1. Understanding MDPs

1.1. Chess as MDP

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.

- State space (S): All possible board configurations, which can be calculated as the number of pieces (6 types for each player) raised to the power of the number of squares (8x8) on the chessboard. However, this estimation includes illegal positions.
- Action space (A): All legal moves for the player whose turn it is. Actions can be represented as a tuple (piece, old_square, new_square).
- Transition function (P): P(s'|s, a) would represent the probability of reaching state s' from state s by taking action a, considering both the player's move and the potential opponent's moves. This adds an element of uncertainty to the transition model, making the game of chess probabilistic in the context of a two-player game.
- Reward function (R): Assigns a value for each state-action pair, usually with higher rewards for capturing pieces and winning the game. We can use the material value of the pieces (pawn = 1, knight = 3, bishop = 3, rook = 5, queen = 9, king = 10) and a large positive reward for checkmate(50), while a large negative reward for getting checkmated(-50).
- Policy (π): π(s) = argmax_a (CaptureValue(s, a))

1.2. LunarLander as MDP

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.

- State space (S): 8 dimensional vector representing the lander's position (x, y), velocity (v_x, v_y), angle, and angular velocity, two booleans that represent whether each leg is in contact with the ground or not.
- Action space (A): Four discrete actions are available: do nothing (0), fire left engine (1), fire main engine (2), and fire right engine (3). These actions influence the lander's position, velocity, and angle.
- Transition function (P): Describes the probability of reaching a new state given the current state and action taken. In the LunarLander environment, this function is deterministic but influenced by the continuous nature of the state space and the environment's physics(gravity).
- Reward function (R): The reward function encourages smooth landing, fuel conservation, and penalizes crashes.
- Policy (π): π(s) =

  - Fire left engine, if angle > angle_upper_threshold

  - Fire right engine, if angle < angle_lower_threshold

  - Fire main engine, if vertical_velocity < velocity_threshold

  - Do nothing, otherwise.

1.3. Model Based RL

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?

Environment dynamics consist of the state transition function (P) and the reward function (R).

The state transition function defines the probability of reaching a new state (s') given the current state (s) and action (a) taken: P(s', r|s, a).

The reward function assigns a reward value (r) for each state-action pair.

Examples of environment dynamics:

- In a GridWorld environment, the state transition function represents the probability of moving from one grid cell (state) to another, given the agent's action (e.g., moving up, down, left, or right). The reward function might assign a positive reward for reaching the goal and a negative reward for entering a hazardous cell (e.g., traps or walls).
- In a chess game, the state transition function defines the probability of reaching a new board configuration (s') given the current board configuration (s) and a legal move (a) made by a player. The reward function could assign higher rewards scaled by the value of the piece for capturing pieces or winning and lower rewards for losing pieces or losing.

Environment dynamics are not always known or easily accessible, and their practical use in solving problems with reinforcement learning depends on several factors:

Complexity: In complex environments with large state and action spaces, it may be unfeasible to enumerate all possible transitions and rewards.

Uncertainty: In many real-world problems, the environment dynamics are probabilistic, making it difficult to predict the exact state transitions and rewards.

Observability: In partially observable environments, where the agent cannot access the complete state information, estimating the environment dynamics becomes challenging.

Although environment dynamics can be used in some cases to solve problems with RL, they are not always known or easily accessible. In practice, many RL problems are addressed using model-free (Q-learning) methods that learn from interaction with the environment or model-based methods that learn an approximate model of the environment dynamics(we get samples, even though P(s', r|s, a) is unobtainable).

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.

- Found by searching: https://github.com/topics/grid-world

- https://github.com/hmeretti/grid-world
- https://github.com/elhamza9/Grid-World-Reinforcement-Learning-ReactJS
- https://github.com/JuliaReinforcementLearning/GridWorlds.jl