Q-Learning de renforcement à partir de zéro en Python avec OpenAI Gym

Apprenez à un taxi à prendre et déposer des passagers aux bons endroits grâce à l'apprentissage par renforcement

la plupart d'entre vous ont probablement entendu parler de l'IA apprenant à jouer à des jeux informatiques par elle-même, un exemple très populaire étant Deepmind. Deepmind a fait la une des journaux lorsque son programme AlphaGo a battu le champion du monde sud-coréen Go en 2016. Il y a eu de nombreuses tentatives réussies dans le passé pour développer des agents dans le but de jouer à des jeux Atari comme Breakout, Pong et Space Invaders.

Chacun de ces programmes suit un paradigme d'apprentissage automatique connu sous le nom d'apprentissage par renforcement. Si vous n'avez jamais été exposé à l'apprentissage par renforcement auparavant, voici une analogie très simple de la façon dont il œuvres.

Analogie de l'apprentissage par renforcement

Considérez le scénario consistant à enseigner de nouveaux tours à un chien. Le chien ne comprend pas notre langue, nous ne pouvons donc pas lui dire quoi faire. Au lieu de cela, nous suivons une stratégie différente. Nous imitons une situation (ou un signal), et le chien essaie de répondre de différentes manières. Si la réponse du chien est celle souhaitée, nous le récompensons avec des collations. Maintenant devinez quoi, la prochaine fois que le chien est exposé à la même situation, le chien exécute une action similaire avec encore plus d'enthousiasme dans l'attente de plus de nourriture. C'est comme apprendre « quoi faire » à partir d'expériences positives. De même, les chiens auront tendance à apprendre ce qu'il ne faut pas faire face à des expériences négatives.

C'est exactement ainsi que fonctionne l'apprentissage par renforcement au sens large :

- Votre chien est un « agent » qui est exposé à l'environnement. La environnement pourrait dans votre maison, avec vous.
- Les situations qu'ils rencontrent sont analogues à un état. Un exemple d'un l'état pourrait être votre chien debout et vous utilisez un mot spécifique dans un certain ton dans votre salon

- Nos agents réagissent en réalisant une action pour passer d'un « état » à un autre « état », votre chien passe de debout à assis par exemple.
- Après la transition, ils peuvent recevoir une récompense ou une pénalité en retour. Tu faites-leur plaisir ! Ou un "Non" en guise de sanction.
- La politique est la stratégie consistant à choisir une action compte tenu d'un état attendu de meilleurs résultats.

L'apprentissage par renforcement se situe entre le spectre de l'apprentissage supervisé et de l'apprentissage non supervisé, et il y a quelques points importants à noter:

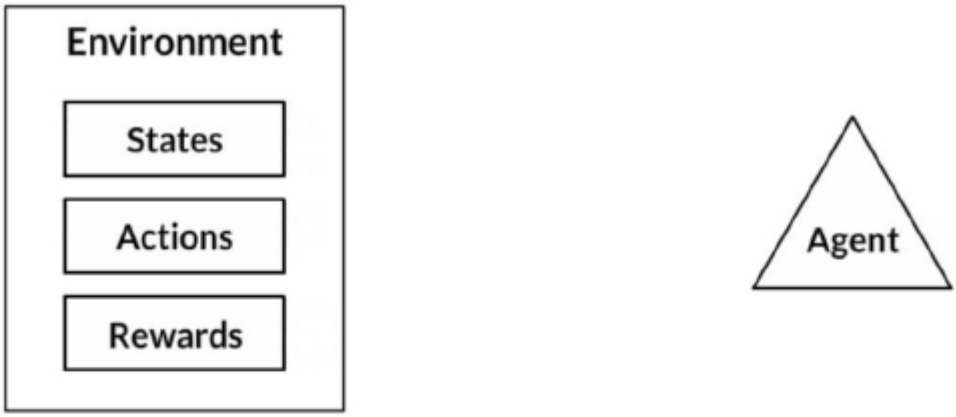
1 Être gourmand ne fonctionne pas toujours

Il ya des choses faciles à faire pour une gratification instantanée, et il y a des choses qui offrent des récompenses à long terme
Le but n'est pas d'être gourmand en recherchant les
récompenses immédiates rapides, mais plutôt d'optimiser pour un maximum de récompenses sur toute la formation.

2.La séquence compte dans l'apprentissage par renforcement

L'agent de récompense ne dépend pas seulement de l'état actuel, mais de toute l'histoiredes états. Contrairement à l'apprentissage supervisé et non supervisé, le temps est important ici.

Le processus d'apprentissage par renforcement



D'une certaine manière, l'apprentissage par renforcement est la science qui consiste à prendre des décisions optimales à l'aide d'expériences. En le décomposant, le processus d'apprentissage par renforcement implique ces étapes simples:

1. Observation de l'environnement
2. Décider comment agir en utilisant une stratégie
3. Agir en conséquence
4. Recevoir une récompense ou une pénalité
5. Apprendre des expériences et affiner notre stratégie
6. Itérer jusqu'à ce qu'une stratégie optimale soit trouvée

Comprenons maintenant l'apprentissage par renforcement en développant en fait un agent pour apprendre à jouer à un jeu automatiquement par lui-même.

Exemple de conception: cabine autonome

Concevons une simulation d'une cabine autonome. L'objectif principal est de démontrer, dans un environnement simplifié, comment vous pouvez utiliser les techniques RL pour développer une approche efficace et sûre pour résoudre ce problème.

Le travail du Smartcab consiste à prendre le passager à un endroit et à le déposer à un autre. Voici quelques éléments dont nous aimerions que notre Smartcab s'occupe:

- Déposez le passager au bon endroit.
- Économisez du temps aux passagers en prenant le moins de temps possible
- Veiller à la sécurité des passagers et aux règles de circulation

Différents aspects doivent être pris en compte ici lors de la modélisation d'une solution RL à ce problème: les récompenses, les états et les actions.

1. Récompenses

Étant donné que l'agent (le conducteur imaginaire) est motivé par la récompense et qu'il va apprendre à contrôler la cabine par des expériences d'essai dans l'environnement, nous devons décider des récompenses et/ou des pénalités et de leur ampleur en conséquence.
Voici quelques points à considérer :

- L'agent devrait recevoir une récompense positive élevée pour un dépôt réussi car ce comportement est fortement souhaité
- L'agent doit être pénalisé s'il essaie de déposer un passager par erreur
- L'agent devrait recevoir une légère récompense négative s'il n'arrive pas à destination après chaque pas de temps. "Léger" négatif car nous préférerions que notre agent arrive en retard au lieu de faire de faux pas en essayant d'atteindre la destination le plus rapidement possible

2. Espace d'état

Dans l'apprentissage par renforcement, l'agent rencontre un état, puis agit en fonction de l'état dans lequel il se trouve.

L'espace d'état est l'ensemble de toutes les situations possibles que notre taxi pourrait vivre. L'état doit contenir des informations utiles dont l'agent a besoin pour effectuer la bonne action.

Disons que nous avons une zone d'entraînement pour notre Smartcab où nous lui apprenons à transporter des personnes dans un parking vers quatre emplacements différents (R, G, Y, B):

Supposons que Smartcab est le seul véhicule dans ce parking. Nous pouvons diviser le parking en une grille 5x5, ce qui nous donne 25 emplacements de taxi possibles.
Ces 25 sites font partie de notre espace étatique. Notez que l'état actuel de l'emplacement de notre taxi est la coordonnée (3, 1).

Vous remarquerez également qu'il existe quatre (4) emplacements où nous pouvons prendre et déposer un passager: R, G, Y, B ou **[(0,0), (0,4), (4,0), (4,3)]** en coordonnées (ligne, col). Notre passager illustré est à l'emplacement Y et il souhaite se rendre
à l'emplacement R.

Lorsque nous tenons également compte d'un (1) état de passager supplémentaire à l'intérieur du taxi, nous pouvons prendre toutes les combinaisons d'emplacements de passagers et d'emplacements de destination pour arriver à un nombre total d'états pour notre taxi.

environnement; il y a quatre (4) destinations et cinq (4 + 1) passagers
Emplacements.

Ainsi, notre environnement de taxi a $5 \times 5 \times 5 \times 4 = 500$ nombre total d'états possibles.

3. Espace Action

L'agent rencontre l'un des 500 états et il entreprend une action. L'action dans notre cas peut être de se déplacer dans une direction ou de décider de ramasser/déposer
un passager.

En d'autres termes, nous avons six actions possibles:

1. **Sud**
2. **Nord**
3. **Est**
4. **Ouest**
5. **Ramassage**
6. **Dépot**

C'est l'espace d'action : l'ensemble de toutes les actions que notre agent peut effectuer dans un état donné.

Vous remarquerez dans l'illustration ci-dessus que le taxi ne peut pas effectuer certaines actions dans certains états en raison des murs. Dans le code de l'environnement, nous fournirons simplement une pénalité de -1 pour chaque mur touché et le taxi ne se déplacera nulle part.
Cela ne fera qu'accumuler des pénalités obligeant le taxi à envisager de faire le tour le mur.

Implémentation avec Python

Heureusement, OpenAI Gym a cet environnement exact déjà construit pour nous.

Gym fournit différents environnements de jeu que nous pouvons connecter à notre code et tester un agent. La bibliothèque s'occupe de l'API pour fournir toutes les informations dont notre agent aurait besoin, comme les actions possibles, le score et l'état actuel. Nous devons juste nous concentrer uniquement sur la partie algorithme.

L'interface du gymnase IA:

L'interface principale de AI Gym est **env** , qui est l'interface d'environnement unifié.

Voici les méthodes **env** qui nous seraient très utiles:

- **env.reset : réinitialise** l'environnement et renvoie un état initial aléatoire.
- **env.step(action) :** incrémente l'environnement d'un pas de temps. Retour
    - observation : Observations de l'environnement
    - récompense: si votre action a été bénéfique ou non
        - done: Indique si nous avons récupéré et déposé avec succès un passager, également appelé un épisode
        - info: informations supplémentaires telles que les performances et la latence pour le débogage fins
    - **env.render: rend** une image de l'environnement (utile pour visualiser l'environnement)

Remarque : Nous utilisons le .env à la fin du **make** pour éviter l'arrêt de la formation à 200 itérations, ce qui est la valeur par défaut pour la nouvelle version de Gym (référence).

Rappel de notre problème

Voici notre énoncé de problème restructuré (de Gym docs):

"Il y a 4 emplacements (étiquetés par des lettres différentes), et notre travail consiste à choisir entre
monter le passager à un endroit et le déposer à un autre.
Nous recevons +20points pour un dépôt réussi et perdre 1point pour chaque pas de temps prend. Il y a aussi une pénalité de 10 points pour le ramassage et le dépôt illégaux

Plongeons davantage dans l'environnement.

```
env.reset() # réinitialise l'environnement à un nouvel état aléatoire
env.render()
```

Espace d'action discret(6)

Espace d'état discret(500)

- Le carré plein représente le taxi, qui est jaune sans passager et vert avec un passager. • Le

tuyau ("|") représente un mur que le taxi ne peut pas franchir.

- R, G, Y, B sont les emplacements de prise en charge et de destination possibles. La lettre bleue représente le lieu actuel de prise en charge des passagers et la lettre violette est la destination actuelle.

Comme le vérifient les impressions, nous avons un espace d'action de taille 6 et un espace d'état de taille 500. Comme vous le verrez, notre algorithme RL n'aura pas besoin de plus d'informations que ces deux choses. Tout ce dont nous avons besoin est un moyen d'identifier un état de manière unique en attribuant un numéro unique à chaque état possible, et RL apprend à choisir un numéro d'action de 0 à 5 où:

- 0 = sud
- 1 = nord
- 2 = est
- 3 = ouest
- 4 = ramassage
- 5 = chute

Rappelons que les 500 états correspondent à un encodage de la position du taxi, de la position du passager et de la position de destination.

L'apprentissage par renforcement apprendra une cartographie des états à l'action optimale à effectuer dans cet état par exploration, c'est-à-dire que l'agent explore l'environnement et prend des mesures en fonction des récompenses définies dans le environnement.

L'action optimale pour chaque état est l'action qui a le plus haut récompense cumulative à long terme.

Retour à notre illustration

Nous pouvons en fait prendre notre illustration ci-dessus, encoder son état et le donner à l'environnement à rendre dans Gym. Rappelons que nous avons le taxi au rang 3, colonne 1, notre passager est à l'emplacement 2 et notre destination est l'emplacement 0. En utilisant la méthode d'encodage d'état Taxi-v2, nous pouvons faire ce qui suit:

Nous utilisons les coordonnées de notre illustration pour générer un nombre correspondant à un état compris entre 0 et 499, qui s'avère être 328 pour l'état de notre illustration.

Ensuite, nous pouvons définir l'état de l'environnement manuellement avec **env.env.s** en utilisant cela

numéro encodé. Vous pouvez jouer avec les chiffres et vous verrez le le taxi, le passager et la destination se déplacent.

Le tableau des récompenses

Lorsque l'environnement Taxi est créé, une table de récompenses initiale est également créée, appelée«P». Nous pouvons le considérer comme une matrice qui a le nombre d'états en lignes et le nombre d'actions en colonnes, c'est-à-dire une matrice.

Étant donné que chaque état se trouve dans cette matrice, nous pouvons voir les valeurs de récompense par défaut attribuées à l'état de notre illustration:

Ce dictionnaire a la structure

{action: [(probabilité, état suivant, récompense,fait)]} .

Quelques points à noter:

- Le 0-5 correspond aux actions (sud, nord, est, ouest, ramassage, débarquement) que le taxi peut effectuer à notre état actuel dans l'illustration.
- Dans cet environnement, la **probabilité** est toujours de 1,0.
- L' **état suivant** est l'état dans lequel nous serions si nous prenions l'action à cet instant index du dict
- Toutes les actions de déplacement ont une récompense de -1 et les actions de ramassage/dépose ont une récompense de -10 dans cet état particulier. Si nous sommes dans un état où le taxi a un passager et est au-dessus de la bonne destination, nous verrions une récompense de 20 à l'action de dépose (5)

- **done** est utilisé pour nous dire quand nous avons réussi à déposer un passager au bon endroit. Chaque dépôt réussi est la fin d'un épisode

Notez que si notre agent choisissait d'explorer l'action deux (2) dans cet état, il irait vers l'Est dans un mur.
Le code source a rendu impossible le déplacement du taxi sur un mur, donc si le taxi choisit cette action, il continuera à accumuler des pénalités de -1, ce qui affectela récompense à long terme.

Résoudre l'environnement sans Apprentissage par renforcement

Voyons ce qui se passerait si nous essayions de forcer brutalement notre chemin pour résoudre le problème sans RL.

Puisque nous avons notre table **P** pour les récompenses par défaut dans chaque état, nous pouvons essayer de faire naviguer notre taxi en utilisant simplement cela.

Nous allons créer une boucle infinie qui s'exécute jusqu'à ce qu'un passager atteigne une destination (un épisode), ou en d'autres termes, lorsque la récompense reçue est de 20. La méthode env.action_space.sample() sélectionne automatiquement une action aléatoire parmi un ensemble de tous actions possibles.

Voyons ce qui se passe:

Pas bon. Notre agent prend des milliers de pas de temps et fait beaucoup de fausses livraisons pour livrer un seul passager à la bonne destination.

C'est parce que nous n'apprenons pas de l'expérience passée. Nous pouvons exécuter cela encore et encore, et cela ne sera jamais optimisé. L'agent n'a aucune mémoire de quelle action était la meilleure pour chaque état, ce qui est exactement ce que l'apprentissage par renforcement fera pour nous.

Viens alors l'apprentissage par renforcement

Nous allons utiliser un algorithme RL simple appelé Q-learning qui donnera à notre agent un peu de mémoire.

Introduction au Q-learning

Essentiellement, le Q-learning permet à l'agent d'utiliser les récompenses de l'environnement pour apprendre, au fil du temps, la meilleure action à entreprendre dans un état donné.

Dans notre environnement Taxi, nous avons le tableau des récompenses, **P** , que l'agent va apprendre. Il fait quelque chose en cherchant à recevoir une récompense pour avoir effectué une action dans l'état actuel, puis en mettant à jour une valeur Q pour se rappeler si cette action a été bénéfique.

Les valeurs stockées dans la table Q sont appelées valeurs Q et correspondent à une **(état, action)** combinaison.

Une valeur Q pour une combinaison état-action particulière est représentative de la "qualité" d'une action prise à partir de cet état. De meilleures valeurs Q impliquent de meilleures chances d'obtenir de plus grandes récompenses.

Par exemple, si le taxi est confronté à un état qui inclut un passager à son emplacement actuel, il est fort probable que la valeur Q pour le **ramassage** soit plus élevée lorsque par rapport à d'autres actions, comme **le débarquement** ou le **nord**

Les valeurs Q sont initialisées à une valeur arbitraire, et lorsque l'agent s'expose à l'environnement et reçoit différentes récompenses en exécutant différentes actions, les valeurs Q sont mises à jour à l'aide de l'équation (...)

(alpha) est le taux d'apprentissage- Tout comme dans l'apprentissage supervisé (paramètres), est la mesure dans laquelle nos valeurs Q sont mises à jour à chaque itération.

- (gamma) est le facteur de réduction ( ) - détermine l'importance que nous voulons accorder aux récompenses futures. Une valeur élevée pour le facteur d'actualisation (proche de 1) reflète l'attribution effective à long terme, alors qu'un facteur de remise de 0 oblige notre agent à ne considérer que la récompense immédiate, ce qui le rend gourmand.

Qu'est-ce que cela dit?

Nous attribuons ( ), ou mettons à jour, la valeur Q de l'état et de l'action actuels de l'agent en prenant d'abord un poids ( ).
La valeur apprise est une combinaison de la récompense d'avoir pris l'action actuelle dans l'état actuel, et la récompense maximum actualisée du prochain état dans lequel nous serons une fois que nous aurons pris l'action en cours.

Fondamentalement, nous apprenons l'action appropriée à prendre dans l'état actuel en examinant la récompense pour le combo état/action actuel et les récompenses maximales pour l'état suivant.

Cela amènera éventuellement notre taxi à envisager l'itinéraire avec les meilleures récompenses enchaînées.

La valeur Q d'une paire état-action est la somme de la récompense instantanée et de la récompense future actualisée (de l'état résultant).
La façon dont nous stockons les valeurs Q pour chaque état et action se fait via une table Q

Tableau Q

Le Q-table est une matrice où nous avons une ligne pour chaque état (500) et une colonne pour chaque action (6). Il est d'abord initialisé à 0, puis les valeurs sont mises à jour après la formation.

Notez que la table Q a les mêmes dimensions que la table des récompenses, mais elle a un objectif complètement différent.

Les valeurs de Q-Table sont initialisées à zéro, puis mises à jour pendant la formation sur des valeurs qui optimisent la traversée de l'agent dans l'environnement pour un maximum de récompenses

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Résumé du processus de Q-Learning

En le décomposant en étapes, on obtient

- Initialiser la Q-table par tous les zéros.
- Commencer à explorer les actions: pour chaque état, sélectionnez l'une des actions possibles pour l'état actuel (S).
- Passer à l'état suivant (S') à la suite de cette action (a).
    - Pour toutes les actions possibles à partir de l'état (S') sélectionner celle avec le Q le plus élevé évaluer.
    - Mettez à jour les valeurs de la table Q à l'aide de l'équation.
    - Définissez l'état suivant comme état actuel.
    - Si l'état d'objectif est atteint, terminez et répétez le processus.

Exploiter les valeurs apprises

Après une exploration aléatoire suffisante des actions, les valeurs Q ont tendance à converger, servant n
otre
agent comme une fonction action-valeur qu'il peut exploiter pour choisir l'action la plus optimale à partir d'
un
état donné.

Il y a un compromis entre l'exploration (choisir une action aléatoire) et l'exploitation (choisir des actions basées sur des valeurs Q déjà apprises). Nous voulons éviter que l'action emprunte toujours le même chemin, et éventuellement un surajustement, nous allons donc introduire un autre paramètre appelé "epsi
lon"
pour répondre à cela pendant la formation.

Au lieu de simplement sélectionner la meilleure action de valeur Q apprise, nous privilégierons parfois une exploration plus approfondie de l'espace d'action. Une valeur epsilon plus faible entraîne des épisodes avec plus de pénalités (en moyenne), ce qui est évident car nous explorons et prenons des décisions aléatoires.

Implémentation de Q-learning en python

Formation de l'agent

Tout d'abord, nous allons initialiser la Q-table à une à une matrice de zéros
Nous pouvons maintenant créer l'algorithme de formation qui mettra à jour cette Q-table au fur et à mesure que l'agent explore l'environnement sur des milliers d'épisodes.

Dans la première partie , nous décidons de choisir une action au hasard ou pour exploiter les Q-values déjà calculées. Cela se fait simplement en utilisant la valeur **epsilon** et en la comparant à la **fonction random.uniform(0, 1) ,** qui renvoie un nombre arbitraire entre 0 et 1.

Nous exécutons l'action choisie dans l'environnement pour obtenir le **next_state** et la **récompense** de l'exécution de l'action. Après cela, nous calculons la valeur Q maximale pour les actions correspondant au **next_state ,** et avec cela, nous pouvons facilement mettre à jour notre Q-value vers la **new_q_value :**

Maintenant que le Q-table a été établi sur 100 000 épisodes, passons voir quelles sont les valeurs Q à l'état de notre illustration

La valeur Q maximale est "nord" (-1,971), il semble donc que Q-learning ait effectivement appris la meilleure action à entreprendre dans l'état de notre illustration!

Évaluation de l'agent

Évaluons les performances de notre agent. Nous n'avons pas besoin d'explorer davantage les actions, donc maintenant l'action suivante est toujours sélectionnée en utilisant la meilleure valeur Q:

Nous pouvons voir à partir de l'évaluation que les performances de l'agent se sont considérablement améliorées et qu'il n'a encouru aucune pénalité, ce qui signifie qu'il a effectué les bonnes actions de prise en charge/dépose avec 100 passagers différents.

Comparer notre agent Q-learning au RL Classique

Avec Q-learning, l'agent commet initialement des erreurs lors de l'exploration, mais une fois qu'il a suffisamment exploré (vu la plupart des états), il peut agir judicieusement en maximisant les récompenses en faisant des mouvements intelligents.
Voyons à quel point notre solution Q-learning est meilleure par rapport à l'agent effectuant des mouvements aléatoires.

Nous évaluons nos agents selon les métriques suivantes,

- Nombre moyen de pénalités par épisode: plus le nombre est petit, plus améliorer la performance de notre agent. Idéalement, nous aimerions que cette métrique soit nulle ou très proche de zéro.
- Nombre moyen de pas de temps par trajet: nous voulons un petit nombre de pas de temps par épisode également puisque nous voulons que notre agent fasse un minimum de pas (c'est-à-dire le chemin le plus court) pour atteindre la destination.
- Récompenses moyennes par déplacement: plus la récompense est importante, plus l'agent fait ce qu'il faut. C'est pourquoi le choix des récompenses est un élément crucial de l'apprentissage par renforcement. Dans notre cas, comme les pas de temps et les pénalités sont récompensés négativement, une récompense moyenne plus élevée signifierait que l'agent atteint la destination le plus rapidement possible avec le moins de pénalités

Mesure	Performance de l'agent aléatoire	Performances de l'agent Q-learning
Récompenses moyennes par coup	-3.9012092102214075	0.6962843295638126
Nombre moyen de pénalités par épisode	920,45	0.0
Nombre moyen de pas de temps par trajet	2848.14	12h38

Ces métriques ont été calculées sur 100 épisodes. Et comme le montrent les résultats, notre agent Q-learning a réussi!

Hyperparamètres et optimisations

Les valeurs de `alpha`, `gamma` et `epsilon` étaient principalement basées sur l'intuition et certains "hit and trial", mais il existe de meilleures façons de trouver de bonnes valeurs.

Idéalement, les trois devraient diminuer avec le temps, car à mesure que l'agent continue d'apprendre, il construit en fait des a priori plus résilients;

- (le taux d'apprentissage) devrait diminuer à mesure que vous continuez à acquérir une plus grande et plus grande base de connaissances.
- Au fur et à mesure que vous vous rapprochez de la date limite, votre préférence pour la récompense à court terme devrait augmenter, car vous ne resterez pas assez longtemps pour obtenir la récompense à long terme, ce qui signifie que votre gamma devrait diminuer au fur et à mesure que nous développons notre stratégie, nous avons moins besoin d'exploration et plus d'exploitation pour obtenir plus d'utilité de notre politique, donc plus les essais augmentent, epsilon devrait diminuer.
-

Réglage des hyperparamètres

Un moyen simple de trouver par programme le meilleur ensemble de valeurs de l'hyperparamètre consiste à créer une fonction de recherche complète (similaire à la recherche de grille) qui sélectionne les paramètres qui donneraient le meilleur

rapport **récompense/time_steps .** La raison de **récompense/time_steps** est que nous voulons

choisir des paramètres qui nous permettent d'obtenir le maximum de récompense le plus rapidement possible. Nous pouvons également souhaiter suivre le nombre de pénalités correspondant à la combinaison de valeurs d'hyperparamètres,

car cela peut également être un facteur décisif (nous ne voulons pas que notre agent intelligent enfreigne les règles au prix d'une atteinte plus rapide). Une façon plus sophistiquée d'obtenir la bonne combinaison de valeurs d'hyperparamètres serait d'utiliser des algorithmes génétiques.

Conclusion

Très bien ! Nous avons commencé par comprendre l'apprentissage par renforcement à l'aide d'analogies du monde réel. Nous avons ensuite plongé dans les bases de l'apprentissage par renforcement et défini une cabine autonome comme un renforcement

Problème d'apprentissage. Nous avons ensuite utilisé Gym d'OpenAI en python pour nous fournir un environnement connexe, où nous pouvons développer notre agent et l'évaluer. Ensuite, nous avons observé à quel point notre agent était terrible sans utiliser aucun algorithme pour jouer au jeu, alors nous avons continué à implémenter l'algorithme Q-learning à partir de zéro. Les performances de l'agent se sont considérablement améliorées après le Q-learning. Enfin, nous avons discuté de meilleures approches pour décider des hyperparamètres de notre algorithme.

Q-learning est l'un des algorithmes d'apprentissage par renforcement les plus simples. Le problème avec Q-learning est cependant qu'une fois que le nombre d'états dans l'environnement est très élevé, il devient difficile de les implémenter avec la table Q car la taille deviendrait très, très grande. Les techniques de pointe utilisent les réseaux de neurones profonds au lieu de la Q-table (Deep Reinforcement Learning). Le réseau neuronal prend les informations d'état et les actions dans la couche d'entrée et apprend à produire la bonne action au fil du temps. Des techniques d'apprentissage en profondeur (comme les réseaux de neurones convolutifs) sont également utilisées pour interpréter les pixels à l'écran et extraire des informations du jeu (comme les scores), puis laisser l'agent contrôler le jeu.

Nous avons beaucoup discuté de l'apprentissage par renforcement et des jeux. Mais l'apprentissage par renforcement ne se limite pas aux jeux. Il est utilisé pour gérer les portefeuilles d'actions et les finances, pour fabriquer des robots humanoïdes, pour la fabrication et la gestion des stocks, pour développer des agents généraux d'IA, qui sont des agents qui peuvent effectuer plusieurs choses avec un seul algorithme, comme le même agent jouant à plusieurs jeux Atari. Open AI dispose également d'une plate-forme appelée univers pour mesurer et former l'intelligence générale d'une IA à travers des myriades de jeux, de sites Web et d'autres applications générales.

Maintenant, à vous de jouer

Si vous souhaitez poursuivre ce projet pour l'améliorer, voici quelques éléments que vous pouvez ajouter:

- Transformez ce code en un module de fonctions pouvant utiliser plusieurs environnements
- Réglez alpha, gamma et/ou epsilon à l'aide d'une décroissance sur les épisodes
- Mettez en œuvre une recherche Grid pour découvrir les meilleurs hyperparamètres