Deep Reinforcement Learning - Homework

Vladis Becker, Simon Beginn, Tobias Schröder

Summer Semester 2022

Contents

1 Homework 1

1.1 Task 01

1. You are tasked with creating an AI for the game of chess. To solve the problem using Reinforcement Learning, you have to frame the game of chess as a Markov Decision Process (MDP). Describe both the game of chess formally as a MDP, also formalize the respective policy.

Solution:

A state in chess can be defined as some kind of distribution of chess pieces over fields on the board. This can be either a list of pieces each associated with a unique position on the board or a list of fields on the board, each associate with one or no piece which stands on it. This state is constrained by some rules of chess. A player can never have more than one bishop on fields of the same color and pawns can never stand in the first row of their side.

The set of all such states is our set of states S.

An action in chess consists of picking a single piece and moving it to another field, the possible target fields being constrained by the type of piece, following typical chess rules. A piece can only move to another field which is empty or has an opposing piece. A pawn has conditional movement rules based on the type of target field (empty or occupied).

The set of all such actions is our set of actions A.

The state transition function is more complicated. While chess is a deterministic game, that is, any action immediately leads into a clear follow up state, the next state that the player says is based on unknown factors. Moving a chess piece is deterministic, the opponents answer is not. As such, the state transition function is dependend on the opponents policy. No optimal policy for chess has been found (which would make this exercise futile in the first place), so assumptions about the opponents play can be made. One possible approach is to assume that the opponent plays the same as we do. As such, the AI would practically train against itself. In this case, the transition function from the perspective of one agent would be given through the rules of chess determining where a piece may move and the AIs own judgement of what the correct follow up move would be.

The reward function should be aimed at winning the game. The most obvious reward would be for checkmating the opposing king. A reward for taking opposing pieces seems sensible, but could become an unnecessary distraction. Taking opposing pieces and losing one's own pieces is irrelevant as long as the opposing king can be checkmated. As such, the purest reward function rewards only moves which cause a checkmate and punish getting checkmated.

The policy now gives us the probability that, given a specific configuration of the chess board, we pick a certain piece and move it to a certain position.

1.2 Task 02

2. Check out the LunarLander environment on OpenAI Gym: Check out this Link!. Describe the environment as a MDP, include a description how the policy is formalized.

Solution:

The state consists of:(map, x,y,rot, xvel,yvel,rotvel, legrotationl,legrotationr, legrotlvel,legrotrvel). possible actions are firing right, left, down or not firing at all. State transitions are calculated using the physics engine and take these actions into account as well as the previous state. These state transitions should therefore be deterministic. The expected reward decreases for every frame the main thrusters are used and for every landing the reward is based on the position(x,y) on the map, the impact speed(xvel,yvel) and whether the legs touch the floor, calculated from the complete state.

A policy $\pi(a \mid s)$ describes the probabilities of choosing actions in every state. Since the states are continuous, the probabilities can't be set for every possible state and have to be calculable. The policy therefore has to be a continuous function of s.

1.3 Task 03

3. Discuss the Policy Evaluation and Policy Iteration algorithms from the lecture. They explicitly make use of the environment dynamics ($p(s', r|s, a)$).
    - Explain what the environment dynamics (i.e. reward function and state transition function) are and give at least two examples.
    - Discuss: Are the environment dynamics generally known and can practically be used to solve a problem with RL?

Solution:

The reward function describes some form of quality judgement for reaching a certain state. This is supposed to describe how good it is to enter a certain state.

The state function is used whenever a decision is made. It describes, given a state and action, what the next state is that will be reached by the time another action is to be decided on, based on what action is taken now.

As examples, a chess game and a race will be used. The reward function in chess would judge how good any move would be given the current state of the game. This could possibly be based on rewarding the taking of pieces, checking the enemy and checkmating the enemy. It could also punish losing pieces, getting checked and getting checkmated. The state transition function in chess can be problematic, even though it already is a very simple problem in any one moment. While the action of the AI causes a very easily predicted change in the state of the game based on chess rules, it also needs to be considered that the opponent gets a move which is outside of the purview of the AI. As such, an assumption needs to be made in the state transition function and can even be incorrect.

The second example is that of a race. A car needs to finish the race by driving along the entire track and reaching the goal. The reward function would then describe how good a decision on acceleration, steering and braking would be. It would do so by judging how much progress is achieved. The state would contain how long the race has been going and how far along the race track the car is. As such, increasing progress and finishing the race ought ot be rewarded, whereas taking more time ought to be punished. The state transition function in this example would be based on some physics engine. It would take in the current state of the car, like speed, direction and other such things, apply the action to it and calculate where the car would end up being in the next time step.

Now that examples are given, it should already be rather easy to see why the environment dynamics can't always be feasibly determined. This can cause the application of reinforcement learning to fail to various degrees. Some problems, like a real-life race car, can not be trivially computed with sufficient accuracy. The chess example also shows that sometimes, part of what determines the next state is functionally not possible to determine until after the action was taken. In such cases, some assumption must always be made.