National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

CS 368: Reinforcement Learning

Lab 07: Model Free Reinforcement Learning - Control

(SARSA Algorithm)

Date: 01 November 2024

Time: 02:00 PM – 5:00 PM

Instructor: Dr. Zuhair Zafar

Lab 07: Model Free Control (SARSA Algorithm)

Objectives

Given a simulated environment, find the optimal policy using SARSA (0) algorithm.

Tools/Software Requirement:

Google Colab, Python, Gymnasium Library

Introduction

Model-free reinforcement learning figures out the best way to act without needing a model of the environment. It's like learning to play a game by trying over and over, rather than reading the rule book. Over time, it learns what actions give good results through trial and error. It works well in situations where we can't predict what will happen next or don't know all the rules.

SARSA Algorithm

In control task, the objective is to find an optimal policy rather than evaluating a policy. SARSA implements a $Q(s,a)$ value-based generalized policy iteration (GPI) and naturally follows as an enhancement from the $\epsilon - greedy$ policy improvement step of MC control.

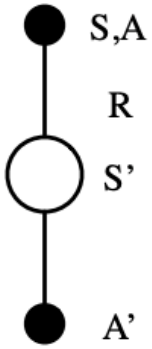
We meet also here the familiar two steps:

1. The first is a technique for learning the Q-function via TD-learning that we have seen in the prediction section. This is similar to evaluating a policy.
2. The second is a method for evolving the policy using the learned Q-function.

SARSA - TD Learning

In the TD prediction section, we have met the TD prediction step for $V(s)$ but for control we need to predict the $Q(s,a)$.

The TD(0), also known as single-step TD, tree for SARSA is shown below:



SARSA action-value backup update tree. Its name is attributed to the fact that we need to know the *State-Action-Reward-State-Action* before performing an update.

Following the value estimate of temporal difference (TD) learning, we can write the value update equation as:

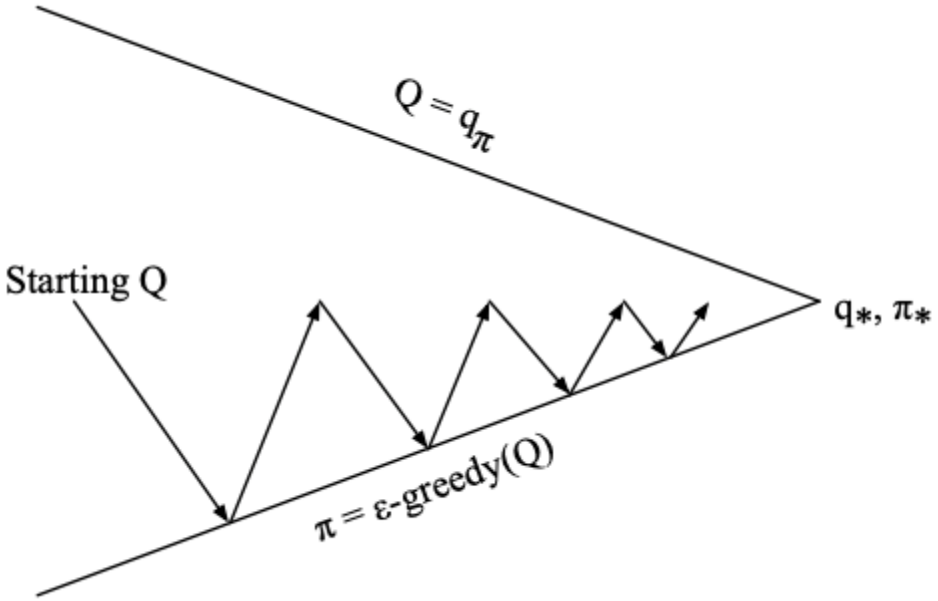
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$

where:

- $Q(S_t, A_t)$ is the current estimate of the state-action value
- α is the learning rate
- R_{t+1} is the reward observed after taking the action
- γ is the discount factor
- $Q(S_{t+1}, A_{t+1})$ is the estimated value of the next state-action pair.

This update is done after every transition from a nonterminal state St. If S_{t+1} is terminal, then $Q(S_{t+1}, A_{t+1})$ is defined as zero. This rule uses every element of the quintuple of events, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$, that make up a transition from one state–action pair to the next.

Effectively the equation above updates the Q function by α times the direction of the TD error. What SARSA does is basically the policy iteration diagram we have seen in the control above but with a twist. Instead of trying to evaluate the policy using episodes as in MC, SARSA does policy improvement on an estimate obtained **over each time step** significantly increasing the iteration rate - this is figuratively shown below:



SARSA on-policy control

The idea is to increase the frequency of the so called ϵ-greedy policy improvement step where we select with probability ϵ a random action instead of the action that maximizes the $Q(s,a)$ function (greedy). We do so, in order to "hit" new states and therefore improve on the degree of exploration of our agent and as a result giving opportunities to the agent to reduce its variance *and* its bias.

The SARSA algorithm is summarized below:

Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Repeat (for each step of episode):
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Lab Tasks

- Learning rate, alpha, α = 0.1
- Discount factor, gamma, γ = 0.95
- Exploration factor, epsilon, ε = 0.2
- Number of Episodes = 20000

1. **Solve the Cart Pole problem given in the Gymnasium library using the SARSA (0) algorithm. You have to simulate the episodes using SARSA algorithm and generalized policy iteration and see its convergence. Remember, you have to implement epsilon greedy policy improvement to pick actions for each state. Plot the returns for episodes.**

2. **Solve the Acrobat problem given in the Gymnasium library using the SARSA (0) algorithm. You have to simulate the episodes using SARSA algorithm and generalized policy iteration and see its convergence. Remember, you have to implement epsilon greedy policy improvement to pick actions for each state. Plot the returns for episodes.**

Deliverable:

Please submit your notebook on LMS before the deadline **(10th November 2024, 11:59pm).**

Lab Rubrics

Assessment	Does not meet expectation (1/2 marks)	Meets expectation (3/4 marks)	Exceeds expectation (5 marks)
Software Problem Realization **(CLO1, PLO1)**	The student struggles to formulate the problem as RL and does not apply SARSA algorithm to solve it. There is a lack of understanding of the problem's requirements and no attempt to find the optimal policy effectively.	The student formulates the problem as RL with some guidance, applies SARSA algorithm with hints, and shows it's working. However, the approach might not be fully optimized or lacks a thorough justification.	The student independently formulates the given problem as RL, applies SARSA algorithm without guidance, and effectively find an optimal policy. The approach is fully optimized and can be applied to different similar problems.
Software Tool Usage **(CLO4, PLO5)**	Code has syntax errors, and the implementation of the SARSA (0) algorithm is incorrect or incomplete. The code is not modular and lacks comments for readability and reuse. The student shows limited ability to use gymnasium library functions where required.	The student has implemented the SARSA algorithm correctly for the given problem with minor mistakes. The code is mostly correct in terms of syntax and functionality but might not be optimized or well-structured for reuse. Some documentation is provided. The student also shows working knowledge of gymnasium library where required.	The student has implemented the SARSA algorithm efficiently and correctly. The code is clean, modular, well-documented, and follows best practices for reproducibility and reuse. The student demonstrates full command of the gymnasium library and its functions.