Apprentissage par Renforcement

M2 DAC

TME 3. Q-Learning

Ce TME a pour objectif d'expérimenter les approches de renforcement value-based vues en cours, notamment l'algorithme Q-Learning (version tabulaire), et les appliquer au problème du GridWorld considéré la semaine dernière. Par contre attention, la commande getMDP est maintenant interdite: on se met dans le cas de figure où les dynamiques de l'environnement sont inconnues. De la même manière, `env.getStateFromObs` n'est pas autorisée, on construit la liste d'états possibles au fur et à mesure de l'exploration, on ne les connaît pas a priori. Il faut explorer pour apprendre les trajectoires efficaces.

Pour aider votre développement, on donne un squelette d'algorithme sur le site de l'UE dans l'archive envTME3.zip. Cette archive contient:

- utils.py : un fichier contenant diverses fonctions utiles pour le chargement de fichiers de configuration yaml et le log de résultats via tensorboard. Pour l'utiliser, vous aurez peut-être à installer pytorch, pyyaml et tensorboard :

  ```
  pip3 install pyyaml --proxy=proxy:3128 --user -U
  pip3 install torch torchvision --proxy=proxy:3128 --user -U
  pip3 install tensorboard --proxy=proxy:3128 --user -U
  ```

- QLearningAgent.py : squelette de l'algorithme à développer. Le chargement des paramètres, l'enregistrement des observations et la création de l'agent sont faits pour vous (du moins dans une version minimale pour la version basique de QLearning). La boucle principale d'apprentissage est également prête à l'emploi (vous n'avez a priori pas à toucher au code donné dans le `main`).

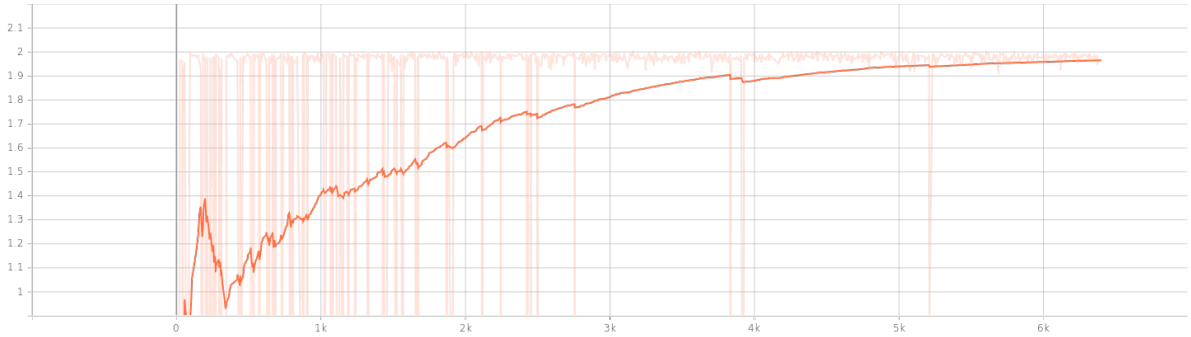
- Répertoire configs : un répertoire visant à contenir des fichiers de configuration yaml. Pour le moment le répertoire ne contient qu'un seul fichier, permettant de paramétrer QLearning sur l'environnement GridWorld. Les valeurs qu'il contient sont les valeurs utilisées pour réaliser la première courbe ci-dessous.

À noter que vous pouvez tester le bon fonctionnement de l'ensemble avant même de commencer à compléter le squelette (recommandé), car il est prévu pour fonctionner par défaut comme un agent aléatoire. Lors du lancement, vous noterez l'affichage dans la console d'une adresse à laquelle vous pourrez observer les courbes de résultats évoluer au cours du temps (vous pourrez bien sûr ajouter d'autres courbes que celles de base prévues dans la version fournie).

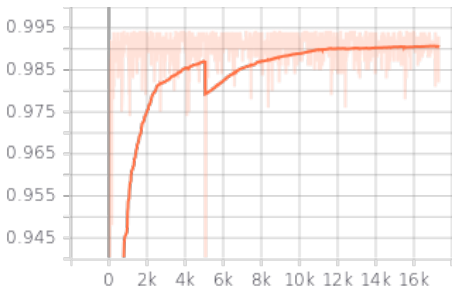
Le travail à réaliser ensuite est le suivant:

- Implémenter la version basique du Q-learning (revient à écrire du code au niveau des TODO du fichier). Tester son fonctionnement sur plusieurs cartes GridWorld et avec divers paramètres d'exploration.
- Implémenter la version SARSA (a priori modification très mineure).
- Implémenter l'extension Dyna-Q (requiert de considérer des structures supplémentaires pour enregistrer le modèle des dynamiques).
- Comparer les résultats sur différentes cartes du jeu (courbes d'apprentissage)
- Bonus facultatif: Tester l'implémentation des traces d'éligibilité

À titre indicatif, voici ce que l'on peut obtenir en apprentissage sur le plan1 de gridworld (et le schéma de reward par défaut), avec Q-learning epsilon-greedy (epsilon initial = 0.1, avec decay de 0.999 à la fin de chaque trajectoire), un discount de 0.99 et un pas d'apprentissage de 0.1 (abscisse=nombre d'épisodes, ordonnée=reward cumulé):



Sur le plan5, on note un net avantage pour DynaQ (à droite, utilisant 100 échantillons du modèle à chaque évènement) par rapport à QLearning classique (même paramètres que précédemment), bien que plus instable et bien plus lent:



Contrairement à QLearning qui n'arrive pas à explorer suffisamment pour aller collecter la récompense du bas de la carte, l'échantillonnage de transitions selon le modèle appris dans DynaQ permet une meilleure propagation des valeurs et évite cette convergence trop rapide vers une politique sous-optimale.