Sharmili Srinivasan

Instructor: Shalabh Bhatnagar

Course name: Reinforcement Learning (CCE)

10, June 2020

Sudoku solving using RL: Comparative study

In this project, we try to solve Sudoku using Reinforcement Learning. In addition, we use this use-case to understand performance and comparison among following RL techniques.

- Monte Carlo On policy (ε-soft policy)
- Temporal Difference – SARSA (on policy)
- Temporal Difference – Q Learning (off policy)

RL concepts: Overview

Reinforcement learning is a machine learning approach to train agents to learn from experience. The main goal is to train models to make decisions (actions) in a given context (state) to maximize a numeric output (reward) received from the environment in response to the action (Figure 1).

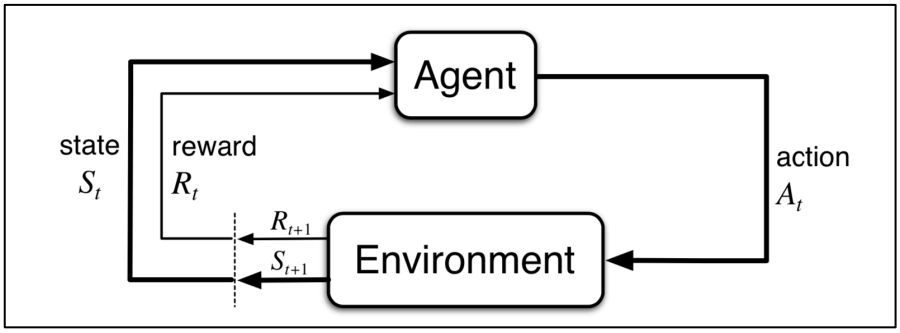

Fig. 1Agent-Environment interaction

Following are the techniques explored in this project:

1. *On-policy every-visit Monte Carlo for ε-soft policy*

   - Monte Carlo methods sample and average returns for each state-action pair.
   - In every-visit method, returns are averaged following all visits to state-action *(s,a)*.
   - The policy used to make decisions is improved in on-policy method.
   - We use a soft policy, i.e. probability of choosing any action in any given state is greater than zero. In specific, we use ε-greedy policy where probability of choosing an action follows below probability

$$\pi\left(\frac{a}{S_t}\right) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|\mathrm{A}(S_t)|} & if\ a = A^* \\ \frac{\epsilon}{|\mathrm{A}(S_t)|} & otherwise \end{cases}$$

   where A^* represents optimal action (greedy)

2. *SARSA: On-policy Temporal Difference*

   - In TD methods, we bootstrap (like in Dynamic Programming) and also learn from episodes (like in Monte Carlo) without any prior knowledge of environment dynamics.
   - Being on-policy, we update and improve the policy we use for making decisions.
   - We use ε-greedy policy, in specific. (Refer above for the policy definition)
   - We use below update rule for updating state-action values.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

3. *Q Learning: Off-policy Temporal Difference*
   - Being off-policy, we use a behavior policy (ε-greedy) to make decisions and improve upon a target policy (greedy).
   - Following is the update rule used

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Problem setup: Sudoku

Sudoku is a number puzzle with an objective to fill given square grid (e.g. *9*9*) with numbers such that number does not repeat in any row, column or a sub-square (e.g. *3*3*).

A Sudoku puzzle grid is provided with few numbers (clues) to start with. Depending on number of clues and their placement, complexity of solution varies from very easy to extremely difficult for a human solver.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Fig. 2 Sudoku grid sample

Solution: Solving Sudoku using RL

For computational simplicity, we consider *Mini Sudoku (4*4)* for our project. Also, we provide minimum *12* clues. This is to ensure convergence as well as plotting of results for multiple runs.

Sudoku can be modelled as a finite MDP (Markov Decision Process) which has following properties.

- Number of states and actions are finite. Though, these can grow exponentially with increase in grid size.

  To be specific,

  - Number of states in a *4*4* grid is *(4+1) ^16*.

    (1 for unfilled state)

  - Number of actions is *4*(unfilled positions)* for every state.
- Each state can be considered to have Markov property – the state depends only on its previous state and not on history of states.

We use three techniques specified in last section to train an agent to solve a *mini sudoku*. We also understand and compare the three methods with this use-case as the base.

Results: Comparative study

Figure 3 provides comparative performance of three techniques for solving a *mini sudoku*. It provides a plot of percentage of success (i.e. grid solved successfully) vs number of episodes. It is plotted *1000* episodes averaged over *10000* runs.

Following are the observations/ comments about the techniques used. These are derived from the above observed experiment.

- Monte Carlo method takes a greater number of episodes than temporal difference methods, initially to learn.
- Initial learning curve of SARSA and Q learning are similar, though Q learning seems to converge faster compared to SARSA.
- Q learning is the quickest to learn with a smaller number of episodes.

- Beyond 100 episodes, all three techniques have stabilized over optimal performance of around 70%.

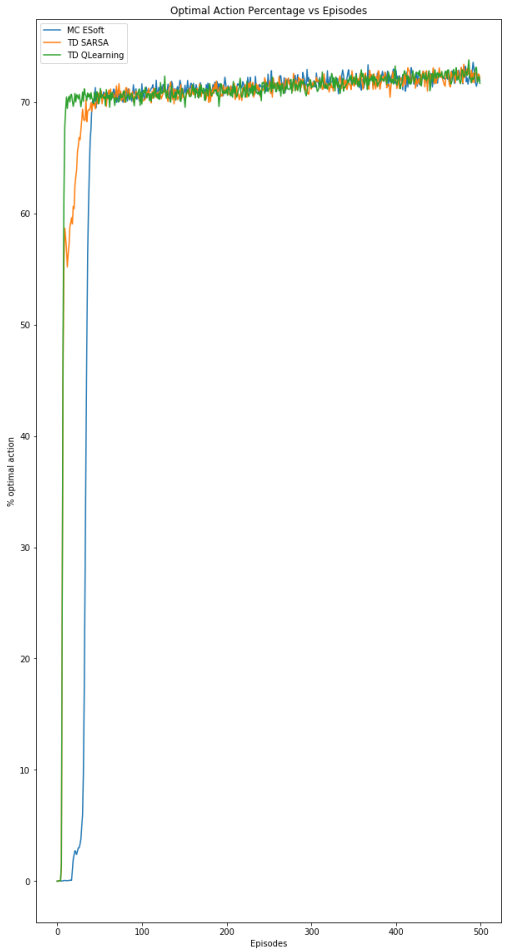

Fig. 3 Comparative study using sudoku episodes

Conclusion: Future scope

This project was performed with following constraints to achieve convergence with limited computation. Each of these can be worked upon further.

- The grid size is limited to *4*4* (mini sudoku). Other sizes like *9*9* can be experimented.
- Number of clues has a minimum limit of *12*. This can be reduced up to *4* for Mini Sudoku (*4*4*).

- The hyperparameters (like learning rate (α) and discount factors (γ)) of the techniques can be tuned for better performances.
- We have limited the scope of this project to study only three of the Reinforcement techniques. We can extend the experiments to explore more RL algorithms.
- Optimization techniques can be employed to handle explosion of state space for higher grids or lower clues.

References

1. GitHub link of the project:

   https://github.com/sharmilisrinivasan/RL/tree/project/IISc_Project

2. Richard S. Sutton and Andrew G. Barto. 2018. Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA.