Reinforcement Learning: An Introduction
Attempted Solutions
Chapter 1

Scott Brownlie & Rafael Rui

1 RL Book

1.1 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?

As there is no reward for drawing, when the reinforcement learning algorithm plays against itself each side will learn a policy which attempts to win at all costs without trying to block moves which lead to wins for its opponent. When the reinforcement learning algorithm plays against a human the algorithm again has no incentive to block moves which would lead to wins for the human, however, a human is likely to have the foresight to at least block the most obvious of moves which would lead to wins for the algorithm.

For example, when the algorithm plays against itself, after a substantial period of learning it would not be unlikely for a game to proceed as follows:

1. X in 1st row, 1st column
2. O in 3rd row, 1st column
3. X in 1st row 2nd column
4. O in 3rd row, 2nd column
5. X in 1st row, 3rd column (X wins)

Here the second player is trying to win by putting three Os on the bottom row with no consideration for blocking the three Xs on the top row. When the algorithm plays against a human, and assuming that the algorithm is playing Xs, this game should never occur because any human with an ounce of sense would block the top row before the algorithm could lay down three Xs. Thus, when playing against a human such a simplistic policy would never be reinforced and the algorithm would be forced to find policies with more foresight.

In fact, we implemented the code for the reinforcement learning algorithm playing against itself and after a significant period of learning almost half the games finished in only five moves (three for the first player, two for the second), which is the minimum possible. Therefore, the algorithm learned very short-sighted policies which only work when it has the first move. A human who knows the rules of the game would never allow the algorithm to win after only five moves and it would have to learn more sophisticated policies to maximise reward.

1.2 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?

Instead of learning the value function for all board states we could learn the value function only for unique states after removing symmetries. So, for example, the state with only an X in the top left corner and a O in the bottom right corner would have the same value as the state with only a O in the top left corner and an X in the bottom right corner. This would significantly reduce the number of states, meaning that each state would be visited more often and the value function would converge in less time steps (assuming that the step-size parameter is reduced properly over time).

If the opponent did not take advantage of symmetries then her value function would take longer to converge. This may actually be beneficial due to increased exploration. Suppose that player X, who is taking advantage of symmetries, won from a suboptimal board state during one of the early games. Then this state would be reinforced along with all symmetrical states, making it even more likely for it or one of its equivalent states to be experienced again and all symmetrical states could eventually become locked into the same suboptimal policy. In contrast, if player O is not taking advantage of symmetries then she could learn different policies for symmetrically equivalent states, increasing her chance of finding the optimal policy for at least some of them.

As the value of a state clearly depends on the opponent's policy, symmetrically equivalent states should only have the same value if the opponent's policy is the same for those symmetrically equivalent states. This should be true for skilled human opponents, but for the general player or indeed another reinforcement learning algorithm there is no guarantee of this.

1.3 Exercise 1.3: Greedy Plan

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?

It would likely play worse than a nongreedy player. To begin with, all non-terminal states have value 0.5, so the greedy player has to select moves randomly. She could get lucky and win from a suboptimal board state, in which case the move would be reinforced and the player could become locked into that move whenever she encountered the possibility of moving into the same state again. Alternatively, the greedy player could be unlucky and lose from a good board state, in which case the state's value would be reduced and moves which led to that state may never be chosen again.

1.4 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?

When we do not learn from exploratory moves the value of each state converges to the true probability of winning from that state, assuming that after convergence we always move greedily. When we do learn from exploratory moves the value of each state converges to the true probability of winning from that state, assuming that after convergence we continue to make exploratory moves.

For example, consider a board state with two empty spaces (and thus two possible moves) such that player X wins if she chooses one of the moves and draws if she chooses the other. As player X can always win from this state, if she only learns from greedy moves then the value of this state will converge to 1. However, if she also learns from exploratory moves, and assuming that the probability of choosing an exploratory move is ϵ, then the value of this state will converge to $1 \cdot (1 - \epsilon) + 0 \cdot \epsilon = 1 - \epsilon$.

Assuming that we do continue to make exploratory moves, the probabilities which are learned from both greedy and exploratory moves would be better to learn and lead to more wins because, as discussed above, they are the true probabilities of winning when making exploratory moves and as such they define the optimal policy for this scenario.

1.5 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?

Instead of backing-up the value of the next state to the previous state only, we could back-up the value to two or more time steps before, perhaps with some discount factor which gives less weight to states further back. In the extreme we could back-up the value to all previous states in the game.

As well as reducing the step-size parameter over time we could also reduce the probability with which we make exploratory moves. For example, we could start with a fairly high probability such as 0.5 and gradually reduce it to 0.01.

As discussed in the solution to Exercise 1.1, setting the value of draws to 0, the losses, leads to the reinforcement learning algorithm learning very short-sighted policies when it plays against itself. We believe that setting the value of draws to 0.5 would force the algorithm to learn more sophisticated policies which would lead to more wins against a human opponent.

To speed up learning the algorithm could play n games in parallel. Once all n games terminated the n different value functions would be averaged and this mean value function would be used as the initial value function in the next round of n games.

Alternatively, we could try to solve the problem using a search tree or a meta-heuristic such as a genetic algorithm or particle swarm optimisation, but we do not believe that these methods would perform better that the reinforcement learning algorithm, which is able to converge to an optimal policy.

2 RL Tutorial

2.1 Problem 1

How is Reinforcement Learning different from Supervised Learning?

In supervised learning we have a set of input-output pairs. Each output tells us the "correct answer" for the corresponding input, and we use these examples to learn a mapping from inputs to outputs. The goal is to learn such a mapping which minimises the generalisation error on new, unseen data.

In reinforcement learning we instead learn a policy which maps states to actions. States are analogous to inputs in supervised learning, but in this case we do not have outputs which indicate how to act in each state. Instead we have a numerical reward signal which gives some feedback about the goodness of the action. This signal may be stochastic and rewards may be delayed. The

goal is to learn a policy which maximises the total long-term reward. For this we must trade-off exploration and exploitation to find the best actions and then use that information to act in a way that maximises the reward.

The comparison here is flawed because it claims that in reinforcement learning there is no direct mapping. In fact, the policy which we try to learn is a mapping from states to actions, the difference is that we need to explore the actions space to find the best policy.

2.2 Problem 2

Provide two different examples of applications of Reinforcement Learning in the industry. What aspects of these problems make them solvable by Reinforcement Learning?

Where I work we are trying to develop an RL system which controls email and push notifications to our customers. This problem is well-suited to RL because a notification that we send today may not have any apparent immediate impact, but may lead to the customer buying something next week. The rewards can therefore be delayed. For the same reason it is also difficult to construct a training set for supervised learning which indicates whether each notification was successful or not. Instead, we need to send many notifications to many customers in a trial-and-error fashion in order to discover the policy which maximises the long-term reward for the business.

Another potential application where I work is using RL to control customer discounts. Here the goal is to maximise the total spent by the customer over his or her lifetime. Again the rewards may be delayed because offering someone a discount today may actually lead to an immediate loss in revenue if the customer was going to buy the product anyway, but the discount may strengthen the customer's affinity for the company which could lead to future purchases. This type of long-term strategy would be difficult to learn without some form of trial-and-error over an extended time period.