Final Project Report on Sokoban for Reinforcement Learning

Author: Le Hoa

October 2024

Faculty of Electrical and Electronic Engineering, Phenikaa University, Yen Nghia, Hanoi, 12116, Vietnam. Graduate University of Sciences and Technology, Vietnam Academy of Science and Technology, Hanoi, Vietnam. Faculty of Technology Education, Hanoi National University of Education

Code Student: 22010984
Class: AI - RB
Instructor: Vu Hoang Dieu
Deadline: 11/10/2024

Abstract

Reinforcement Learning (RL) is a key subfield of machine learning that focuses on training agents to make decisions by interacting with an environment and maximizing cumulative rewards. In contrast to supervised learning, where models learn from labeled data, RL emphasizes learning through trial and error, allowing agents to discover optimal policies by exploring and exploiting available information. Central concepts in RL include states, actions, rewards, and policies, often formalized through a Markov Decision Process (MDP).

RL has made significant contributions across various domains, including gaming, robotics, and finance. Notably, RL has achieved superhuman performance in complex games such as Chess, Go, and StarCraft by discovering novel strategies beyond human capabilities. In robotics, RL enables machines to adapt to dynamic environments, facilitating tasks such as autonomous navigation and manipulation. In finance, RL is used in algorithmic trading and portfolio management to optimize decision-making under uncertain market conditions.

However, despite these advancements, RL faces challenges, including the high computational cost of training, exploration-exploitation trade-offs, and issues related to learning stability. This paper presents the core principles of RL, explores its key applications, and discusses strategies for improving learning efficiency and stability in RL models.

Contents

1 Introduction

Reinforcement Learning in Sokoban

Reinforcement Learning (RL) involves an agent that interacts with an environment to maximize cumulative rewards. Key components of RL include:

- **Agent:** The learner or decision maker that interacts with the environment, which in this case is the Sokoban game.
- **Environment:** The setting in which the agent operates, represented by the Sokoban game board.
- **State (s):** A representation of the environment at a given time, indicating the positions of the player and the boxes on the board.
- **Action (a):** The choices available to the agent, including moving or pushing boxes, which affect the state of the environment.
- **Reward (r):** Feedback received after taking an action in a state, guiding the agent's learning process.
- **Policy ():** A strategy that defines the agent's behavior, mapping states to actions.

A critical concept in RL is the **exploration-exploitation trade-off**, where the agent must balance between exploring new actions (exploration) and leveraging known rewarding actions (exploitation). This is often modeled using **Markov Decision Processes (MDP)**, which provide a formal framework for decision-making where outcomes are partly random and partly under the control of the decision maker.

1.1 Overview of Algorithms

1.1.1 Q-Learning

Q-Learning is an off-policy, model-free algorithm. It learns the value of action-reward pairs (Q-values) without requiring a model of the environment. The core update rule for Q-values is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

where α is the learning rate, γ is the discount factor, and r is the reward received after transitioning from state s to state s' by taking action a.

1.1.2 SARSA

SARSA (State-Action-Reward-State-Action) is an on-policy algorithm, which means it updates the Q-values based on the actions actually taken by the agent. The update rule differs from Q-Learning as it considers the next action chosen by the policy:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s', a') - Q(s,a) \right] \tag{2}$$

1.1.3 Monte Carlo

Monte Carlo is a reinforcement learning algorithm that updates the Q-values based on the actual returns obtained from full episodes. Unlike temporal-difference methods, which update the Q-values step-by-step, Monte Carlo waits until the entire episode finishes and then computes the cumulative reward (return) for each state-action pair encountered in the episode. The Q-values are updated by averaging the returns across multiple episodes, and the update rule can be written as:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[G_t - Q(s,a) \right], \tag{3}$$

where G_t is the return (total accumulated reward) starting from time step t. The main distinction between SARSA and Q-Learning is that SARSA uses the action selected by the policy, while Q-Learning uses the action that maximizes the Q-value.

2 Methodology

2.1 Problem Definition

The Sokoban problem is framed as a grid-world environment where an agent (the player) pushes boxes to designated target locations (goals). The environment consists of the following components:

- **State Space**:Each state represents a unique configuration of the grid, including the position of the agent and the positions of the boxes, the positions of the traps, and the goals. With a grid whose rows are states and whose columns are actions, the state space consists of 100 states, with each state encoded based on the positions of the agent and the boxes.
- **Action Space**: The agent can perform a set of actions, which include:
  - No action
  - Move Up
  - Move Down
  - Move Left
  - Move Right

- Push Up
- Push Down
- Push Left
- Push Right

The game provides 9 actions to interact with the environment. Push and Move actions into the directions Up, Down, Left and Right. The No Operation action is a void action, which does not change anything in the environment. The mapping of the action numbers to the actual actions looks as follows.

- **Reward Structure**: Finishing the game by pushing all on the targets gives a reward of 10 in the last step. Also pushing a box on or off a target gives a reward of 1 respectively of -1. In addition a reward of -0.1 is given for every step, this penalizes solutions with many steps

  we have 1 more trap state. when agent move trap will be minus 5 and return to original position. Similarly when the agent pushes the box into the trap, the box will be reset to its original position and the reward will be -10 to make the agent avoid the trap.

Action	**Reward**
Perform Step	-0.1
Push Box on Target	1.0
Push Box off Target	-1.0
Push all boxes on targets	10.0
Move into Trap	-5.0
Push Box into Trap	-10.0

Table 1: Reward structures.

2.2 Method

For this project, I have chosen to implement the SARSA algorithm for solving the Sokoban problem due to its on-policy nature, which allows for more stable learning in dynamic environments.

The reasons for selecting SARSA over other methods include:

- Exploration: SARSA inherently encourages exploration, as it updates Q-values based on the actions taken by the agent. This can help the agent learn more robust strategies in the Sokoban environment, where optimal paths may require exploring multiple routes.

- Training: The SARSA algorithm performs well, along with the Q-Learning and Monte Carlo algorithms, when the total reward obtained is close to 10.

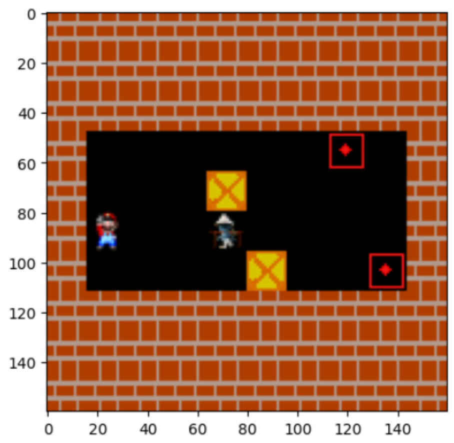

Figure 1: Sokoban environment

- Performs: Here we use the gym library from open Ai, a powerful library for solving the Sokoban game Sokoban Gym is a grid-world environment, in which the actions of the agent take place in a square space like a chessboard. Each Sokoban Gym is a grid environment, in which the agent's actions take place in a square space like a chessboard. Each] x square on the grid can contain an object such as: agent, box or wall, ... and each object will return the coordinate position in the form of an image.

  Each component will be represented by a symbol below. each coordinate will present for each

  0: (-1, 0), 1: (1, 0), 2: (0, -1), 3: (0, 1) article caption graphicx

Symbol	**Description**
#	Wall
@	Player
$	Box
.	Goal
T	Special (Trap)

Figure 2: Sokoban Game Symbols

Action	**Description**
0	Move Up
1	Move Down
2	Move Left
3	Move Right

Figure 3: Sokoban Game Actions

Here, I have set up each state in the environment so that I can control and adjust it easily when I want to change. The coordinates will follow the position of the row and column in the field.

Suppose the origin coordinate of the agent is defined as $[0, 0]$. In this coordinate system, when the agent moves to the right, its updated position becomes $[0, 1]$. Here, the first value remains 0 (indicating the row), while the second value changes to 1 (indicating the column).

3 Reinforcement learning Algorithm

3.1 EXPERIMENTAL DESIGNS

```
Sarsa Algorithm
[1] Initialize $Q\_table$ as a table with dimensions $[numberofstates \times numberofactions]$

each episode Initialize state $s$
Choose action $a$ using epsilon-greedy policy
episode not done Take action $a$, observe reward $r$ and next state $s'$
$Choosenextactiona'$ using epsilon-greedy policy
Update $Q(s, a)$:
```

$$\alpha = 0.85,\ \gamma = 0.90,\ \epsilon = [0.8, 0.5, 0.2]$$

In this study, we utilize the Sokoban environment, a popular setting in reinforcement learning where the player must move boxes to target locations within a maze. This environment is structured as a matrix, with states representing the positions of the player and the boxes. The hyperparameters used in this environment include:

- Learning Rate (α): 0.85
- Discount Factor (γ): 0.90
- Exploration Rate (ϵ): 0.2, 0.5, 0.8
- Number of Episodes: 5000

3.2 Hyperparameter Tuning

Hyperparameter tuning is crucial for optimizing the performance of reinforcement learning algorithms. For each algorithm, we experiment with different values of the learning rate, discount factor, and exploration rate to identify the most effective combinations. The tuning process involves running multiple training sessions, analyzing the convergence behavior, and assessing the stability of the learned policies. This iterative approach allows us to refine the parameters to achieve better performance in the Sokoban environment.

3.3 RESULT

Sarsa algorithm is faster than other algorithms when converging at reward 10. After running 5000 iterations, the average reward of the Sarsa algorithm is up to 10.98. While Q-learning has an average reward of 2.73 and Monte Carlo is only about 0.23.

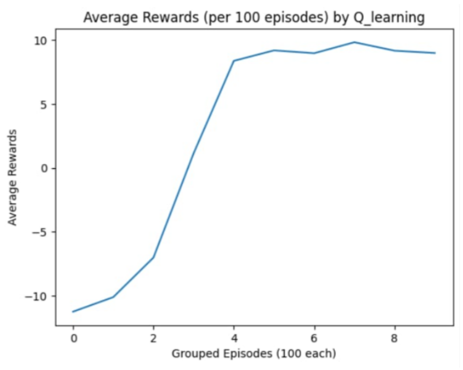

Figure 4: Sarsa

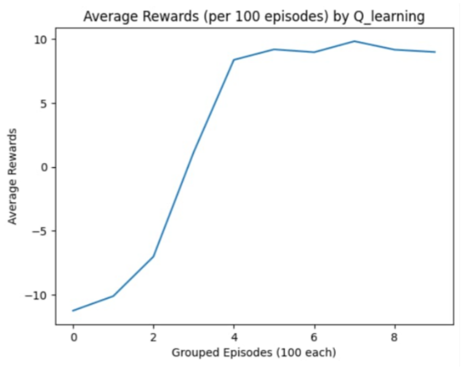

Figure 5: Montel Carlo

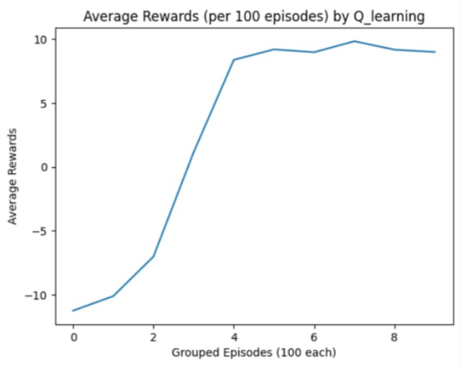

Figure 6: Q learning

[1] Experience with Sarsa.
Sarsa is the best algorithm in this environment. Therefore, I decide to try another episode with epsilon values of 0.2, 0.5, and 0.8 to compare the agent's learning rates. The convergence speeds are similar.

Epsilon	**Average Reward**
0.2	9.85
0.5	9.95
0.8	9.921

Table 2: Rewards of Sarsa with Different Epsilon Values

[2] Additionally, Q-learning runs quite well despite the slower convergence. Experience with Q-learning.

Epsilon	**Average Reward**
0.2	2.3
0.5	-2.6
0.8	-0.39

Table 3: Rewards of Q-learning with Different Epsilon Values

[4] **Comparison of Algorithms**

In this section, we provide a comparison of the Sarsa, Q-learning, and Monte Carlo algorithms based on three key factors: stability, convergence speed, and the optimal policy discovered. Sarsa demonstrates higher stability compared to Q-learning and Monte Carlo in the Sokoban environment, attributed to its on-policy nature, which allows for smoother adjustments of the Q-values during training. Furthermore, Sarsa exhibits faster convergence, achieving an average reward of 10.98 after 5000 iterations, whereas Q-learning reaches only 2.73, and Monte Carlo achieves approximately 0.23. Additionally, Sarsa finds optimal policies more effectively in Sokoban, particularly in complex state scenarios. This superiority is evident in situations with similar states or multiple actions leading to the same outcome, where Sarsa can adjust Q-values based on the actual actions taken, thereby maintaining a more optimal policy. In contrast, Q-learning may converge to a suboptimal policy in such cases due to updates not reflecting the actual actions performed. Overall, Sarsa stands out as a robust algorithm in the Sokoban environment, characterized by high stability, rapid convergence, and the ability to discover effective optimal policies compared to other algorithms.

[5] **Challenge:**
In the Sokoban environment, I have added a feature to receive rewards when approaching the box as quickly as possible. By receiving a reward of +1, the agent is encouraged to adjust its actions to reach the box faster. However, the experimental results indicate that this feature does not significantly improve performance. Training is relatively slow, and after 5000 epochs, the agent only achieves an average reward of 0.82. This suggests that while the idea of rewarding proximity to the box may seem beneficial, it does not sufficiently accelerate the learning process in this context.

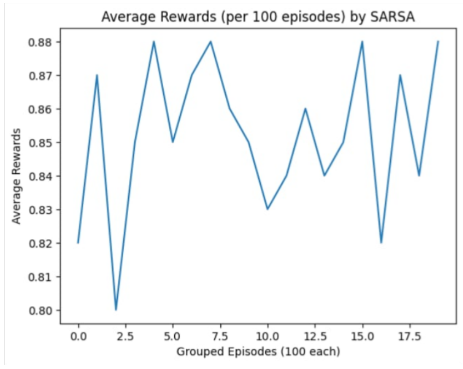

Figure 7: Performance of Sarsa Algorithm in Sokoban Environment

Because 1 box will have 4 faces corresponding to 4 positions. And I am planning to set up the agent to be able to reach the coordinates of the 2 faces in front of the agent. In order to help the agent reach the box as quickly as possible. Minimize the situation of getting lost and not reaching the box.

If multiple boxes are close together, the agent should assess the surroundings to determine which box is the most accessible. It needs to scan for obstacles such as walls, other boxes, or even another agent. By expanding its search around each box, the agent can evaluate the number of free spaces adjacent to it, preferring boxes with more open surroundings for easier movement. Additionally, the agent should be aware of narrow corridors or dead-ends where pushing a box could become difficult or impossible. To further enhance its decision-making, the agent may adjust its epsilon value to favor exploration, allowing it to gather more information about potential obstacles and paths.

4 Conclusion

4.1 Summary

In this project, we compared the performance of traditional reinforcement learning algorithms such as Sarsa, Q-learning, and Monte Carlo in the Sokoban environment. Our findings highlight that Sarsa outperforms the other algorithms in terms of stability and convergence speed, achieving an average reward of 10.98 after 5000 iterations. While Q-learning and Monte Carlo algorithms show slower convergence, they may still have potential in different or less complex environments.

4.2 Lessons Learned

Throughout the project, we learned the significance of selecting the appropriate reinforcement learning algorithm depending on the task. Sarsa's on-policy nature provided better results in this context, emphasizing the importance of policy control in environments like Sokoban. Additionally, we realized that carefully designing the reward structure can significantly impact the agent's learning efficiency and final performance. This reinforces the importance of balancing exploration and exploitation in RL techniques.

4.3 Future Work

For future work, we suggest exploring the use of deep reinforcement learning (DRL) methods such as Deep Q-Networks (DQN) or Proximal Policy Optimization (PPO) to handle more complex environments. Additionally, testing these traditional RL algorithms in dynamic and multi-agent environments may reveal new insights. Improving reward functions and fine-tuning hyperparameters may further enhance agent performance in the Sokoban environment.

I plan to implement a feature that allows two agents to collaborate in the environment. This will enable them to communicate about their positions, the locations of boxes, and any obstacles they encounter. By coordinating their actions, they can maximize their efficiency in tasks like pushing boxes.

For example, if one agent is blocked by an obstacle, the other can navigate around to assist or clear the path. This communication will enhance their problem-solving abilities, making the agents more effective in exploring and interacting with the environment together. Overall, this feature aims to create a more dynamic and cooperative experience.

5 References

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, vol. 135. MIT Press, Cambridge, 1998.

[2] A. Duburcq, F. Schramm, G. Boéris, N. Bredeche, and Y. Chevaleyre, "Reactive stepping for humanoid robots using reinforcement learning: Application to standing push recovery on the exoskeleton Atalante," *arXiv preprint arXiv:2203.01148*, 2022.

[3] L. Wang, J. Liu, H. Shao, W. Wang, R. Chen, Y. Liu, and S. L. Waslander, "Efficient reinforcement learning for autonomous driving with parameterized skills and priors," *arXiv preprint arXiv:2305.04412*, 2023.

[4] M. Schrader, *Gym-Sokoban environment*, GitHub repository. Available at: https://github.com/mpSchrader/gym-sokoban.

[5] J.-C. Walter and G. Barkema, "An introduction to Monte Carlo methods," *Physica A: Statistical Mechanics and its Applications*, vol. 418, pp. 78–87, Jan. 2015.

[6] A. Garriga-Alonso, M. Taufeeque, and A. Gleave, "[Paper title]," *Journal or Conference*, vol. X, pp. 1-10, 2021.