OLCAR- Exercise 2 – Reinforcement Learning

Answers to question related to programming exercise

Handout: 21.04.2015
Due: 06.05.2015

Yi Hao Ng, Raphael Stadler
(Team Number 10)

Exercise 2a: Mountain Car

Question 1: Build a probabilistic model of the system. What is the stochastic element in the modeling process and what is its significance? What modeling parameters have the most effect on the quality of the solution?

The probabilistic element comes from the fact that the transition probabilities, meaning the transition from one state to another, have to be modelled. Discretizing the model with a finite number of bins will not cover every possible continuous state points. As a result of this, the continuous state value inside the bin has effect on the position that the car might end up after a certain input is applied.
The quality of the solution therefore is directly affected by the number of state bins (`MDP_Params.pos_N, MDP_Params.vel_N`) and the number of modeling iterations (`MDP_Params.modeling_iter`).

Question 2: Implement the Generalized Policy Improvement algorithm introduced in Section 2.8.3. Use appropriate terminal conditions for Policy Evaluation and Policy Improvement processes and implement the Policy Iteration and the Value Iteration algorithms. Think about what the optimal solution to this task should be for the Mountain Car system. Was the learning algorithm able to find this solution? If not, why do you think that is the case?

The expected solution should be of the form that the car first bounces back and forth to gain enough velocity to finally be able to reach the top of the hill as fast as possible.
Using the parameters which were suggested, the algorithm in fact is able to find such a solution. Although, choosing `GPI_Params.minDelta_V` too big (e.g. equal to 1) or `GPI_Params.alpha` too small (e.g. equal to 0.1) causes the algorithm not to find a solution which reaches the goal.

Question 3: Now build a deterministic discrete state-action space model of the system (i.e. set the number of modeling iterations to 1). Is it possible to find a policy which reaches the goal? What problems are faced when discretizing the system in this way?

Configuring the system with such discrete state-action space does not allow the algorithm to find an optimal solution - independently of the position in the valley, always the same not-optimal control input (a = -1) is chosen. Initially, in the Policy Improvement step the term which is maximized is the same for every action (as long as the goal was never reached, the reward is -1 for every action in every time step). As a result, the first evaluated action which led to this term value (in our case, a=-1) is chosen.

Exercise 2b: Cliff World

Question 4: First implement the Monte Carlo algorithm. Test the algorithm using different values of epsilon. What impact does epsilon have on the solution? Can the algorithm find the optimal greedy policy?

The value of epsilon characterizes how soft the policy is. With a non-zero probability (epsilon/|U|) a non-greedy policy - a policy which does not directly optimize the value function - is taken.
Smaller epsilon result in more exploitation, while bigger epsilon result in more exploration of the algorithm. The bigger epsilon is, the less probable it gets that the algorithm finds the optimal greedy policy.

Question 5: Now implement the Q=Learning algorithm. First test the algorithm with decreasing exploration during learning (i.e. k_epsilon < 1). Next, test the algorithm using constant exploration during learning (i.e. k_epsilon = 1). How does k_epsilon influence the solution?

Configuring a constant exploration during learning (i.e. k_epsilon = 1) still allows to find the solution, but in contrast to the case where the k_epsilon < 1 (exploration decreases during learning), there are much more "spikes" in the total reward function. This can be explained with the always constant probability of choosing a non-greedy action, other than the currently optimal policy.
In the case where the epsilon decreases, the longer the learning phase is, the smaller the probability gets, that the algorithm chooses a non-greedy action, which leads to a “smoother" evolution of the total reward function.

Question 6: Compare the computational efficiency and performance of the Monte Carlo and Q-Learning algorithms. What are the fundamental reasons why one algorithm performs better than the other?

The off-policy Q-Learning performs much better than the on-policy Monte-Carlo algorithm and chooses a more optimal solution while the Monte-Carlo tends to choose a more safe solution.
Q-Learning combines "Policy Evaluation" and "Policy Improvement" into 1 single step, where for Monte-Carlo this is made in two separate loops one after each other. Instead of taking into account the direct rewards as it is the case for Q-Learning, the Monte-Carlo algorithm always has to calculate the accumulated rewards to be able to improve the policy afterwards, which is also computationally more expensive.