Université Pierre et Marie Curie

Manuel du programmeur

PIAD de Master1 d'Informatique en Intelligence Artificielle et Décision

Semi-supervised Learning Agents

Auteurs :
Lan Zhou
Matthieu Zimmer

Superviseurs :
Paolo Viappiani
Paul Weng

9 mai 2013

Version 1.1

Table des matières

1 Rappel des objectifs

Nous devrons développer une bibliothèque indépendante du domaine dans lequel l'agent évolue. Elle fournira plusieurs algorithmes de base de l'apprentissage par renforcement (Q-Learning, SARSA, ...), plusieurs critères de performance, ainsi qu'une ouverture sur l'apprentissage semi-supervisé : c'est à dire avec les retours de l'expert pris en compte.

Dans un premier temps, l'expert pourra simplement compenser la fonction de récompense en précisant si l'agent a bien ou mal agit. Dans un second temps, si le temps le permet, l'expert pourra également agir sur le choix de l'action à entreprendre lors de l'exploration de l'agent, ou encore dire à l'agent s'il est temps d'exploiter ou d'explorer.

Parallèlement au développement de la bibliothèque, afin d'avoir une application pratique de la théorie, on utilisera ces algorithmes dans le simulateur TORCS sur l'apprentissage automatique de la conduite de voiture sur circuit. Nous modifierons également l'interface TORCS pour intégrer les retours positifs ou négatif de l'expert, voire des retours plus complexes cités précédemment [Zhou and Zimmer, 2013a].

2 Choix d'implémentation

2.1 Séparation Library/Driver

Permettre l'indépendance total entre les 2 entitiés : aucun fichier de la library ne peut inclure un fichier du driver.

2.2 DAction & ActionTemplate

Permet de définir n'importe quelles actions.

2.3 QTable

Transformer un tableau à 5 dimensions en un tableau à 1 dimension pour gagner en performance.

2.4 Fonctor & Features

Flexibilité et la généralité qu'ils apportent.

3 Architecture du logiciel

3.1 Architecture fichiers

Le projet est constitué de 3 dossiers principaux et 2 dossiers auxilières (documentation+rapport) :

- torcs
- library
- driver

torcs/ Ce dossier contient à son tour 3 à 4 dossiers, comme son nom l'indique il relève des fichiers TORCS :

export/ contient simplement des fichiers headers afin de communiquer avec TORCS, ils proviennent directement du code de TORCS et n'ont pas été modifiées.

torcs.base/ (Optionnel) Contient tous les fichiers de TORCS intacts.

torcs.torcs-1.3.4/ (Optionnel) Contient tous les fichiers de TORCS compilés.

xml/ Contient des fichiers de configuration interessants de TORCS qui seront copier dans le répertoire ~/.torcs/ de l'utilisateur. Ils permettent entre autres de gérer la taille de l'écran, mais surtout de définir des courses pour l'apprentissage des agents.

Les 3 scripts contenus dans torcs/ (getTorcs,...) sont décris dans le Manuel de l'Utilisateur [Zhou and Zimmer, 2013b]

library/ Contient tout les algorithmes génériques d'apprentissage par renforcement, ils sont totalement indépendant de TORCS.

lib/ Sert de destination pour la bibliothèque une fois compilée

build/ Contient les fichiers compilés

src/ **include**/ Contient les fichiers sources : bib étant général, et sml pour l'apprentissage par renforcement.

driver/ Contient tout les algorithmes pour l'interfacement entre la bibliothèque et TORCS.

workingState/ Contient des données déjà apprises par les algorithmes (peuvent être utilisé pour ne pas réapprendre de zéro).

cmake/ Contient des fichiers de configuration de CMake, pour permettre une compilation plus aisé.

data/ Contient les fichiers de données nécessaire à l'installation module du robot.

./makeDriver Compile les sources de la library, puis celle des drivers, installe tout les fichiers du module du robot, ainsi que les données d'apprentissage.

./clear Nettoie les fichiers compilés et la bibliothèque générée.

3.1.1 Ajouter un robot

Pour ajouter correctement un robot, il convient de suivre ces différentes étapes :
- Implémenter un robot : choisir si on intègre des feedbacks du clavier, si oui, il doit descendre de DriverFeedback. Si non, il doit descendre de Driver.
- Associer l'instance à un index dans smile.cpp :InitFuncPt et augmenter le nombre de robots si nécessaire (smile.cpp :botname, smile.cpp :botdesc, smile.cpp :NBBOTS).
- Dire à torcs qu'un nouvel index est disponible dans ce module en modifiant le fichier data/smile.xml
- Ajouter une configuration de course qui utilise cet index dans torcs/xml/dtmraceX.xml.
- Recharger le tout avec ./makeDriver

3.1.2 Lancer un apprentissage

Pour lancer un apprentissage, on utilisera ./learnIA <index> [<cpu>].

Avant de lancer un apprentissage, vérifié que le robot a bien son paramètre learn à true, sauf si vous voulez au contraire visualiser ces performances sans ϵ-greedy.

Une fois l'apprentissage lancé, il faut également prendre en compte le paramètre "restrictive_learning" de votre robot, s'il est à true, l'affichage durant l'apprentissage sera moindre car il n'affichera et ne sauvegardera que les apprentissages "pas trop mauvais".

3.2 Developpement

Le projet a été conçu sous une architecture cmake, et pleinement fonctionnelle pour l'IDE Kdevelop (bien que cmake permettent d'utiliser n'importe quel IDE).

4 Description du code

4.1 Eléments utilisés

4.1.1 Boost

On utilise plusieurs fonction de Boost :
- les vectors pour ne pas réinventer la roue
- les unordered_map implémenté via un hachage pour gagner en performance (au lieu d'un arbre)
- la serialization pour sauvegarder/restaurer les objets facilement en xml
- les mutex pour faire de l'exclusion mutuelle entre processus
- filesystem pour tester facilement l'existence d'un fichier
- les functors pour facilité la généralisation du code

4.2 TORCS

On utilise directement ce simulateur dans lequel on intègre notre propre module.

On y utilise notament :
- car.h pour avoir les caractéristiques de la voiture
- track.h pour connaître celles de la route
- robottools.h pour faire des calculs et récupérer des capteurs locaux
- tgfclient.h pour récupérer les évènements claviers

Il convient de lire obligatoirement le tutoriel de TORCS sur les robots pour pouvoir se familiariser au developpement de ce projet [Bernhard, 2003].

4.3 Eléments produits

4.3.1 Library

- Q-Learning
- Q-Learning(λ) avec historique
- Q-Learning(λ) par descente de gradient $Q(s,a) = \theta_0 + \sum_{i=1}^{n} \theta_i \times f_i(s,a)$
- Sarsa
- Sarsa(λ) avec historique

Chacun avec la possibilité de choisir si la trace est accumulative ou non (s'ils ont une trace).
- Actions/Etats discrétisés/continues génériques
- Fonctions d'approximation par tiling prédéfini

4.3.2 Driver

Liste robots prédéfinis et interfacé avec torcs

Index	Algorithme	Etat	Action	Feedback	Classe
0	Q-Learning	distance au centre, tangente à la route	direction	Sans	QLearnDiscr
1	Q-Learning(λ)	distance au centre, tangente à la route	direction, 4 actions vitesse	Sans	QLearnDiscr2
2	Q-Learning(λ) par descente de gradient	distance au centre, tangente à la route	direction, 4 actions vitesse	Sans	QLearnGen
3	Q-Learning(λ) par descente de gradient	distance au centre, tangente à la route vitesse prochain virage	direction, 4 actions vitesse	Contrôle	QLearnGenCmplx
4	Q-Learning(λ) par descente de gradient	distance au centre, tangente à la route	direction, 4 actions vitesse	Superviseur	QLearnGenFdb
5	Sarsa(λ)	distance au centre, tangente à la route, vitesse	direction,	Contrôle	SarsaFdb

- TWorld défini des fonctions de récompenses, de discrétisation, ... tout ce qui a un rapport avec le monde de TORCS
- TFeature prédéfini des fonctors pour les approximations de fonction dans le monde de TORCS
- Driver fourni les méthodes de bases d'intégration d'un agent conducteur
- DriverFeedback fourni les méthodes nécessaires à la récupération des évènements claviers

La fonction de récompense fonctionne de cette manière : le conducteur est récompensé en fonction de la vitesse s'il reste sur les limites de la route

4.4 Paramètres d'implémentation

4.4.1 À la compilation

Durant la compilation de TORCS, on peut choisir ou non la version debug (retirer –enable-debug dans buildTorcs).

Durant la compilation du projet, on peut choisir la version release ou debug, modifier la première ligne de makeDriver et les options du CMakeLists.txt.

4.4.2 Dans le code

Il y a de nombreux paramètres défini pour un robot, en particulier on pourra trouver :

- la structure LearnConfig qui définit si l'apprentissage est restrictif (non enregistré si les performance sont trop mauvaise)
- learn : défini si l'agent apprend ou non
- DECISION_EACH : fréquence de décision d'un agent
- ACTIONS_DIRECTION : le nombre d'actions possibles pour diriger la voiture
- lambda : l'importance de l'historique durant l'apprentissage
- lrate : le taux d'apprentissage

- discount : l'importance du prochain état durant l'apprentissage
- epsilon : politique ϵ-greedy
- road_width : la largeur considérée pour la route
- total_angle : l'angle total considéré
- total_speed : la vitesse total considérée
- aux 3 derniers paramètres il faut associer un nombre d'entier pour les discrétiser
- simu_time : le temps de simulation pour une course
- le choix des associations de dimension pour les fonctions d'approximation
- les malus pour sortie de route dans la fonction de récompense

Références

[Bernhard, 2003] Bernhard, W. (2003). Torcs robot tutorial. `http://www.berniw.org/tutorials/robot/tutorial.html`.

[Zhou and Zimmer, 2013a] Zhou, L. and Zimmer, M. (2013a). *Cahier des charges - PIAD Semi-supervised Learning Agents.*

[Zhou and Zimmer, 2013b] Zhou, L. and Zimmer, M. (2013b). *Manuel de l'Utilisateur - PIAD Semi-supervised Learning Agents.*