[Lecture-2] Reinforcement Learning- David Silver

Notes by Ritobrata Ghosh (ghoshr<at>zohomail<dot>in)

Notes meant to be read *with* the Lectures and Slides. Reading these notes in a standalone manner is not recommended.

---

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/beb78bc4-44c5-4ab4-9068-d1500f1d3a25/MDP.pdf

Course Slides [Provided under Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) Lisence]

Markov Decision process formally describes an environment for Reinforcement Learning. Almost all RL problems can be formalised as MDPs.

Environment is fully observable.

Bandits are MDPs with one state. Arms in bandits are stochastic, and they award rewards in a stochastic manner. They have only one state. For a single-armed bandit, it is clear that it is a one-state MDP. State transition doesn't happen after you pull an arm. The arm is in the same state. Time t progresses, but that has no effect on the state of the bandit.

Markov Property

Whatever happens next depends only on the current state. It does not depend on previous states.

> 🤖 ***The future is independent of the past given the present.***

The state is the sufficient statistics of the future.

So, if we have a state s, there is a probability defined that one state changes to another state s'. We call this transition probability.

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

We can form a complete 2D matrix storing the probability of every state transiting to every other states. We call this state transition matrix. It's like a table.

$$\mathcal{P} = \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix}$$

Markov Process / Markov Chain

A Markov process is a sequence of Markov States, i.e. states with the Markov property (if you know the present state, you don't need the past states).

It is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$.

$\mathcal{S}$ is finite set of states. $\mathcal{P}$ is the full transtion probability matrix among those states.

In a certain environment, there are several states with the transition probability matrix of the environment. There are one or more terminal states. Your episode ends if you reach there.

As there are many states, in different episodes, the trajectory of the state should be different. We call these different trajectories sample episodes. And the process of selecting a few episodes is called sampling. In sampling, the transitions and rewards happen according to the transition probabilities.

Random sequence drawn from the environment according to the dynamics of it has variable length.

Note the states and the transition matrix → Create samples according to the dynamics → You have samples which are Markov Chains

---

Markov Reward Process

A Markov Reward Process is a Markov chain with values.

In a Markov chain, we had $\langle \mathcal{S}, \mathcal{P} \rangle$, we further add $\mathcal{R}$ and γ.

$\mathcal{R}$is reward function, we only care about immediate reward. If the agent is in a state, then, we care about how much award the agent gets from that state only.

$$\mathcal{R}_s = \mathbb{E}[R_{t+1} \mid S_t = s]$$

Our goal becomes to maximize the accumulated sum of rewards.

γ is a discount factor.

Markov Reward Process can be described by this tuple- $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

Return

Return is the discounted sum of rewards from a time step t.

Our goal is to maximize the return. To make this return finite and easy to compute, we multiply rewards with discount factor γ.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$\gamma \in [0, 1]$

(There is no notion of expectation here, because, rewards are noted from samples, which are very distinct, and deterministic, as we are calculating rewards from samples, there is no notion of expectation $\mathbb{E}$.)

$\gamma = 0$ means very short-sighted agent. It only looks at the immediate reward.
$\gamma = 1$ means the agent takes into account all future rewards eqaully.

We discount with γ once for each time step into the future. For one step ahead, we discount with γ, for two time-steps ahead, we discount with $\gamma \times \gamma = \gamma^2$.

We use discount factor because-

- We want to focus on reward now rather than reward later. A very practical and concrete example of this is portfolio management applications. We want $10 now rather than $10 5 months later, because the value of money deteriorates, and we miss the chance to compound.
- And there are also more uncertainty about the future.
- To keep it mathematically convenient.
- Avoid infinite returns in cyclic Markov processes.

If we know all sequences terminate, we can choose to not use discount factor, i.e. $\gamma = 1$.

Value Function

The value function $v(s)$ gives the long term value of state s.

We really care about this quantity. What is the total reward we get if we are in a state? That is value function.

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

So, value function takes in a state, and outputs expected return for being in that state.

State $\rightarrow v(s) \rightarrow$ Value (expected return (total discounted rewards from that state onwards))

We can calculate value functions in different ways.

We can sample random samples and then average over them if we want to calculate the value function.

Value function is not random, it is an expectation over quantities we gained from randomly sampling.

Bellman Equation

Bellman Equation gives us a recursive relation involving value functions of consecutive states. Basically, the value function can be broken down into two quantities- 1. immediate reward, and, 2. the discounted value of the next state.

$$
\begin{aligned}
v(s) &= \mathbb{E}[G_t | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma (R_{t+2} + \gamma R_{t+3} + \ldots) | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma G_{t+1} | S_t = s] \\
&= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]
\end{aligned}
$$

Bellman equation just tells us that we can see value functiuons as the reward of the current state summed with the discounted value function of the next state.

In the backup diagrams, we average over the values of the successor state to get the value of the current state.

We can have a vector version of the Bellman Equation.

$$
\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & \ddots & \vdots \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}
$$

We can calculate the values of all states in one sweep.

The Bellman Equations are linear, and can therefore be solved directly. The Bellman equations just tell us about values, and when we turn to how to

maximize the values, the process is not as simple as solving linear equations.

$$
\begin{aligned}
v &= \mathcal{R} + \gamma \mathcal{P} v \\
v(I - \gamma \mathcal{P}) &= \mathcal{R} \\
v &= (I - \gamma \mathcal{P})^{-1} \mathcal{R}
\end{aligned}
$$

Complexity is $O(n^3)$. Not practical.

To solve these in more efficient manner, we do-

- Dynamic Programming
- MC Evaluation
- TD learning

We do the above just so we can calculate values more efficiently.

Markov Decision Process

So far, we did not have any agency, we did not *do* anything. We just sampled based on environment's statistics.

Markov Decision Process is just Markov Reward Process + Actions.

From now on, our reward depends on the actions we take.

MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

$\mathcal{S}$ is a finite state space.

$\mathcal{A}$ is a finite action space.

$\mathcal{P}$ is a state transition probability matrix. $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$

$\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{P}[R_{t+1} | S_t = s, A_t = a]$

We are factoring in action here. In Markov reward process, we did not have a notion for action. In Markov Decision Process, we take action into account.

Policy

A policy is just a distribution over actions in given states.

You put in a state to π, you get an action back based on the probability distribution.

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$$

Policies are stationary, no matter what time-step you are in, the policy remains the same.

Also, policies don't depend on the history, so MDP policies have Markov property.

You can get an MRP and Markov Chain from MDP by flattening.

Value Function

$v_\pi(s)$ tells us how good it is to be in state s, if we follow policy π.

In state value and action value functions, we consider the immediate reward for the state that we are in. plus the expected state value and/or action value from the next step onwards. The spirit of Bellman Equation holds true here, too.

Bellman Expectation Equations

Bellman Expectation Equations say the same things as before (value function can be decomposed into immediate reward and the discounted value of successor state), but for value functions and action-value functions. Immediate reward + value of successor state if policy π is followed.

$\mathbb{E}_\pi$ basically means the expectation of something when you are following the policy π.

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$
$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

Optimal Value Function

$$v_*(s) = \max_\pi v_\pi(s)$$

Which policy gives us the most rewards if you start from state s, is measured by this.

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Which policy gives you the most reward if you start at s, and take action a.

If you know q_*, you have solved the MDP.

What do we mean when we say "best policy"? What makes a policy better than the other?

$$\boxed{\pi \geq \pi' \text{ iff } v_\pi(s) \geq v_{\pi'}(s), \forall s}$$

One policy is better than the other if and only if it leads to better or equal value **for all states** than the other policy.

For **all MDP**-

- There is always one or more optimal policy
- All optimal policies achieve the optimal value function $\boxed{v_{\pi_*}(s) = v_*(s)}$
- All optimal policies achieve the optimal action value function $\boxed{q_{\pi_*}(s,a) = q_*(s,a)}$

All optimal policies recieve the same amount of reward from the system.

We solve MDPs by finding the optimal set of actions. i.e. the optimal policy. We do this simply by looking backwards.

In case of Bellman Optimality Equations, we take max for actions and average for values.

Bellman Optimality equations are not linear. There is a max, there is an expectation in the equation. We cannot solve this directly. We have to rely on iterative dynamic programming methods, such as-

- Value iteration
- Policy Iteration
- Q-learning

- Sarsa, etc.
