Chapter 1 - Introduction Exercises

Stéphane Liem NGUYEN

September 11, 2021

Exercises with (***corrected***) were corrected based on the Errata.
These are my own answers and mistakes or errors are possible.

Exercise 1.1: Self-Play **(p. 12)** 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?

In this case, if we suppose that draws are better than defeats, I would think that the policies will both converge towards the same optimal one where all outcomes are draws. However, from the book, it's written than draws are assumed to be equally bad as defeats and I would think that for this case, it might fail to arrive at the solution where all outcomes are draws. The players might sometimes win, sometimes lose and sometimes it's a draw.

In both cases, the policies learned are different from the one learned against the random imperfect opponent because the "backups" from two successive states S_t and S_{t+1} depends on how the opponent behaves.

Both sides might learn also similar policies, both trying to win against their opponent and the solution will maybe converge towards a minimax solution.

TODO: maybe program it to observe in numerically

Exercise 1.2: Symmetries **(p. 12)** 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?

To take advantage of symmetries, we can maybe convert any state or board configurations into some canonical form. For example, for any given board configuration, by rotating the board, we have in total maximum 4 symmetrical states (3 other than the given configuration) from this transformation. We can also flip the board left-right (reflection) then apply rotations and it can give a maximum of 4 additional symmetrical states. In practice, there are probably many ways to efficiently have save, compare and use the canonical state configurations and a non-efficient way would be to save each configuration encountered and each time, compare if the current state is symmetrical to the ones we already know.

This change might improve the learning process by reducing the amount of samples required to train the agent because updates would no longer be spread into the other symmetrical states.

If the opponent does not take advantage of symmetries, it is not really the problem of our agent that the opponent is less performant so I would think that we should continue taking advantage of symmetries.

Intuitively, a state that is symmetrical to another one (by applying transformations cited previously) should not change the basis on which the agents take their actions. In other words, values should be the same for symmetrically equivalent positions (expected return from states, not the estimations. But estimations should converge).

***Exercise 1.3: Greedy Play* (p. 12)** 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?

If the RL player was greedy with respect to the value functions, it will play worse in the limit than an epsilon-greedy algorithm with ϵ decaying over time (Greedy in the Limit with Infinite Exploration, see David Silver lecture on exploration-exploitation). The greedy algorithm might be stuck in a suboptimal behavior because it does not try to improve the estimates of the values of other actions that can be potentially better.

In general it also depends on what *nongreedy player* means. For instance, if the opponent is trying to lose all the time, we're unsure if the RL player would be better or not etc.

***Exercise 1.4: Learning from Exploration* (p. 13)** 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?

Let's first recall the TD learning update rule

$$V(S_t) \leftarrow V(S_t) + \alpha \left[V(S_{t+1}) - V(S_t) \right] \tag{1}$$

where α is a small positive step-size parameter.

When learning updates occur only after greedy moves, S_{t+1} is the state after the greedy move and S_t is the state before the greedy move. In the set of probabilities learned in this case, the value of a state would converge to the probability of winning by our player if he follows the optimal greedy policy without any exploration. In other words, the target policy is greedy while the behavior policy (how the agent interacts with the environment to gather information) has some exploration in it.

In constrast, learning updates occurring after all moves including exploratory ones would be like *on-policy* instead of *off-policy*. For each state, the value

would converge to the probability of winning by our player if he follows the target policy as the behavior policy (target policy is the same as the behavior policy and includes exploratory moves) from that state.

Let's take another problem to illustrate what can be better in what scenario. Let's say that the agent has to go from a starting point to an end point with a cliff in the middle. We suppose that actions deterministically move the agent to the next square or do not move the agent if he tries to move into a wall or edge of the grid world. Let's also suppose that the behavior policy is most of the time taking greedy actions with respect to the value function and sometimes exploratory actions.

Intuitively, if we want to behave optimally while still having the tendency to randomly explore, we need to avoid getting too close to the cliff. To do so, we have to learn from all moves, so including exploratory ones. On the other hand, if we just want to behave optimally without the tendency to randomly explore, we can go close to the cliff for the shortest path because the agent won't have the risk of falling.

Coming back to our problem, after learning the optimal value function when we do not learn from exploratory moves, if the player follows the target policy (no exploration) that is greedy with respect to the optimal value function, then the player would obtain the most rewards. However, if we want to get more rewards while interacting with the environment with the behavior policy that has the tendency to explore, we should go for the update rule including exploratory moves.

TODO: code and observe

***Exercise 1.5: Other Improvements* (p. 13)** 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?

One way to maybe improve the RL player might be to specify a different reward for losing and for draws so that they're not equally bad.