DEEP REINFORCEMENT LEARNING 2023

Homework 01

Author
Krishnendu Bose, Jakob Heller, Alexander Wallenstein
Group 02
Osnabrueck University

27.04.2023

1 Understanding MDPs

1.1 Chess

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.

1.1.1 Solution

Definition 1.1 *Markov Decision Process (MDP)*
A finite Markov Decision Process consists of:

- the finite set of States S, with states $s \in S$,
- the finite set of Actions A, with actions $a \in A$,
- the probabilistic state dynamics $p(S_{t+1}|S_t, A_t)$,
- the probabilistic reward dynamics $r(s, a) = \mathbb{E}[R_{t+1}|s, a]$

- **States** S are the possible configurations of the chess board. Although really large (some estimations for *legal* chess positions are around 10^{45}), that number is indeed finite.
- **Actions** A are the possible moves a player can make. This includes all possible moves for all pieces, including castling, capturing, promoting and en passant.
- **State Dynamics** $p(S_{t+1}|S_t, A_t)$ is the probability of the next state S_{t+1} given the current state S_t and the action A_t. Note: given a state S_t and an action A_t of our player, the next state S_{t+1} is the board state after the opposing player has made their move. p is therefore not deterministic, and includes every possible move the opposing player can make after A_t.
- **Reward Dynamics** $r(s, a) = \mathbb{E}[R_{t+1}|s, a]$ is the expected reward after taking action a in state s. The reward is 1 if the game is won and 0 if it is not. We actively decide against introducing a reward for capturing pieces or similar subgoals: otherwise, the agent might learn to play for the reward instead of playing to win. "[...] the reward signal is not the place to impart to the agent prior knowledge about *how* to achieve what we want it to do."[1]

Our policy π, or $\pi(a|s)$, is a probability distribution over actions a given a state s. A complete policy chooses a move (optimally the best move) for every possible board state. In chess, a good policy has very large probabilities for moves that lead to a win, and very small probabilities for moves that lead to a loss.

1.2 LunarLander

Check out the LunarLander environment on OpenAI Gym. Describe the environment as a MDP, include a description of how the policy is formalized.

1.2.1 Solution

There are multiple variations to the Lunar Lander environment — because nothing was specified, we will choose the discrete version without wind.

- Different to the chess example, we only have an observation space, not a state space. The observation space consists of states s, where each state is an 8-dimensional vector. Each state vector contains x and y coordinates of the lander, its x and y linear velocity, its angle, its angular velocity and two boolean values indicating whether the left and right legs are touching the ground.
- The set of **actions** A is discrete and includes the following actions: do nothing, fire left orientation engine, fire main engine, fire right orientation engine.
- The **state dynamics** $p(S_{t+1}|S_t, A_t)$ are deterministic. The next state S_{t+1} is determined by the current state S_t and the action A_t.
- A **reward** is assigned to every action, the reward of the episode is the sum of all rewards. For each step, the reward:
  - is increased/decreased the closer/further the lander is to the landing pad.
  - is increased/decreased the slower/faster the lander is moving.
  - is decreased the more the lander is tilted.
  - is -0.3 for each step where the main engine is firing,
  - is -0.03 for each step where the left or right orientation engine is firing,
  - is $+10$ for each leg that is touching the ground,
  - is -100 if the lander crashes,
  - is $+100$ if the lander is on the landing pad,

  If it scores at least 200 points, the episode is considered a solution.

Since we are dealing with an observation space, we cannot use a policy $\pi(a|s)$ that maps every state to an action. Instead, we use a policy $\pi(a|o)$ that maps every observation to an action. The policy is a probability distribution over actions a given an observation o.

1.3 Model Based RL: Accessing Environment Dynamics

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 ?

1.3.1 Solution

State transition function $p(s'|s, a)$:
Starting in a state s and taking an action a, what state do we end up in?
On the one hand, in the example of the Lunar Lander, we have a deterministic state transition function and can thus easily calculate the next state s' given the current state s and the action a. On the other hand, consider a Gridworld setup with "icy" tiles: when deciding for an action, we might slip or slide further than intended — this is represented by a stochastic state transition function.

Reward function $r(s, a)$:
What reward do we get for taking action a in state s?
The predefined rewards for the Lunar Lander incentivize the agent to land on the landing pad as fast and safe as possible. A typical reward structure for a Gridworld setup could be some small negative reward for each step and a large positive reward if the action causes the agent to reach the goal.

Are the environment dynamics generally known?
If we consider very constructed examples such as Gridworld or the Lunar Lander, then the answer is yes. After all, we defined the environment dynamics ourselves. If we talk about the real world or multi-agent environments, then the answer is almost always no. Considering chess for example, since we already talked about that in this homework: we know what move will result in what board state since we know the rules of chess, but we cannot know what move the opponent will make. Of course we could be able to make reasonable assumptions, but we never know with certainty.

2 Implementing A GridWorld

2.1 Look up some examples

Look up some examples of GridWorld! List at least three links or references to different GridWorlds you could find online.

2.1.1 Solution

- `https://github.com/openai/gym/blob/master/gym/envs/toy_text/frozen_lake.py`
- `https://github.com/MatthewFilipovich/rl-gridworld/blob/master/gridworld.py`
- `https://github.com/hmeretti/grid-world`

The rest of the homework is about implementing a GridWorld environment. The solutions can thus be found in Homework 01/gridworld.py.

3 References

[1] R. S. Sutton, F. Bach, and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press Ltd, 2018.