Dynamic Programming

Dynamic Programming (DP)

- A collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a finite MPD
- Classic DP isn't useful for many Reinforcement Learning because of computational expense - but it is the foundation for many other approximation methods
- Key Idea of DP - Use value functions to organize/structure the search for good policies

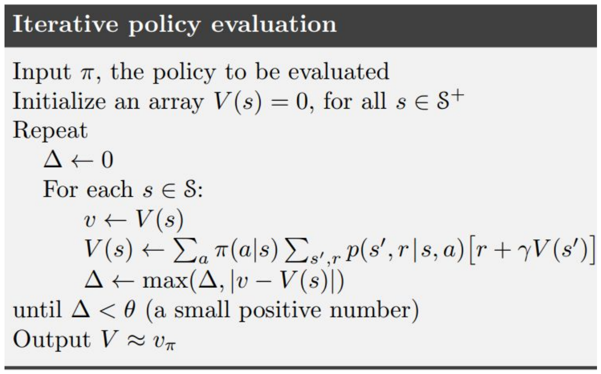
Policy Evaluation (Prediction)

Bellman Equation is used to compute the state-value function for all **s** in policy □

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

Iterative Policy Evaluation

- Pseudo Code
- Two Arrays Method
    - One for the old values **v_k(s)**, and one for the new **v_{k+1}(s)**
- Sweeping Method (Standard DP Method)
    - Values are updated immediately
    - Converges faster than two arrays method
    - If **v(s')** is known, then to compute **v(s)** only one step-lookahead is needed



```
Iterative policy evaluation

Input $\pi$, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$
```

Policy Improvement

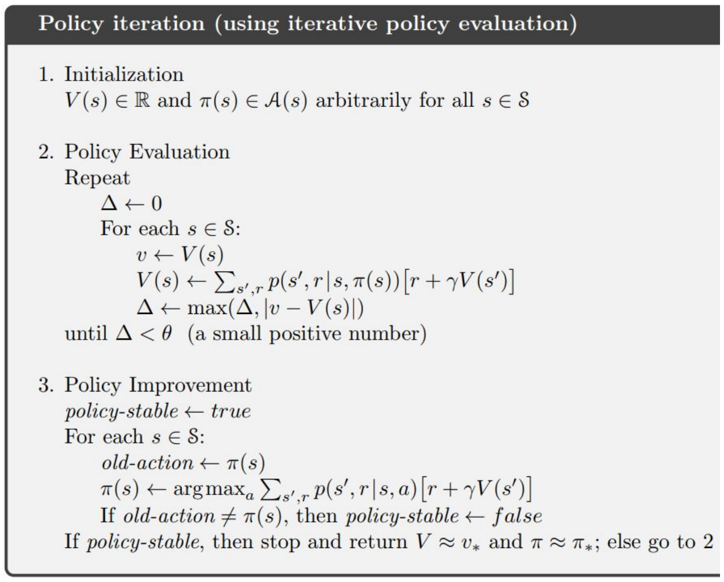
- If the value function for a policy is known:
    - For some state **s** we would like to know if the policy should be changed to choose a different action
    - If the action results in a higher discounted return, than the policy is changed
    - New policy is the exact same, except for when in state **s**
- The new policy, □**'** ≥ □, is greedy
- If □**'** is as good as, but not better than, the old policy □. Then **v**$_{□}$ = **v**$_{□}$,
    - Bellman Equation shows that both □**'** and □ are optimal policies

Policy Iteration

- Method of finding the optimal policy
- Drawback - requires multiple sweeps through the state set
- Perform policy evaluation, then policy improvement
    - Once a policy, □, has been improved using $\mathbf{v}_{□}$ to yield a better policy, □**'**, then $\mathbf{v}_{□'}$ can be computed, and be used to improve policy again and yield □**''**

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*,$$

where $\xrightarrow{\mathrm{E}}$ denotes a policy *evaluation* and $\xrightarrow{\mathrm{I}}$ denotes a policy *improvement*



Policy iteration (using iterative policy evaluation)

```
1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
      $old\text{-}action \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
      If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2
```

Value Iteration

- Policy iteration that is stopped after just one sweep (one update of each state)
    - Doesn't give $\mathbf{v}_{□}$, but an approximation that only looks ahead one step
    - Faster convergence is often achieved by looking ahead by more than one step
- Combines one sweep of policy evaluation and one sweep of policy improvement

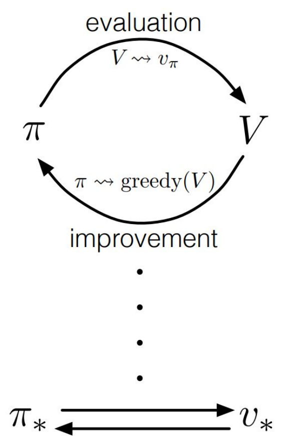
Asynchronous Dynamic Programming

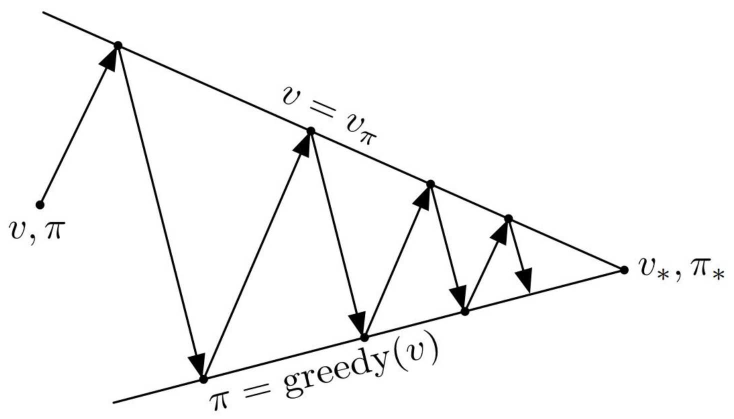
- Methods described above require sweeps of the entire state set
    - If state set is large, than even a single sweep can be prohibitively expensive
- Types of async DP
    - In-place DP
        - Store only one copy of value function
    - Prioritized Sweeping
        - Maintain a priority queue for state-value updates
        - Use magnitude of Bellman error to guide state selection
    - Real-time DP
        - Use agent's experience to guide the selection of states

Generalized Policy Iteration (GPI)

- GPI is the idea of letting policy evaluation and policy improvement interact independent of the granularity and other details of the two processes

The interaction of policy evaluation and policy improvement


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \text{greedy}(V)$
improvement
π_*
v_*



- The evaluation and improvement processes in GPI can be viewed as both competing and cooperating. They compete in the sense that they pull in opposing directions.
    - Policy Evaluation: making the policy greedy with respect to the value function typically makes the value function incorrect for the changed policy
    - Policy Improvement: Making the value function consistent with the policy typically causes that policy no longer to be greedy.
    - In the long run, however, these two processes interact to find a single joint solution: the optimal value function and an optimal policy.

Efficiency of Dynamic Programming

- DP is not practical for very large problems ($S > 10^7$), but compared to other methods for solving MDPs DP methods are very efficient
- In the worst case, the time DP methods take to find an optimal policy is polynomial in terms of the number of states and actions
- A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is k^n
    - n and k denote the number of states and actions,
- DP is exponentially faster than any direct search in policy space could be, because direct search would have to exhaustively examine each policy to provide the same guarantee.