Emmanuel Franck

Année 2019-2020
Mathématiques Appliquées L3

Informatique S6 Projet: Apprentissage par renforcement, première partie

1 Introduction

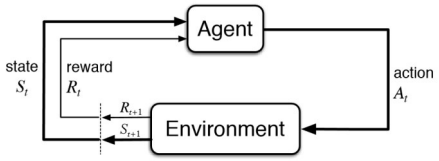
Ce projet se place dans le contexte de l'apprentissage par renforcement. Il s'agit d'une des grandes familles d'apprentissage largement utilisée en robotique ou en IA pour les jeux comme les échecs. Il s'agit d'un environnement ou un agent va effectuer une série d'action au cours du temps. L'agent a un temps t est dans un état $s_t \in S$ et peut effectuer une action $a_t \in A$. Cette action lui permet de se retrouver dans un nouvel état s_{t+1} et il obtient une récompense r_{t+1} qui va évaluer l'action et/ou le nouvel état. Ce nouvel état ainsi que la récompense est donné par l'**environnement**. Le processus continu jusqu'à ce qu'un état terminal soit atteint. L'objectif est de trouver la meilleure série d'action effectuée afin de maximiser les récompenses. On cherche donc ici une fonction appelée **politique** :

$$\mu(s) : S \to A$$

qui a un état renvoie une action. L'enjeu ici sera de construire la meilleure politique μ tel qu'on maximise les gains futurs :

$$\sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

avec $\gamma < 1$. Le principe est détaillé sur la figure suivante On se placera ici dans un cas



où s et a peuvent prendre un nombre fini et faible de valeur. Dans ce cas μ est vecteur de taille le nombre d'état possible (noté n_s). Dans cette première partie on va construire des classes formalisant l'agent (on parle de processus de décision markovien) puis dans les parties suivantes on proposera des méthodes pour calculer μ.

2 Processus de Décision Markovien

Objectif (8.75 points) : On va ici construire deux classes de Processus de décision Markovien) puis un exemple.

2.1 MDP

A partir du cours numéro 5.
Objectif (3.25 points) : Construire la classe **MDP**.
Il s'agit d' un template de classe dépendant de deux classes S et A qui correspondront respectivement au type (int ou double) des états et au type (int ou double) des actions.
La classe devra contenir les attributs suivants :

- Deux entiers pour le nombre d'états et d'actions possibles,
- un Pointeur de type "S" qui contiendra le tableau des états possibles,
- un Pointeur de type "A" qui contiendra le tableau des actions possibles,
- 4 pointeurs de fonction :
  - un pointeur nommé "reset" sur les fonctions qui prennent un entier et un pointeur sur S et qui renvoie un état (classe S) ; La fonction permettra de réinitialiser un MDP avec un nouvel état de départ.
  - un pointeur nommé "possible_action" sur les fonctions qui prennent un état S et un pointeur (par référence) sur les actions A et qui renverra un entier ; la fonction permettra de pour un état s construire la liste des actions possibles (modifier par la fonction) et renverra la taille de cette liste.
  - un pointeur nommé "environnement" sur les fonctions qui prennent un état S,un action A, un double et un état S tous les deux par référence. la fonction permettra, pour un état s et une action a de calculer l'état suivant s_p et la récompense r.
  - un pointeur nommé "is_terminal" sur les fonctions qui prennent un état S et un int "n" et qui renvoie un booléen. la fonction permettra, pour un état s et un temps n de savoir si c'est un état terminal.

Elle devra contenir des accesseurs et mutateurs pour l'ensemble des attributs. Pour les pointeurs de fonction un mutateur modifiera le pointeur (donc la fonction) et l'accesseur évaluera la fonction contenu dans le pointeur pour des paramètres donnés.
Vous devrez écrire le constructeur par défaut, le destructeur, le constructeur par copie et surchargé le "=".
surchargez l'opérateur "[]" qui renverra le ième état possible et surchargez l'opérateur "()" qui renverra la ième action possible. -

2.2 MDP parfait

A partir du cours numéro 7.
Dans la classe précédente on ne connaissait pas les probabilités des états suivants ni les récompenses possibles. Ici on propose un sous ensemble de MDP qui connait ces informations. **Objectif** (2.75 points) : Construire le template de classe **MDP**$\_parfait$ qui hérite du template de classe MDP.
La classe devra contenir les attributs supplémentaires suivants :

- un pointeur de pointeur P de type double. il contiendra les probabilités de transitions entre un l'état et les suivants :

$$P(s_{t+1} \mid (s_t, a_t))$$

  La ligne sera l'index du couple (i_s, i_a) avec i_a l'index de l'action et i_s l'index de

l'état. La colonne sera l'index de i_{s_+} et le résultat sera la probabilité de l'état suivant s_{t+1}.

— un pointeur de pointeur R de type double. il contiendra les récompenses entre un l'état et les suivants :

$$R(s_{t+1} \mid (s_t, a_t))$$

La ligne sera l'index du couple (i_s, i_a), la colonne l'index de i_{s_+} et le résultat sera la récompense associée.

Pour cette classe il faudra coder : le constructeur par défaut, par copie, le destructeur et l'opérateur "=" en utilisant l'héritage des constructeurs/destructeur de la classe mère.

Il faudra faire des accesseurs et mutateurs permettant d'accéder ou de modifier les valeurs de P et R connaissant (i_s, i_a, i_{s_+}). Pour finir il faut fournir une méthode "privé" qui renvoie l'index du couple (i_s, i_a) avec i_a l'index de l'action et i_s l'index de l'état.

2.3 MDP parfait exemple

A partir du cours numéro 7.
Objectif (2.75 points) : Construire un exemple de MDP.
Il s'agit d'un exemple simple d'un robot collecteur de déchets.
On a deux états possibles de batterie : "bas" et "haut" et 3 actions possibles : "chercher" un déchet (fortement consommateur d'énergie), "attendre" qu'un déchet soit à portée (faiblement consommateur d'énergie), "recharger" la batterie. Si le robot fait une action de recherche avec peu de batterie. Il tombe en rade et doit recharger en urgence. Ce genre de cas est pénalisé. Pour ce problème les états seront des "int" (0 pour "bas" et 1 pour "haut"). Idem pour les actions ("0" pour chercher, "1" pour attendre et "2" pour recharger).

On implémente d'abord plusieurs fonctions :

— deux fonctions qui construiront les tableaux des états et actions possibles (trivial)

— une fonction qui construit les tableaux P et R avec

$$P(\text{"}bas\text{"} \mid \text{"}bas\text{"}, \text{"}chercher\text{"}) = 0.1, \quad P(\text{"}haut\text{"} \mid \text{"}bas\text{"}, \text{"}chercher\text{"}) = 0.9$$

$$P(\text{"}bas\text{"} \mid \text{"}haut\text{"}, \text{"}chercher\text{"}) = 0.9, \quad P(\text{"}haut\text{"} \mid \text{"}haut\text{"}, \text{"}chercher\text{"}) = 0.1$$

$$P(\text{"}haut\text{"} \mid \text{"}haut\text{"}, \text{"}attendre\text{"}) = 1.0, \quad P(\text{"}bas\text{"} \mid \text{"}bas\text{"}, \text{"}attendre\text{"}) = 1.0$$

$$P(\text{"}haut\text{"} \mid \text{"}bas\text{"}, \text{"}recharger\text{"}) = 1.0$$

le reste des probabilité est donc a 0. Pour les récompenses on a

$$R(\text{"}bas\text{"} \mid \text{"}bas\text{"}, \text{"}chercher\text{"}) = 2, \quad R(\text{"}haut\text{"} \mid \text{"}bas\text{"}, \text{"}chercher\text{"}) = -3$$

$$R(\text{"}bas\text{"} \mid \text{"}haut\text{"}, \text{"}chercher\text{"}) = 2, \quad R(\text{"}haut\text{"} \mid \text{"}haut\text{"}, \text{"}chercher\text{"}) = 2$$

$$R(\text{"}haut\text{"} \mid \text{"}haut\text{"}, \text{"}attendre\text{"}) = 1, \quad R(\text{"}bas\text{"} \mid \text{"}bas\text{"}, \text{"}attendre\text{"}) = 1$$

$$R(\text{"}haut\text{"} \mid \text{"}bas\text{"}, \text{"}recharger\text{"}) = 0$$

— Une fonction "robot_reset" (de même signature que le pointeur de fonction "reset") qui choisit aléatoirement un état.
— Une fonction "robot_possible_action" (de même signature que le pointeur de fonction "possible_action") qui renvoie la liste des actions possibles à partir d'un état.
— Une fonction robot_reset (de même signature que le pointeur de fonction "reset") qui choisit aléatoirement un état.
— Une fonction "robot_environnement" (de même signature que le pointeur de fonction "environnement") qui renvoie l'action et l'état suivant (on utilisera les mêmes proba et récompenses que dans R et P).
— Une fonction "robot_is_terminal" (de même signature que le pointeur de fonction "is_terminal") qui renvoie true si $n > 30$.

Une fois ces fonctions écrite, dans le "main" on va construire un "MDP_parfait" à qui on donne via les mutateurs : les 4 fonctions précédentes dans les pointeurs de fonctions, les tableaux des états et actions possibles (construits par les 1ères fonctions) et les tableaux P et R (construit par les 1ères fonctions).
On construira aussi un "MDP" a qui on donne via les mutateurs : les 4 fonctions précédentes dans les pointeurs de fonctions, les tableaux des états et actions possibles (construits par les 1ères fonctions).

Remarque : ne pas oublier que P et R sont des tableaux 2D et pas 3D et qu'il faudra utiliser une methode "privé" qui renvoie l'index du couple (i_s, i_a) avec i_a l'index de l'action et i_s l'index de l'état.

Emmanuel Franck

Année 2019-2020
Mathématiques Appliquées L3

Informatique S6 Projet: Apprentissage par renforcement, deuxième partie

1 Introduction

On cherche maintenant à déterminer une **politique** :

$$\mu(s) : S \to A$$

qui à un état renvoie une action. L'enjeu ici sera de construire la meilleure politique μ tel qu'on maximise les gains futurs :

$$\sum_{t=0}^{\infty} \gamma^t r_{t+1}$$

avec $\gamma < 1$. On va regarder un premier algorithme appelé "itération sur les valeurs" qui appartient à la classe d'algorithme appelée "Programmation dynamique". Pour évaluer une politique on définit la **fonction valeur** $V^{\mu}(s)$ qui associe un score à une politique partant d'un état s. On va donc chercher la politique qui a le meilleur score. On va donc chercher la politique optimale, celle qui conduit à la fonction valeur optimale.

2 Programmation Dynamique

On va ici construire deux classes. Une implémentant le cas générale et une implémentant l'algorithme. C'est basé sur deux résultats mathématiques que l'on va énoncer maintenant. La **fonction valeur** optimale satisfait : l'équation suivante :

$$V^{opt}(s) = \max_a \left(\sum_{s'} \left(r(s' \mid s, a) + \gamma V^{opt}(s') \right) P(s' \mid s, a) \right), \quad \forall s \in S \tag{1}$$

et la politique optimale est donnée par

$$\mu^{opt}(s) = \text{argmax}_a \left(\sum_{s'} \left(r(s' \mid s, a) + \gamma V^{opt}(s') \right) P(s' \mid s, a) \right)$$

avec V solution de l'équation précédente.

2.1 ProgDym

A partir du cours numéro 7.
Objectif (2.5 points) : Construire la classe générale **ProgDym**, elle sera commune à tous les algorithmes de programmation dynamique. Ces algorithmes permettent de calculer V

à partir d'un modèle parfait.
Il s'agit d' un template de classe dépendant de deux classes S et A qui correspondront respectivement au type (int ou double) des états et au type (int ou double) des actions.
La classe devra contenir les attributs suivants :

- un double : γ
- un Pointeur sur un tableau de type "int" qui contiendra la politique μ. A chaque état d'index i_s elle donnera l'index de la meilleur action i_a
- un Pointeur sur un tableau de type "double" qui contiendra la fonction valeur V. A chaque état d'index i_s elle donnera le score associé.
- un pointeur vers un "modèle parfait" (partie 1). Ici le modèle sera créé en dehors de la classe et on souhaite juste stocker son adresse dans la classe.

Elle devra contenir le constructeur par défaut, par copie et le destructeur. Il faudra aussi un constructeur qui prendra l'adresse d'un objet "modèle parfait" et qui initialise les tableaux en récupérant le nombre d'états possibles du modèle. On ajoutera deux fonctions :

- une fonction qui applique la politique. Elle prendra un état s et renverra l'action a issue de la politique (pour utiliser la politique il faudra passer par les index des états et actions)
- Une fonction qui calcul la politique à partir de la fonction valeur V en utilisant l'expression :

$$\mu(s) = argmax_a \left(\sum_{s'} \left(r(s' \mid s,a) + \gamma V(s') \right) P(s' \mid s,a) \right)$$

- une fonction virtuelle "résolution" qui appliquera l'algorithme de calcul de la fonction valeur optimale. Elle sera implémentée dans la classe fille.

2.2 IterationValeur

A partir du cours numéro 7.
Il s'agit ici de construire un exemple particulier d'algorithme de programmation dynamique. On parle l'itération sur les valeurs pour calculer la fonction valeur optimale. L'équation (1) est une équation non-linéaire sur V. l'alorithme consiste à utiliser une méthode de Picard. On calcul itérativement une sucession de fonctions valeurs V^k avec :

$$V^{k+1}(s) = max_a \left(\sum_{s'} \left(r(s' \mid s,a) + \gamma V^k(s') \right) P(s' \mid s,a) \right), \quad \forall s \in S \qquad (2)$$

On s'arrête quand $\parallel V^{k+1} - V^k \parallel_2 < \epsilon$ or $k > imax$ avec $imax$ et ϵ des paramètres.
Objectif (2.0 points) : Construire le template de classe **IterationValeur** qui hérite du template de classe ProgDym.
La classe devra contenir les attributs supplémentaires suivants :

- $imax$ un entier,
- ϵ un double.

Ensuite vous devrez coder les constructeurs de la classe (les mêmes qu'au dessus) en utilisant justement les constructeurs de la classe mère.

Pour finir il faudra implémenter la fonction "résolution" qui calcul la valeur V^{opt} avec la méthode de point fixe.

Objectif (2.0 points) : dans le "main", on calcule la politique associée à l'exemple de la partie précédente. Il suffira d'appliquer la résolution puis la fonction de calcul de la politique. Pour la tester on codera la boucle en temps suivante :

- On réinitialise l' état s_t avec le reset de l'environnement.
- tant que $t < 100$ on fait :
  - on prend l'état courant s_t, on applique la politique pour avoir l'action,
  - on applique l'environnement pour obtenir l'état suivant s_{t+1} et la récompense,
  - on ajoute la récompense à une somme cumulée,
  - on dit que $s_t = s_{t+1}$,
  - On teste si l'état est terminal et si oui on arrête.

On va comparer ensuite la somme cumulée des récompenses obtenues avec le cas "sans politique". Pour cela, seule la première ligne est modifiée. L'action est tirée aléatoirement parmis les actions possibles associées à l'état (fonction contenue dans le modèle) Commentez le résultat (dans un fichier séparé et individuel) sur les gains obtenus par la politique et par le cas "aléatoire".

Emmanuel Franck

Année 2019-2020
Mathématiques Appliquées L3

Informatique S6 Projet: Apprentissage par renforcement, Troisième partie

1 Introduction

On cherche maintenant à déterminer une **politique** dans le cas d'un MDP non parfait. Les méthodes de programmation dynamique s'appliquaient au MDP parfait uniquement. Ici on ne cherche pas à déterminer la fonction valeur puis la politique mais la Q-fonction puis la politique. Elle est de la forme :

$$Q(s,a), \forall s \in S, a \in A.$$

On a

$$\mu^{opt}(s) = \text{argmax}_a \, Q(s,a).$$

La Q-fonction sera construite par des algorithmes dit "stochastiques" donc probabilistes. On va ici construire deux classes, l'une implémentant le cas général et une autre implémentant un algorithme. On utilisera une politique aléatoire $\pi(s \mid a)$ qui est une loi de probabilité des actions connaissant les états. Cette politique sera utile pour générer les trajectoires.

2 Modèles Stochastiques

2.1 Modèles Stochastiques

A partir du cours numéro 7.
Objectif (4.0 points) : Construire la classe générale `ModelSto`, elle sera commune à tous les algorithmes stochastiques. Il s'agit d'un template de classe dépendant de deux classes S et A, qui correspondront respectivement au type (int ou double) des états et au type (int ou double) des actions.
La classe devra contenir les attributs suivants :

- trois double : γ, ϵ, d
- un pointeur de pointeur de type `double`. Il contiendra la politique aléatoire $\pi(a \mid s)$ sous la forme $\pi(i_s, i_a)$ (qui renvoie la probabilité pour l'état de numéro i_s et d'état i_a). Il stockera connaissant les numéros de l'état et de l'action, la probabilité associée.
- un pointeur sur un tableau de type `int` qui contiendra (à terme) la politique optimale $\mu^{opt}(s)$. A chaque numéro d'état, il contiendra le numéro de la meilleure action.
- un pointeur vers un `MDP` (partie 1). Ici le modèle sera créé en dehors de la classe et on souhaite juste stocker son adresse dans la classe.
- un pointeur de pointeur de type `double`. Il contiendra la Q fonction. Il stockera connaissant les numéros de l'état et de l'action, la valeur associée de la Q fonction.

La classe `ModelSto` devra contenir également le constructeur par défaut, par copie et le destructeur. Il faudra aussi un constructeur qui prendra l'adresse d'un objet `MDP_parfait` et qui initialise les tableaux en récupérant le nombre d'états possibles du modèle. On ajoutera quatre fonctions :

— une fonction qui, à un état donné, renvoie l'action donnée par la politique optimale. Elle prend un état, calcule son numéro (par une recherche dans l'ensemble des états du modèle), récupère le numéro de la meilleure action dans $\mu^{opt}(s)$ et renvoie l'action associée.

— une fonction qui, à un état donné, renvoie une action suivant la politique aléatoire. Pour cela, on récupère le numéro de l'état i_s, les numéros des actions possibles i_a associées (on suppose n_a actions possibles). On construit une fonction de répartition associée à la loi de probabilité $\pi(i_s, .)$ (tableau de taille $n_a + 1$) ainsi :

$$F(i_a + 1) = F(i_a) + \pi(i_s, i_a)$$

avec $F(0) = 0$. Pour déterminer l'action, on tire un nombre aléatoire p entre 0 et 1 et on renvoie l'action de numéro i_a tel que

$$F(i_a) \leq p < F(i_a + 1)$$

— une fonction qui met à jour la politique optimale à partir de la Q-fonction. Pour cela on remplit le tableau μ^{opt} avec $\mu^{opt}(s) = \text{argmax}_a\, Q(s, a)$.

— une fonction qui calcule la politique aléatoire π. Pour tout état de numéro i_s, on calcule la meilleure action, de numéro noté i_a^* donnée par la politique optimale (fonction 1) puis on définit :

$$\pi(i_s, i_a^*) = 1 - \epsilon + \frac{\epsilon}{n_a}$$

et

$$\pi(i_s, i_a) = \frac{\epsilon}{n_a}$$

si i_a différent de i_a^*.

La classe `ModelSto` contiendra aussi une fonction virtuelle pure appelée `calcul_episodes( etat s)`. Enfin on écrira une fonction `apprentissage`, qui prend en paramètre un entier décrivant le nombre d'épisodes nécessaire à l'apprentissage. Cette fonction fera donc une boucle sur le nombre d'épisodes : à chaque itération, on appelle la fonction `reset` de modèle pour obtenir un état et ensuite on appelle la fonction virtuelle `calcul_episodes` sur cet état.

2.2 Q learning

A partir du cours numéro 7.

Objectif (3.0 points) : L'algorithme Q learning est un cas particulier des algorithmes stochastiques. Il s'agit d'une méthode avec un seul paramètre supplémentaire appelé α (de type double).

Ecrivez la classe nommée `Q_learning` qui hérite de la classe précédente avec comme attribut supplémentaire α. Ecrivez les constructeurs en utilisant ceux de la classe mère.

Ensuite il faudra implémenter la fonction `calcul_episodes( etat s)`. Justifier à l'écrit ou dans le code pourquoi. Cette fonction implémente l'algorithme suivant. On crée un booléen initialisé à faux. Tant qu'il est faux (boucle while) :

- on vérifie si l'état est terminal (fonction du modèle). Si l'état est terminal alors le booléen devient vrai.
- on appelle la politique aléatoire pour récupérer l'action a_t,
- on appelle l'environnement avec l'état et l'action choisie. On récupère r_t et s_{t+1}
- on calcule l'action optimale a^* sur l'état suivant obtenu s_{t+1}.
- On met à jour la Q-fonction. Si l'état s_t est terminal :

$$Q(i_{s_t}, i_{a_t}) = Q(i_{s_t}, i_{a_t}) + \alpha\, r_t$$

sinon

$$Q(i_{s_t}, i_{a_t}) = Q(i_{s_t}, i_{a_t}) + \alpha \left(r_t + \gamma\, Q(i_{s_{t+1}}, i_{a^*})\right)$$

- on prend $s_t = s_{t+1}$ et $\epsilon = d * \epsilon$.

Vous pouvez tester l'algorithme comme dans la partie 2 l'algorithme sur l'itération des valeurs. Il faut prendre $\gamma = 0.95$, $\alpha = 0.5$, $\epsilon = 1.0$ et $d = 0.98$.