FrozenLake

Sujet difficile

1 Présentation

1.1 Le Jeu

L'objectif du jeu est d'amener votre personnage jusqu'à l'endroit où son Frisbee s'est égaré. L'action se déroule sur un lac gelé en hiver et le parcours ne vas pas s'avérer facile. En effet, lorsque vous déplacez votre personnage vers la droite par exemple, celui-ci risque fort de glisser et dévier vers le haut ou vers le bas et ceci de manière aléatoire. Pour corser le jeu, à certain endroit du lac, la glace est fragile et si votre personnage tombe sur une de ces cases, il tombe à l'eau et il recommence à sa position de départ. Pour augmenter encore la difficulté, la glace s'avère de plus en plus glissante en se rapprochant de l'objectif. Ainsi, on peut trébucher, faire une pirouette et tomber en arrière ou sur les côtés, ce qui vient compliquer les déplacements.

1.2 Début du projet

Pour vous familiariser avec ce jeu, nous vous conseillons de jouer à plusieurs parties afin de bien assimiler l'aspect aléatoire des déplacements. Pour cela, lancez le fichier .py qui déclenche directement une partie au clavier. Utilisez les touches q, z, d et s pour vous déplacer.

1.3 Une nouvelle approche : le Q-Learning

Habituellement, nous connaissons les règles du jeu et nous savons à quoi sert chaque action. En effet, dans Pac-Man par exemple, vous savez que la manette permet de se déplacer dans un labyrinthe délimité par des murs. Vous savez aussi que les pac-gommes servent à manger les fantômes. Ainsi, vous programmez un algorithme permettant de générer au mieux les déplacements de Pac-Man.

Nous allons maintenant nous placer à un méta niveau. Dans FrozenLake, nous allons construire un algorithme de Q-Learning qui ignore tout du jeu. L'algorithme associé va être très différent de ce que nous avons fait précédemment. Nous avons cependant accès à une notion de récompense liée à la variation du score. C'est uniquement grâce à cette information que l'IA va apprendre à faire telle ou telle action car elle ne connait rien des règles du jeu :

- L'IA ne sait pas à quoi correspondent les actions possibles. Mais si l'IA bascule le joystick vers la droite amenant le personnage à sortir du terrain, le personnage va rester à la même position sans augmenter le score. A force d'apprentissage, l'IA conclura que cette action est inutile et ne la fera plus.
- L'IA ne connaît pas l'existence des trous dans le sol ainsi que le risque associé. Cependant, en tombant dans un trou, la récompense associée va être négative. Ainsi, à force d'apprentissage, l'IA va chercher à éviter de passer sur ces cases et va, à force d'expérience, chercher à les éviter en prenant des chemins passant loin des trous.
- L'IA, une fois le frisbee trouvé, va choisir des chemins amenant à cette case associée à une forte récompense. Comme nos parties font un nombre de tours constant, l'IA va indirectement choisir le chemin le plus court pour ramasser le frisbee plusieurs fois afin d'augmenter le score final de la simulation.

Ainsi, sur la page Wikipedia, vous trouvez comme description : <<Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state. It does not require

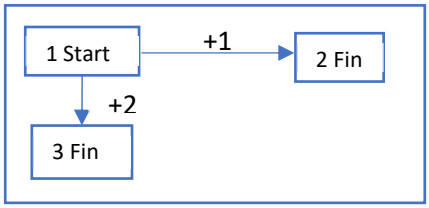
a model of the environment (hence "model-free"), and it can handle problems with stochastic transitions and rewards.>>

2 Principe

Nous allons décrire un jeu comme un ensemble d'états, chaque action amenant d'un état à un autre. Dans le jeu FrozenLake, l'état du jeu correspond à la position du joueur sur la grille car il n'y a pas d'autres éléments dans le jeu et le lac ne change pas ses propriétés au cours de la partie. Les actions correspondent aux quatre déplacements. Chaque état suivi d'une action va rapporter une certaine récompense quantifiable suivant une valeur. Le but est de maximiser le score sans connaissance à priori des règles du jeu.

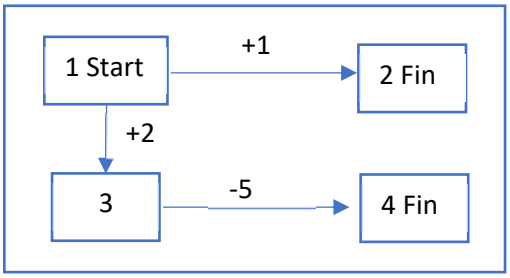
2.1 Rechercher les meilleures récompenses

Nous allons considérer un jeu avec un état de départ et deux actions possibles. Chaque action amène vers un nouvel état et une récompense est associée. Le jeu se termine après une action. Les deux actions possibles sont : Aller à droite, ce qui amène à l'état 2 et rapporte 1 point et Aller en bas, ce qui amène à l'état 3 et rapporte 2 points.



Graph de transitions du Jeu 1

Si vous deviez jouer à ce jeu, il est évident qu'après quelques parties, vous choisiriez tout le temps d'aller vers le bas car c'est l'option la plus rentable. Il nous suffit de choisir l'action qui rapporte le plus de points. Mais ce n'est pas toujours aussi évident. Observons un deuxième jeu :



Graph de transitions du Jeu 2

Cette fois, même si aller vers le bas rapporte 2 points, pour terminer la partie, il faut aller jusqu'à l'état 4 au risque de faire une action rapportant -5 points. Ainsi la route vers le bas s'avère peu rentable avec un total de 2-5=-3 points par rapport à l'autre option. Trouver la meilleure stratégie ne se résume donc pas à rechercher les actions associées aux meilleures récompenses.

2.2 La notion de politique

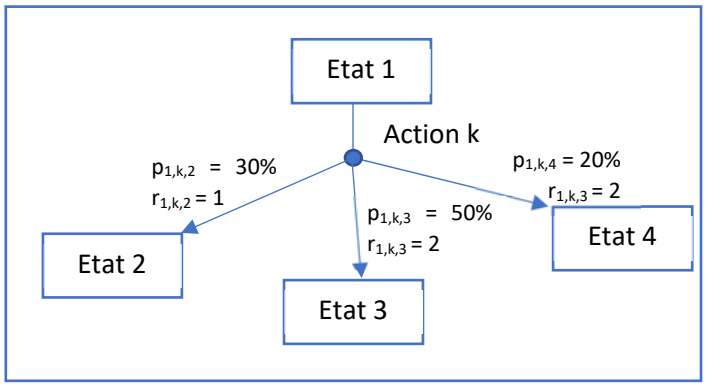
Pour notre I.A. il va falloir à un moment lui donner des règles dès qu'elle ne va plus jouer au hasard. Ces choix de jeu, appelés « politique » vont influencer le gain moyen. Supposons par exemple que dans le jeu 1, la politique soit la suivante :

- 60% du temps aller en bas
- 40% du temps aller à droite

Quel serait le gain moyen de cette politique ? Il suffit pour cela d'estimer le gain moyen associé à ces probabilités : 60% * 2 + 40% * 1 = 1,4

2.3 Un monde plus complexe

Nous avons vu pour l'instant le cas classique des jeux de plateau où faire une action amène toujours vers le même état. La vie de tous les jours n'est pas toujours aussi simple. Par exemple, lancer une pièce est une action et le résultat associé est à 50% pile et à 50% face. Nous pouvons dans le graph des transitions modéliser cette réalité en indiquant pour chaque action les probabilités d'arrivée sur un nouvel état :



Ainsi, la notation $p_{1,k,2}$ correspond à la probabilité depuis l'état 1 d'atteindre l'état 2 par l'action *k*.

2.4 La grandeur Q

La grandeur $Q_{e,a}$ nous donne le gain moyen que l'on peut espérer depuis l'état *e* en effectuant l'action *a*. Nous définissons aussi le gain moyen que l'on peut espérer depuis un état *e* noté V_e. Ainsi, nous pouvons écrire :

$$Q_{e,a} = \sum_{k \in K} p_{e,a,k}(r_{e,a,k} + V_k)$$

Où l'ensemble *K* désigne l'ensemble des états accessibles depuis l'état *e* après avoir effectué l'action *a*. Il reste à savoir comment exprimer V_k à partir de la grandeur *Q*. En fait, il suffit de se rappeler que depuis un état donné, nous pouvons choisir n'importe quelle action. Il suffit donc de choisir l'action qui donne un gain moyen maximal depuis cet état. Ainsi, nous pouvons écrire :

$$V_k = max_{a' \in A_k} Q_{k,a'}$$

Où l'ensemble A_k désigne l'ensemble des actions depuis l'état *k*. Ainsi, nous obtenons en combinant les deux formules :

$$Q_{e,a} = \sum_{k \in K} p_{e,a,k}(r_{e,a,k} + max_{a' \in A_k} Q_{k,a'})$$

Si cette formule semble simple à mettre en œuvre, elle comporte un défaut majeur. En effet, si dans un jeu on peut effectuer une série d'actions ramenant à un état antérieur avec un gain positif, cela sous-entend que nous sommes en mesure de construire une valeur Q infinie. Pour éviter cela, nous rajoutons un facteur γ compris entre 0 et 1 et permettant de limiter l'influence des cycles infinis :

$$Q_{e,a} = \sum_{k \in K} p_{e,a,k}(r_{e,a,k} + \gamma . max_{a' \in A_k} Q_{k,a'})$$

3 Mise en place

3.1 Estimer les valeurs Q

Nous allons associer à chaque case de la grille de jeu un numéro allant de 0 à *n-1*.

N'ayant aucune information sur le jeu, le programme va d'abord simuler plusieurs parties totalement aléatoires. Chaque partie va ainsi générer une séquence correspondant à une suite d'états / actions. Dans le jeu FrozenLake, nous ne pouvons pas savoir si une partie est terminée car nous n'avons pas accès à cette information. En effet, lorsque le joueur meurt dans un trou, cela amène un malus et c'est la seule chose que nous voyons car la partie « recommence » à la position initiale. Pour l'IA, les trous dans la glace pourraient tout aussi bien être des téléporteurs avec un coût d'utilisation chaque fois que l'on traverse leurs cases ! Nous allons donc jouer un nombre constant de tours pour chaque simulation et ainsi toutes les séquences seront de même longueur.

Une fois les séquences générées, nous avons suffisamment d'information sur le jeu pour estimer les grandeurs $p_{e,a,k}$ et $r_{e,a,k}$. Pour cela, nous devons comptabiliser :

- Le nombre de fois où nous sommes passés par l'état *e* et où l'action *a* a été choisie
- Le nombre de fois où nous sommes arrivés à l'état *k* depuis l'état *e* en ayant choisi l'action *a*
- Les récompenses associées

Ces trois tableaux peuvent être conservées d'un run à l'autre car ces observations décrivent « les règles » du jeu qui sont indépendantes de la politique choisie. En les conservant, on augmente donc notre connaissance du jeu.

Nous avons étudié l'ensemble des séquences de jeu et nous avons mis à jour les tableaux correspondants, les séquences peuvent à ce niveau être oubliées car elles ne sont plus utiles.

Nous pouvons maintenant construire une estimation des valeurs $p_{e,a,k}$ et $r_{e,a,k}$.

Il nous reste maintenant à construire les valeurs Q. Nous allons utiliser l'approche suivante :

- 1 : Initialisez les valeurs $Q_{e,a}$ à zéro
- 2 : Pour chaque état *e* et pour chaque action *a* appliquez la formule encadrée
- 3 : Recommencez l'étape 2 précédente

Bien sûr, vous n'allez pas effectuer une boucle infinie. Il faut donc trouver un critère d'arrêt judicieux.

3.2 Mise en place d'une politique de jeu

Vous pouvez à ce niveau construire une politique de jeu optimale par rapport aux explorations que nous avons effectuées. Pour cela, il suffit lorsque le joueur se trouve sur une certaine case correspondant à l'état *e* de choisir parmi les actions possibles celle qui offre la meilleure valeur Q :

$$max_{a \in A_e} Q_{e,a}$$

Vous pouvez ainsi effectuer un affichage d'une séquence de jeu utilisant cette politique. Cela vous permet de faire un contrôle des performances actuelles de l'IA.

3.3 On améliore !

Il se peut que le résultat de votre premier run soit peu satisfaisant. En effet, si lors des simulations aléatoires, la case du Frisbee n'a jamais été trouvée, l'IA n'est pas en mesure d'aller vers la bonne direction. Il se peut que votre joueur IA joue la sécurité en tournant en rond sur quatre cases en évitant les trous dans la glace !

Dans la mise en place d'un apprentissage, il existe un équilibre entre « l'exploitation » des bonnes stratégies connues et « l'exploration » de nouvelles approches. Jouer uniquement la carte de l'exploitation en rejouant des scénarios similaires aux meilleurs scénarios est un choix confortable car il vous garantit que vous obtiendrez des performances proches des meilleures performances connues. Par contre, il ne faut pas compter sur l'exploitation pour trouver un passage secret ou le super bonus. En effet, c'est le travail de la stratégie « d'exploration » qui consiste à tester de nouveaux chemins et de nouvelles options. Cette stratégie rapporte peu car la plupart des séquences générées vont être sans grande valeur, cependant, elle nous permettra de trouver de nouvelles approches qui vont nous permettre d'aller plus loin dans le jeu.

La stratégie mise en place par le *maxQ* est une stratégie d'exploitation. En début d'apprentissage, cette stratégie ne permet de revisiter que les endroits du jeu déjà visités. Il faut donc intégrer une stratégie qui permette aussi l'exploration durant nos simulations.

Encourager l'exploration

Cela peut, indépendamment du jeu, être mis en place en fournissant une récompense pour tout nouvel état exploré durant une partie. On peut par exemple fixer cette récompense à +1. Il faut que le nombre d'états à explorer soit inférieur à la récompense de la case bonus, sinon nous allons construire l'équivalent du jeu Tron où l'objectif va être de parcourir un maximum de cases du jeu !

Equilibrer exploitation et exploration

Choisissez une valeur ε entre 0 et 1. A chaque tour de jeu durant les simulations, vous pouvez choisir la politique optimale avec une probabilité ε ou un choix aléatoire avec une probabilité 1-ε.

Laisser une chance aux choix moins rentables

Nous pouvons choisir une politique de jeu moins stricte que celle du *maxQ* qui nous force à choisir depuis un état toujours la même action. Nous allons utiliser pour cela une pondération utilisant une fonction $\sigma(x)$. Supposons que nous sommes dans un certain état et que nous avons le choix entre m actions chacune associée à un certain poids noté K_i. Nous allons donc définir une probabilité de choisir l'action i de la manière suivante :

$$p_i = \frac{\sigma(K_i)}{T} \qquad \text{avec} \qquad T = \sum_{j=0}^{m-1} \sigma(K_j)$$

Vous pouvez remarquer que la valeur T est constante et commune à l'ensemble des valeurs p_i. Prenons comme exemple les valeurs dans le tableau ci-dessous avec la fonction $\sigma(x) = e^x$:

Action	K_i	$exp(K_i)$	T	p_i
0	3	20	21,41	93,41%
1	0.2	1,22		5.70%
2	-2	0,13		0.61%
3	-5	0,06		0.28%

Cette normalisation appelée Softmax permet de transformer tout type de poids en probabilité. Elle est intéressante car elle accepte des poids négatifs ce qui est notre cas avec les valeurs Q.