DRL Homework 1 - Group 25

Tom Pieper: 982845, Dario Hol Kieslich: 984741, Lorenz Leisner: 984643

April 22, 2022

1 Task 1

The game of chess is a finite MDP as its environment is a 8x8 chess board with a limited bound of possible positions (for example a white pawn can never be on the first rank). The states are all possible legal chess positions: an upper bound can be estimated to be 4.51045. See the link below: github.io/chess

The set of actions are all the possible moves one can perform with one's pieces (if it is still on the board). With pawns one also has to differentiate whether one captures another piece or just moves it forward. We suggest a policy which tries to maximise the reward with the following policy. You receive a reward of 100 for winning (checkmating), 9 for capturing the queen, 5 for the rook, 3 for the bishop and the knight and 1 for a pawn (losing those pieces also costs the same amount). Additionally each possible move you have in one position gives you a reward of 0.01 and each move costs you 1.

For the policy you receive a small cost for each move but a reward for a good position (having more pieces than the enemy is generally a good thing, the more squares your your own pieces can go to the better). An additional reward(e.g. 0.2) for moves that guards another figure should be implemented to support a useful setup. With this general guidelines which can also be seen as partial goals, the policy should find the optimal move for every possible state, that maximises the actors chance of winning.

2 Task 2

The state space is all the possible pixel combinations there are in the environment (600x400 pixel values). Additionally there are 8 observations: The x-coordinate of the position of the rover, the y-coordinate of the position of the rover, weather the right leg of the rover is on the surface, weather the left leg of the rover is on the surface, the velocity of the rover in x and in y direction (with also negative values for up and down, and left and right movements), its angle and its angular velocity. There are four actions: Do nothing, fire left orientation engine, fire right orientation engine, fire left orientation engine.

The corresponding policy the actor should be running is to maximise the reward i.e. landing the spaceship on the landing pad with as few engine firings as possible. This is realised with a reward function with in between rewards of giving 100 points when the rover lands (120 when rover lands on both feet) and costs for each frame of firing the main engine.

3 Task 3

The Policy evaluation algorithm offers an encompassing way of assigning values to states in our state space. The environment dynamics can be described by (p(s, r—s, a)). In real world scenarios they are rarely given but need to be sampled from interaction with the environment. These dynamics describe the probability of getting a new state s' and a reward r in the state when taking action a at state s. For each possible state, all returns that are possibly reachable from this specific state on are accumulated and averaged to compute the respective value described by a random variable of this state under the given policy. Using this dynamics we can evaluate every state according to our policy. By finding the optimal value function to maximise the state values, we can then extract and optimal policy function from the value function. Since we are maximising our returns and therefore our policy, once we cannot improve the policy anymore, we have found a unique and optimal policy.

The policy iteration algorithm contains the policy evaluation algorithm as well as an update function where the current policy is changed. Until no further improvement of the given policy is possible i.e. the optimal policy was found, the steps of evaluating the policy, hence the state-values and update the policy accordingly are repeated (theoretically infinitely) in a loop. For this we make use of Q-values which are computed by taking an random action at s to s' and from there on make moves according to the policy. This allows exploring new states due to the difference of this single move in comparison to the move that the current policy would predict.

The environment dynamics of a Markov-Decision-Process (p(s, r—s, a)) are based on the Markov-Property, that a state transition e.g. taking action a in state s and the reward this action obtains are conditionally independent from every situation that was before s. Therefore we do not need to consider doubled states in the process of finding optimal policy. For each state in the finite set of states an action is taken from a finite set of actions depending on the state we are in and our policy. The reward function is used to calculate the value of the current state and the action taken from s to s'. This is done by discounting rewards that lie in the future with a discount factor (a reward in the next move e.g. Checkmate is better than the same reward in 10 moves). For each state all returns from this state on are calculated by discounting the future rewards for all possible moves, getting all possible returns(discounted sum of rewards till a terminal state is reached) that are reachable from this state and then averaging over them. In the scenario of a self/driving car that is facing a sharp right turn, a good state transition function would suggest slowing down before the curve as most of the situations that can be computed from the reward function yield higher rewards i.e. not crashing or moving in the wrong lane. Another example for the discounting of rewards would be the LunarLander environment, where the reward for landing the spaceship fast is higher than hovering for a few seconds over the landing pad as this costs a lot of fuel. Here as well are actions according to a good policy moving the ship in direction of the landing pad as well as keeping the ship steady. An action a is taken in state s, leading to state s'. This new state gets an Value by our value function according to the average of returns that could result from this new state.

As already stated above, in most RL applications, especially in real-world situations the environment dynamics are unknown to the actor and we do not know every consequence our action s could have in state s e.g. bring us to a desired state s'. Therefore we need to interact with the environment, take actions, and learn from them. The policy still needs to be updated by the rewards we get from the environment, but we need to achieve these rewards by 'sampling actions' or 'simply trying out actions in certain situations.