Chapter 4 - Dynamic Programming

Atul Balaji - EE16B002

February 20, 2019

General dynamic programming methods often require a perfect model of the environment and a large amount of computation. However, in Reinforcement Learning, the DP methods used do not rely on this assumption and do not require a very high degree of computation,

1 Policy Evaluation

- Policy evaluation refers to the method of computing the value function v_π for some policy π, given the MDP. It is implemented in an iterative manner, following from the **Bellman Equation**.
- We start with an arbitrary v_0 and then successive approximation is done to obtain v_{k+1} in terms of v_k.

$$v_{k+1}(s) = E_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')]$$

2 Policy Improvement

- Having determined the value function according to a policy π, we want to know whether it is possible to improve the policy. Here, action-value function q_π can be used to determine whether changes need to be made to the original policy π.
- For two policies π and π', we can say that π' is a better policy than π if $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ for all states s. π' is the improved policy which has been obtained by making π greedy with respect to v_π.

$$\pi'(s) = \underset{a}{argmax}\ q_\pi(s, a) = \underset{a}{argmax} \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{1}$$

  π' is better than π is all cases except when π is already optimal, in which cases both are identical.

3 Policy Iteration

- After finding π', we can once again find the new value function using policy evaluation and again find an improved policy π''. This process can be done successively to keep improving the policies.
- Since we are dealing with finite MDPs, using this method it is guaranteed to converge to the optimal policy and value function.

4 Value Iteration

- In policy iteration, the policy evaluation and improvement steps are repeated one after the other till convergence.
- By contrast, in value iteration, the optimal value function is first calculated and then the best **policy extraction** happens.
- The iterative algorithm can be derived as:

$$v_{k+1}(s) = \underset{a}{max} \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{2}$$

- This is a small modification of the Bellman Equation where we use the **max** function additionally. This algorithm also converges to the optimal value function in the case of a finite MDP. It can be stopped when difference between successive estimates are sufficiently small.

- After convergence, the **optimal policy** can be estimated using:

$$\pi(s) = \underset{a}{argmax} \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{3}$$

5 Asynchronous DP

- The drawback of the above algorithms is that they require a complete sweep of the state space. This is especially true when the MDP has a very large number of states. This makes a sweep of the entire state space unfeasible.
- In asynchronous DP algorithms, we update the values of only select states in an iteration. We are given liberty to decide which states to skip and to focus the updates to select states. This kind of approximation leads to faster convergence.

6 Generalized Policy Iteration

- The **policy iteration** algorithm is quite generic. The value function stabilizes only when it is consistent with the current policy and the policy stabilizes only when it is greedy with respect to the current value function.
- Improvement of policy is done by being greedy with respect to the current estimate of the value function V. The two processes compete and co-operate until optimality is reached.

7 Efficiency of DP Algorithms

- The Dynamic programming algorithms described find the optimal policy in time which is polynomial in n and k.
- It also outperforms other methods such as **direct search** and **linear programming**, which are not feasible with large number of states.