Projet RL-lib

Bibliothèque de Reinforcement Learning en Java avec intégration de ProB

Fergouch Yahya Rayyan Hedidar Chaillot Louis

April 27, 2025

Abstract

Ce document présente en détail la bibliothèque **RL-lib** développée dans le cadre du projet de Méthodes Numériques pour l'IA. Elle fournit une infrastructure Java permettant de charger et d'exécuter des modèles formels de type B Machines via l'outil ProB, et d'expérimenter différents algorithmes d'apprentissage par renforcement (*Reinforcement Learning*). Utilisez le sommaire pour naviguer rapidement entre les sections.

Contents

1 Structure du code **2**
- 1.1 Explication des composants . . . 2
- 1.2 Résumé . . . 3

2 Utilisation **3**
- 2.1 Compilation et exécution . . . 3
- 2.2 Déroulement de l'application . . . 4

3 Organisation et choix de conception **4**
- 3.1 Séparation claire des responsabilités . . . 5
- 3.2 Philosophie d'architecture . . . 5
- 3.3 Choix techniques spécifiques . . . 5

4 Algorithmes utilisés, choix de récompenses, difficultés et apprentissages **6**
- 4.1 Correspondance Algorithme - Problématique . . . 6
- 4.2 Choix des fonctions de récompense . . . 6
- 4.3 Difficultés rencontrées . . . 6
- 4.4 Apprentissages issus du projet . . . 7
- 4.5 Analyse des compétences développées . . . 7

5 Conclusion **7**

1 Structure du code

Voici l'organisation du projet `rl-lib` :

```
|- agent
|  |- Agent.java
|  |- BanditGradientAgent.java
|  |- EpsilonGreedyAgent.java
|  |- PolicyIterationAgent.java
|  |- QLearningTicTacToeAgent.java
|  |- UCBAgent.java
|  -- ValueIterationAgent.java
|- analysis
|  -- AgentAnalyzer.java
|- graph
|  -- GraphGenerator.java
|- reward
|  |- RewardFunction.java
|  |- SimpleRewardFunction.java
|  |- TicTacToeRewardFunction.java
|  -- YouTubeRewardFunction.java
|- tictactoe
|  -- TicTacToeEpisodeManager.java
|- App.java
|- Evironnement.java
|- MyProb.java
|- Runner.java
|- SchedulerRunner.java
|- SimpleRunner.java
|- TicTacToeRunner.java
-- YouTubeRunner.java
```

1.1 Explication des composants

- **agent** : Contient les différentes stratégies d'agents d'apprentissage par renforcement (ϵ-*Greedy*, *UCB*, *Gradient Bandits*, *Policy Iteration*, *Value Iteration*, *Q-Learning* pour TicTacToe). Tous héritent de `Agent.java`.
- **analysis** : Outils d'analyse et d'évaluation des agents. `AgentAnalyzer.java` pour comparer leurs performances.
- **graph** : Génération de graphes pour visualiser les résultats expérimentaux (ex. : convergence des stratégies).
- **reward** : Définit les fonctions de récompense adaptées à chaque environnement : Simple, YouTube, TicTacToe. `RewardFunction.java` est une interface générale.
- **tictactoe** : Gestion spécifique d'épisodes de jeu pour TicTacToe (exécution de parties, suivi des états et des récompenses).
- **App.java** : Point d'entrée principal du projet, lance les différents tests d'agents et d'environnements.

- **Evironnement.java** : Interface simplifiée pour interagir avec un environnement B-Machine chargé dans ProB.
- **MyProb.java** : Encapsulation de l'API ProB pour charger et manipuler les machines B.
- **Runner.java** : Classe abstraite pour exécuter une machine B (initialisation, exploration des transitions).
- **SimpleRunner.java** : Exécution de la machine `SimpleRL.mch` (problème de bandit à 3 bras, films AB/C).
- **SchedulerRunner.java** : Exécution de la machine `scheduler_main.mch` (planification de processus).
- **TicTacToeRunner.java** : Exécution de la machine `tictac.mch` (jeu du morpion).
- **YouTubeRunner.java** : Exécution de la machine `YouTube.mch` (modélisation d'un système de recommandations vidéo).

1.2 Résumé

Le projet est organisé de manière modulaire :

- **Agents** : différentes stratégies d'apprentissage par renforcement.
- **Environnements** : différentes spécifications de problème (Simple, Scheduler, TicTacToe, YouTube).
- **Outils** : Analyse, graphes, fonctions de récompense.

2 Utilisation

2.1 Compilation et exécution

Pour utiliser le projet `rl-lib`, suivez les étapes suivantes :

1. Nettoyer le projet (supprimer les anciens fichiers compilés) :

```
mvn clean
```

2. Compiler le projet :

```
mvn compile
```

3. Lancer l'application principale (`fr.polytech.mnia.App`) :

```
mvn exec:java
```

2.2 Déroulement de l'application

Après lancement, l'utilisateur interagit avec l'application à travers plusieurs étapes :

- **Choix de l'environnement** :
  1 SimpleRL
  2 YouTube
  3 TicTacToe
- **Sélection des agents** :
  - Environnements SimpleRL et YouTube :
    1 ϵ-Greedy (nécessite de définir ϵ)
    2 UCB
    3 Bandit Gradient (nécessite de définir α)
  - Environnement TicTacToe :
    4 Value Iteration (paramètres γ, θ)
    5 Policy Iteration (paramètre γ)
    6 Q-Learning (paramètres α, γ, ϵ)
- **Mode d'affichage pendant l'entraînement** :
  1 Affichage détaillé (verbose)
  2 Mode silencieux
- **Définir les paramètres d'entraînement** :
  - Nombre d'étapes pour SimpleRL et YouTube.
  - Nombre d'itérations maximum pour TicTacToe.
- **Exécution de l'entraînement** :
  - Chaque agent sélectionné est entraîné sur l'environnement choisi.
  - Une analyse est effectuée après l'entraînement (`AgentAnalyzer`).
  - Si applicable (ex: Q-Learning pour TicTacToe), l'agent affiche le chemin optimal appris.
- **Fin de l'expérience** :
  - L'application termine après avoir entraîné et analysé tous les agents.

3 Organisation et choix de conception

Le projet `rl-lib` a été conçu avec une approche modulaire et extensible, afin de faciliter l'ajout de nouveaux agents d'apprentissage, de nouveaux environnements et de nouveaux outils d'analyse sans modifier la structure existante.

3.1 Séparation claire des responsabilités

Nous avons structuré le projet en plusieurs modules, chacun ayant une responsabilité bien définie :

- **agent** : Contient les différentes stratégies d'apprentissage par renforcement. Chaque agent est une classe indépendante implémentant des algorithmes spécifiques (ϵ-*Greedy*, *UCB*, *Bandit Gradient*, *Value Iteration*, *Policy Iteration*, *Q-Learning*).
- **runner** : Chaque environnement B est encapsulé dans un `Runner` spécifique (`SimpleRunner`, `SchedulerRunner`, `TicTacToeRunner`, `YouTubeRunner`), facilitant l'initialisation et l'exploration de l'état.
- **reward** : Définit les fonctions de récompense propres à chaque environnement, permettant d'adapter facilement le critère d'apprentissage sans modifier les agents.
- **analysis** : Outils pour évaluer les performances des agents (`AgentAnalyzer`) et générer automatiquement des graphes (`GraphGenerator`).
- **tictactoe** : Gestion spécifique d'épisodes pour l'environnement TicTacToe, avec la classe `TicTacToeEpisodeManager`.
- **App.java** : Point d'entrée unique, qui guide l'utilisateur dans l'expérimentation (choix d'environnement, d'agents, paramètres d'entraînement, analyse).

3.2 Philosophie d'architecture

Notre approche repose sur les principes suivants :

- **Modularité** : Chaque fonctionnalité est isolée dans son propre module, facilitant l'évolutivité du projet.
- **Extensibilité** : Ajouter un nouvel agent, un nouvel environnement ou une nouvelle fonction de récompense nécessite uniquement d'ajouter une nouvelle classe sans toucher aux classes existantes.
- **Clarté** : Le découpage en packages permet de retrouver rapidement où chaque élément est défini et utilisé.
- **Réutilisabilité** : Les classes d'agent et les fonctions de récompense peuvent être réutilisées dans plusieurs environnements différents.
- **Expérimentation facilitée** : L'application guide l'utilisateur pas à pas pour configurer ses expériences, sans avoir besoin de modifier le code.

3.3 Choix techniques spécifiques

- Utilisation de l'outil **ProB** pour animer et explorer automatiquement les B-Machines, afin de s'abstraire de la complexité de la modélisation.
- Utilisation de **Maven** pour la gestion du cycle de vie du projet (compilation, exécution), garantissant portabilité et simplicité d'installation.
- Génération automatique de graphes de résultats avec **GraphGenerator**, permettant une analyse rapide des performances des agents.
- Interaction utilisateur en ligne de commande (`Scanner`) pour rendre le projet plus interactif et adaptable.

4 Algorithmes utilisés, choix de récompenses, difficultés et apprentissages

4.1 Correspondance Algorithme - Problématique

Dans ce projet, nous avons associé différents algorithmes d'apprentissage par renforcement à des problématiques spécifiques selon la nature de l'environnement :

- **SimpleRL (Bandits à 3 bras)** : Nous avons utilisé ϵ-Greedy, UCB et Bandit Gradient. Le problème est de choisir entre trois actions (A, B, C) et de maximiser la récompense, ce qui correspond exactement au cadre classique des bandits manchots. Chaque agent apprend à privilégier les films préférés.
- **YouTube (Recommandations adaptatives)** : Nous avons également utilisé ϵ-Greedy, UCB et Bandit Gradient. Ici, l'objectif est de recommander des vidéos en fonction du comportement utilisateur qui évolue au fil du temps. Le Bandit Gradient est particulièrement pertinent car il adapte dynamiquement la politique de sélection.
- **TicTacToe (Jeu tour par tour)** : Nous avons utilisé Value Iteration, Policy Iteration et Q-Learning. Le problème est beaucoup plus complexe : il s'agit d'apprendre une politique optimale dans un environnement séquentiel avec états et transitions. L'apprentissage par valeur d'état est donc plus adapté que les stratégies simples de bandits.

4.2 Choix des fonctions de récompense

Pour chaque environnement, nous avons conçu une fonction de récompense spécifique :

- **SimpleRewardFunction** : Donne une récompense fixe selon que l'action (film) est aimée ou non.
- **YouTubeRewardFunction** : Utilise la variable `duration` pour refléter l'intérêt de l'utilisateur pour une vidéo (par exemple, regarder 80% d'une vidéo = forte récompense).
- **TicTacToeRewardFunction** : Évalue la grille pour déterminer si un joueur gagne, perd ou continue, avec des scores adaptés pour guider l'agent vers la victoire.

Le choix de créer des fonctions de récompense séparées permettait de rendre le code flexible et de pouvoir expérimenter facilement différentes stratégies d'entraînement.

4.3 Difficultés rencontrées

Nous avons rencontré plusieurs difficultés tout au long du projet :

- **Intégration avec ProB** : Comprendre le fonctionnement de ProB et comment interagir dynamiquement avec des B-Machines a demandé un temps d'adaptation important.
- **Gestion des environnements dynamiques** : L'environnement YouTube est non déterministe (les préférences changent), ce qui rendait l'apprentissage plus difficile à stabiliser.
- **Paramétrage des agents** : Trouver les bons hyperparamètres (ϵ, α, γ, θ) pour obtenir une convergence rapide et éviter l'overfitting a nécessité beaucoup d'expérimentations.

4.4 Apprentissages issus du projet

Ce projet nous a permis de développer de nombreuses compétences techniques et méthodologiques :

- **Apprentissage du Reinforcement Learning** : Comprendre en profondeur les algorithmes classiques et leur mise en œuvre concrète.
- **Modélisation formelle** : Travailler avec des B-Machines nous a forcés à être rigoureux dans la définition des états et des transitions.
- **Organisation d'un projet Java complexe** : Structurer un projet modulaire, extensible et propre (Maven, packages clairs, documentation).
- **Expérimentation scientifique** : Paramétrer, tester et analyser de manière méthodique les résultats obtenus.

4.5 Analyse des compétences développées

En réalisant ce projet, nous avons renforcé plusieurs compétences clés :

- **Compétences en programmation orientée objet (Java)** : Design de classes abstraites, encapsulation, modularité.
- **Compétences en Machine Learning** : Implémentation d'algorithmes RL, analyse critique des performances.
- **Capacité d'adaptation** : Prendre en main un outil complexe (ProB) et construire autour un projet fonctionnel.
- **Gestion de projet** : Structurer le travail, planifier les tâches, documenter proprement, rendre le projet facilement utilisable par d'autres.

5 Conclusion

Ce projet nous a permis de mettre en pratique de manière concrète les concepts fondamentaux de l'apprentissage par renforcement, en les appliquant à des environnements variés modélisés par des B-Machines.

Nous avons conçu une architecture modulaire et extensible, qui facilite l'ajout de nouveaux agents, de nouveaux environnements et de nouvelles fonctions de récompense. Ce travail nous a également appris à interagir avec un outil complexe comme ProB, à gérer la difficulté d'environnements dynamiques, et à expérimenter différents paramètres d'apprentissage.

Au-delà des compétences techniques en programmation orientée objet et en Machine Learning, ce projet nous a permis de développer une méthodologie de travail rigoureuse : structuration du code, gestion de projet, documentation claire, et capacité à analyser des résultats expérimentaux.

Enfin, ce projet ouvre des perspectives intéressantes : il serait possible d'aller plus loin en implémentant d'autres algorithmes (comme DQN pour TicTacToe), en testant d'autres stratégies adaptatives pour YouTube, ou en généralisant l'environnement pour d'autres jeux de décision séquentiels.

Nous sommes satisfaits du travail accompli, tant sur le plan technique que sur le plan méthodologique, et nous avons pris plaisir à explorer en profondeur les enjeux de l'apprentissage par renforcement.