Lecture notes - Introduction to Reinforcement learning with David Silver

Lecture 2

By - Amal Sunny

Markov Decision Process

- An MDP is a formal representation of the environment, such that we can apply tools to it/work with it.
- Case 1: Environment is fully observable
  - Current state tells us all we need to know about how the environment will react onwards
  - Almost all RL problems can be formalised as MDPs
    - Partially observable problems can be converted into MDPs
    - Bandits are MDPs with one state
      - Bandit is a reference to the multi-arm bandit problem, where we are given multiple arms to pull and get different rewards from each arm. Our task is to maximize the reward there.

Markov Property

> A state S_t is Markov if and only if
>
> $$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t]$$

- Which states one single state is all that is needed to determine reward from action at that state. The rest of this history isn't needed for that (so no need to store).

State Transition Matrix

- If we have a state s and its successor s', we can define the probability of it transitioning to that state. This **state transition probability** is a well-defined statistic (due to each state having adequete info for the next one) given by

$$P_{ss'} = P[S_{t+1} = s' | S_t = s]$$

- With that property, we can have an entire matrix storing the transition probability between **all** states.
    - i.e each row has the probability of transitioning from one particular state to all others.
        - Thus, each row would have the sum 1.

$$P = from \begin{bmatrix} P_{11} & ... & P_{1n} \\ \vdots & & \\ P_{n1} & ... & P_{nn} \end{bmatrix}$$

Markov Process

- Markov process is a sequence of random states $(S_1, S_2, ...)$ that has the Markov property.
- Formally, A Markov Process (or Markov Chain) is a tuple $< S, P >$
    - S is a (finite) set of states
    - P is a state transition probability matrix for all states
        - $P_{ss'} = P[S_{t+1} = s' | S_t = s]$

Sample episode (Sequence)

- Sample episode (or sequence) is the series of states followed in one episode/rotation where it starts and ends at fixed states. Essentially a set of random states sampled from the chain following its rules
- It has to always terminate.
- If the probabilty of the state transitioning should be changing depending on other factors (say for instance how long it has been at one state), then we represent each iteration of staying there as a series of states with varying probability for each of them. That ensures the MDP structure holds

Markov Reward Process

- Markov process with value judgments (based on rewards)
- Formally,

- Markov Reward Process is a tuple $< S, P, R, \gamma >$
- S is a (finite) set of states
- P is a state transition probability matrix for all states
    - $P_{ss'} = P[S_{t+1} = s' | S_t = s]$
- R gives the immediate reward from that state to the next
    - $R_s = E[R_{t+1} | S_t = s]$
- γ is the discount factor, $\gamma \epsilon [0, 1]$

Return

- Return is the cumulative reward for the sequence/sample.
- Return G_t is total discounted reward from time-step t.

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

- The discount factor $\gamma \epsilon [0, 1]$ shows how much we care about the future rewards.
    - 0 means no consideration for future, greedy (myopic)
    - 1 means max consideration for future (far-sighted)
- The value for recieving reward R after $k + 1$ steps is $\gamma^k R$ at current step.

Why do we need a discount factor?

- Uncertainity in the model, lack of accuracy. Essentially a fail safe against the model itself.
- Mathematical convenient to discount reward.
- Avoids infinite returns in a cyclic Markov processes
- Financially, immediate rewards may be more profitable
- Cognitive models show preference for immediate reward.
- Still, sometimes we can use undiscounted Markov reward processes $(\gamma = 1)$. Eg: If all sequences terminate (won't run into an infinite loop).

Value Function

- It gives the long-term value of that state s, i.e how profitable we can end up from that state. Or how good it is to be in that state.
- Formally, state value function $v(s)$ of an MRP is the expected return starting from state s.

$$v(s) = E[G_t | S_t = s]$$

- If we sample a few sequences starting from a point, the return averaged across those sequences would give us a good estimate of the value function of that state.

Bellman Equation for MRPs

- The bellman equation decomposes the value function into two parts
  - The immediate reward R_{t+1}
  - discounted value of successor state $\gamma v(S_{t+1})$

It achieves this by representing the trailing parts of the return function

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

as

$$G_t = R_{t+1} + \gamma G_{t+1}$$

Plugging that in the value function equation,

$$v(s) = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$$

> Note: R_{t+1} referes to the reward obtained on exiting state S_t. Convention dictates the t+1 part of the reward - This confusion arises in MRPs usually

Matrix form

- Bellman equation can be represented in matrix form.

  $$v = R + \gamma P v$$

  where v is a vector with one entry per state
- Each R_i corresponds to reward obtained from leaving state S_i.

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \dots & P_{1n} \\ \vdots & & \\ P_{n1} & \dots & P_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix}$$

- Bellman equation is linear, and so can be solved directly

$$v = R + \gamma P v (1 - \gamma P) v = R v = (1 - \gamma P)^{-1} R$$

- Complexity for this process is $O(n^3)$ for n states.

- So only possible for small MRPs.
- For larger MDPs, there are many iterative methods:
  - Dynamic Programming
  - Monte-Carlo
  - Temporal Difference learning

Markov Decision Process

- Is a markov reward process with decisions involved (The A in the equation).
- An environment in which all states are Markov.
- Markov Decision Process is a tuple $< S, A, P, R, \gamma >$
  - S is a (finite) set of states
  - A is a finite set of actions
  - P is a state transition probability matrix for all states
    - $P_{ss'}^{a} = P[S_{t+1} = s' | S_t = s, A_t = a]$
  - R gives the immediate reward from that state to the next
    - $R_s^a = E[R_{t+1} | S_t = s, A_t = a]$
  - γ is the discount factor, $\gamma \epsilon [0, 1]$

Policy

- Is the mapping of the probability of all actions agent can take in each state. We decide the probabilities there.
- Formally, a policy π is a distribution over actions given states

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

- Policy defines the behaviour of an agent.
- MDP policies are stationary (time-independent). Only depend on state.
- i.e the probability of an action wont change over time, just over state.

Value function

- Tells us how good it is to be in this state.
- The state value function $v_\pi(s)$ of an MDP is expected return starting from state s, while following police π

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

Action Value function

- Tells us how good it is to take a particular action from this state.
- The action-value function $q_\pi(s,a)$ is the expected return starting from state s , taking action a, and then following policy π.

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a]$$

Bellman Expectation Equation

- The state value function can be decomposed similarly to MRPs.

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

- And action-value function too

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

Accordingly, going by the defination for both - we can relate policy, action-value and state value functions if we're in a state taking deciding on the best action by the following:

$$v_\pi(s,a) = \sum_{a \epsilon A} \pi(a|s) q_\pi(s,a)$$

- Intuitively, value of a state = probability of taking an action * max potential reward from that action
- Let's say the situation is flipped, we've taken an action and can end up in some state. How good is that state?
  - To evaluate that, we intuitively can reckon it depends on reward in going to that state + potential future rewards. Equation form

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_\pi(s')$$

We can combine both these equations to get $v_\pi(s)$ in terms of $v_\pi(s')$ in a recursive relationship. Which we can exploit to solve it.

This is called the bellman equation

$$v_\pi(s,a) = \sum_{a \epsilon A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_\pi(s') \right)$$

- We can do the same for q_π too.

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a \sum_{a \epsilon A} \pi(a|s) q_\pi(s,a)$$

- Bellman equation can be expressed in matrix form as well.

$$v_\pi = R^\pi + \gamma P^\pi v_\pi$$

And thus can be solved as,

$$v_\pi = (I - \gamma P^\pi)^{-1} R^\pi$$

Optimal Value Function

- We would want to find the optimal way to solve problems.
- Optimal state-value function $v_*(s)$ is the maximum value function over all policies.
- It doesn't tell the best policy, but the maximum reward extractable from the system.

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

- Optimal action-value function $q_*(s,a)$ is the maximum action-value function over all policies.

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

- So given you start with a particular state s, and take an action a - this would give the max reward you can potentially get after that.
- Essentially if we get q_* for all state action pairs, we've solved the problem.

Optimal Policy

- To compare policies, we have to define partial ordering over policies.
  $\pi \geq \pi\prime$ if $v_\pi(s) \leq v_{\pi\prime}(s), \forall s$

> Theorem
> For any MDP,
>
> - There exists an optimal policy π* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$

> - All optimal policies achieve the optimal value function, i.e $v_{\pi*}(s) = v_*(s)$
> - All optimal policies achieve the optimal action-value function, $q_{\pi*}(s,a) = q_*(s,a)$

- An optimal policy is one which maximizes $q_*(s,a)$ over all (s,a)

$\pi_*(a|s) = \{1$ if $a = argmax_{a \epsilon A} q_*(s,a)$, 0 otherwise}

- There's always a deterministic optimal policy for any MDP
- And all we have to find is $q_*(s,a)$.

Bellman Optimality Equation

- In a particular state, instead of averaging the possible rewards by future actions, we pick the action that gives higher possible reward (the one with greater q_*)
- Thus, the optimal value function of a state $v_*(s) = max_a q_*(s,a)$
- Flipping it for the particular action, we can’t choose the state now. We're at the mercy of the environment sending us to some state.
- Hence, we have to average over the states here to find how good our action is.

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_*(s')$$

- We put these two equations together, to get a recurrance relation for v_* that we can solve.

$$v_*(s) = max_a \left(R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a v_*(s') \right)$$

- Similarly, we can do the same for q_* to get an equation just in terms of it.

$$q_*(s,a) = R_s^a + \gamma \sum_{s' \epsilon S} P_{ss'}^a max_a q_*(s,a)$$

Solving Bellman's optimality equation

- Equation is non-linear
- No closed form solution (in general)
- Many iterative solution methods

- Value iteration
- Policy iteration
- Q-learning
- Sarsa

Problems

Problem 1: Return function (taken from Barton and Sutto, Chapter 3)

Suppose $\gamma = 0.5$ and the following sequence of rewards is received $R_1 = -1, R_2 = 2, R_3 = 6, R_4 = 3$ and $R_5 = 2$, with $T = 5$. What are $G_0, G_1, ..., G_5$?

Ans:
Return function $G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Or alternatively, $G_t = R_{t+1} + \gamma G_{t+1}$

Now calculating from behind we can see G_5 would have to be set as 0, as no rewards are specified for R_6 (and can't be specified either)
So,

$$\begin{gathered}G_4 = R_5 + 0.5 * 0 = 2 \\ G_3 = R_4 + 0.5 * G_4 = 3 + 1 = 4 \\ G_2 = R_3 + 0.5 * G_3 = 6 + 2 = 8 \\ G_1 = R_2 + 0.5 * G_2 = 2 + 4 = 6 \\ G_0 = R_1 + 0.5 * G_1 = -1 + 3 = 2\end{gathered}$$

Problem 2: Return function, infinite inputs (same as above)

Supposed $\gamma = 0.9$ and the reward sequence is $R_1 = 2$ followed by an infinite sequence of 7s. What are G_1 and G_0 ?

Ans:
As shown above,

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

Going backwards again,

$$G_1 = G_2 + \gamma G_3 + \gamma^2 G_4 + ...$$
$$G_1 = 7 + 7 * \gamma + 7 * \gamma^2 + ...$$
$$G_1 = 7(1 + \gamma + \gamma^2 + ...)$$

Sum of infinite $GP = a/(1 - r)$. Here Sum = $1/(1 - 0.9) = 10$

$$G_1 = 7 * 10 = 70$$
$$G_0 = R_1 + \gamma G_1 = 2 + 0.9 * 70 = 65$$