Le Reinforcement Learning : une introduction théorique et pratique

Comprendre le Reinforcement Learning : ses enjeux, ses méthodes, son design

Bref historique

L'apprentissage par renforcement (que nous noterons reinforcement learning) est une branche très récente du machine learning. Le site d'Open AI, organisation connue pour fournir des "environnements" (nous allons voir ce que signifie ce mot) open-source de reinforcement learning, donne à cette adresse des articles importants dans ce domaine : à quelques exceptions près (des publications en 1990, 2000 ou 2008), tous ont été publiés après 2010. Et pour cause, le reinforcement learning a été popularisé par un article intitulé *Playing Atari with Deep Reinforcement Learning* et publié par des ingénieurs de Google DeepMind. Dans cet article, les auteurs montrent comment il est possible d'entraîner un ordinateur à jouer à n'importe quel jeu Atari. L'ordinateur commence par faire des erreurs puis s'améliore graduellement - très vite - jusqu'à devenir imbattable. L'idée est de faire jouer l'ordinateur un grand nombre de fois et de le récompenser pour ses bonnes actions tout en le punissant pour ses mauvaises actions. Comme un humain, l'ordinateur va fuir les actions qui sont susceptibles de lui amener une punition et préférer des actions susceptibles de lui procurer une grande récompense. Cette idée, simple à vulgariser, extrêmement performante sur des jeux simples, a beaucoup plu et a été développée par la suite de nombreuses fois. Alpha Go, l'ordinateur célèbre pour avoir battu 4-1 en 2016 Lee Sedol (considéré comme le meilleur joueur de Go du monde de 2000 à 2010) a également été développé par DeepMind avec un apprentissage par renforcement. Bref, la réputation du reinforcement learning n'est plus à faire tant les résultats qui ont été obtenus sont spectaculaires.

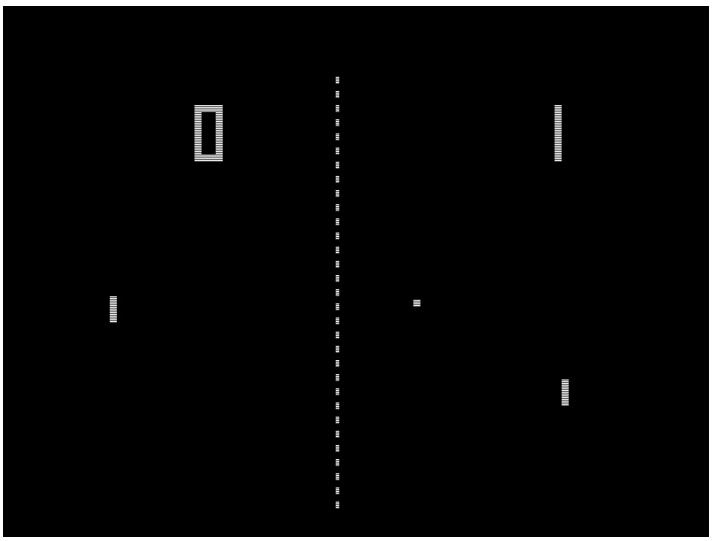
Formalisation théorique

L'apprentissage par renforcement ne s'oppose pas à l'apprentissage supervisé ou non supervisé : au contraire, il se surajoute, se superpose. C'est une manière de transformer un prédicteur statique en un prédicteur séquentiel. Les algorithmes classiques (*random forest*, réseau de neurones, ...) sont à l'origine statiques et cette staticité pose problème dans un grand nombre de cas, surtout lorsqu'on cherche à travailler sur des comportements essentiellement adaptatifs (les comportements humains typiquement). Disons quelques mots du cadre général dans lequel on s'inscrit lorsqu'on développe une méthode de reinforcement learning. Dans ce cadre théorique, on trouve :

- Un **agent** : c'est celui qui prend les décisions et dont on veut optimiser le processus de décision. Schématiquement, c'est l'humain que le système se propose de remplacer ;
- Un **environnement** : c'est le cadre dans lequel l'agent évolue, qui contraint les possibilités d'action et de perception ;
- Des **états**, que l'agent est capable de percevoir. Ce sont les indicateurs qu'il va rapprocher de ce qu'il "connaît" (ou non), en fonction desquels la meilleure décision va être choisie ;
- Des **actions** par lesquelles l'agent interagit avec son environnement ;

- Une fonction de **récompense** cruciale car c'est elle qui va conduire l'agent à améliorer ses actions par la suite. Mieux la fonction de récompense est construite, plus l'agent va apprendre vite de ses erreurs ;
- Une **politique** qui dicte la façon de sélectionner l'action selon l'état de l'environnement.

Exemple : Pong (Atari)



Comment entraîner un ordinateur à jouer à Pong ? En utilisant le reinforcement learning : **l'agent** (la **raquette de gauche**) observe un **environnement** (**une image constituée de pixels noirs et de pixels blancs – ou plus mathématiquement un tableau de 0 et de 1**) dont il analyse des **états** (**position du seul pixel blanc** en dehors des raquettes) et décide ensuite d'une **action (aller d'un pixel en haut ou en bas ou ne pas bouger**) pour maximiser sa **récompense** (**opposé de la distance entre la raquette et la balle par exemple** – plus cette récompense est élevée, donc plus la distance est faible, mieux c'est pour le joueur).

Le but, dans ce cadre, est de trouver la politique qui va maximiser l'ensemble des récompenses reçues. On voit d'ores et déjà les difficultés inhérentes à tout projet de reinforcement learning : définir précisément chacun de ces objets, tordre le problème concret pour le faire entrer dans ce cadre contraignant, définir une fonction de récompense qui valorise justement chaque aspect positif d'une décision tout en pénalisant ses aspects négatifs (par exemple en finance : un trade est-il bon s'il génère un rendement tout en exposant l'investisseur à un risque important ?), ...

Ce cadre théorique a beau être relativement contraignant, il reste très vague. De nombreuses lignes de fracture traversent le reinforcement learning et permettent de séparer différents champs de recherche, différentes méthodes d'optimisation. En voici quelques-unes :

- ***Model-based* ou *model-free*** : cette ligne de partage correspond grossièrement à celle qui sépare les statistiques paramétriques des statistiques non paramétriques. Dans la plupart des cas, on peut voir l'environnement comme une chaîne de Markov sur laquelle l'agent se déplace : discrète ou continue, il est tentant d'en apprendre la probabilité de transition. Si l'on choisit des distributions dont on cherche les meilleurs paramètres, on est dans le cadre *model-based* ; si l'on opte pour une méthode *trial-and-error*, on est dans le cadre *model-free* ;

- ***On-policy* ou *off-policy*** : ici, on se demande si l'algorithme apprend de ses actions effectives ou des actions qu'il aurait pu effectuer. Autrement dit, si l'on est capable, à une date t, de voir l'effet de chaque action sur l'environnement et donc sur la récompense, on peut avoir un apprentissage *off-policy* où l'on apprend de tout ; en revanche, si l'on ne peut voir que l'impact des actions ayant vraiment été effectuées, alors l'apprentissage est nécessairement *on-policy*. On peut penser, pour ce dernier cas, à un médecin qui doit choisir entre plusieurs médicaments pour des patients : on ne peut observer que l'effet du médicament effectivement pris.
- ***Value method* ou *policy method*** : cette distinction sépare les algorithmes qui cherchent à trouver la politique qui va maximiser la somme actualisée des récompenses (optique de long terme) - c'est la *policy method* - des algorithmes qui cherchent à toute date l'action qui va maximiser la récompense suivante - *value method*.

Ceci constitue un bref aperçu de l'idée qui sous-tend tout le domaine du reinforcement learning. Pour aller plus loin, de nombreuses ressources sont disponibles en ligne et une section y est dédiée dans la bibliographie (elle est intitulée "Introduction au reinforcement learning"). Cette section regroupe des articles qui permettent de rapidement cerner le fonctionnement et les enjeux du reinforcement learning.

Fonctionnement du Reinforcement Learning

Pour comprendre le Reinforcement Learning, il est à ce stade impératif de lire l'excellent cours de Nicolas Baskiotis ([9]), qui pose clairement les bases mathématiques du reinforcement learning. A la lecture attentive (à de maintes reprises) de ce cours, on peut faire les remarques suivantes :

- **Comme il a été dit, le reinforcement learning ne remplace pas l'apprentissage supervisé.** Là où l'apprentissage supervisé cherche à prédire ponctuellement un label (par exemple : prédire la probabilité de défaut d'un client en fonction de son emploi, de son revenu annuel, de sa CSP, de son genre, de son âge etc), **le reinforcement learning cherche à optimiser une trajectoire**. Le problème n'est pas ponctuel mais intertemporel et, par exemple, des choix peu judicieux peuvent être faits en route tant que la trajectoire est bonne. Ce qui est maximisé, c'est l'agrégation des récompenses (via une somme, ou une somme actualisée) et non à chaque point la récompense obtenue.
- Le reinforcement learning **arbitre systématiquement un dilemme entre exploitation et exploration**. Schématiquement, l'agent, comme un agent humain, a le choix, face à une situation donnée, entre exploiter les états similaires qu'il a déjà vus et prendre une décision « éclairée », et explorer des actions jusqu'alors jamais entreprises. Ce dilemme peut être arbitré de différentes manières. La **politique gloutonne** (« greedy » dans la littérature, ou déterministe) choisit, à tout instant, l'action qui donne la meilleure récompense espérée ; à l'inverse, la **politique aléatoire** choisit, à tout instant, une action de manière probabiliste (les probabilités étant proportionnelles à la récompense espérée), avec en filigrane l'idée que la politique aléatoire converge vers la politique « greedy » (plus on a vu de situations, plus l'action à choisir est claire et plus les probabilités sont discriminantes). Enfin, un hybride est très pratiqué : la **politique « ε - greedy »** choisit une action aléatoire avec probabilité ε (faible) et une action déterministe avec probabilité (1- ε).
- **Le problème à résoudre est un problème intertemporel markovien** : l'agent effectue des actions séquentiellement et chaque action ne dépend que de l'état actuel. Fondamentalement, l'agent essaie à chaque étape d'optimiser sa récompense future sachant

que cette dernière dépend de son action mais également de l'état que son action va engendrer. En d'autres termes, si V_t est un proxy pour la valeur à maximiser, alors $V_t = r(a_t) + V_{t+1}$ (où a_t est l'action choisie et r la fonction de récompense). On est en présence d'une **équation de Bellman**. Si les probabilités de transition de la chaîne de Markov sont connues, alors cette équation de Bellman a une solution exacte et le reinforcement learning est inutile ; en revanche, si ces probabilités de transition sont inconnues, le reinforcement learning prend tout son sens et la fonction V_t est celle que l'on cherche à optimiser. Elle est une fonction intermédiaire qui répond à la problématique suivante **: résoudre ponctuellement un problème dynamique. Le proxy choisi (la fonction Q, la fonction A dans la littérature) dépend des algorithmes de reinforcement learning**.

- Dans les premiers cas traités (comme les jeux Atari), les espaces des états et des actions sont discrets (la raquette a trois actions possibles : monter, descendre, ne pas bouger ; l'environnement est un tableau à 64 entrées composées de 0 ou de 1). La recherche a évolué pour pouvoir résoudre de plus en plus de problèmes grâce au reinforcement learning, et pour cela elle a relâché cette contrainte spatiale. Les états peuvent être continus (observations de prix de marché en finance, qui peuvent prendre toutes les valeurs réelles positives), les actions aussi (choix de poids d'actions dans un portefeuille, ces poids variant librement entre 0 et 1). **Les types d'actions ou d'états observés (discrets ou continues) sont eux aussi différents selon les algorithmes utilisés**.

Une fois ces lignes de fracture assimilées, on peut rentrer dans le détail et appréhender les différents algorithmes. Voici la liste Wikipédia des principaux algorithmes (avec les caractéristiques décrites ci-dessus) :

Algorithm	Description	Model	Policy	Action Space	State Space	Operator
Monte Carlo	Every visit to Monte Carlo	Model-Free	Off-policy	Discrete	Discrete	Sample-means
Q-learning	State–action–reward–state	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA	State–action–reward–state–action	Model-Free	On-policy	Discrete	Discrete	Q-value
Q-learning - Lambda	State–action–reward–state with eligibility traces	Model-Free	Off-policy	Discrete	Discrete	Q-value
SARSA - Lambda	State–action–reward–state–action with eligibility traces	Model-Free	On-policy	Discrete	Discrete	Q-value
DQN	Deep Q Network	Model-Free	Off-policy	Discrete	Continuous	Q-value
DDPG	Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
A3C	Asynchronous Advantage Actor-Critic Algorithm	Model-Free	On-policy	Continuous	Continuous	Advantage
NAF	Q-Learning with Normalized Advantage Functions	Model-Free	Off-policy	Continuous	Continuous	Advantage
TRPO	Trust Region Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
PPO	Proximal Policy Optimization	Model-Free	On-policy	Continuous	Continuous	Advantage
TD3	Twin Delayed Deep Deterministic Policy Gradient	Model-Free	Off-policy	Continuous	Continuous	Q-value
SAC	Soft Actor-Critic	Model-Free	Off-policy	Continuous	Continuous	Advantage

Pour comprendre dans le détail chacun des algorithmes, on trouvera à cette adresse un descriptif complet.

Cette section n'ayant pour but que d'introduire les fondements théoriques et intuitifs du reinforcement learning, elle se clôt ici et le lecteur curieux est invité à lire attentivement chacun des articles fournis dans le corps du texte et dans la bibliographie : ces derniers ont été choisis pour offrir un panel large et supposément non redondant.

Pratiquer le Reinforcement Learning : Python et Open AI Gym

Les ressources disponibles

Une fois le cadre théorique assimilé, on peut réfléchir à la manière de développer de tels algorithmes. Evidemment, ces derniers sont complexes, demandent de nombreux calculs dans leur exécution**, il est donc exclu de les recoder à la main**. En outre, la qualité des librairies disponibles gratuitement rend superflue toute tentative de recommencer de 0.

Parmi tous les codes que l'on peut trouver en ligne, aucun n'est écrit en SAS ; une librairie est disponible sur R et semble fonctionnelle sur des exemples simples ; **de nombreuses librairies et répertoires Github utilisent le langage Python, aussi c'est ce dernier qu'on préfère utiliser en reinforcement learning**.

Des organisations privées comme Google, Facebook ou DeepMind ont rendu publiques certaines librairies utilisées en interne : c'est le cas de Google Horizon, Facebook Dopamine ou de DeepMind Truffle (TRFL). Ces librairies semblent éprouvées et fonctionnelles mais présentent le désavantage, pour l'amateur, d'être un peu difficiles d'accès. On peut les utiliser dans un second temps, une fois que l'architecture de ces librairies est plus claire pour celui qui a pratiqué. A l'extrême inverse, on trouve des comptes Github comme ceux de Tal Perry (développeur chez Google, fondateur de la plateforme d'annotation de texte par NLP LightTag.io) ou de Denny Britz (ex Google Brain, Stanford, Berkeley) qui permettent à l'amateur de comprendre le fonctionnement de codes de reinforcement learning grâce à des exemples simples et détaillés.

Entre les deux (sans d'ailleurs exclure ni une grande performance ni une simplicité d'accès), on trouve Open AI et sa librairie Gym.

Open AI Gym

Une section entière est dédiée à Open AI Gym, tant cette librairie est devenue le standard en termes de reinforcement learning. Selon Wikipédia,

OpenAI (« AI » pour Artificial Intelligence, ou Intelligence artificielle) est une entreprise à « but lucratif plafonné » en intelligence artificielle, basée à San Francisco. Avant mars 2019, elle était reconnue association à but non lucratif. L'objectif de cette société est de promouvoir et développer une intelligence artificielle à visage humain qui bénéficiera à toute l'humanité.

Open AI travaille sur des librairies, essentiellement en Python, qui sont accessibles au grand public et très bien documentées. Fondamentalement, la logique d'appropriation par l'utilisateur suit un format « Plug and play » : l'utilisateur donne un minimum d'information requis par la librairie, le reste se branche automatiquement. Dit autrement, l'utilisateur fournit les informations sur l'environnement, l'agent, la récompense et la politique, le reste est standard et s'adapte aux définitions de l'utilisateur. Ce fonctionnement est très pratique car il permet d'utiliser un grand nombre d'algorithmes de manière simple, sans les recoder, donc en profitant de leur programmation efficace.

Open AI offre de nombreuses possibilités. **Pour commencer, elle offre de quoi comprendre le reinforcement learning avec des exemples simples**. De nombreux jeux Atari ont été utilisés pour faire des exemples d'utilisation de Gym (la librairie d'Open AI dédiée au reinforcement learning). Chaque jeu correspond à un « environnement » (terminologie dédiée) qui est accessible via une API. On trouve à cette adresse un exemple basique d'utilisation de Gym.

Dans le même temps, Open AI a construit une librairie qui s'appelle baselines et qui implémente de nombreux algorithmes de reinforcement learning (A2C, ACER, ACKTR, DDPG, DQN, GAIL, PPO2, SAC, TD3, TRPO, …). **On trouve à cette adresse une documentation excellente et très complète, qui guide l'amateur pas à pas de l'installation des environnements existants à la création d'environnements spécifiques**. C'est cette documentation qu'il faut suivre scrupuleusement et étudier de près pour commencer à entrer dans la pratique de reinforcement learning. Une fois un environnement créé sur mesure pour répondre à un problème donné, l'ensemble des algorithmes cités peuvent être « pluggés » et utilisés sans difficulté. De nombreux avantages peuvent être mentionnés comme la grande rapidité calculatoire et l'utilisation facilité de TensorBoard (l'outil de visualisation de TensorFlow).

En pratique, on souhaite résoudre un problème spécifique à l'aide du reinforcement learning. Pour utiliser Open AI Gym à des fins spécifiques, il convient de définir 4 fonctions nommées par Gym. En quelque sorte, **il s'agit d'un code à trous que l'utilisateur remplit**. Pour comprendre comment remplir ce code à trou, on peut suivre ce guide. En quelques mots, ces quatre fonctions sont :

__init__ : cette fonction (ou « méthode ») définit l'environnement, la manière d'interagir avec l'agent, l'espace d'actions (discrètes ou continues, bornes, …), l'espace des observations (idem) et d'autres items que l'utilisateur veut utiliser par la suite. Pour faire simple, elle contient tout ce qui est spécifique au problème de l'utilisateur ;

step : cette fonction implémente un pas temporel dans l'environnement en fonction d'une action. Elle renvoie l'observation de l'étape suivante, la récompense induite par l'action, un booléen qui donne l'information de fin de trajectoire et un fichier de format JSON qui permet à l'utilisateur d'avoir des informations en temps réel sur la performance de l'algorithme ;

reset : cette fonction porte un nom clair, elle renvoie l'environnement tel qu'au début de la trajectoire. Dans la mesure où le reinforcement learning optimise une trajectoire d'agent au sein d'un environnement, la fonction « reset » permet simplement de recommencer une trajectoire simplement. Elle renvoie la première observation ;

render (optionnelle) : enfin, la fonction « render » renvoie quoi que ce soit qui permette d'afficher la trajectoire en temps réel. Pour un jeu, cela permet d'observer le comportement de l'agent au cours du jeu, du début à la fin. Cela peut être ignoré pour des questions de gain de temps.

Evidemment, pour que tout fonctionne correctement, il convient que les quatre méthodes citées ci-dessus respectent des **conditions de format** (sur les arguments et sur ce que les méthodes renvoient en sortie) et de nom. Le gros du travail consiste donc d'une part à **formaliser un environnement qui bien souvent est plus complexe que le cadre simplifié du reinforcement learning**, d'autre part à **écrire une fonction de récompense qui encapsule parfaitement l'ensemble du comportement recherché pour l'agent, but recherché mais également effets pervers compris.**

Il a été écrit ci-dessus que Gym est devenu un standard en termes de reinforcement learning. Ceci est vrai dans la mesure où de nombreux codes que l'on peut trouver en ligne, qui explorent l'une ou l'autre facette du reinforcement learning, ont comme prérequis l'emploi d'une classe de type Gym, avec ses méthodes, ses attributs, ses propriétés. Par conséquent quiconque veut utiliser ces codes est contraint de créer un environnement Gym au départ, ce qui peut être fastidieux pour celui qui a utilisé autre chose. Le parti pris peut donc être d'utiliser Gym dès le départ pour simplifier le processus d'exploration.

Ceci clôt cette partie « pratique », encore théorique. Le lecteur est encouragé à ouvrir chacun des liens cités dans le corps du texte (contrairement à un article de recherche, ces références sont mises en évidence non pas pour légitimer le propos de l'auteur, mais pour le compléter) et à lire la seconde documentation, qui rend compte du projet de reinforcement learning qui a été mené chez Aequam Capital au sujet de l'optimisation d'allocation d'actifs, un sujet très pertinent par sa dimension d'optimisation temporelle. Le lecteur y trouvera un exemple détaillé d'implémentation de reinforcement learning, de la formalisation théorique d'un problème pratique dans la structure du reinforcement learning à l'utilisation effective des librairies suscitées.

Bibliographie

Introduction au reinforcement learning

1. Mnih et al., *Playing Atari with Deep Reinforcement Learning*, 2017, DeepMind
2. TowardsDataScience.com, *Introduction to Various Reinforcement Learning Algorithms*. Part I (Q-Learning, SARSA, DQN, DDPG)
3. LearnDataSci.com, *Reinforcement Q-Learning from Scratch in Python with OpenAIGym*
4. Csaba Szepesvari, *Algorithms for Reinforcement Learning*, 2009, Morgan & ClaypoolPublishers
5. Tal Perry for Medium.com, *I Went To The German Alps and Applied ReinforcementLearning To Financial Portfolio Optimization*
6. Gordon Ritter, *The Usefulness of Reinforcement Learning in Finance*, 2018, NewYork University, Rutgers University, Baruch College
7. Skymind.ai, *A Beginner's Guide to Deep Reinforcement Learning*
8. Louiskirsch.com, *A Map of Reinforcement Learning*
9. Nicolas Baskiotis, *Apprentissage par renforcement*, 2018, Laboratoire d'Informatique de Paris VI (LIP6), Sorbonne Université
10. Andrej Karpathy blog, *Deep Reinforcement Learning : Pong from Pixels*