CS5500 : Reinforcement Learning

Assignment № 2

Due Date : 06/03/2020

Teaching Assistants : Venkatesh Vayyavuru and Razat Shanker

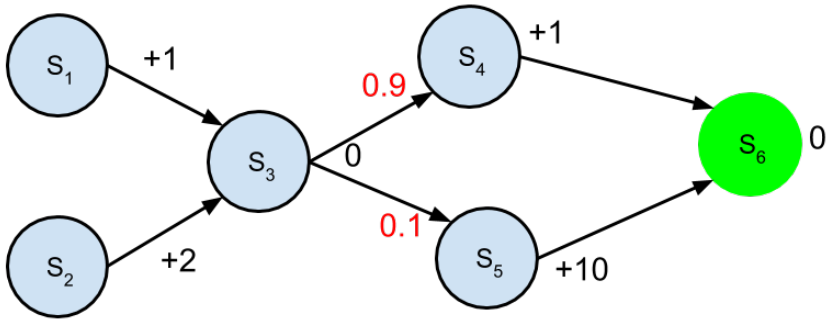
Easwar Subramanian, IIT Hyderabad 23/02/2020

Assignment Policy

Same as in Assignment 1. Refer to relevant Piazza post on assignment policies. Specifically, for programming parts, even if you consult the internet, code it yourself without plagiarism. Please include instructions to run the code along with a short write-up of your observations.

Problem 1 : Model Free Prediction

Consider the following Markov reward process. The MRP has six states and all corresponding Markov chain trajectories start from either state s_1 or state s_2. All trajectories terminate at state s_6. From state s_3, one could go to state s_4 with probability 0.9 or to state s_5 with probability 0.1. Rewards associated with each state are mentioned near to the states. Assume the discount factor to be $\gamma = 1$.



(a) Evaluate the **true** value of each state of the MRP. (2 Points)

Consider the following samples of Markov chain trajectories of the above MRP to answer the questions below.

(1) $s_1 \rightarrow s_3 \rightarrow s_4 \rightarrow s_6$

(2) $s_1 \rightarrow s_3 \rightarrow s_5 \rightarrow s_6$

(3) $s_1 \rightarrow s_3 \rightarrow s_4 \rightarrow s_6$

(4) $s_1 \rightarrow s_3 \rightarrow s_4 \rightarrow s_6$

(5) $s_2 \rightarrow s_3 \rightarrow s_5 \rightarrow s_6$

(b) Evaluate $V(s_1)$ and $V(s_2)$ using Monte-Carlo method (2 Points)

(c) Evaluate $V(s_1)$ and $V(s_2)$ using TD(0) method (2 Points)

(d) Evaluate $V(s_2)$ using maximum likelihood estimation. (2 Points)
[Hint : A MLE based value estimation is computed from sample trajectories. Here, to compute $V(s_2)$ we need to compute $V(s_3)$ and one need to calculate state transition probabilities to go from state s_3 to s_5 and s_4 respectively using samples. Use the transition probabilities obtained to compute $V(s_3)$.]

(e) We now have three estimates of $V(s_2)$ one from MC method, one from TD method and one from MLE. Which estimate is closer to the true value of state, namely, $V(s_2)$? And which is far from the true value $V(s_2)$? Explain with reasons. (2 Points)

Problem 2 : On Learning Rates

In any TD based algorithm, the update rule is of the following form

$$V(s) \leftarrow V(s) + \alpha_t[r + \gamma V(s') - V(s)]$$

where α_t is the learning rate at the t-th time step. In here, the time step t refers to the t-th time we are updating the value of the state s. Among other conditions, the learning rate α_t has to obey the Robbins-Monroe condition given by,

$$\begin{aligned} \sum_{t=0}^{\infty} \alpha_t &= \infty \\ \sum_{t=0}^{\infty} \alpha_t^2 &< \infty \end{aligned}$$

for convergence to true $V(s)$. Other conditions being same, reason out if the following values for α_t would result in convergence. (5 Points)

(1) $\alpha_t = \frac{1}{t}$

(2) $\alpha_t = \frac{1}{t^2}$

(3) $\alpha_t = \frac{1}{t^{\frac{2}{3}}}$

(4) $\alpha_t = \frac{1}{t^{\frac{1}{2}}}$

Generalize the above result for $\alpha_t = \frac{1}{t^p}$ for any positive real number p (i.e. $p \in \mathbb{R}^+$)

Problem 3 : Policy Improvement

Given an MDP and a policy π, what does it mean to do ϵ-greedy exploration ? Prove that, for that any ϵ-greedy policy π, the ϵ-greedy policy π' with respect to Q^π is an improvement over π. (3 Points)

Problem 4 : λ Return

In the TD(λ) algorithm, we use λ returns as the target. The λ return target is given by,

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

where $G_t^{(n)}$ is the n-step return defined as,

$$G_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1}r_{t+n} + \gamma^n V(s_{t+n}).$$

The parameter λ is used to determine the weights corresponding to each of the n-step returns in the λ-return target. We know that the weights decay exponentially with n. Therefore, in the G_t^λ sequence, after some terms, the weights of subsequent terms would have fallen by more than half as compared to the weight of the first term. Let $\eta(\lambda)$ denote the time by which the weighting sequence would have fallen to half of its initial value. Derive an expression that relates the parameter λ to $\eta(\lambda)$. Use the expression derived to compute the value of λ for which the weights would drop to half after 3 step returns. (3 Points)

Problem 5 : Q-Learning

Consider an MDP with three states $\{s_1, s_2, s_3\}$ and three actions $\{a_1, a_2, a_3\}$ with discount factor $\gamma = 0.5$. There is no noise in the environment and therefore all actions result in intended state transitions. The reward for transitioning into a state s_i is i. For example, if any action $a_k, k\{1, 2, 3\}$ pushes the agent into state s_3, then the reward is 3. We consider a Q-learning agent that uses the ϵ-greedy strategy. When the Q-values from a particular state are same for more than one action, the agent breaks ties by choosing the action a_k with lowest k. Let us initialize the Q table to zeros for all state-action pairs and let the learning rate be set to $\alpha = 0.7$. For $\epsilon \neq 0$, could the Q-learning agent generate the following trajectory given by

$$(s_1, a_1, 1, s_1, a_2, 2, s_2)$$

If yes, reason out, which of the two action is greedy and which of it is random ? (4 Points)

Problem 6 : Game of Pac-Man

Consider a simplified version of the game Pac-Man which is played in a grid world of suitable size. The goal is to develop an agent that moves across this grid world with randomly placed food-pellets and a ghost that moves across the grid. The agent receives small positive rewards for touching (and thus consuming) food pellets and receives heavy negative rewards for touching the ghost or moving across the walls which are the boundaries of the grid world.

Both Pac-Man and the ghost can move up, down, left or right from their current location. The Pac-Man's action could include staying put in the same square of the grid world. An episode of the Pac-Man game terminates when the Pac-Man hits a boundary or is consumed by the

ghost. However, when the ghost hits a boundary, it disappears and re-appears at one end of Pac-Man's row or column. Note that the ghost hitting the boundary does not end in game termination. If during game, if all food pellets are consumed, then a random number of them reappear at random positions and the game continues. The goal of is this problem is to train an agent using tabular Q-learning and SARSA to play this game of Pac-Man.

(a) Develop a suitable grid world environment to play the game pf Pac-Man. The size of the grid world should be an input to the **init** function of the environment class along with the number of food pellets. An initialize function in the environment should place the Pac-Man and all food pellets at random locations in the grid world. The ghost is placed (randomly) at one end of Pac-Man's row or column. The environment class may also include a display function that renders the current state of the game. (5 Points)

(b) Implement a tabular Q-learning and SARSA based agent to play Pac-Man. The Pac-Man can perceive

- His position
- The position of the ghost
- The locations of the walls
- The positions of each food pellet
- The total number of food pellets still available
- His current score in the game

In addition, the Pac-Man can also determine the next state of the environment given an action. (10 Points)

For this assignment, along with your code, please submit instructions to run the code and include a small write-up that summarises your assumptions on the grid world, MDP structure and a note on final performance of the trained agent. Note that the implementation of Q-learning and SARSA differs only in the update rule and hence you may want to compare the performance of the trained SARSA and Q-learning agents by letting them play few episodes of Pac-Man.

Problem 7 : Game of Tic-Tac-Toe

Consider a 3×3 Tic-Tac-Toe game. The aim of this problem is to implement a Tic-Tac-Toe agent using Q-learning. This is a two player game in which the opponent is part of the environment.

(a) Develop a Tic-Tac-Toe environment with the following methods. (5 Points)

(1) An **init** method that starts with an empty board position, assigns both player symbols ('X' or 'O') and determines who starts the game. For simplicity, you may assume that the agent always plays 'X' and the opponent plays 'O'.

(2) An **act** method that takes as input a move suggested by the agent. This method should check if the move is valid and place the 'X' in the appropriate board position.

(3) A **print** method that prints the current board position

(4) You are free add other methods inside the environment as you deem fit.

(b) Develop two opponents for the Q-learning agent to train against, namely, a random agent and safe agent (5 Points)

(1) A **random agent** picks a square among all available empty squares in a (uniform) random fashion

(2) A **safe agent** uses the following heuristic to choose a square. If there is a winning move for the safe agent, then the corresponding square is picked. Else, if there is a blocking move, the corresponding square is chosen. A blocking move obstructs an opponent from winning in his very next chance. If there are no winning or blocking moves, the safe agent behaves like the random agent.

(c) The Q-learning agent now has the task to learn to play Tic-Tac-Toe by playing several games against **safe** and **random** opponents. The training will be done using tabular Q-learning by playing 10,000 games. In each of these 10,000 games, a fair coin toss determines who makes the first move. After every 200 games, assess the efficacy of the learning by playing 100 games with the opponent using the full greedy policy with respect to the current Q-table. Record the number of wins in those 100 games. This way, one can study the progress of the training as a function of training epochs. Plot the training progress graph as suggested. In addition, after the training is completed (that is after 10,000 games of training is done), the trained agent's performance is ascertained by playing 1000 games with opponents and recording the total number of wins, draws and losses in those 1000 games. The training and testing process is described below. (10 Points)

(1) Training is done only against the random player. But the learnt Q-table is tested against both random and safe player.

(2) Training is done only against the safe player. But the learnt Q-table is tested against both random and safe player.

(3) In every game of training, we randomly select our opponent. The learnt Q-table is tested against both random and safe player.

(4) Among the three agents developed, which agent is best ? Why ?

(5) Is the Q-learning agent developed unbeatable against any possible opponent ? If not, suggest ways to improve the training process.

[**Note** : A useful diagnostic could be to keep count of how many times each state-action pair is visited and the latest Q value for each state-action pair. The idea is that, if a state-action pair is visited more number of times, Q value for that state-action pair gets updated frequently and consequently it may be more close to the 'optimal' value. Although, it is not necessary to use the concept of afterstate discussed in the class, it may be useful to accelerate the training process]

ALL THE BEST