Utilisateur:Agerussi/Brouillon

L'**apprentissage par renforcement hors ligne** (ou **batch**) est un cas particulier de l'apprentissage par renforcement, qui est une classe de problèmes d'apprentissage automatique dont l'objectif est de déterminer à partir d'expériences une stratégie (ou politique) permettant à un agent de maximiser une récompense numérique au cours du temps.

Dans le cadre de l'apprentissage par renforcement purement hors ligne, l'agent ne peut pas interagir avec l'environnement : une base d'apprentissage lui est fournie au départ et il l'exploite pour apprendre une politique. Contrastant avec les algorithmes en-ligne, où l'agent à la possibilité d'interagir comme bon lui semble avec l'environnement, les algorithmes hors-ligne tentent d'exploiter au maximum les exemples d'apprentissage dont ils disposent, sans compter uniquement sur la possibilité d'exploration. Cette approche est donc particulièrement avantageuse quand il n'est pas possible d'effectuer des expériences ou lorsque ces expériences sont coûteuses (casse de matériel possible, obligation d'avoir recours à une assistance humaine pendant les expériences, etc). En général cependant, les techniques d'apprentissage par renforcement batch peuvent être utilisées dans un cadre plus large, où la base d'apprentissage peut évoluer au cours du temps. L'agent peut alors alterner entre des phases d'exploration et des phases d'apprentissage. Les algorithmes hors-ligne sont en général des adaptations d'autres algorithmes comme le Q-Learning, eux-mêmes inspirés par les algorithmes de programmation dynamique résolvant les MDPs.

1 Classification des batchs

Définir précisément ce qu'est ou n'est pas un algorithme hors-ligne n'est pas forcément évident. Une des positions que l'on peut adopter est de considérer “hors-ligne” tout algorithme qui n'est pas purement en-ligne. Ceci amène alors à distinguer trois types de méthodes batch :

- Sans interaction sur l'environnement et en prenant toutes les expériences disponibles : *pure batch*.

Le principe du batch pure est d'être complètement hors ligne. C'est à dire qu'il prend une fois les expériences et apprend sa politique optimale sur ce jeu uniquement. Si une nouvelle expérience arrive, elle ne peut pas être incluse toute seule, il faut l'inclure dans l'ensemble des expériences précédentes et réapprendre sur ce nouvel ensemble.

- Avec interaction sur l'environnement et en prenant toutes les expériences disponibles : *growing batch*.

Dans la situation de pure-batch il peut être difficile de trouver une politique optimale si l'ensemble des expériences suit une distribution qui n'est pas celle du système réel sous-jacent. Cela peut se traduire par des représentations de probabilités de transitions erronées ou des transitions manquantes. La meilleure solution pour palier à ce manque d'information est donc d'interagir avec le système quand c'est possible et d'utiliser à chaque fois l'ensemble de l'information disponible. C'est le principe d'exploration (effectué par l'interaction) et d'exploitation (effectué par les expériences) que l'on retrouve dans les méthodes classique d'apprentissage par renforcement.

- Avec interaction sur l'environnement et en ne prenant pas toutes les expériences disponibles : *semi-batch*.

Le semi-batch se situe entre les deux précédentes. Le système met à jour plusieurs expériences en même temps ce qui fait que le semi-batch n'est pas purement en ligne mais il n'utilise qu'une seule fois les expériences, ce qui fait qu'il n'est pas complètement hors-ligne.

2 Modèle théorique sous-jacent

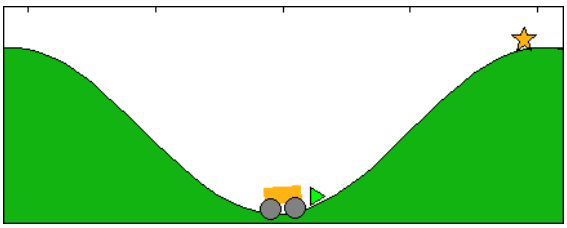
La modélisation classique de l'apprentissage par renforcement (en ligne ou hors ligne) utilise la notion sous-jacente de processus de décision markovien (MDP). Rappelons-en les principales notations :

- le MDP est noté $\{S, A, T, R\}$;
- la fonction de valeurs des états V ;
- celle des valeurs des états-actions Q ;
- les politiques sont notées π .

3 Exemples

3.1 Mountain Car

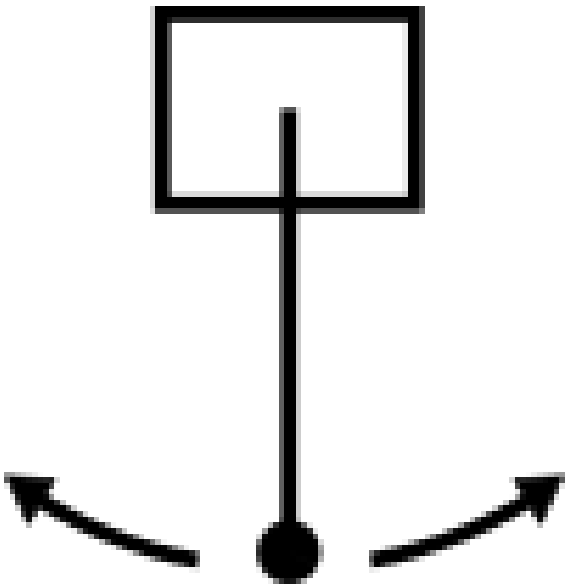
Le Moutain car est un problème classique de la littérature de l'apprentissage par renforcement, dans lequel une voiture doit atteindre le haut d'une colline.



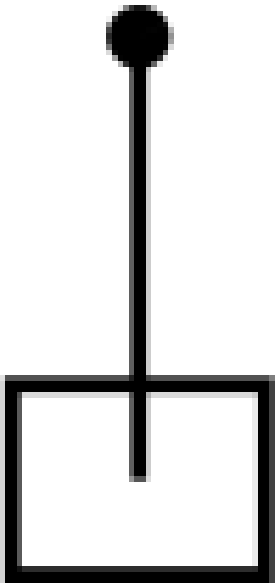
Problem du Mountain car

- Les états sont décrits par la position de la voiture ($p \in [-1.2, 0.6]$) et sa vitesse ($s \in [-0.07, 0.07]$) ;
- Les actions sont les accélérations possibles de la voiture : accélérer vers l'avant, vers arrière, ou rester en roue libre.
- La fonction de transition est obtenue en effectuant la somme des forces qui s'applique à la voiture ;
- La fonction de récompense vaut $hauteur - 1$

3.2 Pendule inversé



État initial



État d'équilibre

Ce problème est un autre exemple classique, dans lequel un agent doit contrôler une perche munie d'un poids à l'aide d'un moteur pour la placer en équilibre en position verticale, avec le poids en haut. Le moteur n'étant pas assez puissant pour faire passer de la perche de sa position initiale à la position d'équilibre d'un coup, l'agent doit apprendre à faire balancer la perche.

- Les états sont la position de la perche et sa vitesse ;
- Trois actions sont possibles : faire tourner le moteur dans les deux sens, ou arrêter le moteur ;
- La fonction de transition est inconnue, ses valeurs sont obtenues en observant la perche ;
- La fonction de récompense vaut 0 si la perche est à la position d'équilibre (avec une marge de 0.5°) et -1 sinon.

4 Les techniques batch

4.1 Généralités

Il existe une grande variété d'algorithmes hors-ligne dans la littérature, mais de manière générale ils brodent tous

autour de quelques principes fondamentaux qui les distinguent des algorithmes purement en-ligne ou de ceux dans lesquels le MDP est connu. Initialement pensés pour résoudre les difficultés spécifiques à l'apprentissage hors-ligne, ces principes sont également avantageux dans un cadre plus général.

4.1.1 Experience Replay

Pour un Q-learning purement en ligne, l'agent est libre d'explorer et de générer de l'expérience comme bon lui semble. Ces expériences sont utiles par deux aspects :

1. effectivement explorer l'environnement, c'est-à-dire expérimenter de nouvelles transitions vers des états non encore visités et récolter de nouvelles récompenses ;

2. faire converger l'algorithme, car celui-ci demande un grand nombre d'itérations pour propager les informations et finalement se stabiliser vers une politique optimale. Dans la situation hors ligne, on ne peut se contenter de n'utiliser qu'une seule fois une expérience donnée car le nombre d'expériences est limité par définition, et ne permettrait pas de faire converger l'algorithme.

Cette difficulté est levée en “rejouant” plusieurs fois les mêmes expériences, c'est-à-dire que le jeu de données disponible est exploité encore et encore comme s'il s'agissait de nouvelles expériences.

4.1.2 Fitting

Dans les méthodes classiques d'apprentissage par renforcement, il est commun de faire la mise à jour de la fonction de valeur de manière désynchronisée. C'est à dire qu'après chaque observation d'une transition, la valeur d'un état ou d'un état-action est mise à jour localement et les autres états sont laissés inchangés. Ensuite, ce n'est que lors de nouvelles interactions que les autres états seront évalués sur la base de la valeur mise à jour précédemment.

Les algorithmes batch, dans le but d'exploiter au maximum les données disponibles, de s'adapter à des espaces d'états continus et de contrer certains problèmes d'instabilités lors de l'apprentissage (cas de non convergence), ne se contentent pas de stocker simplement les valeurs pour chaque état ou action et d'appliquer des mises à jour de type programmation dynamique. Ils ajoutent au dessus une forme ou une autre de globalisation et de régularisation des mises à jour qui peut se voir comme un apprentissage supervisé des fonctions de valeurs.

Par exemple, dans les algorithmes que nous présentons ci-dessous, on trouve l'utilisation d'un noyau gaussien qui permet de moyenner localement les valeurs, d'une base de fonctions dans laquelle on exprime la fonction de valeurs, ou plus explicitement d'un algorithme d'apprentissage supervisé (réseau de neurones, k-plus-proches-voisins...) pour apprendre la fonction de valeurs.

De cette manière chaque expérience utilisée va permettre de mettre à jour globalement l'ensemble de la fonction de valeurs (des états ou états-actions) sans nécessiter d'attendre que de multiples itérations propagent l'information de manière plus ou moins aléatoire.

4.2 Présentation de quelques-uns des algorithmes principaux

4.2.1 Kernel-Based Approximate Dynamic Programming

Principe général L'algorithme Kernel-Based Approximate Dynamic Programming (KADP) suppose qu'un ensemble de transitions observées de façon arbitraire $(s, a, r, s') \in F$ est à sa disposition. Il suppose ensuite que l'espace des états est muni d'une distance, à partir de laquelle la notion de moyennage basé sur un noyau Gaussien prend sens. Ce noyau est utilisé pour mettre à jour itérativement la fonction de valeurs V en *tous* les états, à partir du jeu d'échantillons F . On voit bien ici l'application de l'idée d'experience replay (on utilise répétitivement toutes les données) et celle de fitting puisque le noyau permet à l'ensemble des valeurs d'être mise à jour par chaque transition.

Algorithme L'algorithme démarre d'une fonction de valeurs des états $\hat{V}_0$ arbitrairement choisie puis chaque itération du KADP consiste en deux phases :

Phase 1 La première phase évalue la mise à jour du processus décisionnel :

$\hat{Q}_a^{i+1}(\sigma) := \sum_{(s,a,r,s')\in F_a} k(s,\sigma)[r + \gamma \hat{V}^i(s')]$

avec F_a un sous ensemble de F qui n'utilise que les actions a.

Cette équation calcule la moyenne pondérée de la mise à jour habituelle : $r + \gamma \hat{V}^i(s')$ avec comme pondération $k(s, \sigma)$. Le noyau est choisi de telle sorte que les transitions les plus éloignées aient une moins grande influence que les plus proches.

Phase 2 La seconde phase consiste à calculer les nouvelles valeurs des états selon la politique gloutonne :

$\hat{V}^{i+1}(s) = \max_{a\in A} \hat{Q}_a^{i+1}(s)$

L'algorithme va donc itérer sur les équations définies au dessus pour évaluer $\hat{V}^i$ avec $i = 1, 2, \ldots$ et espérer converger vers une unique solution $\hat{V}$.

Exemple d'application Le problème du choix du portfolio optimal décrit dans [1] est un cas pratique du KADP. C'est un problème d'investissement financier où l'agent décide d'acheter ou non des actions. Le but étant à la fin de la simulation de maximiser le gain de l'agent.

Sa représentation comme un MDP est la suivante :

- s_t l'état symbolisant la valeur de l'action à un instant t
- a_t l'action représentant l'investissement de l'agent dans l'actions avec $A \equiv 0, 0.1, 0.2, \ldots, 1$
- La richesse de l'agent au temps t est W_t et sa richesse au temps $t+1$ est $W_{t+1} = (1 + a_t \frac{s_{t+1} - s_t}{s_t})W_t$

L'agent va donc au cours du temps investir (ou pas) dans une action pour maximiser sa richesse avec comme but d'être le plus riche possible à la fin du temps maximum défini. Pour rendre le problème le plus réaliste possible, l'agent adopte un comportement tel qu'il a peur de perdre ses gains autant qu'il a l'envie d'en gagner.

4.2.2 Fitted Q iteration (FQI)

Cet algorithme permet de calculer une approximation de la fonction de valeurs optimale des états-actions Q à partir d'un jeu d'échantillons du MDP. Connaissant Q , il est alors possible de déduire la politique correspondante : $\pi(s) = \arg\max_{a \in A} Q(s, a)$. Il repose sur le même principe que l'algorithme d'itération sur la valeur des états-actions, qui consiste à calculer de manière itérative le point fixe de l'équation de Bellman :

$Q^{i+1}(s, a) = \sum_{s' \in S}[R(s, a, s') + \gamma \max_{a' \in A} Q^i(s', a')]T(s, a, s')$

On cherche à calculer ce point fixe à partir d'un jeu d'échantillons $\mathcal{F}$ (un ensemble de transitions de la forme (s, a, r, s')). L'algorithme procède de la manière suivante :

1. Il crée une base de donnée d'apprentissage P constituée des couples entrée-sortie $(s, a) \rightarrow r + \gamma \max_{a' \in A} \hat{Q}^i(s, a)$ pour chaque transition $(s, a, r, s') \in \mathcal{F}$. La valeur ainsi calculée pour chaque (s, a) est en quelque sorte la « meilleure estimation » que l'on puisse faire de la valeur de $Q(s, a)$ dans l'état actuel des connaissances décrit par Q^i .
2. Il en déduit une fonction $\hat{Q}^{i+1}$ à partir de l'ensemble P en utilisant une méthode d'apprentissage supervisé. Cette fonction est donc une approximation de la fonction Q^{i+1} obtenue à la i+1ème itération de l'algorithme d'itération sur la valeur des états-actions.
3. Il incrémente i et retourne à la première étape tant qu'une condition d'arrêt n'est pas vérifiée.

Cet algorithme peut-être utilisé avec n'importe quelle méthode d'apprentissage supervisé, en particulier des méthodes non paramétriques, qui ne font aucune hypothèse sur la forme de Q . Cela permet d'obtenir des approximations précises indépendamment de la taille des espaces d'états et d'actions, y compris dans le cas d'espaces continus.

On peut consulter [2] pour une mise en oeuvre dans laquelle la méthode d'apprentissage est une régression régularisée moindres carrés ; un réseau de neurones est utilisé dans [3] ; enfin une approche basée sur différents types d'arbres de régression est analysée dans [4].

De manière générale, la convergence de l'algorithme n'a été démontrée que pour une certaine classe de méthodes d'apprentissage. Différentes conditions d'arrêt peuvent-être utilisées ; en pratique, fixer le nombre d'itérations permet d'assurer la terminaison de l'algorithme, et ce quelque soit la méthode d'apprentissage utilisée.

Dans [4], plusieurs problèmes ont été utilisés pour évaluer l'algorithme avec différentes méthodes d'apprentissage supervisé, et avec plusieurs types de batch (pure batch et growing batch). Sur ces problèmes, l'algorithme de fitted Q iteration est très performant, en particulier en utilisant des méthodes d'apprentissage non paramétriques (comme la méthode des k plus proches voisins).

4.2.3 Itération moindres carrés de la politique (LSPI : Least-Squares Policy Iteration)

Bien adapté à l'apprentissage pûrement hors-ligne, LSPI est une approche reconnue pour exploiter efficacement un jeu de données fixe pré-existant ([5], [6]).

Description de l'algorithme Cet algorithme est une adaptation de l'algorithme d'itération de la politique exprimé sur la fonction Q d'états-actions. Il se focalise exclusivement sur l'évaluation de Q , et la politique n'y est jamais représentée explicitement, mais simplement déduite à la volée, par le choix glouton classique :

$$\pi(s) = \arg\max_{a \in A} Q(s, a)$$

Par rapport à PI, LSPI opère deux grands changements :

1. la fonction Q est exprimée comme une combinaison linéaire de fonctions ϕ_i , $i = 1, \ldots, k$:

$$Q(s, a) = \sum_{i=1}^{k} w_i \phi_i(s, a).$$

Les fonctions ϕ_i forment donc une base d'un sous-espace $\mathcal{Q}$ de l'espace des fonctions états-actions, choisie une fois pour toutes au départ. Ainsi en interne, la fonction Q est simplement représentée par les k coefficients w_i .

Ce modèle possède un avantage : quels que soient le nombre d'états et d'actions, seuls k coefficients sont utilisés pour représenter l'ensemble des valeurs de Q , contre

$|S| \times |A|$ si l'on utilisait une représentation tabulaire classique. Ceci permet de traiter efficacement des MDP possédant un grand nombre d'états et/ou d'actions. On peut même, dans ce formalisme, considérer des espaces d'états ou d'actions continus.

L'inconvénient, en revanche, est que l'on est maintenant restreint aux fonctions de $\mathcal{Q}$. Dans l'algorithme PI avec états-valeurs, la phase d'évaluation de la politique π consiste à déterminer le point fixe Q^π de l'opérateur

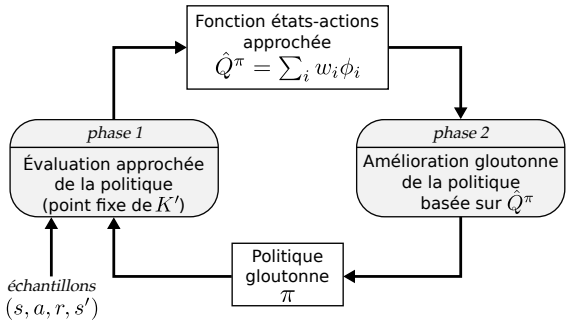
$$K(Q)(s,a) = \sum_{s' \in S} [R(s,a,s') + \gamma Q(s', \pi(s'))] T(s,a,s'),$$

soit par la résolution d'un système linéaire, soit par calcul itératif sur K .

Or même si par définition $Q \in \mathcal{Q}$, on n'a pas en général $K(Q) \in \mathcal{Q}$. L'idée dans LSPI est alors de projeter (au sens des moindres carrés, d'où le nom de l'algorithme) la mise à jour $K(Q)$ sur $\mathcal{Q}$:

$$K'(Q) = \mathcal{P}_{\mathcal{Q}}^{\perp}(K(Q)),$$

et donc d'obtenir une approximation $\hat{Q}^\pi$ de Q^π qui est stable par la règle de mise à jour suivie de la projection. Dans [7] ce point fixe est obtenu par résolution d'un système linéaire.



L'algorithme général d'itération de la politique, avec projection moindres carrés.

2. La deuxième différence est que LSPI ne suppose pas le MDP connu ; l'algorithme se base uniquement sur un jeu d'échantillons du MDP, donnés sous la forme de quadruplets (s,a,r,s') . Ces échantillons peuvent être donnés directement au départ de l'algorithme (mode pure-batch) ou progressivement (mode semi-batch ou en-ligne). Concrètement, l'algorithme se contente d'effectuer les mises à jour précédentes sur le jeu d'échantillon, éventuellement de façon répétée pour provoquer la convergence (« experience replay »). On peut montrer qu'en fait cela revient à appliquer la mise à jour sur le véritable MDP, mais avec les probabilités de transitions $T(s,a,s')$ obtenues empiriquement selon la distribution des échantillons. Ainsi, dans l'hypothèse où cette distribution est conforme aux véritables probabilités de transition du MDP sous-jacent, le résultat converge asymptotiquement vers la véritable valeur Q^π , et donc finalement vers la fonction optimale Q^* . On peut également remarquer que chaque échantillon contribue linéairement à chaque itération (il ajoute un terme dans la somme définissant K et l'opérateur de projection est linéaire), ce qui permet de mettre en place des optimisations lorsque les échantillons arrivent progressivement.

La figure ci-contre résume schématiquement l'algorithme LSPI.

Exemple d'application Dans [7], plusieurs applications de l'algorithme LSPI sont proposées. L'une d'elles, dans lequel l'algorithme est particulièrement performant notamment par rapport à un algorithme de Q-Learning, est le problème de l'*équilibre et la conduite d'un vélo* [8].

Dans ce problème l'apprenant connaît à chaque pas de temps l'angle et la vitesse angulaire du guidon, ainsi que l'angle, la vitesse et accélération angulaire de l'angle que forme le vélo avec la verticale. Il doit alors agir quelle force rotatoire appliquer au guidon (choisie parmi trois possibilités), ainsi que le déplacement de son centre de masse par rapport au plan du vélo (choisie également parmi 3 possibilités), de manière d'une part à rester en équilibre (c'est-à-dire ici ne pas dépasser un angle vélo/sol de +/- 12°) et d'autre part atteindre une destination cible.

La mise en œuvre est purement hors-ligne et consiste à observer le comportement d'un agent aléatoire pour collecter de l'ordre de quelques dizaines de milliers d'échantillons sous forme de trajectoires. Les trajectoires collectées sont ensuite coupées après quelques pas de temps, pour ne garder que la partie intéressante avant qu'elles ne rentrent dans un scénario de chute inexorable. Les récompenses sont en lien avec la distance atteinte par rapport à l'objectif. Quelques passes de l'algorithme sont ensuite effectuées sur ces échantillons (« experience replay ») et une convergence très rapide vers des stratégies viables est observée.

5 Bibliographie générale

- (en) Sascha Lange, Thomas Gabel, and Martin Riedmiller, *Batch Reinforcement Learning*, dans Marco Wiering, Martijn van Otterlo éd., *Reinforcement Learning : State-of-the-Art*, Berlin Heidelberg, Springer-Verlag, coll. « Adaptation, Learning and Optimization, vol. 12 », 2012 (ISBN 978-3-642-27644-6)

6 Références

[1] (en) Dirk Ormoneit and Peter Glynn, « Kernel-Based Reinforcement Learning in Average-Cost Problems : An application to Optimal Portfolio Choice », Advances in

Neural Information Processing Systems, 2000, p. 1068-1074

[2] (en) Farahmand Amirmassoud, Ghavamzadeh Mohammad, Szepesvári Csaba, Mannor Shie, *Regularized Fitted Q-Iteration : Application to Planning*, dans Girgin Sertan, Loth Manuel, Munos Rémi, Preux Philippe, Ryabko Daniil, *Recent Advances in Reinforcement Learning*, Berlin Heidelberg, Springer, col. Lecture Notes in Computer Science, 2008, (ISBN 978-3-540-89721-7) p55-68

[3] (en)Martin Riedmiller, *Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method*, dans Gama João, Camacho Rui, Brazdil PavelB, Jorge AlípioMário, Torgo, Luís éd., *Machine Learning : ECML 2005*, Springer Berlin Heidelberg, Lecture Notes in Computer Science vol. 3720, 2005, (ISBN 978-3-540-29243-2) pp 317-328.

[4] (en) Damien Ernst, Pierre Geurts, Louis Wehenkel, « Tree-Based Batch Mode Reinforcement Learning », Journal of Machine Learning Research 6, 2005, p. 503–556

[5] (en) Christophe Thiery and Bruno Scherrer, *Least-Squares λ Policy Iteration : Bias-Variance Trade-off in Control Problems*, International Conference on Machine Learning, Haifa, Israel, 2010, [lire en ligne]

[6] (en) Li L., Williams J. D., and Balakrishnan S, *Reinforcement learning for dialog management using least-squares Policy iteration and fast feature selection*. In INTERSPEECH, 2009, pp. 2475-2478.

[7] (en) Michail Lagoudakis, Ronald Parr, « Least-Squares Policy Iteration », Journal of Machine Learning Research 4, 2003, p. 1107-1149

[8] (en) Jette Randløv and Preben Alstrøm, « Learning to Drive a Bicycle Using Reinforcement Learning and Shaping », dans Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98), 1998, Jude W. Shavlik (Ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 463-471.

7 Sources, contributeurs et licences du texte et de l'image

7.1 Texte

- **Utilisateur:Agerussi/Brouillon** *Source :* https://fr.wikipedia.org/wiki/Utilisateur%3AAgerussi/Brouillon?oldid=121250593 *Contributeurs :* Agerussi, Lseguinot et Anonyme : 3

7.2 Images

- **Fichier:Least_Squares_Policy_Iteration.svg** *Source :* https://upload.wikimedia.org/wikipedia/commons/d/da/Least_Squares_Policy_Iteration.svg *Licence :* CC BY-SA 4.0 *Contributeurs :* Travail personnel *Artiste d'origine :* Agerussi
- **Fichier:Mcar.png** *Source :* https://upload.wikimedia.org/wikipedia/commons/1/18/Mcar.png *Licence :* CC0 *Contributeurs :* https://en.wikipedia.org/wiki/File:Mcar.png *Artiste d'origine :* BoonTheMoon
- **Fichier:Pole-swing-balanced.png** *Source :* https://upload.wikimedia.org/wikipedia/commons/e/e1/Pole-swing-balanced.png *Licence :* CC0 *Contributeurs :* Travail personnel *Artiste d'origine :* Lseguinot
- **Fichier:Pole-swing-initial.png** *Source :* https://upload.wikimedia.org/wikipedia/commons/b/b6/Pole-swing-initial.png *Licence :* CC0 *Contributeurs :* Travail personnel *Artiste d'origine :* Lseguinot

7.3 Licence du contenu