Notes on *Reinforcement Learning: An Introduction, Second Edition* by Richard S. Sutton and Andrew G. Barto

Luke Renchik

March 24, 2024

1 Session Outline

1.1 Sections Covered

- Preface to the Second Edition
- Chapter 1: Introduction

1.2 Concepts Covered

- General Notation.
- Definitions of ML & RL Methodologies.
- The four main sub-elements of a RL Model.
- Example: Training a RL model to play tic-tac-toe.

2 Preface to the Second Edition

2.1 Important Notes

- Omit Sections with a * on first read, as they are more advanced.
- Reinforcement learning is inherently hedonistic, as it actualizes a system of learning that promotes a modification of behaviors to actualize some desired state or signal from its environment.

3 Chapter 1

3.1 Key Concepts

- *Trial-and-Error Search* and *Delayed Reward* are the two distinguishing features of reinforcement learning.
- The problem of reinforcement learning uses ideas from dynamical systems theory, specifically optimal control of incompletely-known Markov decision processes. Markov decision processes are intended to include just three aspects–sensation, action, and goal.
- Exploration-Exploitation Dilemma - A learning agent must explore a group of solutions to gain an understanding of how those actions will effect the environment and the reward mechanism underlying it. The agent can then utilize these explored paths as future exploits to maximize reward. The challenge in building these agents arises in balancing exploration for higher-order rewards and maximizing exploitations.
- An agent can be a component of a larger system, and it need not see all signals from the parent system. An example of a child agent is a monitoring system that tracks the charge level of a robot's battery and sends commands to the robot's control architecture.
- Four Main Sub-elements of a Reinforcement Learning System - *a policy, a reward signal, a value function*, and optionally a *model* of the environment.
- A *policy* is mapping from perceived states of the environment to the actions to be taken.
- A *reward signal* governs short term decision making.
- A *value function* governs long term goals.
- A *model of the environment* is a system that allows the agent to predict how its actions will affect the environment. Models are used for planning a course of action and thinking multiple states ahead of the current position.
- The "state" represents all information available to the agent about its environment, this text will not investigate how to maximally create the signals to describe the environment, but instead how to make decisions within the environment given the information provided by the states.
- Evolutionary Methods - These models do not use estimated value functions (unlike many of the examples within the text). If the space of policies is sufficiently small, or good policies are easy to find, or if a lot of time is available for the search, then evolutionary methods can be effective. Evolutionary methods have advantages on problems in which the learning agent cannot sense the complete state of its environment.
- Evolution methods are ignored within this text because the learning outcomes are slower and less predictable, this is due to the agents not being able to make decisions and inferences on individual actions, only the general trends of a population.
- A *greedy* move is one in which the agent selects a transition to the state with the highest available value (estimated probability of winning).

- A *exploratory* move is one which a transition is selected randomly, because they cause us to experience states which may otherwise not be seen.
- While playing the game (tic-tac-toe) we modify the probabilities of winning after greedy moves by adjusting the previous state to be closer to the value of the later state.
- If we let S_t denote the state before the greedy move, S_{t+1} is the state after that move, then the update to the estimated value of S_t, denoted $V(S_t)$, can be written as: $V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$
- α is a small positive fraction called *step-size parameter*, which influences the rate of learning
- This is considered a *temporal-difference* learning method because its changes are based on a difference between two estimates at successive times.
- The examples in this book will be in discrete space, although possible to model continuous space, the theory becomes increasingly complicated and out of the scope of the text.

3.2 Important Formulas

Updating the estimated value of S_t: $V(S_t) \leftarrow V(S_t) + \alpha[V(S_{t+1}) - V(S_t)]$

3.3 Personal Insights

- I spent a large majority of my time as a youth purely exploiting the environment around me, very timid towards exploratory actions, this strategy allowed me to get very skilled at certain disciplines, but limited the scope of actions I felt confident performing in many given states. I wonder how the development of individual machines differ based on their propensity to explore vs exploit, and how much control us as the creators have over this balance.

3.4 Exercises

- 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?
- In this case, my suspicion is that both would eventually work out the optimal solution to the game, both models begin as non-skilled players who are able to make mistakes, but winning patterns will be rewarded and losing patterns penalized; my intuition is that this will eventually converge on the same model as the model trained against an unskilled opponent.
- 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?
- This one is challenging, abstracting the board away and only visualizing the relationships is something I am unable to do with my current skillset. In a crude implementation it seems possible to "rotate" each state and see if it has any matches with other existing states, then tie the increase/decrease in the value of each of these states together. This process would decrease the amount of total iteration necessary to achieve the optimal set of weights, thus increasing

the speed of learning. If the opponent reacts differently to symmetrically similar positions, and our goal is explicitly to win the maximum number of games possible, we should not view these cases the same, this is because if there is a path the opponent does not understand in a specific orientation, we could learn to exploit that path and win more games, even if that is not the optimal solution for other symmetric versions of that board state.

- 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 non-greedy player? What problems might occur?

- This player would be inherently flawed, as it would continue to select the same strategy repeatedly, until it became less valuable than another strategy, until it settles on a single set of strategies for any given board state, but this player has a large possibility space of unexplored strategies that may lead to better outcomes but were initially rated lower than the "current optimal strategy", this would plateau the greedy player.

- 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?

- The Optimized Set and The Confused Set - We choose to learn from the optimal moves because the state previous is the one that will lead us to that (hopefully winning) position. If we were to adjust the values during our exploratory moves, we would be wrongly decreasing the value of our initial state due to our selection of a sub-optimal move. I dub this the confused set as it will never converge on a understood strategy as the exploratory moves will continue to shift the values.

4 Closing Thoughts

There seems to be a lot of new notation with this book, especially with the Markov process, but the general notation is grounded in probability and computation, which is familiar.

Reinforcement learning seems to closely model the study of human behavior, but yet feels distinctly different. I suspect teaching a machine to perceive and learn will be uniquely challenging as a machines inputs are vastly different than that of a human, there is no emotion, mood, confidence, fatigue, or other preferences except those that we as designers of the system define. When teaching a human, it is important to build trust, break any tension about the subject being studied, formally introduce a concept, then informally demonstrate the idea, before letting the participant explore their new idea "sandbox". I suspect these steps will have a transformation when applied to teaching machines, but to understand where these heuristics align and where alternative mental models must be constructed, more knowledge is required.

This seems like an extraordinarily intriguing area of study, I'm curious to see how steep the difficulty curve is on the mathematics and implementation details.

Continue at Part I: Tabular Solution Methods

5 Appendix

5.1 General Notation

Random Variables - Upper Case
Instantiated Variables - Lower Case
At time step t:
State = S_t
Action = A_t
Reward = R_t
The specific values would be s, a, r.
Value functions - lower case (e.g., v_π)
Tabular Estimates - upper case (e.g., Q_t(s,a))
Approximate value functions are deterministic functions of the random parameters and written in lowercase ($\hat{v}(s, \mathbf{w_t}) \approx v_\pi(s)$)
Bold Letters represent vectors
Uppercase Bold is used for matrices
Transition for State Function - $p(s', r|s, a)$
$\doteq$ - Equality relationship that is true by definition
$\propto$ - Proportional To
$\Pr\{X = x\}$ - Probability that a random variable X takes on the value x
$X \sim p$ - random variable X selected from distribution $p(x) \doteq \Pr\{X = x\}$
$\mathbb{E}[\mathbb{X}]$ - Expectation of a random variable X, i.e., $\mathbb{E}[\mathbb{X}] \doteq \sum_x p(x)x$)
$\ln x$ - Natural Log of X
e^x, exp(x) - The base of the natural log, e carried to the power x; $e^{\ln x} = x$
$\mathbb{R}$ - Set of real numbers
$f : X \to y$ - Function f from elements of set X to elements of set y
$\leftarrow$ - Assignment
ϵ - Probability of takin a random action in an ϵ greedy policy.
α, β - step-size parameters
γ - Discount Rate Parameter λ - decay-rate parameter for eligibility traces
$\mathbb{I}_{predicate}$ = indicator function ($\mathbb{I}_{predicate} \doteq 1$ if the *predicate* is true, else 0)

5.2 Markov Decision Process Notation

TODO: Add the MDP Notation to Template.

5.3 Important Figures

As shown in Figure 1, ...

5.4 Code Snippets

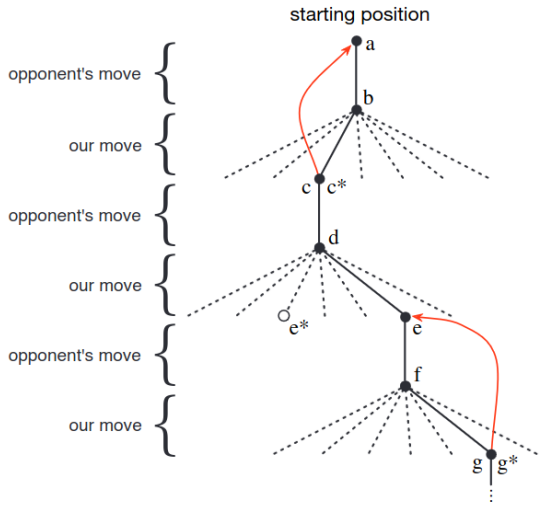

Figure 1.1: A sequence of tic-tac-toe moves. The solid black lines represent the moves taken during a game; the dashed lines represent moves that we (our reinforcement learning player) considered but did not make. The * indicates the move currently estimated to be the best. Our second move was an exploratory move, meaning that it was taken even though another sibling move, the one leading to e^*, was ranked higher. Exploratory moves do not result in any learning, but each of our other moves does, causing updates as suggested by the red arrows in which estimated values are moved up the tree from later nodes to earlier nodes as detailed in the text.

Figure 1: Chapter 1.5 TicTacToe Example