Answers to Exercises

for

Reinforcement Learning: An Introduction

2nd Edition

Richard S. Sutton and Andrew G. Barto

Answers to Exercises
Reinforcement Learning: 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?

Answer: Yes, it would learn a different move-selection policy. Contrary to what one might at first suppose, this new way of playing will not involve the same few games being played over and over. Random exploratory moves will still cause all positions to be encountered, at least occasionally. Just as playing a static imperfect opponent results in the reinforcement learning agent learning to play optimally (maximum probability of winning) against that opponent (except for exploratory moves), so playing itself will result in it learning to play optimally against itself (except for exploratory moves).

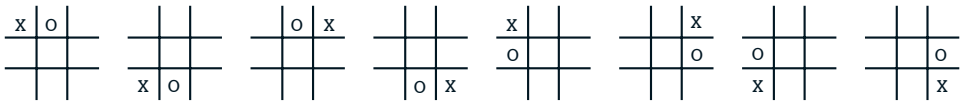
What is less clear is exactly what optimal play against itself would mean. There is a tendency to think that the self-play player would learn to play "better," closer to a universal "optimal" way of playing. But these terms don't have any meaning except with respect to a particular opponent. The self-play player may learn to play better against many opponents, but I think we could easily construct an opponent against whom it would actually fare worse than the original player.

What *can* we say about how the self-play player will play? Clearly all positions in which it can force a win will be played correctly (except for exploratory moves). Because it does make occasional exploratory moves, it will favor lines of play in which it is unlikely to not win even if it explores. And because occasional random moves are made by the opponent, the reinforcement learning agent will prefer positions in which many such moves are losing moves. What about positions in which one side cannot win, but could either lose or draw? Because that player doesn't care about the difference between these two outcomes, it will not try to avoid losses. The other player will prefer these positions over those in which its opponent cannot lose.

One person answering this question pointed out to me that technically it is not possible for the reinforcement learning agent as described to play against itself. This is because the described reinforcement learning agent only learned to evaluate positions after second-player moves. The player who moves first needs to learn the values of positions after first-player moves. Because these two sets of positions are disjoint, and the reinforcement learning agent as described learns separately about each individual position, learning about the two kinds of positions will be totally separate. Thus the reinforcement learning agent cannot play against itself, but only against a copy of itself adjusted to learn after first player moves.

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?

Answer: There are three axes of symmetry in Tic-Tac-Toe: up-down, right-left, and along the diagonal. By reflecting all positions into a standard form, the set of positions whose value needs to be learned can be reduced by a factor of about eight. For example, only one of the following positions need be represented and learned about—the learning will generalize automatically to the other seven:



This would improve the reinforcement learning agent by reducing its memory requirements and reducing the amount of time (number of games) needed to learn. However, if the opponent is imperfect and does not play symmetrically, then this may be counterproductive. For example, if the opponent played correctly in the first position above but incorrectly in the fourth, then our agent ("O") should prefer playing to the fourth position. That is, these symmetrically equivalent positions should not really have the same value. This would not be possible if they were represented as the same because of symmetries.

What is the right solution? One good way to proceed would be to use two tables, one smaller table that collapsed symmetric positions onto the same entry, and one larger table which did not. The approximate value of a position would be the average of the two table entries, one in each table, that applied to the position. Both table entries that applied to the position would also be updated. The result would be that the system would generalize somewhat between symmetrically equivalent positions, but if needed it would also be able to learn distinct values for them. This approach would not provide any savings in memory or computation time per move (in fact it requires more), but it should speed learning in terms of number of games played, without sacrificing asymptotic performance. This is a first step towards using generalization and function approximation rather than table lookup. We will consider many such possibilities later in the book. □

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. Would it learn to play better, or worse, than a non-greedy player? What problems might occur?

Answer: A greedy reinforcement learning player might actually end up playing worse. The problem is that it might get permanently stuck playing less than optimal moves. Suppose there is a position where one move will win 60% of the time but a better move will win 90% of the time. Initially both moves are valued at 0.5, just as all positions are. Suppose the first time it encountered the position by chance it played the 60% move and won. The value of that move will be bumped up, say to 0.51. As a result, that move will become the greedy choice, and next time it will be selected again. It's value will go to 0.6, while the other move, the 90% winning move, will stay valued at 0.5 and never be tried. □

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 set of probabilities. What 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?

Answer: When we don't learn from exploratory moves we learn the probabilities of winning from each position if we played optimally from then on. But of course we don't play exactly optimally from then on—we occasionally make exploratory moves. When we learn from exploratory moves we learn the probability of winning from each position taking into account the fact of these explorations. The best moves given explorations may be different from the best ones without exploration. Because we do continue to explore, the second approach will actually end up winning more games than the first. □

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?

Answer: Of course many other improvements are possible. Some of these are:

- Using generalizing function approximation to speed learning.
- Using search when choosing moves to look ahead farther than one step.
- Incorporating a priori expert knowledge. A natural way to do this is in the initial value function.
- When a new position is encountered, adjustments could be made not just to the immediately preceding position, but to earlier positions as well. This might speed learning.
- We might learn a model of the opponent (how frequently he makes each move in each positions) and use it off-line to improve the value function.
- We might learn to play differently against different opponents.
- Against a stationary opponent, we might reduce the frequency of exploration over time.

□