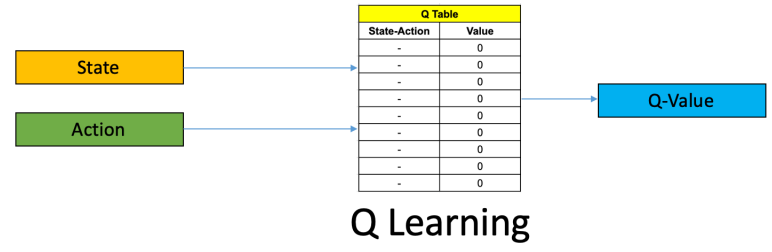
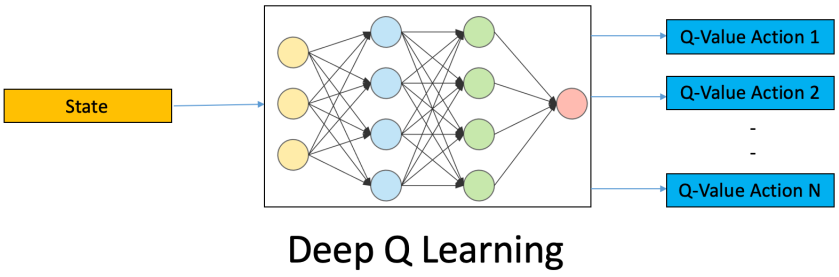
Part III
Deep reinforcement learning

1 Théorie

L'algorithme traditionnel de Q-learning nécessite, à chaque itération d'apprentissage, une recherche par grille (élément par élément) dans l'ensemble de la table Q. Dans l'exemple où on aurait 10000 état et 1000 actions par état, la table Q sera composée de 10 millions d'éléments. La mémoire utilisée peut vite devenir astronomique, et le temps nécessaire à l'exploration d'autant plus important.

Dans le cas d'un problème de gestion de trafic routier, de nombreux paramètres macroscopiques par exemple, la vitesse, la densité et le nombre de véhicules entrants à chaque intersection peuvent être organisés dans la représentation vectorielle de l'état, tandis que les actions sont principalement modélisées par un nombre fini de scalaires par exemple, un nombre prédéfini de phases de feux de signalisation désignées par des indices. La visite de chaque paire état-action dans les structures de type tableau en utilisant l'approche classique de recherche par grille peut devenir infaisable du point de vue informatique.





Pour résoudre les problèmes de haute dimensionnalité de l'espace continu, de nombreuses méthodes d'approximation de fonctions non linéaires sont envisagées. L'idée de l'approximation de fonction est d'éviter de calculer la valeur Q exacte en calculant son approximation qui couvre l'ensemble de l'espace état-action. L'application de l'approximation de fonctions dans l'apprentissage automatique ne produit des résultats significatifs que dans les cas où les états sont de faible dimension et conçus manuellement avec des fonctions de valeur ou de politique linéaires. Dans le contexte de l'apprentissage automatique, les approximateurs suivants sont utilisés : les arbres de décision, les réseaux neuronaux artificiels (ANN) et les méthodes de régression.

2 Application

2.1 Application sur la conduite autonome de véhicule

On se propose d'appliquer cet algorithme sur l'apprentissage automatique de la conduite d'une voiture. On utilise la biliothèque highway-env, une collection d'environnements pour la conduite autonome sur python.

On choisira d'appliquer l'algorithme sur Roundabout-v0, un environnement où le véhicule s'approche d'un rond-point où le trafic est fluide. Il suit automatiquement son itinéraire prévu, mais doit gérer les changements de voie et le contrôle longitudinal pour passer le rond-point le plus rapidement possible tout en évitant les collisions.

2.2 Implémentation

Pour implémenter cet algorithme, on commence par définir notre modèle (réseau de neurone et nombre de couches), ensuite on construit notre agent qui choisira les actions à prendre en fonction de la politique choisie. Ensuite, il suffira de compiler le modèle et d'utiliser la commande de prédiction du modèle.

2.3 Changement de politique (policy)

Malgré les assez-bons résultats de la politique choisie (Epsilon greedy), le modèle a tendance à obtenir des résultats (scores) sous-optimaux.

On se propose alors de changer la politique de jeu.

2.3.1 Explication Boltzmann

Dans la politique Epsilon greedy ci-dessus, l'agent choisira une action aléatoire avec une probabilité constante au lieu de suivre sa politique.

Le problème est qu'il traite toutes les actions de la même manière lorsqu'il décide de l'action à entreprendre. Mais que se passe-t-il si certaines actions semblent plus prometteuses que d'autres ?

La méthode Epsilon Greedy ne peut pas gérer une telle situation. Le fait est que, dans les simulations, toutes les actions ne sont pas créées égales.

Ainsi, on choisit l'action a_i avec une probabilité:

$$P(a_i) = \frac{e^{\frac{Q[s,a_i]}{\tau}}}{\sum_i e^{\frac{Q[s,a_i]}{\tau}}}$$

où τ est un paramètre positif appelé la température.

Les températures élevées font que les actions sont toutes équiprobables. Les températures basses entraînent une plus grande différence dans la probabilité de sélection pour les actions qui diffèrent dans leurs estimations de valeur. Dans la limite où , la sélection d'actions par Boltzmann devient identique à la sélection d'actions par epsilon greedy. L'action ayant la valeur Q[s,a] la plus élevée a donc beaucoup plus de chances d'être sélectionnée.

2.3.2 Résultats

En changeant de politique de jeu, on remarque une augmentation considérable sur la valeur moyenne, toutefois l'environnement n'est pas résolu pour autant. En effet, en moyenne, la politique Epsilon Greedy à un score de 9.2524 en moyenne, mais en la changeant pour la politique Boltzmann la moyenne devient de 9.5228.

La valeur moyenne a augmenté mais la résolution de l'environnement reste incomplète

2.3.3 Problème Experience Replay

Bien que l'agent ait obtenu de bons résultats après la formation initiale, il lui a fallu un temps exponentiel pour s'améliorer davantage. Cela s'explique par le fait qu'il faut un temps plus long à l'agent pour atteindre un scénario où il y'a un accident et, même dans ce cas, il ne peut tirer qu'une seule leçon de celui-ci.

2.4 Importance de l'Experience Replay

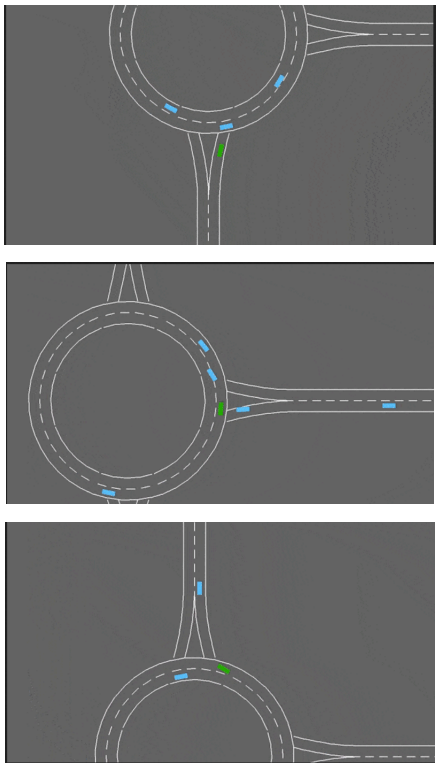
En introduisant la reproduction de l'expérience, l'agent peut tenter le scénario difficile plusieurs fois jusqu'à ce qu'il le surmonte ou qu'il soit bloqué dans une boucle (fixée à un nombre de tentatives).

Il s'agit d'un élément clé lorsque l'acquisition d'une expérience du monde réel est coûteuse, afin d'en tirer pleinement parti. Les mises à jour de l'apprentissage Q sont incrémentielles et ne convergent pas rapidement, donc plusieurs passages avec les mêmes données sont bénéfiques.

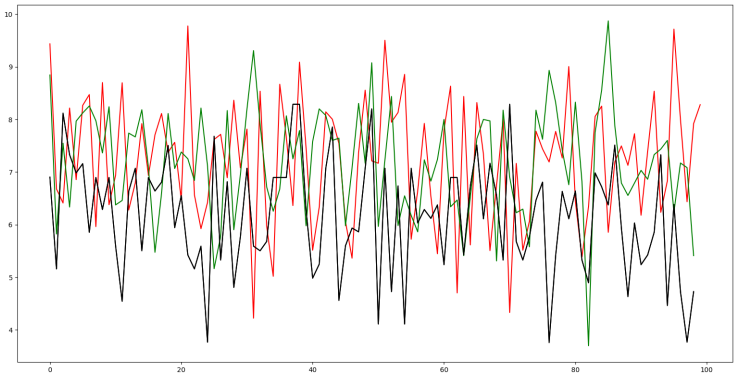
Ainsi on implémente un nouvel agent DQN (Deep Q Network) de la bilbliothèque stable_ baselines3 qui utilise la reproduction de l'experience.

2.5 Comparaison et conclusion

On affiche ci-dessous la solution trouvée:



Puis, on applique les differents algorithmes sur notre environnement:



On remarque que notre agent (en rouge) est en moyenne meilleur, mais qu'il y a toujours des fluctuations dû au temps court d'entraînement et la complexité de l'environnement de travail.