EE675A - IITK, 2022-23-II

Lecture 11: Bellman optimality equation and Backup Diagrams

15th february 2023

Lecturer: Subrahmanya Swamy Peruru *Scribe: Shrinivas Sontakke*

1 Recap

1.1 Recap from last lecture

The last lecture focused on Vector form of Bellman expectaion equation (V_π) which is:

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

where,
Immediate Reward, $R^\pi = E_\pi[R_t + 1|S_t = s]$
P^π = Transition Probability matrix corresponding to policy π

We have also define optimal policy $V^*(s)$. If we can able to find policy π^* such that

$$V_{\pi^*} \geq V_\pi \, , \forall \pi \neq \pi^*$$

and,

$$V^*(s) = \max_\pi V_\pi(s)$$

When we say optimal policy π exist then there must exist policy π^* such that $V_{\pi^*}(s) = V^*(s)$
Similarly,

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

2 Bellman Optimality Equation

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 the action with the max value.

Suppose our agent is in state S and from that state it can take two actions (a). So, we look at the action-values for each of the actions and unlike, Bellman Expectation Equation, instead of taking the average our agent takes the action with greater q* value. This gives us the value of being in the state S. so,

$$V^*(S) = \max_a q^*(s, a)$$

Suppose, our agent has taken an action a in some state s. Now, its on the environment that it might blow us to any of these states (s'). We still take the average of the values of both the states, but the only difference is in Bellman Optimality Equation we know the optimal values of each of the states.Unlike in Bellman Expectation Equation we just knew the value of the states.
Mathematically, we can expressed:

$$q_{\pi_*}(s, a) = R_s^a + \nu \sum_s^{'} V^*(s')P_{ss'}^a$$

In order to find the value of state S we simply average the Optimal values of the States(s). This gives us the value of being in state S.Using the above equations we can express:

$$V^*(s) = \max_a [R_s^a + \nu \sum_s^{'} V^*(s')P_{ss'}^a]$$

This is Called Bellman Optimality Equation and this is not linear equation. Now lets try to find optimal policy from given equations. We know the deterministic policy is given as :

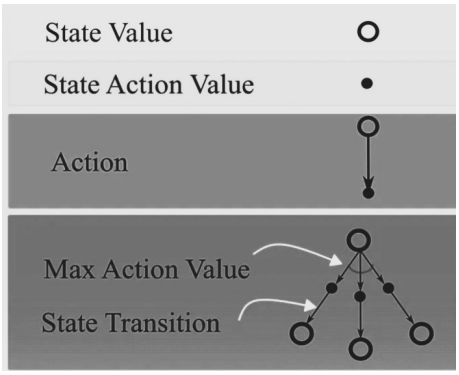
$$\pi(s) = \arg\max_a q^*(s, a)$$

We Know V*(s), so optimal policy can be written as:

$$\pi^*(s) = \arg\max_a (R_s^a + \nu \sum_s^{'} V^*(s')P_{ss'}^a)$$

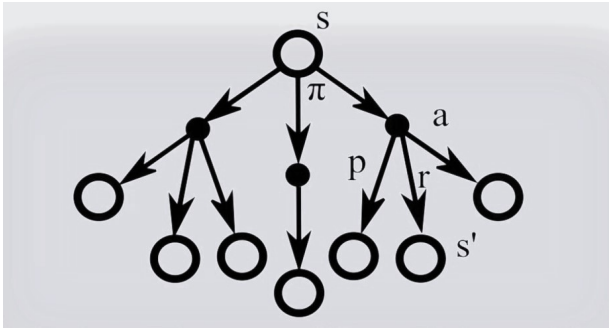
3 Backup Diagrams

In a backup diagram state value is represented by a hollow circle while a state-action value or action value is represented by a solid circle. Action is represented by a arrow starting from a state. Reward is conventionally shown after the action value. Action that result in maximum action value is shown as a arc starting from a state. Refer to below diagrams for standard representation of state value, action value, action, maximum action value, state transition

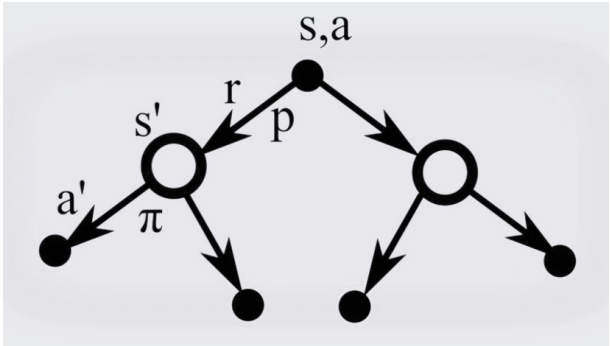


3.1 State Value function under a stochastic policy

s is the starting state and it is the root node from state s there can be three action as shown by arrow and agent take action as per policy state action value or action is shown by solid circle (it is the convention that action is taken from a state and reward is received after the action is taken. Once action is taken the agent can end up in different state with certain state transition probability. after taking right most action agent land into three possible states

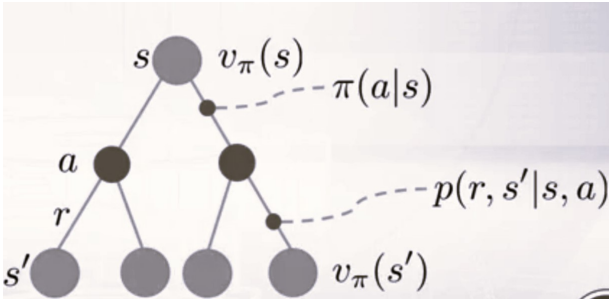


3.2 State-Action Value function under a stochastic policy



Similar to State value function we can create backup diagram for action value function or state-action value function. In this case root node is solid circle as its a specific action from a specific state.

3.3 Backup Diagram for bellman Expectation Equation



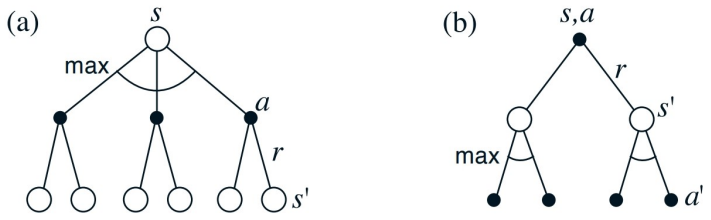


Figure 1: Backup diagrams for (a)$v_* and (b) q_*$

4 Dynamic Programming

Dynamic Programming is a mathematical optimization approach typically used to improvise recursive algorithms. It basically involves simplifying a large problem into smaller sub-problems. There are two properties that a problem must exhibit to be solved using dynamic programming :

- **Optimal Substructure** : If we have V*(s') then we can find V*(s)
- **Overlapping subproblems** : To calculate value function of state S we have to calculate value function for state $s_1, s_2, s_3, ...$ so dynamic programming provide structured way of solving overlapping subproblems instead of solving it again and again.

There are two algorithms used to solve V* :

- **Policy iteration**
- **Value iteration**

References

1. Chapter 2 and 3 of **Reinforcement Learning: An Introduction** Second edition Richard S. Sutton and Andrew G. Barto 2.Internet