Reinforcement Learning: An Introduction Solutions to the Second Edition

Joseph Early, University of Southampton

June 24, 2018

1 Introduction

Exercise 1.1: Self-Play

The main consideration when the reinforcement learning (RL) algorithm plays itself is that its opponent is no longer constant - the opponent is also learning so its policy will change over time. As stated in the example, if the step size remains constant then the algorithm can remain competitive against an opponent that changes strategy over time, so a constant step size would be required in this instance. The overall level of both agents is likely to be greater than against a static opponent as both will learn from their strengths but also have their weaknesses exploited by their opponent. The final policies are likely to be better against any static opponent as a more general policy will have been found rather than one that just exploits a single opponent's weaknesses.

Exercise 1.2: Symmetries

By using the symmetrical properties of tic-tac-toe, the number of states in the value function can be significantly reduced, which will speed up the learning process. However, this assumes that the opponent makes the same move according to the symmetry of the position, otherwise symmetrical states would have different values, in which case there is a potential weakness in the opponents policy that is not exploited. Since the state space for tic-tac-toe is relatively small, it is more important to look for an opponent's weakness by treating symmetric states as independent than using symmetries to reduce the number of states, especially when a draw is considered as bad as a loss.

Exercise 1.3: Greedy Play

An RL agent that acted greedily would still be able to beat a constant player, however it would likely take longer to find a good strategy, and its policy may not be optimal, as once it had found a strategy that worked it would always exploit it rather than exploring for potentially better strategies. A greedy agent is likely to be more brittle as well - it would not have seen as many states as a non-greedy agent, therefore if it came up against a different opponent it would have less understanding of certain positions compared to a non-greedy agent that had explored those positions already.

Exercise 1.4: Learning from Exploration

When learning from just greedy moves, the probabilities converge to the probability of winning from a state given that the best move is made. On the other hand, if learning occurs from all moves (greedy and exploratory), the probabilities converge to the probability of winning from a state given that the best move is made with probability $1 - p$ and a random move is made with probability p, where p is the rate of exploration. If an agent continues to make exploratory moves after convergence, it is better to use the second probability set (the set produced by learning from both greedy and exploratory moves), as it is a more accurate representation of the agent's strategy so should result in more wins.

However, it is unnecessary for the final agent to keep making exploratory moves as it has already determined which moves are the best (assuming a constant opponent) - it is actually detrimental to keep exploring. An agent that doesn't explore once the probabilities have converged will achieve more wins than an agent that continues to explore, however the greedy agent should use the probability set produced by only learning from greedy moves as this represents the strategy that the agent actually follows. Therefore the best way to maximise wins for the final agent is to only learn from greedy moves and to reduce the exploration rate over time.

Exercise 1.5: Other Improvements

One way to produce an agent that is a better overall tic-tac-toe player is to expose it to multiple opponents with different strategies. This means the agent will learn a more general strategy that will be better against any opponent rather than just the one is has played against, and that the agent will experience a greater number of states than it would do by playing against a single opponent. In this manner it could also learn strategies for playing as both noughts and crosses. This could be implemented by choosing an opponent at random (from a pool of possible opponents) for each game the agent plays.

Since the number of states is relatively small for tic-tac-tie, an alternative method would be to present each possible state to the opponent (assuming it plays deterministically) and establish a model for which action it will take in each state. It is then trivial to find an optimal policy to use against this opponent by adapting minimax search for the opponent's policy.

2 Finite Markov Decision Processes

Exercise 3.1: Three MDP Examples

1. **Stock Market Agent** Buying and selling shares on the stock market or cryptocurrency exchanges could be presented as a finite MDP. The states would be the various stock/currency information such as opening price, closing price, high prices and low prices for some period of time. The possible actions would be to buy and sell various assets, with the rewards being an increase or decrease in overall value of the currently held assets.

2. **Football Player** This could be presented as an individual player (e.g. a striker) or a 'team controller' as seen in video games such as Fifa. This example focuses on the latter case, as the former could be considered a partially observable MDP. The state would be the current player positions, the ball position, and statistics such as the score and previous opposition tactics (to capture everything previously seen about the game and thus satisfy the Markov property). The actions would be a vector of commands for each player such as pass here, run here, shoot etc. The obvious reward is the overall score, however this is too sparse a reward to be of any use. More fine-grained rewards such as completing a successful pass and getting a shot on target would make it easier to learn how to play, however the reward function would need to be carefully constructed such that the agent doesn't just pass the ball between defenders as this gives reward without much risk of losing possession.

3. **Spam Detection** The problem of spam detection, although much less of an issue in recent times, could be presented as a MDP. The states would be the incoming messages, along with a history of which messages this particular user had considered spam and which they hadn't. The agent only has two actions - mark a message as spam or allow it to enter the user's inbox. One way to formulate the reward function is to give a negative reward for every email that enters the user's inbox and is subsequently marked as spam - this represents a message 'slipping through'. Only the other hand, the agent could incorrectly identify messages as spam that are actually important to the user, this would result in a negative reward if the user marks a message as not spam.

Exercise 3.2: MDP Limitations

Beginning with states, not all possible states will satisfy the Markov property - there might be information about previous states that aren't represented in the current state, for example an important object passing out of site of an agent's sensors will appear to no longer to exist unless the agent stores information regarding the previous states in some form of memory. This assumes the agent's memory is under complete control of the agent, viz. it is not part of the environment. This leads to an extension of the MDP framework - the partially observable MDP - in which the state the agent receives from the environment is not the complete state of the environment.

With actions, the agent is limited based on the rules of the environment. This is fine in most cases if the action set is well designed, however more optimal strategies could be missed if an agent is given an incomplete action set. For example, if an agent only has the actions *move left* and *move right*, the agent is forced to move on every step therefore it might not find the best strategy if it could benefit from not moving in certain states. Although this is a trivial case, when applying RL to more complex problems it may be beneficial to make the agent able to take actions that weren't originally considered in the problem specification, as this will allow the agent to find more optimal strategies.

Finally, a limitation of the rewards used in MDPs is that they are restricted to a single number. This is fine when optimising for a single objective, but often it is useful to optimise for two or more objectives, usually when looking for a trade-off between different criteria. While it is possible to condense any number of objectives into a single number, this brings additional parameters that need tuning to specify the importance of each different reward function. Although not considered 'true' RL methods by the text, genetic algorithms can be used for multi-objective optimisation to produce a range of Parteo-optimal solutions to a problem, thus highlighting a potential advantage they have over methods that just solve MDPs as outlined in the text.