AI 3000 / CS 5500 : Reinforcement Learning

Assignment № 2

Due Date : 04/10/2024

Couse Instructor : Easwar Subramanian 20/09/2024

Problem 1 : Dynamic Programming and Model Free Methods

(a) Let M be an infinite horizon MDP and let π be a policy. Suppose if the value iteration algorithm to calculate V^π is terminated after $k+1$ iterations with $\left\| V^\pi_{k+1} - V^\pi_k \right\|_\infty < \epsilon$ for some chosen $\epsilon > 0$, how far is the estimate V_{k+1} from the true value function V^π ? Provide details of your derivation. (3 Points)

(b) Prove that the Bellman evaluation operator $\mathcal{L}^\pi$ satisfies the monotoniciity property. That is, for any two value functions u and v such that $u \leq v$ (this means, $u(s) \leq v(s)$ for all $s \in \mathcal{S}$), we have $\mathcal{L}^\pi(u) \leq \mathcal{L}^\pi(v)$ (2 Points)

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

$$G^\lambda_t = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G^{(n)}_t$$

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

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

The parameter λ is used to determine the weights corresponding to each of the n-step returns in the λ-return target and the weights decay exponentially with n. 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. (4 Points)

(d) Assume an MDP where four transitions are provided as shown in the snapshot below. Fill in the blank cells of the table below with the Q-values that result from applying the Q-learning update for the 4 transitions specified by the episode below. You may leave Q-values that are unaffected by the current update blank. Use learning rate $\alpha = 0.5$. Assume all Q-values are initialized to 0. After running the Q-learning algorithm using the four transitions given above, construct a greedy policy using the current values of the Q-table in states C, E and F. (3 Points)

s	a	r	s	a	r	s	a	r	s	a	r	s
C	jump	4	E	right	1	F	left	-2	E	right	+1	F

	Q(C, left)	Q(C, jump)	Q(E, left)	Q(E, right)	Q(F, left)	Q(F, right)
Initial	-10	-10	-10	-10	-10	-10
Transition 1						
Transition 2						
Transition 3						
Transition 4						

Problem 2 : 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. (4 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 (4 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**, it may be useful to accelerate the training process]

ALL THE BEST