1 Lecture two: Markov Decision Process (MDP)

1.1 Markov Process

The current state characterises the process -¿ we are told the state –¿ environment is fully observable

Almost all RL problems can be formalised as MDPs
–¿ Optimal control primarily deals with continious MDPs
–¿ Any partially observable problems can be converted into MDPs
–¿ Bandits are MDPs with one state

"The future is independent of the past given the present"

$$\mathbb{P}\left[S_{t+1} \mid S_t\right] \;=\; \mathbb{P}\left[S_{t+1} \mid S_1, \ldots, S_t\right] \tag{1}$$

What happens next only depends on what happend on the state before - you can throw away anything else.

For a Markov state s and successor state s', the *state transition probability* is defined as

$$\mathcal{P}_{ss'} \;=\; \mathbb{P}\left[S_{t+1} \;=\; s' \,|\, S_t \;=\; s\right] \tag{2}$$

State transition matrix ρ defines transition probabilites from all states s to all successor states s'.

$$\mathcal{P} \;=\; from \begin{pmatrix} & to & \\ \mathcal{P}_{11} & \cdots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \cdots & \mathcal{P}_{nn} \end{pmatrix} \tag{3}$$

Each row of the matrix sums to 1!
A Markov process is a memoryless random process, i.e. a sequence of random states, S_1, S_2, ... with the Markov propoerty. It is defined as a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$.

- S is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix

1.2 Markov Reward process (MRP)

The Markov Reward Process is defined as a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s \;=\; \mathbb{E}[R_{t+1} \mid S_t = s]$

- γ is a discount factor, $\gamma \in [0, 1]$

The *return* $\mathcal{G}_t$ is the total discounted reward from time-step t.

$$\mathcal{G}_t \;=\; R_{t+1} \;+\; \gamma R_{t+2} \;+\; \ldots \;=\; \sum_{k=0}^{\infty} \gamma^k \, R_{t+k+1} \tag{4}$$

- The discount γ is the present value of future rewards - the closer γ is to zero, the less are later rewards accounted (e.g. more 'short-sighted').
- The value of receiving reward $\mathcal{R}$ after $k + 1$ time-steps is $\gamma^k R$.

Why discount?

- Unless you really trust your model and believe that everything turns out as planned, you need to discount in deviations - Uncertainty about the future may not be fully represented
- Mathematically convenient to discount rewards, avoids infinite returns
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behaviour shows preference for immediate reward
- It is sometimes possible to use *undiscounted* Markov reward process (i.e. $\gamma = 1$), e.g. if all sequences terminate.

The value function

The value function *v(s)* gives the long-term value of state s. It defines the expected return in a MRP starting from state s:

$$v(s) \;=\; \mathbb{E}\left[G_t \mid S_t \;=\; s\right] \tag{5}$$

The value function can be decomposed into two parts:

- immediate reward $R + 1$
- discounted value of successor state $\gamma v(S_{t+1})$

This resolves into the **Bellman equation for MRPs**:

$$v(s) \;=\; \mathbb{E}\left[G_t \mid S_t \;=\; s\right] \;=\; \mathbb{E}\left[R_{t+1} \;+\; \gamma v(S_{t+1}) \mid S_t \;=\; s\right] \tag{6}$$

By averaging all possible outcomes we get

$$v(s) \;=\; \mathcal{R}_s \;+\; \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} \, v(s') \tag{7}$$

The Bellman equation can be concisely using matrices, where v is a column vector with one entry per state

$$v \;=\; \mathcal{R} \;+\; \gamma\,\mathcal{P}\,v \quad \rightarrow \quad \begin{pmatrix} v(1) \\ \vdots \\ v(n) \end{pmatrix} \;=\; \begin{pmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{pmatrix} \;+\; \gamma \begin{pmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{pmatrix} \begin{pmatrix} v(1) \\ \vdots \\ v(n) \end{pmatrix} \tag{8}$$

The Bellman equation is linear. It can be solved directly by $v = (I - \gamma\mathcal{P})^{-1}\mathcal{R}$.

- The Computational complexity is $\mathrm{O}(n^3)$ for n states.
- Direct solution only possible for small MRPs
- Iterative methods for large MRPs, e.g. Dynamic programming, Monte-Carlo evaluation, Temporal-Difference learning

1.3 Markov Decision Process (MDP)

A MDP is a MRP with decisions. It is an *environment* in which all states are Markov.
A MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{A}$ is a (finite) set of actions
- $\mathcal{P}$ is a state transition probability matrix
  $\mathcal{P}^a_{ss'} \;=\; \mathbb{P}\left[S_{t+1} \;=\; s' \,|\, S_t \;=\; s,\; A_t \;=\; a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s \;=\; \mathbb{E}[R_{t+1} \mid S_t = s,\; A_t \;=\; a]$
- γ is a discount factor, $\gamma \in [0, 1]$

A policy π is a distribution over actions given states. It fully defines the behaviour of an agent.

$$\pi(a|s) \;=\; \mathbb{P}\left[\; S_t = s,\; A_t \;=\; a \,\right] \tag{9}$$

In an MDP, the policies depend on the current state (not the history). Policies are stationary: $A_t = \pi(\dot{|}S_t), \forall t > 0$

Given an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and a policy π:

- The state sequence $S_1, S_2, \dots$ is a Markov process $\langle \mathcal{S}, \mathcal{P}^\pi \rangle$
- The state and reward sequence $S_1, , R_2, S_2, \dots$ is a Markov reward process $\langle \mathcal{S},\, \mathcal{P},\, \mathcal{R},\, \gamma \rangle$ where

$$\mathcal{P}^\pi_{s,s'} \;=\; \sum_{a \in \mathcal{A}} \pi(a|s)\, \mathcal{P}^a_{s,s'} \qquad \mathcal{R}^\pi_s \;=\; \sum_{a \in \mathcal{A}} \pi(a|s)\, \mathcal{R}^a_s \tag{10}$$

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π:

$$v_\pi(s) \;=\; \mathbb{E}_\pi[G_t \,|\, S_t \;=\; s] \tag{11}$$

The *action-value function* $q_\pi(s, a)$ of an MDP is the expected return starting from state s, taking action a, and then following policy π:

$$q_\pi(s, a) \;=\; \mathbb{E}_\pi[G_t \,|\, S_t \;=\; s,\; A_t \;=\; a] \tag{12}$$

The state-value and action-value functions can again be decomposed into Bellman equations consisting of immediate reward plus discounted value of successor state:

$$v_\pi \;=\; \mathbb{E}_\pi \left[R_{t+1} \;+\; \gamma v_\pi(S_{t+1}) \;\,|\, S_t \;=\; s\right] \tag{13}$$

$$q_\pi \;=\; \mathbb{E}_\pi \left[R_{t+1} \;+\; \gamma v_\pi(S_{t+1},\; A_{t+1}) \;\,|\, S_t \;=\; s,\; A_t \;=\; a\right] \tag{14}$$

Basically, the state-value averages over the different actions that can be taken:

$$v_\pi(s) \;=\; \sum_{a \in \mathcal{A}} \pi(a|s)\, q_\pi(s,\; a) \tag{15}$$

The other way around, by using the probabilities of the transition dynamics we can average through the values of the successing states we can evaluate a certain action:

$$q_\pi(s,\; a) \;=\; \mathcal{R}_s^a \;+\; \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a\, v_\pi(s') \tag{16}$$

- v is is telling how good it is to be in a particular state
- q is telling how good it is to take a particular action

Sticking these together in order to solve MDPs end up in a two-step lookahead:

- Consider all action we might take next (v)
- Consider all the things the environment might do to us (q)
- Evaluate the successor state after what the environment did after that point

By double-averaging over the policy as well as the transistion probability we get the answer to how good it is to be in a particular state (as the Bellman Equation):

$$v_\pi(s) \;=\; \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a \;+\; \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a\, v_\pi(s') \right) \tag{17}$$

Starting from a particular action, we can also do the same two-step lookahead and see where the wind blows us, and from there consider which action might be taken next. By averaging the same way above we get same recursive relationship:

$$q_\pi(s,\, a) \;=\; \mathcal{R}_s^a \;+\; \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s')\, q_\pi(s',\, a') \tag{18}$$

This shows how the value- and action function of the next step relates to itself, we therefore get a recursive relationship. **After all, the value function of the current time step is equal to the immediate reward plus the value function of where you end up.**

To flatten the MDP into a MRP one could average out the matrices as in (7) following policy π and then solve accordingly as in (8). Thereby we can determine the value function.

.

_____ **For more info on undiscounted MDPs:**

Neurodynamic Programming / Dynamic Programming; Dimitri Bertsekas Dynamic Programming and Optimal Control (2 Vol Set) _____

1.4 The Optimal Value Function

The optimal *state-value* and *action-value* functions are the maximum functions over all policies

$$v_*(s) \;=\; \max_\pi \; v_\pi(s) \qquad q_*(s,a) \;=\; \max_\pi \; q_\pi(s,a) \tag{19}$$

- The optimal value function specifies the best performance in an MDP
- An MDP is 'solved' when we know the optimal value function

If you know q_*, you're basically done. It tells you under all different ways to behave which one gives the most reward.

There is always one optimal policy. This optimal policy will then achieve the optimal *state-value* and *action-value* function v_{π_*} and q_{π_*}. So we can define a partial ordering over policies:

$$\pi \;\geq\; \pi' \quad if \quad v_\pi(s) \geq v_{\pi'}(s), \forall s \tag{20}$$

An optimal policy can be found by maximising over $q_{\pi_*}(s,a)$:

$$\pi_*(a|s) \;=\; \begin{cases} 1 & \text{if } a \;=\; argmax_{a \in \mathcal{A}} \; q_{\pi_*}(s,a) \\ 0 & \text{otherweise} \end{cases} \tag{21}$$

The Bellman Optimality Equation

The optimal value functions are recursively related by the Bellman optimality equations:

Instead of taking the average of all actions, you pick the maximum. That determines the optimal value function of a state:

$$v_* = \max_a \; q_*(s, a) \tag{22}$$

Now looking at the action-value function, we can inductively assume that each of the states we might end up in has a optimal state-value v_*. Therefore, similar as for (16), to determine the action-value we add the immediate reward to the average of the optimal state-values where we could end up in:

$$q_*(s,\, a) \;=\; \mathcal{R}_s^a \;+\; \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \; v_*(s') \tag{23}$$

As we did before in (17), we can put these two together to create the recursive relationship with the two-step lookahead. For the optimal solution, we pick the maximum instead of the average. This is the Bellman optimality equation for v_*:

$$v_*(s) \;=\; \max_a \left(\mathcal{R}_s^a \;+\; \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \; v_*(s') \right) \tag{24}$$

By reordering the same idea, we can again define the Bellman optimality equation for q_*.

$$q_*(s,\, a) \;=\; \mathcal{R}_s^a \;+\; \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \max_{a'} \; q_*(s', a') \tag{25}$$

Solving the Bellman Optimality Equation

- The Bellman Optimality Equation is non-linear
  → Solving by inverting matrices like in (7) will not work
- No closed form solution (in general)
- Many iterative solution methods
  → Value iteration, policy interation, **Q-Learning**, Sarsa