4. Dynamic Programming

Notes

- Dynamic programming (DP) refers to a collection of algorithm that can be used to computed optimal policies given a perfect model of the environment as a MDP. They assume a perfect environment and requires high computational costs, thus are not practical. However, it provides a solid foundation for the solving RL problems.
- Key idea of DP is the use of value functions to organise and structure the search for good policies.
- Policy Evaluation (Prediction Problem):
    - In an environment with known dynamics, the system will have $|\mathcal{S}|$ simultaneous linear equations with $|\mathcal{S}|$ unknowns. Solving this analytically is tedious and expensive computation. We can use an *iterative policy evaluation:*

      $$v_{k+1}(s) \doteq \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

      where v_0 is chosen arbitrarily, and the value function v_k is updated at each iteration. The sequence $\{v_k\}$ converges to v_π as $k \to \infty$.
    - The process of successive approximation of v_{k+1} from v_k by using the old values of all the successor states of s and their expected immediate rewards is known as *expected update*. All updates done in DP are called expected updates, and there are many different kinds. It is essentially updates based on the expectation over all possible next states rather than on a sample next state.
    - Implementation: Have a single array of $v_k(s)$, and update the values in-place. Here, the new values of other states can be used to compute the new value of a state, and this is generally faster than computing the new value of a state from the old value of other states from a step ago. The process of iteration is halted once the difference between successive values is small enough.
- Policy Improvement:
    - For a deterministic policy $\pi(s)$, we want to know whether if we should change the policy, such that $a \neq \pi(s)$. The *policy improvement theorem* states that for a pair of deterministic policies π and π', if $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, then the policy π' must be as good as, if not better than π. This also means that $v_{\pi'}(s) \geq v_\pi(s)$.

- Considering changes at all states, we can get a new greedy policy π' given by

  $$\pi'(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

  Suppose that the new greedy policy π' is as good as, but not better than, the old policy π. By the Bellman optimality equation, both π and π' must be optimal policies.

- For a stochastic policy, if there are multiple actions that are optimal, then they are given a portion of the probability, as long as all sub-maximal actions are given zero probability.

- Policy Iteration:
  - We can iterate through the process of finding an optimal policy, where the policy undergoes an evaluation, followed by an improvement, and this repeats until an optimal policy is reached. This is called *policy iteration*. For example, a policy π is improved using $v_\pi$to give a better policy π', and we can then compute $v_{\pi'}$ to give an even better π''. This ensures a sequence of monotonically improving policies and value functions. (page 80)
  - Given that a finite MDP has a finite number of policies, policy iteration must converge to an optimal policy and optimal value function in a finite number of iterations.

- Value Iteration:
  - Policy evaluation involves interactively updating the state value to converge to the optimal state value function v_*. It is possible to stop short and truncate policy evaluation, since there is likely no change in the greedy policy in the later iterations of policy evaluation.
  - *Value iteration* is when policy evaluation is stopped after just one update/sweep, and combines policy improvement in it. This also converges to the optimal state value function, and is show below:

    $$v_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')]$$

    Value iteration combines one sweep of policy evaluation and one sweep of policy improvement in a single sweep. Faster convergence can actually be achieve by having more sweep of policy evaluation between each policy improvement sweep. (page 83)

- Asynchronous DP:
    - A major drawback to these DP methods is that they involve the entire state set, and if the state set becomes very large, it can be very computational expensive.
    - Asynchronous DP updates state values in any order, using states that happen to be available to them. To converge correctly, the algorithm cannot ignore any state after some point in its computation. However, this still gives flexibility to select states to update
    - For example, we can try to order the updates of states such that it improves the rate of progress of the algorithm, or we can choose to skip states that are not relevant to optimal behaviour.
    - Asynchronous DP can also combine computation and real-time interaction, where the iterative DP algorithm runs at the same time as an agent experiencing its MDP environment. The agent's experience can be used to determine the states for the DP algorithm to update, whilst the value and policy information from the DP algorithm can be used to guide the agent's decision making. This makes it possible to focus the DP algorithm onto relevant parts of the state set.
- Generalised Policy Iteration (GPI):
    - This refers to the general idea of letting policy-evaluation and policy-improvement processes interact, and independent of the granularity and other details of the two processes.
    - Almost all RL problems are well described as GPI, where there are identifiable value functions and policies, and both components are always being improved with respect to the other. The stabilisation of both evaluation and improvement process means that there is no longer change in the value function and policy, thus optimal. These two processes can be viewed to be both competing and cooperating.
- DP methods can be considered to be quite efficient, where it runs in polynomial time to the number of states and action. It is much better than any direct search methods, and perform better than linear programming methods for large state spaces.
- One can argue that DP methods has the curse of dimensionality, but this is a problem faced by RL, rather than DP methods. Asynchronous DP can also be used in large state sets to improve efficiency.
- *Bootstrapping* is the idea of using estimates of successor state values to update the estimate of the current state value.

Exercises

4.1 If π is the equiprobable random policy, $q_\pi(11, \texttt{down}) = -1$ and $q_\pi(7, \texttt{down}) = 1[-1 + v_\pi(11)] = -1 - 14 = -15$, assuming $\gamma = 1$.

4.2 Given a new state 15, assuming $\gamma = 1$:

$$\begin{aligned} v_\pi(15) &= 0.25[(-1 + v_\pi(12)) + (-1 + v_\pi(13)) + (-1 + v_\pi(14)) + (-1 + v_\pi(15))] \\ &= 0.25[-23 - 21 - 15 - 1 + v_\pi(15)] \\ v_\pi(15) &= -20 \end{aligned}$$

Updating it once with the *iterative policy evaluation* does not change the value of either $v_\pi(13)$ or $v_\pi(15)$, where they remain at -20.

4.3 For the action-value function $q_\pi(s, a)$:

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s,a)[r + \gamma v_\pi(s')] \\ &= \sum_{s',r} p(s', r|s,a)\left[r + \gamma \sum_{a'} \pi(a|s) q_\pi(s', a')\right] \\ q_{k+1}(s,a) &= \sum_{s',r} p(s', r|s,a)\left[r + \gamma \sum_{a'} \pi(a|s) q_k(s', a')\right] \end{aligned}$$

4.4 The two or more policies that are equally good will have different actions for each state. This means that we need to keep track of policies which give the same state-value function, and store such information if they do. This ensures that if an equally good policy is encountered, the $policy\_stable$ variable will remain $true$.

An alternative method is if after the first sweep of policy evaluation in each policy iteration, if $\Delta < \theta$, then the policy can be considered to be stable, and the policy improvement part can be skipped, and the current policy be deemed the final optimal policy.

4.5

```
1. Initialisation
   $Q(s,a) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop :
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$ :
         Loop for each $a \in \mathcal{A}(s)$ :
            $q \leftarrow Q(s,a)$
            $Q(s,a) \leftarrow \sum_{s',r} p(s',r|s,a)[r + \gamma \sum_{a'} \pi(a'|s')Q(s',a')]$
            $\Delta \leftarrow \max(\Delta, |v - Q(s,a)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of the estimation

3. Policy Improvement
   $\textit{policy-stable} \leftarrow \textit{true}$
   For each $s \in \mathcal{S}$ :
      $\textit{old-action} \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a Q(s,a)$
      If $\textit{old-action} \neq \pi(s)$, then $\textit{policy-stable} \leftarrow \textit{false}$
      Else $\textit{policy-stable} \leftarrow \textit{true}$
   If $\textit{policy-stable}$, then stop and return $V \approx v_*$ and $\pi \approx \pi_8$; else go to 2
```

4.6 Restricted to considering only policies that are ϵ-soft, meaning that the probability of selecting each action in each state, s, is at least $\epsilon/|\mathcal{A}(s)|$. This means that there are no actions that have zero probability, and this is a way to introduce exploration into the policy.

In Step 3: The $\pi(s) \leftarrow \arg\max_a Q(s,a)$ needs to be replaced with an ϵ-greedy selection rule. This means that instead of picking the maximum action-value function, there is an ϵ probability that a random action-value function is selected.

In Step 2: Need to include $\pi(a|s)$ in the Bellman equation, such that $V_(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$

In Step 1: $\pi(s)$ should be defined to be an ϵ-soft method, and a value for ϵ should be initialised.

4.7 With the modified reward system, it is largely favourable for a car to be moved from the first to second location, than to not move any cars, since there is no penalty for this. In addition,

4.8 With a $p_h = 0.4 < 0.5$, and given that the gambler wins when the coin is a heads, this means that the gambler will lose eventually if the bet size is constant. Thus, it seems that the optimal policy given is trying to win the fewest number of flips, since the law of large numbers would mean that the gambler will eventually lose money. (I think...)

4.9 At $p_h = 0.25$, the state values are even more curved (looks more quadratic) than the curve for $p_h = 0.40$, whilst at $p_h = 0.55$, the curve skews towards the top left corner. The optimal policy for $p_h = 0.25$ looks very similar to the curve for $p_h = 0.40$, whilst the optimal policy for $p_h = 0.55$ is to bet 1 at every state.

4.10 For the action-value function $q_{k+1}(s, a)$:

$$q_{k+1}(s,a) = \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} \pi(a'|s') q_k(s',a')\Big]$$