Dynamic Programming for Planning Problems

Why Dynamic Programming?

- **Dynamic:** Sequential/Temporal component to the problem
- **Programming:** optimizing a policy
- Use DP where we can break up a complex problem into sub-problems and solve these and put them back together to get the final solution.

Requirements for Dynamic Programming

- **Optimal Substructure:** Ability to break problem into sub problems, solve them in an optimal way. The overall solution can be derived from an optimal solution to those pieces.
- **Overlapping Subproblems:** Subproblems recur several times and hence we can cache and reuse the results across the overall problem.
- **MDP satisfies both these properties:** Bellman equation gives the recursive decomposition of the value function into immediate reward and future value. The value function stores and reuses the values of future states.

Planning by Dynamic Programming

- DP assumes full knowledge of the MDP
- It is used for planning the MDP in 2 ways
    - **Prediction Problem:** Figuring out how much reward an agent is going to receive if it behaves in a certain way in a certain environment
        - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π i.e. a MRP $\langle \mathcal{S}, \mathcal{P}^{\pi}, \mathcal{R}^{\pi}, \gamma \rangle$
        - Output: value function v_{π}
    - **Control Problem:** Figuring out what is the best possible policy or behaviour for an agent in a certain environment.
        - Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
        - Output: optimal value function v_* and optimal policy π_*

Policy Evaluation

- Problem: evaluate a given policy π for a given MDP
- Solution: Use Bellman expectation equation and apply it iteratively
- We iterate through multiple value functions: v1 -> v2 -> ... -> v_{π}, updating the values of state pairs at each step to eventually reach v_{π}. We start off with a random initial value.
- To do this, we use synchronous backups(update the values of all states at each step) using Bellman Expectation Equation
- $v_{k+1}(s_t) = \mathbb{E}[R_{k+1} + \gamma v_k(S_{t+1})]$ from k = 0 to k -> ∞ (until convergence).
- From each state(k), determine the sum of reward received by performing an action and the

value of the successor state according to the previous policy ($R_{k+1} + v_k(S_{t+1})$). Do this for all possible actions and find the average. This is the new value of the state(k+1).

Policy Iteration

- Problem: Improve the existing policy π for a given MDP
- Solution: First we evaluate the policy by **Policy Evaluation** to get the value for each state. Then we act greedily with respect to the value function, i.e. $\pi' = greedy(v_\pi)$[**Policy Improvement**]. Eventually, we should get to $\pi' = \pi^*$
- Consider a deterministic policy, π.
- We can improve this policy by acting greedily $\pi'(s) = \underset{a \in A}{argmax}\ q_\pi(s, a)$.
- If we follow the greedy policy π' for one step and thereafter follow the usual policy π, we should get as good or better value than just following policy π.

$$q_\pi(s, \pi'(s)) = \underset{a \in A}{max}\ q_\pi(s, a) \geq q_\pi(s, \pi(s)) = v_\pi(s)$$

- Hence, if we use greedy policy for all steps is better than using the previous policy.

Value Iteration

- Problem: Find the optimal policy
- Solution: Iterative application of Bellman Optimality Function
- $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_*$
- At each step, we update the value function for states until convergence to v_*
- It is as if we are improving value function, doing policy improvement and acting greedily with respect to the new policy and then repeating. There is no explicit policy generated and the value functions do not necessarily correspond to any policy.
- $v_{k+1}(s) = \underset{a \in A}{max}\ (R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_k(s'))$
- This combines both the policy evaluation and policy iteration in one step.