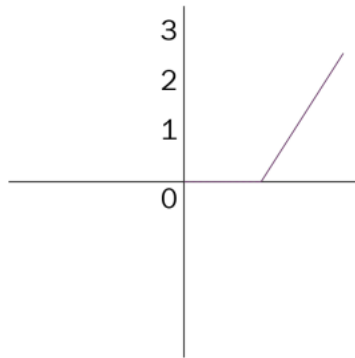


### - La fonction ReLU :

La ReLU est également connue sous le nom d'unité linéaire rectifiée. C'est l'une des fonctions d'activation les plus utilisées. La fonction ReLU peut être définie comme :

$$f(z) = \max(0, z)$$

c'est-à-dire que  $f(z)$  est égal à 0 lorsque  $z$  est inférieur à 0 et  $f(z)$  est égal à  $z$  lorsque  $z$  est supérieur ou égal à 0 :



### - La fonction Softmax :

La fonction softmax est en fait la généralisation de la fonction sigmoïde. Elle est généralement appliquée sur la dernière couche du réseau et lors de l'exécution de tâches de classification multiclasse. Elle donne les probabilités que chaque classe soit une sortie et donc que la somme des valeurs de softmax soit toujours égale à 1. Elle peut être définie comme :

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

## 2.5 Description du projet

Ce présent travail s'inscrit dans le cadre des nouvelles techniques technologiques d'intelligence artificiel qui viennent d'être appliqué dans les métiers d'ingénierie financière.

Dans cet mémoire, nous allons présenter un type particulier de machine learning appelé **RL-Reinforcement Learning** . ce type de machine learning se base sur le principe d'**actions tasks**, la chose qui le diffère par rapport aux autres types de machine learning à savoir supervised et Unsupervised.

L'objectif de mes travaux au sein de l'équipe de la front office est de mettre à la disposition du responsable Desk Equity un outil d'aide à la décision qui permet de détecter d'une manière ou d'autres les positions optimaux d'achat ou de vente sur le marché Equity Marocain en se basant sur les techniques de Reinforcement Learning .

Une études théorique générale s'avère indispensable pour la bonne compréhension du sujet, nous allons découvrir par la suite un rapprochement du principe de RL, ainsi ces différentes types et finalement une étude pratique et une interpretation des résultats obtenus .

## Deuxième partie

### Focus sur RL : Théorie vers Application

## Les bases de RL

Considérez que vous apprenez au chien à attraper une balle, mais vous ne pouvez pas enseigner explicitement au chien à attraper une balle ; au lieu de cela, vous allez simplement lancer une balle et chaque fois que le chien attrape la balle, vous lui donnez un cookie. S'il n'attrape pas le ballon, vous ne donnerez pas de cookie. Le chien déterminera quelles actions lui ont fait recevoir un cookie et répétera ces actions.

De même, dans un **environnement RL**, vous n'enseignerez pas à l'**agent** quoi faire ou comment faire à la place, vous récompenserez l'agent pour chaque **action** qu'il effectue. La récompense peut être positive ou négative. Ensuite, l'agent commencera à effectuer des actions qui lui ont fait recevoir une **récompense** positive. Il s'agit donc d'un processus d'essais et d'erreurs. Dans l'analogie précédente, le chien représente l'agent. Donner un cookie au chien en attrapant le ballon est une récompense positive, et ne pas donner de cookie est une récompense négative[4].

L'agent RL peut **explorer** différentes actions qui pourraient fournir une bonne récompense ou il peut **exploiter** (utiliser) l'action précédente qui a abouti à une bonne récompense. Si l'agent RL explore différentes actions, il y a une grande possibilité que l'agent reçoive une mauvaise récompense car toutes les actions ne seront pas les meilleures. Si l'agent RL n'exploite que la meilleure action connue, il y a également une grande possibilité de passer à côté de la



meilleure action, ce qui pourrait offrir une meilleure récompense. Il y a toujours un compromis entre l'exploration et l'exploitation. Nous ne pouvons pas effectuer à la fois l'exploration et l'exploitation en même temps. Nous discuterons en détail de l'exploration-exploitation dans les prochains chapitres.

### 3.1 RL - Algorithme

Les étapes impliquées dans l'algorithme-RL sont les suivantes :

- 1- Tout d'abord, l'agent interagit avec l'environnement en effectuant une action ;
- 2- L'agent exécute une action et passe d'un état à un autre ;
- 3- Et puis l'agent recevra une récompense en fonction de l'action qu'il a effectuée ;
- 4- En fonction de la récompense, l'agent comprendra si l'action a été bonne ou mauvaise ;
- 5- Si l'action a été bonne, c'est-à-dire si l'agent a reçu une récompense positive, alors l'agent préférera effectuer cette action ou bien l'agent essaiera d'effectuer une autre action qui se traduira par une récompense positive. Il s'agit donc essentiellement d'un processus d'apprentissage par essais et erreurs.

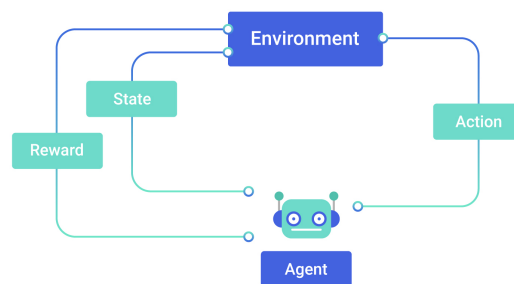


FIGURE 3.1 – RL - Algorithme

## 3.2 Les éléments de RL

Au-delà de l'agent et de l'environnement, on peut identifier quatre sous-éléments principaux d'un système d'apprentissage par renforcement : une politique(Policy), un signal de récompense, une fonction de valeur et, éventuellement, un modèle de l'environnement.

### a- Agent :

Les agents sont les logiciels qui prennent des décisions intelligentes et ils sont essentiellement des apprenants en RL. Les agents agissent en interagissant avec l'environnement et ils reçoivent des récompenses en fonction de leurs actions.

### b- Environnement :

Chaque agent interagit avec ce qu'on appelle un environnement. L'environnement est le monde extérieur. Il comprend tout en dehors de l'agent. Il existe différents types d'environnement :

- **Environnement déterministe** : Un environnement est dit déterministe lorsque nous connaissons le résultat en fonction de l'état actuel. Par exemple, dans une partie d'échecs, nous connaissons le résultat exact du déplacement d'un joueur.
- **Environnement stochastique** : Un environnement est dit stochastique lorsque nous ne pouvons pas déterminer le résultat en fonction de l'état actuel. Il y aura un plus grand niveau d'incertitude. Par exemple, nous ne savons jamais quel nombre apparaîtra lors du lancement d'un dé.
- **Environnement entièrement observable** : Lorsqu'un agent peut déterminer l'état du système à tout moment, il est appelé entièrement observable. Par exemple, dans une partie d'échecs, l'état du système, c'est-à-dire la position de tous les joueurs sur l'échiquier, est disponible tout le temps pour que le joueur puisse prendre une décision optimale.

- **Environnement partiellement observable** : Lorsqu'un agent ne peut pas déterminer à tout moment l'état du système, il est appelé partiellement observable. Par exemple, dans un jeu de poker, nous n'avons aucune idée des cartes de l'adversaire.
- **Environnement discret** : Lorsqu'il n'y a qu'un état fini d'actions disponibles pour passer d'un état à un autre, cela s'appelle un environnement discret. Par exemple, dans une partie d'échecs, nous n'avons qu'un ensemble fini de coups.
- **Environnement continu** : Lorsqu'il existe un état infini d'actions disponibles pour passer d'un état à un autre, il s'agit d'un environnement continu. Par exemple, nous avons plusieurs itinéraires disponibles pour voyager de la source à la destination.
- **Environnement épisodique et non épisodique** : L'environnement épisodique est également appelé environnement non séquentiel. Dans un environnement épisodique, l'action actuelle d'un agent n'affectera pas une action future, tandis que dans un environnement non épisodique, l'action actuelle d'un agent affectera une action future et est également appelée environnement séquentiel. Autrement dit, l'agent effectue les tâches indépendantes dans l'environnement épisodique, alors que dans l'environnement non épisodique, toutes les actions des agents sont liées.
- **Environnement mono et multi-agents** : Comme les noms le suggèrent, un environnement à agent unique n'a qu'un seul agent et l'environnement multi-agents a plusieurs agents. Les environnements multi-agents sont largement utilisés lors de l'exécution de tâches complexes. Il y aura différents agents agissant dans des environnements complètement différents. Les agents d'un environnement différent communiqueront entre eux. Un environnement multi-agents sera principalement stochastique car il présente un niveau d'incertitude plus élevé.

**c- Récompenses :**

**Définition 3.2.1.** *Un **signal de récompense** définit l'objectif d'un problème d'apprentissage par renforcement. À chaque pas de temps, l'environnement envoie à l'agent d'apprentissage par renforcement un numéro unique appelé récompense. Le seul objectif de l'agent est de maximiser la récompense totale qu'il reçoit à long terme. Le signal de récompense définit ainsi quels sont les bons et les mauvais événements pour l'agent[5].*

À partir de la définition ci-dessus, nous pourrions présenter le signal de récompense comme suit :

- Une récompense  $R_t$  est un signal de rétroaction scalaire ;
- Indique la performance de l'agent à l'étape  $t$  ;
- Le travail de l'agent consiste à maximiser la récompense cumulative.
- .

**d- La politique :**

La politique représente une classe spéciale de comportements qui, comme nous le verrons bientôt, jouent un rôle important dans la théorie des **processus de décision de Markov** (MDP). Ils sont spécifiés par un mappage  $\pi$ , qui mappe les états aux actions .

Une politique définit donc le comportement de l'agent dans un environnement c'est à dire La manière dont l'agent décide quelle action effectuer.il existe deux types des politiques à savoir une politique déterministe et l'autre stochastique comme je définie ci-dessous .

On définit :

$\mathcal{S}$  est l'ensemble d'états non vide dénombrable.

$\mathcal{A}$  est l'ensemble des actions .

**Définition 3.2.2 (Politique déterministe).** .

Soit :

$$\pi : \mathcal{S} \longrightarrow \mathcal{A}$$

Le fait de suivre  $\pi$  signifie qu'à tout moment  $t \geq 0$ , l'action  $a_t$  est sélectionnée à l'aide de :

$$a_t = \pi(s_t)$$

Pour tout,  $t \geq 0$ ,  $s_t \in \mathcal{S}$

### Définition 3.2.3 (Politique stochastique).

Pour une telle politique stochastique  $\pi$  on associe les états aux distributions sur l'espace d'action.

En se référant à une telle politique  $\pi$ , nous utiliserons  $\pi(a|s)$  pour désigner la probabilité que l'action  $a$  soit sélectionnée par  $\pi$  dans l'état  $s$  (**i.e** :  $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$  ) [6]. Notez que si une politique stochastique est suivie dans un MDP, on aura :

$$a_t \sim \pi(.|s_t), \quad t \in \mathbb{N}$$

#### e- La fonction de valeur :

Une fonction de valeur indique à quel point il est bon pour un agent d'être dans un état particulier. Elle dépend de la politique et est souvent désignée par  $v(s)$ . Il est égal à la récompense totale attendue reçue par l'agent à partir de l'état initial. Il peut y avoir plusieurs fonctions de valeur ; la fonction de valeur optimale est celle qui a la valeur la plus élevée pour tous les états par rapport aux autres fonctions de valeur. De même, une politique optimale est celle qui a la fonction de valeur optimale.

De plus, la fonction de valeur peut être fonction de l'état ou du couple état-action. Le premier cas est appelé une fonction de valeur d'état, tandis que le second est appelé une **fonction de valeur d'action**.

#### f- Le Modèle :

Le modèle est la représentation par l'agent d'un environnement. L'apprentissage peut être de deux types : **l'apprentissage basé sur un modèle et l'apprentissage sans modèle**.

Dans l'apprentissage basé sur un modèle, l'agent exploite les informations précédemment apprises pour accomplir une tâche, tandis que dans l'apprentissage sans modèle, l'agent s'appuie simplement sur une expérience d'**essai et d'erreur** pour effectuer la bonne action [4].

**Exemple 3.2.1.** *Dites que vous souhaitez atteindre votre bureau plus rapidement depuis votre domicile. Dans l'apprentissage basé sur un modèle, vous utilisez simplement une expérience précédemment acquise (carte) pour atteindre le bureau plus rapidement, tandis que dans l'apprentissage sans modèle, vous n'utiliserez pas une expérience précédente et essaierez toutes les différentes voies et choisirez la plus rapide.*

### 3.3 Les Processus décisionnels de Markov - MDP

Avant d'entrer dans MDP, comprenons d'abord la chaîne de Markov et le processus de Markov, qui forment la base de MDP.

**Définition 3.3.1** (La Propriété de Markov). .

*Un état  $S_t$  est Markov si et seulement si :*

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \dots, S_t]$$

Pour un état de Markov  $s$  et un état successeur  $s'$ , la probabilité de transition d'état est définie par :

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

Le passage d'un état à un autre est appelé transition et sa probabilité est appelée probabilité de transition. On peut formuler les probabilités de transition sous forme de matrice, donc La matrice de transition d'états  $P$  définit les probabilités de transition de tous les états  $s$  à tous les états successeurs  $s'$  est donnée comme suit :

$$P = \begin{pmatrix} P_{11} & \dots & \dots & P_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ P_{n1} & \dots & \dots & P_{nn} \end{pmatrix}$$

Où chaque ligne de la matrice est égale à 1.

**Définition 3.3.2** (Processus de Markov). .

*Un processus de Markov est un tuple  $(\mathcal{S}, \mathcal{P})$*

*Avec :*

- $\mathcal{S}$  est un ensemble (fini) d'états ;
- $\mathcal{P}$  est une matrice de probabilité de transition d'état ,  $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$

Comme nous l'avons définie, La propriété de Markov déclare que l'avenir ne dépend que du présent et non du passé. La chaîne de Markov est un modèle probabiliste qui dépend uniquement de l'état actuel pour prédire l'état suivant et non les états précédents, c'est-à-dire que l'avenir est conditionnellement indépendant du passé. La chaîne de Markov suit strictement la propriété Markov.

En effet, Un MDP est une extension de la chaîne Markov. Il fournit un cadre mathématique pour modéliser des situations de prise de décision. Presque tous les problèmes d'apprentissage par renforcement peuvent être modélisés comme MDP.

**Définition 3.3.3.** *Un processus de décision de Markov est un tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$*

- $\mathcal{S}$  est un ensemble (fini) d'états ;
- $\mathcal{A}$  est un ensemble fini d'actions ;
- $\mathcal{P}$  est une matrice de probabilité de transition d'état,  $\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$
- $\mathcal{R}$  est la fonction de récompense,  $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$  ;
- $\gamma$  est le facteur d'actualisation,  $\gamma \in [0, 1]$ .

### 3.3.1 Retours et récompenses

Comme nous l'avons appris, dans un environnement RL, un agent interagit avec l'environnement en effectuant une action et passe d'un état à un autre. En fonction de l'action qu'il effectue, il reçoit une récompense. Une récompense n'est rien d'autre qu'une valeur numérique .

Un agent essaie de maximiser le montant total des récompenses (récompenses cumulatives) qu'il reçoit de l'environnement au lieu de récompenses immédiates. Le montant total des récompenses que l'agent reçoit de l'environnement est appelé retours. Ainsi, nous pouvons formuler

le montant total des récompenses (retours) reçues par les agents comme suit :

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (3.1)$$

**Remarque 3.3.1.** *Le facteur d'actualisation  $\gamma$  détermine l'importance que nous accordons aux récompenses futures et aux récompenses immédiates. Un facteur d'actualisation égale à 0 signifie que les récompenses immédiates sont plus importantes, tandis qu'un facteur d'actualisation égale à 1 signifierait que les récompenses futures sont plus importantes que les récompenses immédiates.*

### 3.3.2 Politiques et fonctions de valeur

Presque tous les algorithmes d'apprentissage par renforcement impliquent l'estimation de fonctions de valeur - fonctions d'états (ou de paires état-action) qui estiment à quel point il est bon pour l'agent d'être dans un état donné (ou à quel point il est bon d'effectuer une action donnée dans un état donné) en termes de récompenses futures qui peuvent être attendues ou, pour être précis, en termes de rendement attendu. Bien sûr, les récompenses que l'agent peut s'attendre à recevoir à l'avenir dépendent des actions qu'il entreprendra. En conséquence, les fonctions de valeur sont définies par rapport à des manières particulières d'agir, appelées politiques.

**Définition 3.3.4** (La fonction de valeur de l'état). *La fonction de valeur d'un état  $s$  sous une politique  $\pi$ , notée  $v_\pi(s)$ , est le retour attendu à partir de l'état  $s$ , puis en suivant la politique  $\pi$ . Pour les MDP, nous pouvons définir  $v_\pi$  formellement par :*

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right], \forall s \in \mathcal{S}, \quad (3.2)$$

**Définition 3.3.5** (La fonction de valeur de l'état-action-La fonction Q). *De même, nous définissons la valeur de l'action  $a$  dans l'état  $s$  sous une politique  $\pi$ , notée  $q_\pi(s, a)$ , comme le retour attendu à partir de  $s$ , en prenant l'action  $a$ , puis en suivant la politique  $\pi$  :*

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right], \forall s \in \mathcal{S}, a \in \mathcal{A}, \quad (3.3)$$



La fonction de valeur d'état peut à nouveau être décomposée en récompense immédiate plus la valeur actualisée de l'état successeur,

En effet, à partir de l'équation (3.1) on a :

$$\begin{aligned} G_t &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) \\ &= R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+1+k+1} \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

Nous développons l'équation (3.2) ,

$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_{\pi}[R_{t+1} | S_t = s] + \gamma \mathbb{E}_{\pi}[\mathbb{E}_{\pi}[G_{t+1} | S_{t+1} = s'] | S_t = s] \\ &= \mathbb{E}_{\pi}[R_{t+1} | S_t = s] + \gamma \mathbb{E}_{\pi}[v_{\pi}(s') | S_t = s] \end{aligned}$$

Or,

$$\begin{aligned} \mathbb{E}_{\pi}[R_{t+1} | S_t = s] &= \mathbb{E}[R_{t+1} | S_t = s, A_t \sim \pi(a|s)] \\ &= \sum_a \pi(a|s) \mathcal{R}_s^a \end{aligned} \quad (1)$$

Et de même,

$$\mathbb{E}_{\pi}[v_{\pi}(s') | S_t = s] = \sum_a \pi(a|s) \sum_{s'} \mathcal{P}_{ss'}^a v_{\pi}(s') \quad (2)$$

À partir de (1) et (2), On trouve :

$$v_{\pi}(s) = \sum_a \pi(a|s) [\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a v_{\pi}(s')], \forall s \in \mathcal{S} \quad (3.4)$$

L'équation (3.3) est l'**équation de Bellman** pour  $v_{\pi}$ . Il exprime une relation entre la valeur d'un état et les valeurs de ses états successeurs.

**Définition 3.3.6** (L'équation de Bellman - représentation Matricielle). .

*L'équation de Bellman peut être éxprimer sous la forme suivante :*

$$v_{\pi} = \mathcal{R}_{\pi} + \gamma \mathcal{P} v_{\pi} \quad (3.5)$$

À condition que  $(1 - \gamma \mathcal{P})$  est inversible, (3.5) Il peut être résolu directement :

$$v_{\pi} = (1 - \gamma \mathcal{P})^{-1} \mathcal{R}_{\pi}$$

Avec :

- $v_\pi$  est le vecteur des états ;
- $\mathcal{R}_\pi$  est le vecteur des récompenses ;
- $\gamma$  est le facteur d'actualisation ( $\gamma \in [0, 1]$ ) ;
- $\mathcal{P}$  est la matrice de transition.

Nous pourrions schématiser la formule (3.3) que nous venons d'exprimer comme la montre la figure ci-dessous :

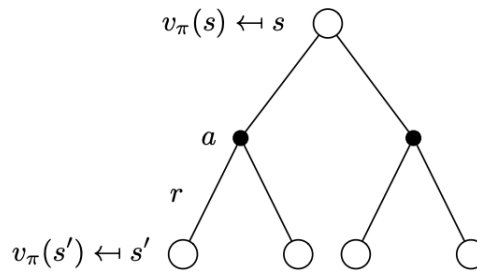


FIGURE 3.2 – Diagramme de  $v_\pi$

Chaque cercle ouvert représente un état et chaque cercle plein représente une paire état-action. En partant de l'état  $s$ , le nœud racine en haut, l'agent peut effectuer n'importe quel ensemble d'actions (deux sont illustrées dans le diagramme) en fonction de sa politique  $\pi$ . À partir de chacun de ceux-ci, l'environnement pourrait répondre avec l'un des nombreux états suivants,  $s'$ , (deux sont représentés sur la figure), avec une récompense,  $r$ , en fonction de sa dynamique donnée par la probabilité de transition.

L'équation de Bellman fait la moyenne de toutes les possibilités, pondérant chacune par sa probabilité de se produire. Il indique que la valeur de l'état de départ doit être égale à la valeur (actualisée) de l'état suivant attendu, plus la récompense attendue en cours de route.

Chaque nœud plein sur la figure (3.2) représente un ensemble (État-Action), Et nous pourrions encore représenter notre diagramme sous la forme suivante :

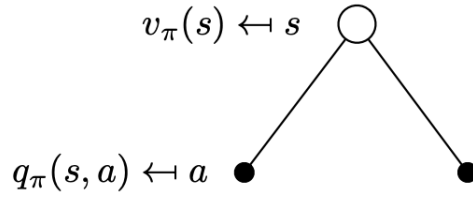


FIGURE 3.3 – Diagramme de  $v_\pi$  et  $q_\pi$

En effet,

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

Et à partir de l'équation (3.3), On a :

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a v_\pi(s') \quad (3.6)$$

Nous remplaçons dans l'équation (3.6), Nous déduisons l'équation de Bellman pour la fonction  $Q$  :

$$q_\pi(s, a) = R_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \sum_{a'} \pi(a'|s') q_\pi(s', a') \quad (3.7)$$

### 3.4 L'équation de Bellman et l'optimalité

L'équation de Bellman, du nom de Richard Bellman, mathématicien américain, nous aide à résoudre MDP. Il est omniprésent dans RL. Lorsque nous disons résoudre le MDP, cela signifie en fait trouver les politiques optimales et les fonctions de valeur. Il peut y avoir de nombreuses fonctions de valeur différentes selon différentes politiques. La fonction de valeur optimale est celle qui donne la valeur maximale par rapport à toutes les autres fonctions de valeur.

**Définition 3.4.1.** *La fonction de valeur d'état optimale  $v^*(s)$  est la fonction de valeur maximale sur toutes les politiques :*

$$v^*(s) = \max_{\pi} v_\pi(s)$$

*La fonction de valeur  $Q$  optimale  $q^*(s, a)$  est la fonction de valeur  $Q$  maximale sur toutes les politiques :*

$$q^*(s, a) = \max_{\pi} q_\pi(s, a)$$

**Définition 3.4.2.** Une politique  $\pi$  est définie comme étant supérieure ou égale à une politique  $\pi'$  si son rendement attendu est supérieur ou égal à celui de  $\pi'$  pour tous les États. En d'autres termes,  $\pi' \geq \pi$  si et seulement si  $v_{\pi'}(s) \geq v_{\pi}(s)$  pour tous  $s \in \mathcal{S}$ .

**Théorème 3.4.1** (La politique Optimale). .

Pour toute MDP :

- Il existe une politique optimale  $\pi^*$  qui est supérieur ou égal à toutes les autres politiques ,  
 $\pi^* \geq \pi, \forall \pi$ ,
- Toutes les politiques optimales atteignent la fonction de valeur optimale d'état ,

$$v_{\pi^*}(s) = v^*(s)$$

- Toutes les politiques optimales atteignent la fonction de valeur Q optimale ,

$$q_{\pi^*}(s, a) = q^*(s, a)$$

Une politique optimale peut être trouvée en maximisant  $q^*(s, a)$ ,

$$\pi_*(a|s) = \begin{cases} 1 & \text{si } a = \arg \max_{a \in \mathcal{A}} q_*(s, a) \\ 0 & \text{sinon} \end{cases}$$

Étant donné que la fonction de valeur optimale est celle qui a une valeur plus élevée par rapport à toutes les autres fonctions de valeur (c'est-à-dire le rendement maximum), ce sera le maximum de la fonction Q. Ainsi, la fonction de valeur optimale peut facilement être calculée en prenant le maximum de la fonction Q comme suit :

$$v_*(s) = \max_a q_*(s, a)$$

### La fonction d'optimalité de Bellman

Par définition, On a :

$$\begin{aligned} q_*(s, a) &= \max_{\pi} q_{\pi}(s, a) \\ &= R_s^a + \gamma \max_{\pi} \sum_{s'} \mathcal{P}_{ss'}^s v_{\pi}(s') \\ &= R_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^s v_*(s') \end{aligned}$$

Alors ,

— **La fonction d’optimalité de Bellman pour La fonction du valeur :**

$$\begin{aligned} v_*(s) &= \max_a q_*(s, a) \\ &= \max_a R_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a v_*(s') \end{aligned}$$

— **La fonction d’optimalité de Bellman pour La fonction du valeur Q :**

$$q_*(s, a) = R_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$$

## 3.5 Résolution de l’équation d’optimalité de Bellman

Nous pouvons trouver les politiques optimales en résolvant l’équation d’optimalité de Bellman. Pour résoudre l’équation d’optimalité de Bellman, nous utilisons des technique spéciales tel que **DP (Dynamic programming)**, **MC (Monte-Carlo)** et des techniques de **TD (Temporal Difference)**.

Dans ce mémoire, nous nous intéresserons plus particulièrement à la méthode la résolution de la fonction d’optimalité de Bellman en se basant sur des techniques TD à savoir Q-Learning simple et Q-Learning avec Deep Neural Networks qui est sont des méthode off-policy control qui se base sur aucun modèle (Le modèle est inconnu à l’avance- i.e : **Les probabilités des transitions et les récompenses futures ne sont pas connus**).

### 3.5.1 Q-Learning

Avant de commencer de préseter la méthode Q-learning nous allons définir une techniques très intéressant dans le domaine de RL, c’est la **politique epsilon-greedy** qui va nous aider de bien explorer-exploiter notre agent dans l’environnement .

L’algorithme Epsilon-Greedy utilise le compromis exploration-exploitation par :

— demander à l’ordinateur d’explorer (c.-à-d. choisir une option aléatoire avec probabilité epsilon)

- et exploiter (c'est-à-dire choisir l'option qui semble jusqu'à présent être la meilleure) le reste du temps.

**Algorithme d'epsilon-greedy :**

- Génère un nombre aléatoire  $r \in [0, 1]$  ,
- Si  $r < \epsilon$  choisissez une action dérivée des valeurs Q (qui donne l'utilité maximale),
- Sinon, choisissez une action aléatoire.

**- Q-learning : Off-policy TD Control**

Q-Learning est une politique de contrôle TD hors politique. il ne suit pas une politique pour trouver la prochaine action  $A'$  mais choisit plutôt l'action d'une manière **greedy**.

son objectif est d'évaluer les valeurs Q et sa règle de mise à jour est :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Avec :  $\alpha$  est également appelé le taux d'apprentissage. Sa valeur est comprise entre 0 et 1.

**Q-learning (off-policy TD control) pour l'estimation de  $\pi \approx \pi^*$**

les Paramètres d'algorithme :  $\alpha \in [0, 1], \epsilon > 0$

Initialisons Q(S,A), Pour tout  $S \in \mathcal{S}, A \in \mathcal{A}$ , arbitrairement sauf que Q (terminal,  $\cdot$ ) = 0

Boucle pour chaque épisode :

Initialisons S

Boucle pour chaque étape de l'épisode :

Choisissez A parmi S en utilisant une politique dérivée de Q (i.e  $\epsilon$ -greedy)

Prendre l'action A, observez R,  $S'$

$$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$$

$$S \leftarrow S'$$

jusqu'à ce que S soit terminal.

Dans Q Learning, nous sélectionnons une action en utilisant la politique epsilon-greedy. Soit nous explorons une nouvelle action avec la probabilité epsilon, soit nous sélectionnons la meilleure action avec une probabilité  $1 - \epsilon$ .

**Remarque 3.5.1.** *N'oubliez pas qu'en choisissant l'action à entreprendre, nous appliquons la politique epsilon-greedy : nous explorons de nouvelles actions avec une probabilité epsilon ou prenons une action qui a une valeur maximale avec une probabilité  $1 - \epsilon$ . Lors de la mise à jour de la valeur  $Q$ , nous n'exécutons pas la stratégie epsilon-greedy, nous sélectionnons simplement l'action qui a une valeur maximale.*

Les méthodes Q-Learning impliquent de stocker et de mettre à jour chaque valeur séparément dans une table. Ces approches ne sont pas évolutives car, pour un grand nombre d'états et d'actions, les dimensions de la table augmentent de façon exponentielle et peuvent facilement dépasser la capacité de mémoire disponible. En particulier, nous nous concentrerons sur **les réseaux de neurones profonds** qui sont appliqués à Q-Learning.

Nous pouvons maintenant ignorer les tables et représenter les fonctions de valeur avec un **approximateur de fonction**. L'approximation des fonctions nous permet de représenter des fonctions de valeur dans un domaine de contraintes en utilisant uniquement une quantité fixe de mémoire. Des exemples d'approximation de fonction sont **linear functions, decision trees, nearest neighbor algorithms, artificial neural networks,....**

En particulier, des réseaux neuronaux artificiels profonds ou, pour être bref, des réseaux neuronaux profonds (DNN) sont utilisés. Leur popularité est due à leur efficacité et leur capacité à apprendre des fonctionnalités par eux-mêmes, créant une représentation hiérarchique à mesure que les couches cachées du réseau augmentent.

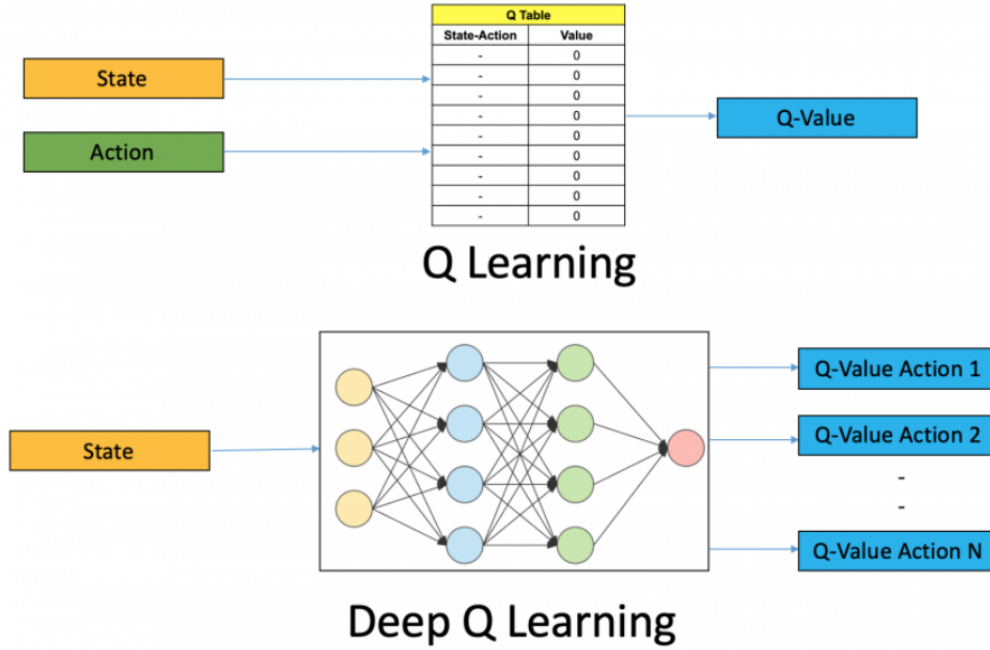


FIGURE 3.4 – Différence entre Q-Learning et Deep Q-Learning

### 3.5.2 Q-Learning avec Deep Neural Networks : DQN

Dans Q-Learning, un réseau neuronal profond apprend un ensemble de poids pour approximer la fonction de valeur  $Q$ . Ainsi, la fonction de valeur  $Q$  est paramétrée par (les poids du réseau) et écrite comme suit :

$$Q_{\theta}(s, a)$$

Pour adapter le Q-learning aux réseaux de neurones profonds (cette combinaison prend le nom de deep Q-learning), nous devons trouver une fonction de perte (ou objectif) à minimiser [2]. D'après l'algorithme Q-Learning, nous avons :

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

Nous appelons  $s, a$  l'état et l'action dans la présente étape, tandis que  $s', a'$  appelé état et action à l'étape suivante.

Avec le réseau neuronal, notre objectif est d'optimiser le poids,  $\theta$ , de sorte que  $Q_{\theta}$  ressemble à la fonction optimale de valeur  $Q$ . Mais comme nous n'avons pas la fonction- $Q$  optimale,



nous ne pouvons que faire de petits pas vers elle en minimisant l'erreur de Bellman,  $R + \gamma \max_{a'} Q(s', a') - Q(s, a)$ , en suivant presque la même étape qu'on a fait pour l'algorithme de Q-Learning à différence que au niveau de Deep Q Learning nous ne mettons pas à jour  $Q(s, a)$ . Au lieu de cela, nous prenons le gradient de la fonction  $Q$  par rapport à  $\theta$  :

$$\theta \leftarrow \theta - \alpha [R + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a)] \nabla_\theta Q_\theta(s, a) \quad (3.8)$$

Avec :

- $\nabla_\theta Q_\theta(s, a)$  est la dérivée partielle de  $Q$  par rapport à  $\theta$  ,
- $\alpha$  est le taux d'apprentissage,

Cette approche que nous venons de voir ne donne pas une bonne approximation. En pratique, Nous utilisons l'erreur quadratique comme fonction de perte et de passer d'une Q-itération en ligne vers une Q-itération par batch. Ces modifications produisent la fonction de perte suivante :

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')} [(y_i - Q_\theta(s_i, a_i))^2] \quad (3.9)$$

Où,

$$y_i = r_i + \gamma \max_{a'_i} Q_\theta(s_i, a'_i) \quad (3.10)$$

Ensuite, le paramètre réseau,  $\theta$ , est mis à jour par descente de gradient sur la fonction de perte (**MSE**) :

$$\theta = \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$$

**Remarque 3.5.2 (Les Problèmes d'instabilités dûs à l'apprentissage DQN)).** .

- Dans **l'apprentissage supervisé**, lors de l'exécution de SGD, les mini-batches sont toujours échantillonnés de manière aléatoire à partir d'un ensemble de données pour s'assurer qu'ils sont indépendants et distribués de manière identique (**IID**). En RL, c'est la politique qui rassemble l'expérience. Et parce que les états sont séquentiels et fortement liés les uns aux autres, l'hypothèse *i.i.d* est immédiatement perdue, provoquant de graves instabilités lors de l'exécution de SGD.

- Une autre cause d'instabilité est due à la non-stationnarité du processus d'apprentissage  $Q$ . vous pouvez voir que le même réseau neuronal mis à jour est également celui qui calcule les valeurs cibles. Ceci est dangereux, étant donné que les valeurs cibles seront également mises

*à jour pendant l'entraînement. C'est comme tirer sur une cible circulaire en mouvement sans tenir compte de son mouvement.*

Le DQL est mal compris théoriquement mais, comme nous le verrons bientôt, il existe un algorithme qui déploie quelques astuces pour augmenter le i.i.d des données et atténuer le problème de la cible mobile. Ces astuces rendent l'algorithme beaucoup plus stable et flexible.

### **Les solutions du problème d'instabilité des DQN**

Les principales innovations apportées par DQN impliquent un buffer de relecture (**replay buffer**) pour surmonter l'inconvénient de la corrélation des données et un réseau cible (**target network**) distinct pour surmonter le problème de non-stationnarité.

#### **a- Replay memory**

Pour utiliser plus de données IID pendant les itérations SGD, DQN a introduit une mémoire de relecture (également appelée **Experience Replay**) pour collecter et stocker l'expérience dans un grand buffer. Ce buffer contient idéalement toutes les transitions qui ont eu lieu pendant la durée de vie de l'agent. Lors de la réalisation de SGD, un mini-batch aléatoire sera collecté à partir de la relecture expérimentée et utilisé dans la procédure d'optimisation. Étant donné que le buffer de mémoire de relecture possède une expérience variée, le mini-batch qui en est échantillonné sera suffisamment diversifié pour fournir des échantillons indépendants. Une autre caractéristique très importante derrière l'utilisation d'une Experience Replay est qu'elle permet la réutilisation des données car les transitions seront échantillonnées plusieurs fois. Cela augmente considérablement l'efficacité des données de l'algorithme.

#### **b- Target network**

Le problème des cibles mobiles est dû à la mise à jour continue du réseau pendant la formation, ce qui modifie également les valeurs cibles. Néanmoins, le réseau neuronal doit se mettre à jour afin de fournir les meilleures valeurs d'état-action possibles. La solution employée dans les DQN consiste à utiliser deux réseaux de neurones. L'un est appelé le réseau en ligne, qui est constamment mis à jour, tandis que l'autre est appelé le

réseau cible, qui n'est mis à jour que toutes les  $N$  itérations. Le réseau en ligne est utilisé pour interagir avec l'environnement tandis que le réseau cible est utilisé pour prédire les valeurs cibles. De cette façon, pour  $N$  itérations, les valeurs cibles produites par le réseau cible restent fixes, empêchant la propagation des instabilités et diminuant le risque de divergence.

DQN est entraîné en minimisant la fonction de perte que nous avons déjà présentée (3.9), mais avec l'utilisation ultérieure d'un Q-target network séparé,  $\hat{Q}$ , avec un poids  $\theta'$ , en mettant tout ensemble, la fonction de perte devient :

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma \max_{a'} \hat{Q}_{\theta'}(s', a') - Q_{\theta}(s, a))^2] \quad (3.11)$$

Ici,  $\theta$  sont les paramètres de notre actual network.

L'optimisation de la fonction de perte différentiable (3.11) est réalisée avec la méthode de la descente de gradient en mini-batch (**mini-batch gradient descent**). En d'autres termes, la mise à jour d'apprentissage est appliquée aux mini-batches qui ont été extraits uniformément du buffer expérimenté. La dérivée de la fonction de perte est la suivante :

$$\nabla_{\theta} \mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma \max_{a'} \hat{Q}_{\theta'}(s', a') - Q_{\theta}(s, a)) \nabla_{\theta} Q_{\theta}(s, a)] \quad (3.12)$$

### Algorithme Deep Q-learning avec Experience Replay

En utilisant la technique **Experience Replay**, nous stockons les expériences de l'agent à chaque pas de temps,  $e_t = (s_t, a_t, r_t, s_{t+1})$  dans une **Data-set**  $\mathcal{D} = e_1, \dots, e_N$  regroupés sur plusieurs épisodes dans une replay memory. Pendant la boucle interne de l'algorithme, nous appliquons des mises à jour Q-learning, ou des mises à jour de mini-batch, à des échantillons d'expérience,  $e \sim \mathcal{D}$ , tirés au hasard à partir des échantillons stockés.

Après avoir qu'on a performé notre **experience replay**, l'agent sélectionne et exécute une action selon une politique  $\epsilon$ -greedy. Étant donné que l'utilisation d'historique de longueur arbitraire comme entrées dans un réseau de neurones peut être difficile, notre fonction Q fonctionne à la place sur une représentation de longueur fixe des historiques produites par une fonction  $\phi$ .



FIGURE 3.5 – Replay memory

L'algorithme complet, que nous appelons DQN, est présenté dans l'algorithme suivant [3] :

---

**Algorithm Deep Q-learning avec Experience Replay**


---

Initialiser la replay memory  $\mathcal{D}$  à la capacité  $N$

Initialiser la fonction de valeur  $Q$  avec des poids aléatoires

**for** episode = 1,  $M$  **do**

    Initialiser sequence  $s_1$  et prenons  $\phi_1 = \phi(s_1)$

**for**  $t = 1, T$  **do**

        Avec une probabilité  $\epsilon$ , sélectionnez une action aléatoire  $a_t$

        sinon sélectionnez  $a_t = \max_a Q_\theta^*(\phi(s_t, a))$

        Exécuter l'action  $a_t$  et observer la récompense  $r_t$

        affecter  $s_{t+1} = s_t$ , récupérer  $a_t$  et prendre  $\phi_{t+1} = \phi(s_{t+1})$

        Ajouter  $(\phi_t, a_t, r_t, \phi_{t+1})$  dans  $\mathcal{D}$

        prendre un échantillon mini-batch aléatoire de transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  à partir de  $\mathcal{D}$

        Affecter  $y_j = \begin{cases} r_j & \text{. for } \text{terminal}\phi_{j+1} \\ r_j + \gamma \max_{a'} Q_\theta(\phi_{j+1}, a') & \text{. for } \text{non-terminal}\phi_{j+1} \end{cases}$

        Effectuez une descente de gradient à  $(y_j - Q_\theta(\phi_j, a_j))^2$  selon l'équation (3.12)

**end for**

**end for**

---

**- Explication de l'algorithme :**

- Tout d'abord, nous pré-traitions et alimentons notre DQN, qui renverra les valeurs Q de toutes les actions possibles dans l'état.
- Maintenant, nous sélectionnons une action en utilisant la politique epsilon-greedy : avec la probabilité epsilon, nous sélectionnons une action aléatoire a et avec la probabilité 1-epsilon, nous sélectionnons une action qui a une valeur Q maximale.
- Après avoir sélectionné l'action a, nous effectuons cette action dans un état s et passons à un nouvel état s' et recevons une récompense.
- Nous stockons cette transition dans notre replay buffer.
- Ensuite, nous échantillonnons des batches de transitions aléatoires à partir du replay buffer et calculons la perte comme la différence au carré entre target Q et predicted Q.
- Nous effectuons une descente de gradient par rapport à nos paramètres de réseau  $\theta$  afin de minimiser cette perte.
- Après chaque T étape, nous copions les poids de notre actual network  $\theta$  dans les poids de notre target network  $\theta'$ .
- Nous répétons ces étapes pour M nombre d'épisodes.

### 3.5.3 Q-Learning avec Deep Neural Networks : DDQN

Le problème avec DQN est qu'il a tendance à surestimer les valeurs de Q. Cela est dû à l'opérateur max dans l'équation de Q-Learning. L'opérateur max utilise la même valeur pour sélectionner et évaluer une action.

Nous pouvons résoudre ce problème en prenant deux fonctions Q distinctes, chacune apprenant indépendamment. Une fonction Q est utilisée pour sélectionner une action et l'autre fonction Q est utilisée pour évaluer une action. Nous pouvons implémenter cela en modifiant simplement la fonction cible de DQN .

$$y_i = r + \gamma \max_{a'} \hat{Q}_{\theta'}(s', a')$$

Nous pouvons modifier notre fonction cible comme suit :

$$y_i^{DDQN} = r + \gamma Q_{\theta'}(s, \arg \max_a Q_{\theta^-}(s, a)) \quad (3.13)$$

Dans l'équation précédente, nous avons deux fonctions Q chacune avec des poids différents. Ainsi, une fonction Q avec des poids  $\theta'$  est utilisée pour sélectionner l'action et l'autre fonction Q avec des poids  $\theta^-$  est utilisée pour évaluer l'action.

# Implémentation d'un framework de Reinforcement Learning sous PYTHON

Pour les besoins de ce mémoire, un framework de Reinforcement Learning a été entièrement développé dans le langage Python. Si ce framework a été développé spécifiquement pour le présent problème, nous l'avons voulu assez généraliste afin de permettre de l'adapter ou de l'étendre aussi facilement que possible.

Vous trouvez ci-dessous les composantes principales définissent ce framework .

## 4.1 L'environnement

L'environnement définit l'espace dans lequel va évoluer l'agent. Dans notre implémentation, Cet espace est un simple chart sur lequel notre agent pourra se performer et de détecter (dans le sens de maximisation des récompenses) les positions à prendre sur le marché equity Marocain dès que possible .

une position d'achat est mentionnée par une boule bleu et une position de vente par une boule orange, qui pourront être visualiser sur le chart de la manière suivante :



### 4.1.1 Identification d'un état

Pour le présent travail, nous prenons à chaque pas du temps Le vecteur **PX\_LAST** comme indicateur de détection de la position actuelle de notre état, On se basant donc sur le dernier prix de clôture du jour précédent pour définir notre état courant du jour suivant, afin de performer l'agent pour nous choisirons la position optimale à prendre.

Nous récupérons le vecteur des états comme suit :

```
def get_stock_data(stock_file):  
    """Reads stock data  
    """  
    df = pd.read_excel(stock_file)  
    #df = pd.read_excel(r'stock file')  
    return list(df['PX_LAST'])
```

FIGURE 4.1 – Récupération du vecteur des états à partir de notre data

## 4.2 L'agent

L'objet agent est un élément très essentielle dans l'implémentation de notre framework, comme nous l'avons déjà mentionné un agent pourra se performer dans un environnement afin d'exécuter ses ordres.

L'agent est considéré comme le cerveau de notre framework, nous faisons appel aux méthodes de deep Q learning qui sont généralement des réseaux de neurones multi-layers artificiels .



Notre Objet Agent prend comme entré Le vecteur des états **PX\_LAST**, et le réseau neuronal DQN et DDQN nous donnons comme output les actions (autrement les Q value fuction) à performer dans notre environnement, et nous appliquons l'algorithme d'epsilon-greedy pour que l'agent pourra s'explorer dans l'environnement pour une première étape afin de performer ses actions et de l'exploiter pour une deuxième étape .

L'objet Agent est définie de la manière suivante :

```
class Agent:
    def __init__(self):
        ....~
    def _model(self):
        """Creates the model
        """
        ....~
        return model

    def remember(self):
        ....~
    def act(self, state, is_eval=False):
        """Take action from given possible set of actions
        """
        ....~

    def train_experience_replay(self):
        ....~

    def save(self):
        ....~

    def load(self):
        ....~
```

FIGURE 4.2 – Objet Agent

Le constructeur de la classe Agent contient les éléments que nous aurons besoin tout au long de la vie de l'agent à savoir les paramètres de configuration de l'agent ainsi que les paramètres de configuration du modèle. La fonction **def act()** prend des actions aléatoires afin de lancer l'expérience. La fonction **def train\_experience\_replay()** donne la définition des algorithmes DQN et DDQN afin d'enregistrer l'expérience en utilisant la fonction **def save()**.

### 4.2.1 Actions possibles

Dans tout le framework, une action est désignée par un nombre entier et correspond à un mouvement de l'agent concerné.

Dans notre problème, nous prenons comme nombre d'action possible 3 à savoir :

- 1 :BUY
- 2 :SELL
- 0 :HOLD

### 4.2.2 Reward

Comme nous l'avons vu, le principe même du Reinforcement Learning repose sur la notion de reward. Le reward est propre à chaque agent et il est donc logique que le reward pour notre problème est le PnL réalisé sur chaque position.

Sur un marché action, on dit qu'une position est dénouée si nous vendons la position déjà qu'on a achetée. donc notre agent prend une position d'achat dès que possible et le reward (+ ou -) se calcule à partir du moment où l'action sera vendue. Notre agent pourra même ne pas effectuer aucune action et la position sera HOLD et aucun PnL n'est pas attaché.

Au moment du lancement de notre expérience, notre agent calcule le reward de chaque action pris dans l'environnement afin de nous retourner à la fin de notre expérience un **total\_profit**.

## 4.3 Exploration/Exploitation

Toutes les actions sont essayées avec une probabilité non nulle (epsilon) :

- Avec une probabilité epsilon nous explorons différentes actions au hasard ;
- Avec une probabilité 1-epsilon nous choisissons une action qui a une valeur maximale (c'est-à-dire que nous ne faisons aucune exploration)

.

Alors, au lieu d'exploiter la meilleure action tout le temps, avec la probabilité epsilon, nous explorons différentes actions au hasard. Si la valeur de l'epsilon est définie sur zéro, nous ne ferons aucune exploration. C'est simplement la politique greedy, et si la valeur de epsilon est définie sur un, alors il ne fera que de l'exploration.

La valeur de l'epsilon diminuera avec le temps car nous ne voulons pas l'explorer pour toujours. Au fil du temps, notre politique exploite donc les bonnes actions.

## 4.4 Déroulement du programme

Cet application s'est crée sur **JetBrains-PyCharm** en utilisant le langage de programmation Python. La première étape consiste à créer un environnement virtuel propre pour cette application sur lequel nous récupurons toutes les bibliothèques que nous devons utilisées tout au long de ce projet .

nous créons un dossier **data** pour stocker nos bases de données à savoir les fichier "**MASI.xlsx**", "**MASI\_Train.xlsx**" et "**MASI\_Eval.xlsx**". un autre dossier **trading\_bot** sur laquelle nous donnons notre classe Agent créer sur le fichier "**Agent.py**" et différentes techniques et outils que nous aurons besoin, ces deux dernier fichier porte le nom "**methods.py**" et "**utils.py**".

Pour le fichier **main()**, j'ai créer deux fichier, le premier porte le nom "**train.py**" qui permet en premier lieu de faire le "**training**" de notre data(i.e Les observations de l'ensemble

d'apprentissage forment l'expérience que l'algorithme utilise pour apprendre ) et en deuxième lieu, un fichier "**eval.py**" pour évaluer la performance de notre modèle en se basant sur ce qu'il a pris notre agent dans la phase de d'entraînement .

Nous montrons ci-dessus la structure du projet qui englobe tous ces éléments :

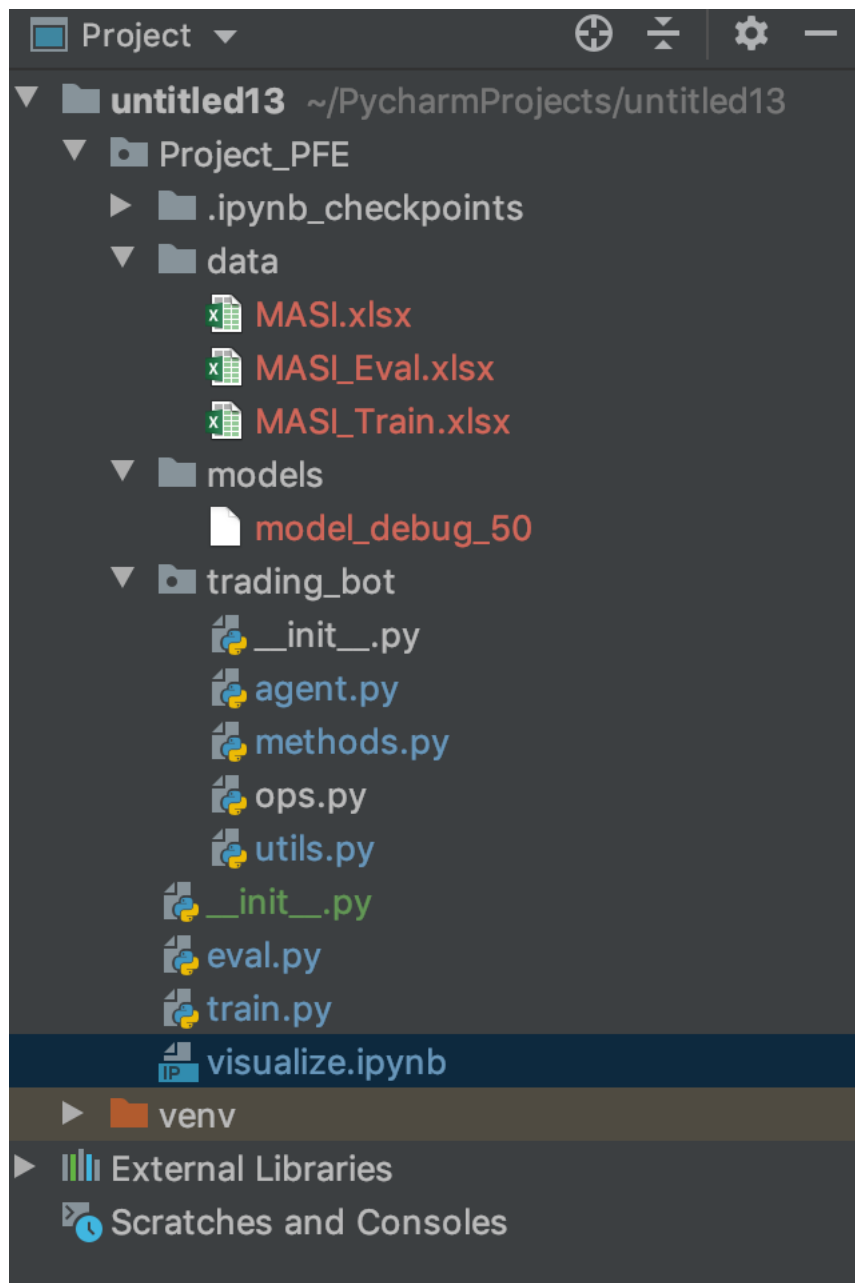


FIGURE 4.3 – La structure du projet

Avant de lancer le programme, il nous faut définir des éléments très essentiels dans la phase d'apprentissage des réseaux neuronnals. Ces éléments sont indispensables dans le sens de définir la manière et la façon comment ces données vont se propager.

En effet, nous ne pouvons pas transmettre toutes les données à l'ordinateur en même temps. Donc, pour surmonter ce problème, nous devons diviser les données en plus petites tailles et les donner à notre ordinateur un par un et mettre à jour les poids des réseaux de neurones à la fin de chaque étape pour les adapter aux données fournies.

### **Epochs :**

Une époque se produit lorsqu'un ensemble de données est transmis en avant et en arrière à travers le réseau neuronal une fois seulement.

Passer l'ensemble de données à travers un réseau de neurones une seule fois ne suffit pas. Et nous devons transmettre le jeu de données complet plusieurs fois au même réseau neuronal.

Malheureusement, il n'y a pas un bon nombre d'epochs. Un bon nombre d'epochs est différent pour différents ensembles de données, mais vous pouvez dire que le nombre d'époques est lié à la diversité de vos données.

### **Batch Size :**

Nombre total d'exemples de formation présents dans un seul batch.

Vous ne pouvez pas passer tout l'ensemble de données dans le réseau neuronal à la fois. Ainsi, vous divisez l'ensemble de données en nombre de batches ou ensembles ou pièces.

### **Le nombre d'itérations :**

Itérations est le nombre de batches nécessaires pour terminer une epoch.

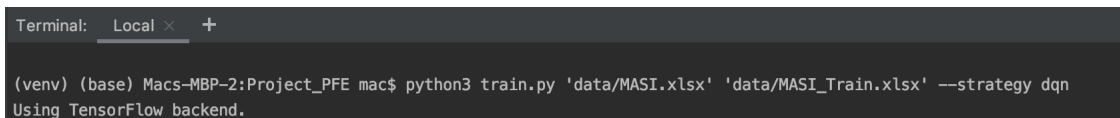
**Remarque 4.4.1.** *Le nombre de batches est égales aux nombres d'itérations pour une époque.*

**Exemple 4.4.1.** *Disons que nous avons 2000 exemples de formation que nous allons utiliser. Nous pouvons diviser l'ensemble de données de 2000 exemples en batches de 500, puis il faudra 4 itérations pour terminer 1 epoch.*

*Où la taille du batch est de 500 et les itérations de 4, pour 1 epoch complète.*

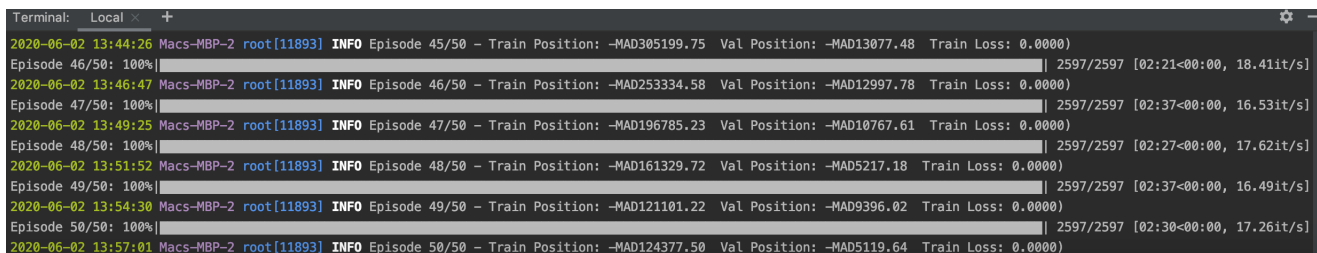
### 4.4.1 DQN

Pour lancer ce programme, nous ouvrons un terminal et commençons à former notre agent en entrant cette instruction et nous mentionnons le type de la stratégie à utiliser, ici sur la figure ci-dessous j'ai utilisé la stratégie **dqn** :



```
Terminal: Local x +
(venv) (base) Macs-MBP-2:Project_PFE mac$ python3 train.py 'data/MASI.xlsx' 'data/MASI_Train.xlsx' --strategy dqn
Using TensorFlow backend.
```

Comme titre d'exemple, pour essayer de lancer le programme nous utilisons un **ep\_count = 50 (nombre d'epochs)** qui correspond aux nombre des épisodes d'entraînement que notre agent a effectué, et nous observons ainsi sur le terminal le résultat :



```
Terminal: Local x +
2020-06-02 13:44:26 Macs-MBP-2 root[11893] INFO Episode 45/50 - Train Position: -MAD305199.75 Val Position: -MAD13077.48 Train Loss: 0.0000
Episode 46/50: 100% | 2597/2597 [02:21<00:00, 18.41it/s]
2020-06-02 13:46:47 Macs-MBP-2 root[11893] INFO Episode 46/50 - Train Position: -MAD253334.58 Val Position: -MAD12997.78 Train Loss: 0.0000
Episode 47/50: 100% | 2597/2597 [02:37<00:00, 16.53it/s]
2020-06-02 13:49:25 Macs-MBP-2 root[11893] INFO Episode 47/50 - Train Position: -MAD196785.23 Val Position: -MAD10767.61 Train Loss: 0.0000
Episode 48/50: 100% | 2597/2597 [02:27<00:00, 17.62it/s]
2020-06-02 13:51:52 Macs-MBP-2 root[11893] INFO Episode 48/50 - Train Position: -MAD161329.72 Val Position: -MAD5217.18 Train Loss: 0.0000
Episode 49/50: 100% | 2597/2597 [02:37<00:00, 16.49it/s]
2020-06-02 13:54:30 Macs-MBP-2 root[11893] INFO Episode 49/50 - Train Position: -MAD121101.22 Val Position: -MAD9396.02 Train Loss: 0.0000
Episode 50/50: 100% | 2597/2597 [02:30<00:00, 17.26it/s]
2020-06-02 13:57:01 Macs-MBP-2 root[11893] INFO Episode 50/50 - Train Position: -MAD124377.50 Val Position: -MAD5119.64 Train Loss: 0.0000
```

FIGURE 4.4 – Le training de notre data avec un **ep\_count = 50**

Un fichier **model\_debug\_50** sera créer dans le dossier models qui permet de stocker les paramètres de notre modèle. ces paramètres vont être utilisés dans la phase de test afin de mesurer la performance de la stratégie effectuée.

## CHAPITRE 4. IMPLÉMENTATION D'UN FRAMEWORK DE REINFORCEMENT LEARNING SOUS PYTHON

Nous montrons ci-dessus l'instruction à exécuter pour lancer notre phase de test et de voir les positions qui sont performées par notre agent en interaction avec l'environnement.

```
Terminal: Local x +
(venv) (base) Macs-MBP-2:Project_PFE mac$ python3 eval.py 'data/MASI_Eval.xlsx' --model-name model_debug_50 --debug
```

Un résultat de la forme suivante sera affiché :

```
Use tf.where in 2.0, which has the same broadcast rule as np.where
2020-06-02 15:10:04 Macs-MBP-2 tensorflow[12348] WARNING From /Users/mac/PycharmProjects/untitled13/Project_PFE/trading_bot/agent.py:21: add_dispatch_support.<locals>.wrapper
(from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /Users/mac/PycharmProjects/untitled13/venv/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

2020-06-02 15:10:05 Macs-MBP-2 tensorflow[12348] WARNING From /Users/mac/PycharmProjects/untitled13/venv/lib/python3.7/site-packages/keras/backend/tensorflow_backend.py:422:
The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12597.09_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12577.50_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12468.24_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12531.90_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12288.16_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12312.97_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12180.99_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12186.39_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Sell at: 12034.32_MAD | Position: -MAD562.77
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12204.42_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12340.24_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12274.85_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12279.24_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12506.96_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Sell at: 12406.97_MAD | Position: -MAD170.53
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12442.24_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12410.25_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12380.29_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 12189.28_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 11961.71_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] DEBUG Buy at: 11881.83_MAD
2020-06-02 15:10:05 Macs-MBP-2 root[12348] INFO model_debug_50: -MAD733.30
```

Après le test de notre agent, une position d'achat s'est passée au prix d'achat 12597.89 MAD, 7 positions d'achat encore s'est passée aux prix successives 12577.50 MAD, 12468.24 MAD, 12531.90 MAD, 12288.16 MAD, 12312.97 MAD, 12180.99 MAD et 12186.39 MAD afin de clôturer ces positions au niveau du prix 12034.32 MAD. Un PnL de -562.77 MAD s'est obtenu .

De même, Une position long s'est ouverte à hauteur de 12204.42 MAD. 4 positions d'achat encore s'est passée aux prix successives 12340.24 MAD, 12274.85 MAD, 12279.24 MAD et 12506.96 MAD afin de clôturer ces positions au niveau du prix 12406.97 MAD. Un PnL de -170.53 MAD s'est obtenu.

Six positions d'achats sont pris par la suite et non pas encore clôturer, pour résumer donc notre agent pendant cette épisode à réaliser un PnL de -733.30 MAD. Notre agent donc à perdu sur ces positions une somme de -733.30 MAD.

et nous pouvons visualiser sur le graphe ci-dessous les positions present par notre agent :

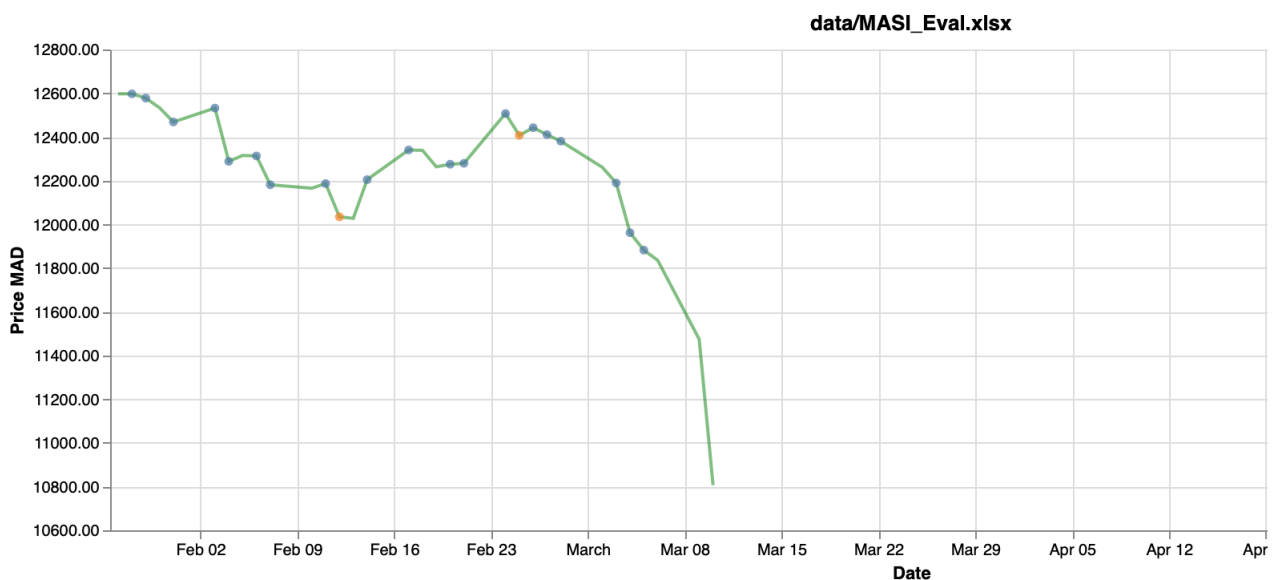


FIGURE 4.5 – Les positions present par l'agent pour la stratégie dqn



### 4.4.2 DDQN

De la même façon que nous avons suivi pour lancer notre programme pour la stratégie DQN. Nous passons la même instruction comme mentionner ci-dessous en choisissant la stratégie DDQN comme suit :

```
Terminal: Local +
(venv) (base) Macs-MBP-2:Project_PFE mac$ python3 train.py 'data/MASI.xlsx' 'data/MASI_Train.xlsx' --strategy double-dqn
Using TensorFlow backend.
2020-06-08 12:26:09 Macs-MacBook-Pro-2.local root[21144] DEBUG switching to TensorFlow for CPU
WARNING:tensorflow:From /Users/mac/PycharmProjects/untitled13/Project_PFE/trading_bot/agent.py:21: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
2020-06-08 12:26:10 Macs-MacBook-Pro-2.local tensorflow[21144] WARNING From /Users/mac/PycharmProjects/untitled13/Project_PFE/trading_bot/agent.py:21: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

Encore une autre fois, un autre fichier sera créer sur le dossier **models** et porte le nom **model\_debug\_50** qui rassemble toutes les informations concernant la phase d'apprentissage de notre agent.

Par la suite, nous exécutons l'instruction suivante pour lancer le test de notre méthode et de voir les actions proposées par notre agent pour les performées dans l'environnement :

```
Terminal: Local +
(venv) (base) Macs-MBP-2:Project_PFE mac$ python3 eval.py 'data/MASI_Eval.xlsx' --model-name model_debug_50 --debug
Using TensorFlow backend.
2020-06-08 23:45:23 Macs-MBP-2 root[22085] DEBUG switching to TensorFlow for CPU
WARNING:tensorflow:From /Users/mac/PycharmProjects/untitled13/Project_PFE/trading_bot/agent.py:21: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
2020-06-08 23:45:23 Macs-MBP-2 tensorflow[22085] WARNING From /Users/mac/PycharmProjects/untitled13/Project_PFE/trading_bot/agent.py:21: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

Après l'exécution de l'instruction précédente, un résultat de la forme suivante est affiché :

```
2020-06-08 23:45:25 Macs-MBP-2 root[22085] DEBUG Buy at: 12597.09_MAD
2020-06-08 23:45:25 Macs-MBP-2 root[22085] DEBUG Buy at: 12204.42_MAD
2020-06-08 23:45:25 Macs-MBP-2 root[22085] DEBUG Buy at: 12274.85_MAD
2020-06-08 23:45:25 Macs-MBP-2 root[22085] DEBUG Buy at: 11881.83_MAD
2020-06-08 23:45:25 Macs-MBP-2 root[22085] INFO model_debug_50: USELESS
```

Finalement, Notre agent nous informe de passer quatre positions d'achats aux niveaux des prix successives suivant : 12597.09 MAD, 12204.42 MAD, 12274.85 MAD et 11881.83 MAD. aucune de ces positions n'est pas dénoué par notre agent. En effet, nous pouvons pas calculer notre PnL pour avoir un moyen de comparaison par rapport à la méthode DQN.

Pour les deux stratégies nous avons un nombre très réduits d'épisodes pour la phase d'apprentissage à savoir 50 épisodes pour le training de notre data. ce nombre d'épisode prend comme durée d'exécution à peu près trois heures. En effet, En se basant sur les résultat données par nos agent pour les deux stratégies, nous pouvons pas faire une comparaison pour définir la stratégie le mieux adapté avec notre problème.

En outre, un nombre plus supérieur que celui qu'on a utilisé prendra comme durée plus de dix heures et plus. en augmentant le nombre d'épisodes notre agent apprend bien nos données afin d'avoir des résultats mieux que celles qu'on a obtenu.

**Remarque 4.4.2.** *pour les deux stratégies dans la phases d'apprentissage de nos données, je me suis basé sur un autre paramètre d'optimisation de la perte générée par notre modèle à savoir la fonction de perte de Huber.*

# Conclusion générale et perspectives

Ce rapport a présenté des nouvelles techniques de machine learning appliquer aux stratégies d'investissement. Nous avons présenté une variante du Q-learning avec une mémoire de lecture d'expérience pour faciliter la formation des réseaux profonds pour RL.

Le but de ce stage, c'est de porter ou de proposer des techniques de ML pour la détermination des positions optimaux sur le marché action Marocain. la première étape consiste de créer un agent qui pourra interagir avec son environnement de trading et d'explorer ses actions. La deuxième étape à pour but d'exploiter les positions optimaux afin de se déplacer dans le sens de la maximisation de **PnL**.

Mes travaux au sein de l'équipe d'UCM vont continuer et nous souhaitons de bien développer ce travail afin de lui permettre de prendre en considération plusieurs paramètres techniques et fondamentales .

Les perspectives de ce stage pour la période restante est de revoir si on peut développer ou améliorer notre agent pour faire des tâches plus compliquées que celle present.

# Bibliographie

- [1] AMMC. Guide pédagogique : Comprendre les opcv. <http://www.ammc.ma>, Mis en ligne 2018.
- [2] A. Lonza. *Reinforcement Learning Algorithms with Python*. Packt Publishing Ltd, 2019.
- [3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmille. Playing atari with deep reinforcement learning. *arXiv :1312.5602*, pages 4,5, jan 2013.
- [4] S. Ravichandiran. *Hands-On Reinforcement Learning with Python*. Packt Publishing Ltd, 2018.
- [5] R. S. Sutton and A. G. Barto. *Reinforcement Learning : An Introduction*. 2018.
- [6] C. Szepesvári. *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2010.

## Les fonctions d'erreurs

### Gradient Descent

Nous devons quantifier l'erreur de manière à ce que le signe soit toujours positif et à ce que les erreurs importantes soient amplifiées. On définit l'erreur de somme au carré (SSE), désignée par  $E$  :

$$E = \frac{1}{2} \sum_{\mu} \sum_j [y_j^{\mu} - \hat{y}_j^{\mu}]^2 \quad (\text{A.1})$$

Avec :

- $y$  : La valeur juste qu'on attend de recevoir par notre modèle ;
- $\hat{y}$  : La valeur prédite ;
- $j$  : unités de sortie du réseau ;
- $\mu$  : les points de données .

la prédiction  $\hat{y}$  d'un NN est déterminée par ses poids :

$$\hat{y}_j^{\mu} = f\left(\sum_i w_{ij} x_i^j\right)$$

Avec :

- $f$  est la fonction d'activation ;
- $x_i^j$  est l'entrée  $i$  pour l'unité de sortie du réseau  $j$ .

En intégrant ainsi l'équation de  $\hat{y}$  dans l'équation SSE, on peut voir que l'erreur dépend des poids :

$$E = \frac{1}{2} \sum_{\mu} \sum_j [y_j^{\mu} - f(\sum_i w_{ij} x_i^j)]^2$$

Par conséquent, notre objectif est de déterminer les poids qui minimiseront l'erreur. Ce dernier est parfois appelé **fonction de coût**, l'idée étant que nous plaçons un terme de pénalité sur les prédictions erronées.

### Gradient :

Le gradient est un vecteur qui est tangent à une fonction et pointe dans la direction de la plus grande augmentation de cette fonction. Le gradient est nul pour un maximum ou un minimum local car il n'y a pas de direction unique d'augmentation. En mathématiques, le gradient est défini comme une dérivée partielle pour chaque variable d'entrée de la fonction.

Il est défini comme suit :

$$\nabla f(x_1, x_2, \dots, x_n) = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right) \quad (\text{A.2})$$

### Idée qui se cache derrière cette méthode :

Comme gradient est un vecteur pointant vers la plus grande augmentation d'une fonction, le gradient négatif est un vecteur pointant vers la plus grande diminution d'une fonction. Par conséquent, nous pouvons minimiser une fonction en déplaçant légèrement un peu dans la direction du gradient négatif. C'est la logique de la descente de gradient.

Prenons le cas le plus simple et supposons qu'on a une seule unité de sortie ( $j=1$ ), on peut prendre la dérivée de l'erreur  $E$  par rapport à une pondération :

$$\frac{\partial E}{\partial w_i} = (y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i}$$

Soit  $\hat{y}$  le résultat d'une seule fonction d'activation  $f$  qui prend une entrée  $h$  telle que  $h$  prend des entrées et les multiplie par leurs poids :

$$\hat{y} = f(h)$$

$$h = \sum_i w_i x_i$$

Remplacer dans la dérivée de l'erreur :

$$\frac{\partial E}{\partial w_i} = (y - \hat{y}) f'(h) \frac{\partial}{\partial w_i} \sum_i w_i x_i$$

On peut substituer dans la dérivée partielle de E , on trouve :

$$\frac{\partial E}{\partial w_i} = (y - \hat{y})f'(h)x_i$$

On définit le pas des poids par :

$$\Delta w_i = \eta(y - \hat{y})f'(h)x_i$$

Pour rendre notre notation plus facile en bout de ligne, nous définissons un terme d'erreur  $\delta$  comme erreur multipliée par la dérivée de la fonction d'activation en h, on pose :

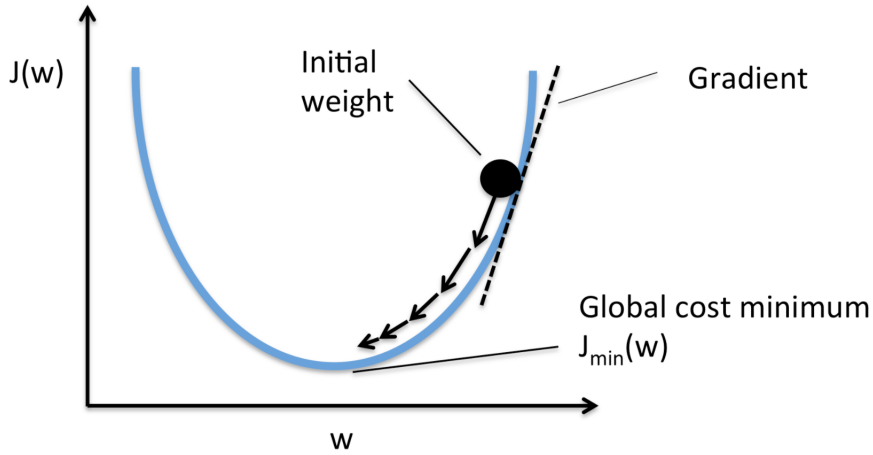
$$\delta = (y - \hat{y})f'(h) \quad (\text{A.3})$$

Cela simplifie le pas de poids pour :

$$\Delta w_i = \eta\delta x_i$$

Généralisation :

$$\Delta w_{ij} = \eta\delta_j x_{ji} \quad (\text{A.4})$$



### Value Function Approximation par Gradient Descent

Nous définissons la fonction de perte comme suit :

$$J(w) = \mathbb{E}_{\pi}[(v_{\pi}(s) - \hat{v}(s, w))^2]$$

Gradient Descent trouve un minimum local

$$\begin{aligned} \Delta w &= -\frac{1}{2}\alpha \nabla_w J(w) \\ &= \eta \mathbb{E}_{\pi}[(v_{\pi}(s) - \hat{v}(s, w)) \nabla_w \hat{v}(s, w)] \end{aligned}$$

### Huber loss

La fonction de perte de Huber décrit la pénalité encourue par une procédure d'estimation de  $f$ .  
la fonction de perte par morceaux est :

$$L_{\delta}(a) = \begin{cases} \frac{1}{2}a^2 & . \quad si \quad |a| \leq \delta \\ \delta|a| - \frac{1}{2}\delta^2 & . \quad sinon \end{cases}$$

Cette fonction est quadratique pour les petites valeurs de  $a$  et linéaire pour les grandes valeurs, avec des valeurs et des pentes égales des différentes sections aux deux points où  $|a| = \delta$ .  
La variable  $a$  fait souvent référence aux résidus, c'est-à-dire à la différence entre les valeurs observées et prédites  $a = y - f(x)$ , de sorte que :

$$L_{\delta}(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & . \quad si \quad |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & . \quad sinon \end{cases}$$



## Code source - Class Agent

```
1 import random
2 from collections import deque
3 import numpy as np
4 import tensorflow as tf
5 import keras.backend as K
6 from keras.models import Sequential
7 from keras.models import load_model, clone_model
8 from keras.layers import Dense
9 from keras.optimizers import Adam
10
11
12 def huber_loss(y_true, y_pred, clip_delta=1.0):
13
14     error = y_true - y_pred
15     cond = K.abs(error) <= clip_delta
16     squared_loss = 0.5 * K.square(error)
17     quadratic_loss = 0.5 * K.square(clip_delta) + clip_delta * (K.abs(error) - clip_delta)
18     return K.mean(tf.where(cond, squared_loss, quadratic_loss))
19
20
21 class Agent:
22
23
24     def __init__(self, state_size, strategy="t-dqn", reset_every=1000, pretrained=False, model_name=None):
25         self.strategy = strategy
26
27         # agent config
28         self.state_size = state_size          # normalized previous days
29         self.action_size = 3                  # [sit, buy, sell]
30         self.model_name = model_name
31         self.inventory = []
32         self.memory = deque(maxlen=10000)
33         self.first_iter = True
```

```
34
35     # model config
36     self.model_name = model_name
37     self.gamma = 0.95 # affinity for long term reward
38     self.epsilon = 1.0
39     self.epsilon_min = 0.01
40     self.epsilon_decay = 0.995
41     self.learning_rate = 0.001
42     self.loss = huber_loss
43     self.custom_objects = {"huber_loss": huber_loss} # important for loading the model from memory
44     self.optimizer = Adam(lr=self.learning_rate)
45
46     if pretrained and self.model_name is not None:
47         self.model = self.load()
48     else:
49         self.model = self._model()
50
51     # strategy config
52     if self.strategy in ["t-dqn", "double-dqn"]:
53         self.n_iter = 1
54         self.reset_every = reset_every
55
56     # target network
57     self.target_model = clone_model(self.model)
58     self.target_model.set_weights(self.model.get_weights())
59
60     def _model(self):
61         """Creates the model
62         """
63         model = Sequential()
64         model.add(Dense(units=128, activation="relu", input_dim=self.state_size))
65         model.add(Dense(units=256, activation="relu"))
66         model.add(Dense(units=256, activation="relu"))
```

```
67         model.add(Dense(units=128, activation="relu"))
68         model.add(Dense(units=self.action_size))
69
70         model.compile(loss=self.loss, optimizer=self.optimizer)
71         return model
72
73     def remember(self, state, action, reward, next_state, done):
74         """Adds relevant data to memory
75         """
76         self.memory.append((state, action, reward, next_state, done))
77
78     def act(self, state, is_eval=False):
79         """Take action from given possible set of actions
80         """
81         # take random action in order to diversify experience at the beginning
82         if not is_eval and random.random() <= self.epsilon:
83             return random.randrange(self.action_size)
84
85         if self.first_iter:
86             self.first_iter = False
87             return 1_# make a definite buy on the first iter
88
89         action_probs = self.model.predict(state)
90         return np.argmax(action_probs[0])
91
92     def train_experience_replay(self, batch_size):
93         """Train on previous experiences in memory
94         """
95         mini_batch = random.sample(self.memory, batch_size)
96         X_train, y_train = [], []
97
98         # DQN
99         if self.strategy == "dqn":
100             for state, action, reward, next_state, done in mini_batch:
101                 if done:
102                     target = reward
103                 else:
104                     # approximate deep q-learning equation
105                     target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])
106
107                 # estimate q-values based on current state
108                 q_values = self.model.predict(state)
109                 # update the target for current action based on discounted reward
110
111                 X_train.append(state[0])
112                 y_train.append(q_values[0])
113
114             # Double DQN
115             elif self.strategy == "double-dqn":
116                 if self.n_iter % self.reset_every == 0:
117                     # reset target model weights
118                     self.target_model.set_weights(self.model.get_weights())
119
120                 for state, action, reward, next_state, done in mini_batch:
121                     if done:
122                         target = reward
123                     else:
124                         # approximate double deep q-learning equation
```

```
126         target = reward + self.gamma * self.target_model.predict(next_state)[0]
127         [np.argmax(self.model.predict(next_state)[0])]
128
129         # estimate q-values based on current state
130         q_values = self.model.predict(state)
131         # update the target for current action based on discounted reward
132         q_values[0][action] = target
133
134         X_train.append(state[0])
135         y_train.append(q_values[0])
136
137     else:
138         raise NotImplementedError()
139
140     # update q-function parameters based on huber loss gradient
141     loss = self.model.fit(
142         np.array(X_train), np.array(y_train),
143         epochs=1, verbose=0
144     ).history["loss"][0]
145
146     # as the training goes on we want the agent to
147     # make less random and more optimal decisions
148     if self.epsilon > self.epsilon_min:
149         self.epsilon *= self.epsilon_decay
150
151     return loss
152
153 def save(self, episode):
154     self.model.save("models/{}_{}".format(self.model_name, episode))
155
156 def load(self):
157     return load_model("models/" + self.model_name, custom_objects=self.custom_objects)
158
```