Summary - Chapter 3

Ameet Deshpande

December 2, 2017

1 Summaries

1.1 Section 3.1 - The Agent-Environment Interface

- The agent's policy determines from which state, with what probability are the actions taken. It is thus a mapping between states and actions, denoted by $\pi_t(s, a)$, which denotes the probability with which the action a is taken when in state s. The time dependence is obvious here given the fact that the policy changes as time proceeds, allowing the agent to learn.
- In general, anything that an agent cannot control is considered as part of the environment. Note that this does *not* mean that the agent does not know anything about the environment. The agent is expected to have an idea about how the reward is calculated as a function of the states and the actions. That is at the end of the day what we expect in Reinforcement Learning. But the reward computation we assume happens outside of the agent and the agent can obviously not change the function arbitrarily.
- In general, the state that the agent is in could be a list, or a vector of parameters. Could be the temperature, fan speed, efficiency etc., of a machine. But reward attained by transitions are always single numbers.
- Though we place the reward function outside of the agent, it does not preclude the agent on defining internal rewards based on which it decides to take actions. In fact, many reinforcement learning methods are known to adapt this technique.

1.2 Section 3.2 - Goals and Rewards

- It needs to be well understood that an agent's goal is to maximize the total reward and not just the immediate reward.
- The reinforcement framework is built on top of the principle that the agent maximizes the reward. Thus, our reward function should be engineered in such a way that achieving the goal is in tandem with maximizing the reward. The reward signal can thus be considered as a way to communicate to the agent what exactly is expected out of it.
- Though we place the reward function outside of the agent, it does not preclude the agent on defining internal rewards based on which it decides to take actions. Infact, many reinforcement learning methods are known to adapt this technique.

1.3 Section 3.3 - Returns

- In general, the aim is to maximize the expected return. In the simplest case, $R_t = r_{t+1} + \cdots + r_T$. This definition makes sense where there is a notion of final step, else it would become an infinite sum.
- The above approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent-environment interaction breaks naturally into subsequences, which we call *episodes*. On the contrary, there are *continual* tasks which are like never ending games.
- Continual tasks force us to introduce the concept of discounts. The discounted reward is calculated as, $\sum \gamma^k \times r_k$. The parameter γ is called the discount rate. It can be observed from the formula that as γ goes closer to 1, the agent becomes more far-sighted.

1.4 Section 3.4 - A Unified Notation for Episodic and Continual Tasks

- For *episodic* tasks a new state called the absorbing state is introduced which always churns out zero rewards. This way, episodic tasks can be treated like continual tasks.

1.5 Section 3.5 - The Markov Property

- If a state signal retains all the important information required from the past, it is called a Markov State. It thus does not matter from which state this state was reached, the history does not matter. This state signal contains all the important information embedded in its state signal.
- $P\{s_{t+1} = s^f, r_{t+1} = r | s_t, a_t\}$ basically is as accurate as the probability which conditions based on all previous states.
- In many problems considered, the state might not actually be Markov but it is considered so as an approximation.

1.6 Section 3.6 - Markov Decision Processes

- An RL problem which satisfies the Markov State is called Markov Decision Processes.
- $P^a_{ss^f}$ and $R^a_{ss^f}$ represent the probability of going to s^f and expected reward respectively.
- *Transition Graphs* are useful to represent Markov Decision Processes. Note that the same action can take s to different states based on probabilities.

1.7 Section 3.7 - Value Functions

- Value, denoted by $V^\pi(s)$ is the expected reward when the agent is in state s and then uses policy π. Similarly, $Q^\pi(s, a)$ is the action value function. V and Q are generally maintained for each state, but if the number of states are too many, then we would have to maintain the values in a parameterized form so that they can be calculated on the fly.

- The Bellman equation for the value function V is $V^\pi(s) = \sum \pi(s,a) \sum P^a_{ssf} \times [R^a_{ssf} + \gamma V^\pi(s^f)]$.
- *Backup diagrams* are used to show the relationships that form the basis for the backup that happens in the Bellman Equations.

1.8 Section 3.8 - Optimal Value Functions

- It can be easily argued that there is one policy π^* which is at least as good as or better than every other policy. This is an optimal policy. Clearly there could be more than one optimal policies.
- When we are choosing the optimal action, we would clearly want to choose the action which gives the best rewards. So in any state, we would want to pick the action which has the highest return. Thus, $V^*(s) = \max_a Q^*(s,a)$. This way note that the optimal policy will be such that it always chooses the same action. One possible violation could occur when there are multiple actions which have the same *action-values*. If $Q(s,a_1) = Q(s,a_2)$, the policy π could be such that it choose these actions with probabilities p_1 and p_2 such that $p_1 + p_2 = 1$. There could thus be a horizon of policies available. This can easily be generalized to the case of multiple action-values being equal, $\sum p_i = 1$
- For finite MDPs, the Bellman optimality condition has a unique solution.
- In approximating optimal behavior, the agent might play very badly on states it thinks it will encounter with very low probabilities. This way, it can probably cut down on a large amount of computational cost.

2 Exercises

2.1 Exercise 3.1

Devise three example tasks of your own that fit into the reinforcement learning framework, identifying for each its state representations, actions, and rewards. Make the three examples as different from each other as possible. The framework is very abstract and flexible and can be applied in many different ways. Stretch its limits in some way in at least one of your examples.

- Playing the game of GO. Label states of the game as states. The exponential number of states might force the agent to use tricks like symmetry etc.
- Teaching a robot how to walk. This is essentially a task of control and coordination.
- Letting a robot figure out the fastest paths between different points. We need to design the rewards based on the time it takes to reach the points.

2.2 Exercise 3.2

Is the reinforcement learning framework adequate to usefully represent all goal-directed learning tasks? Can you think of any clear exceptions?

A lot of engineering will go into designing the correct reward function. Also, will the problem be able to handle infinite states with continuous time? The formulation in the book says that time is a discrete unit.

2.3 Exercise 3.3

Consider the problem of driving. You could define the actions in terms of the accelerator, steering wheel, and brake, i.e., where your body meets the machine. Or you could define them farther out, say where the rubber meets the road, considering your actions to be tire torques. Or, you could define them farther in, say where your brain meets your body, the actions being muscle twitches to control your limbs. Or you could go to a really high level and say that your actions are your choices of where to drive. What is the right level, the right place to draw the line between agent and environment? On what basis is one location of the line to be preferred over another? Is there any fundamental reason for preferring one location over another, or is it just a free choice?

The choice will depend on what we want to control. For example, if we want to control the path taken when we are driving, we might choose the brain plus the car as the agent. Anything that constraints the path. We might not include the accelerator however because we maybe interested in path length. At the same time, we might want to use the accelerator and breaks as the actions if the task is to control the car rather than choose the right path.

2.4 Exercise 3.4

Suppose you treated pole-balancing as an episodic task but also used discounting, with all rewards zero except for a -1 upon failure. What then would the return be at each time? How does this return differ from that in the discounted, continual formulation of this task?

If the task is episodic, then we are probably looking at a task of balancing the pole for a time τ. The returns will thus be 0 whenever the agent is able to balance the pole for that time period and it will be negative when the pole drops. What is interesting is that the pole might be falling unstably at time instant τ but the agent will not care about this as it has achieved the maximum possible reward. In the continuous case, the returns are always negative (0 in limiting case).

2.5 Exercise 3.5

Imagine that you are designing a robot to run a maze. You decide to give it a reward of +1 for escaping from the maze and a reward of zero at all other times. The task

seems to break down naturally into episodes—the successive runs through the maze—so you decide to treat it as an episodic task, where the goal is to maximize expected total reward (3.1). After running the learning agent for a while, you find that it is showing no improvement in escaping from the maze. What is going wrong? Have you effectively communicated to the agent what you want it to achieve?

It is not conveyed to the robot that it has to minimize the number of steps. It is possible that the robot gets stuck in loops with no incentive to minimize the number of steps it has taken. It thus gets stuck with zero reward.

2.6 Exercise 3.6

Broken Vision System. Imagine that you are a vision system. When you are first turned on for the day, an image floods into your camera. You can see lots of things, but not all things. You can't see objects that are occluded, and of course you can't see objects that are behind you. After seeing that first scene, do you have access to the Markov state of the environment? Suppose your camera was broken that day and you received no images at all, all day. Would you have access to the Markov state then?

Borrowed Answer: It depends on what your task is. The Markov state is based on what information provides a complete state of the information necessary to make decisions. For any environment there are assumptions made about the possible changes, so as long as the environment continues to operate under those assumptions then you can have a Markov state.

If the camera was broken you wouldn't necessarily be able to determine the next reward you received (again, dependent on the task). In general this would not be considered the Markov state

2.7 Exercise 3.7

Assuming a finite MDP with a finite number of reward values, write an equation for the transition probabilities and the expected rewards in terms of the joint conditional distribution in (3.5).

Refer write up.