Exercise solutions for Reinforcement Learning: An Introduction [2nd Edition]

Mátyás Pólya

September 10, 2024

Chapter 1

Exercise 1.1: Self-Play
Suppose, instead of playing against a random opponent, the reinforcement learning algorithm described above played against itself, with both sides learning. What do you think would happen in this case? Would it learn a different policy for selecting moves?

Solution:

- Early games would be random and unstructured.
- Over time, the agent would improve by reinforcing winning moves and avoiding losing ones.
- Eventually, the agent would converge toward optimal play, leading to most games ending in a draw.
- The RL agent would effectively learn to play tic-tac-toe perfectly, unable to lose but also unable to win against a similarly skilled opponent (itself).

Exercise 1.2 Symmetries
Many tic-tac-toe positions appear different but are really the same because of symmetries. How might we amend the learning process described above to take advantage of this? In what ways would this change improve the learning process? Now think again. Suppose the opponent did not take advantage of

symmetries. In that case, should we? Is it true, then, that symmetrically equivalent positions should necessarily have the same value?

Solution:

- Rather than treating each possible board configuration as unique, the RL agent should group board positions that are equivalent under symmetry. This would reduce the number of unique states the agent needs to learn/investigate.

- No, we shouldn't exploit symmetries if the opponent doesn't exploit them either. If the opponent has a policy that behaves differently at states that are symmetric to each other, then the agent couldn't expoit this if it treated symmetric states the same. In this case symmetrically equivalent positions should not have the same value.

Exercise 1.3 Greedy Play
Suppose the reinforcement learning player was greedy, that is, it always played the move that brought it to the position that it rated the best. Might it learn to play better, or worse, than a nongreedy player? What problems might occur?

Solution:

- Case 1, The opponent is deterministic (always plays the same move at a certain state):
  In the beginnig, all of the non-winning states have a 50% rating, so the greedy player would choose a random move. If this move leads to winning, its estimate increases, so in the following rounds the agent will choose it again and again. If it leads to losing, then its estimate decreases, so the agent will choose another move in the following rounds. This way the agent will find a set of steps that will always lead to winning (if they exist).

- Case 2, The opponent is non-deterministic:
  Suppose at first try the agent finds a set of steps which always produce an estimate that is $> 50\%$. The greedy agent will always choose these steps (the others were initialized at 50%), but there might be a set of steps that achieve better winrate. In this case there is no guarantee

that the agent finds the best policy.
(This is a special case, the winrate can be arbitrarily small)

Exercise 1.4 Learning from Exploration
Suppose learning updates occurred after all moves, including exploratory moves. If the step-size parameter is appropriately reduced over time (but not the tendency to explore), then the state values would converge to a different set of probabilities. What (conceptually) are the two sets of probabilities computed when we do, and when we do not, learn from exploratory moves? Assuming that we do continue to make exploratory moves, which set of probabilities might be better to learn? Which would result in more wins?

Solution:

- If we don't update after exploratory moves: the value is the probability that a greedy agent wins from that state.
- If we update after exploratory moves: the value is the probability that an agent that is prone to explore wins from that state.
- If we continue to explore, then we should include the exploratory moves into the updates.

Exercise 1.5 Other Improvements
Can you think of other ways to improve the reinforcement learning player? Can you think of any better way to solve the tic-tac-toe problem as posed?
Solution:

- Incorporate domain knowledge (Heuristics and Rules): e.g. always take center.
- Use of Value Function Approximation: use a neural network or other function approximator to estimate the value function.