Reinforcement Learning with Markov Models

Caroline Sun and Krish Ganotra

April 2020

1 Introduction

In typical reinforcement learning problems, the agent is the learner and decision maker. The environment provides rewards and a new state based on the actions of the agent. So, in reinforcement learning, we do not teach an agent how it should do something but, instead, present it with rewards positive or negative based on its actions. The root question we are trying to answer in this lecture is how to formulate any problem in reinforcement learning mathematically. This is where the Markov Decision Process (MDP) comes in. In order to do this, we must develop our understanding of a couple of concepts, discussed next.

2 Background

2.1 Agent-Environment Relationship

Anything the agent cannot arbitrarily change is part of the environment. However, this does not mean the agent is altogether unaware of the environment. For example, reward calculation is part of the environment, and the agent knows a bit about reward calculations based on its actions. Further, even if the agent were to be fully aware of its environment, that does not guarantee it can play well. For example, we might know how to play a Rubik's cube, but still not be able to solve it. From this, we can determine that the agent-environment relationship represents the limit of the agent control and not its knowledge.

2.2 The Markov Property

"Future is independent of the past given the present." Mathematically, we can express this as:

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots\ldots, S_t]$$

This intuitively means that our current state already captures the information of the past states.

For reinforcement learning, the Markov property can be represented by the *state transition probability matrix*. In the matrix, $p_{i,j}$ is the probability that the agent will choose state i given the current state is j. Notice the agent does not take any previous states into account. Each column should add to 1:

$$P = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,j} & \cdots \\ p_{2,1} & p_{2,2} & \cdots & p_{2,j} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots \\ p_{i,1} & p_{i,2} & \cdots & p_{i,j} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \ddots \end{pmatrix}.$$

Figure 1: State Transition Probability Matrix

2.3 Rewards and Returns

Rewards are the numerical values that the agent receives on performing some action at some state(s) in the environment. The numerical value can be positive or negative based on the actions of the agent.

In Reinforcement learning, we care about maximizing the cumulative reward (all the rewards agent receives from the environment) instead of, the reward agent receives from the current state (also called immediate reward). This total sum of reward the agent receives from the environment is called returns.

2.4 Episodic vs Continuous Tasks

Episodic Tasks are the tasks that have a terminal state. We can say they have finite states. For example, in racing games, we start the game (start the race) and play it until the game is over (race ends!). This is called an episode. Once we restart the game it will start from an initial state and hence, every episode is independent.

Continuous Tasks are the tasks that have no terminal state. These types of tasks will never end. For example, learning!

It's easy to calculate the returns on episodic tasks as they are finite. However, if we sum rewards from continuous tasks, they will go on to infinity! Thus, we must redefine returns for continuous tasks.

This is where a discount factor comes into play. A discount factor is a value between 0 and 1, determining the importance given to immediate rewards vs future rewards. It has a value between 0 and 1. A value of 0 means that more importance is given to the immediate reward and a value of 1 means that more importance is given to future rewards. In practice, a discount factor of 0 will never learn as it only considers immediate reward and a discount factor of 1 will go on for future rewards which may lead to infinity.

What value of a discount rate should we use? It depends on the game. For example, in chess, we would want to use a larger discount rate, as the ultimate goal of capturing the king is far more important than capturing a pawn.

2.5 The Markov Reward Process

In summary: each state has a reward, choose the best one! We still have to choose a function that defines the reward at each state.

2.6 Bellman Equation for Value Function

Bellman Equation helps us to find optimal policies and value function.We know that our policy changes with experience so we will have different value function according to different policies. Optimal value function is one which gives maximum value compared to all other value functions.

Bellman Equation states that value function can be decomposed into two parts - immediate reward and discounted value of successor states.

3 Bellman Expectation Equation

A quick review of Bellman Equation we talked about in the previous section:

$$v(s) = \mathbb{E}\left[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s\right]$$

Figure 2: Bellman Equation for Value Function (State-Value Function)

From the above equation, we can see that the value of a state can be decomposed into immediate reward (R[t+1]) plus the value of successor state (v[S (t+1)]) with a discount factor. This still stands for Bellman Expectation Equation. But now what we are doing is we are finding the value of a particular state subjected to some policy(π). This is the difference between the Bellman Equation and the Bellman Expectation Equation.

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\right]$$

Figure 3: Bellman Expectation Equation for Value Function (State-Value Function)

4 Optimal Value Function and Optimal State-Action Value Function (Q - function)

In an MDP environment, there are many different value functions according to different policies. The optimal Value function is one which yields maximum value compared to all other value function. When we say we are solving an MDP it actually means we are finding the Optimal Value Function.

Similarly, Optimal State-Action Value Function tells us the maximum reward we are going to get if we are in state s and taking action a from there on-wards.

5 Optimal Policy

How do we define an optimal policy? We say that one policy(π) is better than other policy (π') if the value function with the policy π for all states is greater than the value function with the policy π' for all states.

6 Bellman Optimality Equation for State-Value Function

Bellman Optimality equation is the same as Bellman Expectation Equation but the only difference is instead of taking the average of the actions our agent can take we take the action with the max value. For large MDPs, we would use dynamic programming algorithms like policy iteration and value iteration.

7 Applications

Many of the applications of reinforcement learning can utilize Markov Models, namely, ones that are mainly concerned with the present state over past states. Applications of reinforcement learning with Markov models are wide ranging and include:

- Natural Language Processing - Speech Transcription, Translation
- Computer Vision - Recognition, Detection, Perception
- Finance - Managing Portfolios, Trading, risk management
- Games - Go, Poker, Chess
- Transportation - Adaptive Traffic Signals

8 References

- http://www.cs.cmu.edu/ 10601b/slides/MDP_RL.pdf
- https://medium.com/@jonathan_hui/machine-learning-hidden-markov-model-hmm-31660d217a61
- https://hackernoon.com/from-what-is-a-markov-model-to-here-is-how-markov-models-work-1ac5f4629b71
- https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da
- https://towardsdatascience.com/reinforcement-learning-markov-decision-process-part-2-96837c936ec3
- https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf