Lecture - 3

Dynamic Programming

In RL, DP refers to a collection of algorithms used to compute optimal policies given a perfect model of the environment as an MDP.

Dynamic Programming

- Sequential / temporal component to the problem
- Optimizing a "program"; i.e. a policy

→ DP solves a problem by breaking them into subproblems

A DP problem has two components

1. Optimal substructure
   - Principle of optimality applies
   - Optimal soln can be decomposed into subproblems
2. Overlapping subproblems
   - Subproblems recur many times
   - Their soln can be cached & reused

→ MDP satisfies both these properties

1. Bellman eqn gives recursive decomposition
2. Value fun stores & reuses soln's

Planning by DP

- DP assumes full knowledge of the MDP
- It is used for planning in an MDP

for prediction

Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ and policy π (and action a)

Output: Value fun^n V_π (q_π and) → Policy Evaluation

for control

Input: MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

output: optimal value fun^n V_*, q_* → ① Policy iteration ② Value iteration

optimal policy π_*

Policy Evaluation

Given an arbitrary policy π, compute the (state) value fun^n of π V_π.

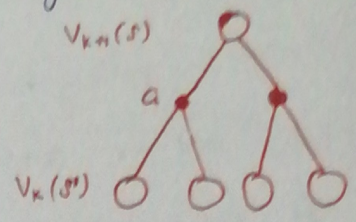
from Bellman expectation eqn:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}(s,a,s') + \gamma V_\pi(s') \right)$$

where $\mathcal{R}(s,a,s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s']$

→ If the environment's dynamics are known, above eqn is a system of $|\mathcal{S}|$ simultaneous eqns in $|\mathcal{S}|$ unknowns $(V_\pi(s), s \in \mathcal{S})$

→ Iterative application of above Bellman eqn gives an iterative method to compute $V_\pi(s)$.



$$V_1 \longrightarrow V_2 \longrightarrow \cdots \longrightarrow V_\pi(s)$$

$$V_{k+1}(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma V_k(S_{t+1}) \mid S_t = s\right]$$

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s'} P^a_{ss'} \left(r(s,a,s') + \gamma V_k(s')\right)$$

↔ The above sequence can be shown, in general, to converge to V_π as $k \to \infty$. This algo. is called iterative policy evaluation.

Iterative policy evaluation, for estimating $V \approx v_\pi$

① Input π (policy) and θ (algorithm threshold)

② Initialize $V(s)$, $\forall s \in \mathcal{S}$ arbitrarily except that $V(\text{terminal}) = 0$

③ Loop

  ⓐ $\Delta \leftarrow 0$

  ⓑ Loop for each $s \in \mathcal{S}$

    (i) $v \leftarrow V(s)$

    (ii) $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P^a_{ss'} \left(r(s,a,s') + \gamma V(s')\right)$

    (iii) $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

Policy Improvement

Once a policy has been evaluated, its value fun^n can be used to construct another better policy.

A new policy π' can be constructed by acting greedily w.r.t. value fun^n of previous policy.

$$\pi' = \text{greedy}(V_\pi)$$

This results in a deterministic policy π', which improves over policy π.

i.e. $$\pi'(s) = \underset{a \in A}{\operatorname{argmax}}\ q_\pi(s,a) = \sum_{s'} P_{ss'}^{a} \left(\gamma(s,a,s') + \gamma V_\pi(s') \right)$$

* This improves the value from any state s over one step

$$q_\pi(s, \pi'(s)) = \max_{a \in A} q_\pi(s,a) \geq q_\pi(s, \pi(s)) = V_\pi(s)$$

* We can also show that new ~~poly~~ policy π' is better than policy π overall. i.e. $V_{\pi'}(s) \geq V_\pi(s)$.

$$\begin{aligned}
V_\pi(s) \leq q_\pi(s, \pi'(s)) &= \mathbb{E}\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s) \right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma V_\pi(S_{t+1}) \mid S_t = s \right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s \right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) \mid S_t = s \right] \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \cdots \mid S_t = s \right] \\
&= V_{\pi'}(s)
\end{aligned}$$

$$\boxed{V_\pi(s) \leq V_{\pi'}(s)}$$

This is true for all states $s \in \mathcal{S}$. So, the new policy π' improves over original policy π.

When the improvement stops, i.e. when for two successive iterations ~~$V(\pi') = V(\pi)$, then~~,

$$V_{\pi'}(s) = \max_a \mathbb{E}\left[R_{t+1} + \gamma V_{\pi'}(S_{t+1}) \mid S_t = s, A_t = a\right]$$

$$= \max_a \sum_{s'} P^a_{ss'} \left(\gamma(s,a,s') + \gamma V_{\pi'}(s')\right)$$

The Bellman optimality eqn has been satisfied, and we can conclude that $\boxed{\pi' = \pi_*}$

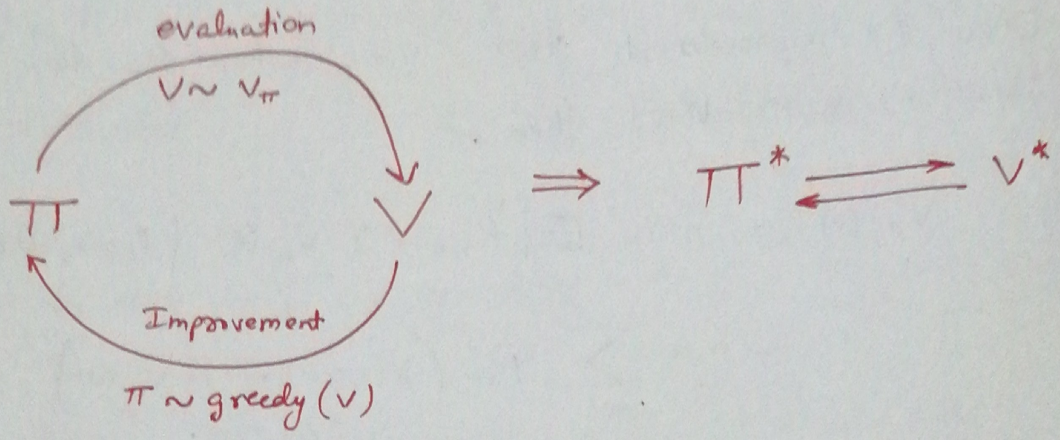
i.e. π' must be an optimal policy, as well as π.

Policy Iteration

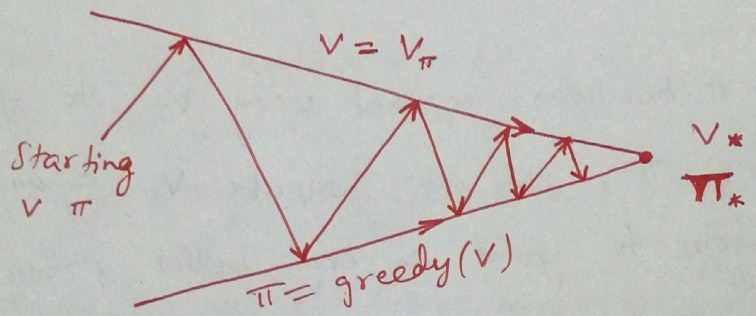
Once a policy π has been improved using V_π to yield a better policy π', we can compute $V_{\pi'}$ again & improve it again to yield an even better policy π''. This way of finding optimal policy is called policy iteration.

☞ Policy iteration consists of two simultaneous, interacting processes, one making the value fun^n consistent with the current policy & other making policy greedy w.r.t. current value fun^n. This is in general not necessary to alternate b/w PEval & PImp. In asynchronous DP, in some cases a single

state is updated in one process before returning to other states. As long as both processes continue to update all states (infinitely as $t \to \infty$), convergence to optimal policy is guaranteed.



→ Generalized policy Iteration (GPI) is the general idea of letting policy-evaluation & policy-improvement processes interact, independent of granularity and other details of the two processes.



Policy iteration (using iterative policy eval.) for estimating $\pi \approx \pi_*$ using V_*

1. Initialize $V(s) \in \mathbb{R}$ & $\pi(s) \in A(s)$ arbitrarily
2. Perform policy evaluation for policy π to get V
3. Policy improvement
   for each state $s \in S$
   $$\pi(s) \leftarrow \underset{a}{\text{argmax}} \sum_{s'} P^a_{ss'} (r(s,a,s') + \gamma V(s'))$$
4. Go to step 2 until convergence

Excercise 4.5 (Sutton)

Policy iteration to estimate π_* using action-value fun^n q_*

(1) Initialization

$Q(s,a) \in \mathbb{R}$ and $\pi(s) \in A(s)$ arbitrarily $\forall\, s \in \mathcal{S}$

(2) Policy evaluation

Loop

$\Delta \leftarrow 0$

Loop for each $s \in \mathcal{S}$ and $a \in A(s)$

$q \leftarrow Q(s,a)$

$$Q(s,a) \leftarrow \sum_{s'} P^{a}_{ss'} \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q(s',a') \right)$$

$\Delta \leftarrow \max(\Delta, |q - Q(s,a)|)$

until $\Delta < \theta$ (threshold determining accuracy)

(3) Policy improvement

policy-stable $\leftarrow$ True

for each $s \in \mathcal{S}$ and $a \in A(s)$

old-action $\leftarrow \pi(s)$

$$\pi(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s'} P^{a}_{ss'} \left(r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q(s',a') \right)$$

if old-action $\notin \{a_i\}$ (set of equiprobable policies (actions) from $\pi(s)$)

then policy-stable $\leftarrow$ false

if policy-stable then stop and return $Q \approx q_*$ and $\pi \approx \pi_*$

else goto step (2)

Value iteration

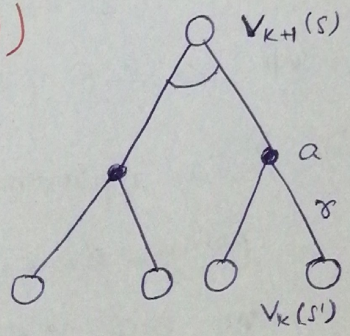
Policy iteration has a drawback that each of its iterations involves policy evaluation, which may itself be an iterative process requiring multiple sweeps through the state set $\mathcal{S}$.

However, without losing convergence guarantees policy evaluation can be truncated in several ways. One special case is when policy evaluation is stopped after just one sweep (one update of each state). This is called value iteration algorithm.

$$V_{k+1}(s) \doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma V_k(S_{t+1}) \mid S_t = s, A_t = a\right]$$

$$= \max_a \sum_{s'} P_{ss'}^a \left(r(s,a,s') + \gamma V_k(s')\right)$$

$$\forall s \in \mathcal{S}$$



* This can again be shown to converge to V_* under conditions that guarantee the existence ~~convergence~~ of V_*

Principle of Optimality

A policy $\pi(a|s)$ achieves the optimal value from state s, $V_\pi(s) = V_*(s)$ iff

* for any state s' reachable from s, π achieves the optimal value from s'. $V_\pi(s') = V_*(s')$

Value iteration can be thought of as applying principle of optimality iteratively.

$$V_*(s) = \max_{a \in \mathcal{A}} R_s^a + \gamma \sum_{s' \in \mathcal{S}} P_{ss'}^a V_*(s')$$

$$= \max_{a \in \mathcal{A}} \sum_{s'} P_{ss'}^a \left(r(s,a,s') + \gamma V_*(s') \right)$$

Value iteration for estimating $\pi \approx \pi_*$

1. Initialize $V(s) \quad \forall s \in \mathcal{S}^+$ arbitrarily except $V(\text{terminal}) = 0$
2. Loop

   $\Delta \leftarrow 0$

   Loop for each $s \in \mathcal{S}$

   $v \leftarrow V(s)$

   $V(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a \left(r(s,a,s') + \gamma V(s') \right)$

   $\Delta \leftarrow \max(\Delta, |V(s) - v|)$

   until $\Delta < \theta$
3. ~~Output~~ Output deterministic policy $\pi \approx \pi_*$ s.t

$$\pi(s) = \arg\max_a \sum_{s'} P_{ss'}^a \left(r(s,a,s') + \gamma V(s') \right)$$

Synchronous DP algorithms

Problem	Bellman Eqn	Algorithm
Prediction	Bellman expectation eqn	Iterative policy evaluation
Control	Bellman exp. eqn + Greedy policy improvement	Policy iteration
Control	Bellman optimality eqn	Value iteration

Asynchronous DP

One major drawback of DP methods discussed so far is that they used synchronous backups; they require sweeps of the state set. This is very expensive if state set is very large.

One soln to this is asynchronous DP, which backs up states individually in any order. It, for each selected state, applies the appropriate backup.

* This is also guaranteed to converge if all states continue to be selected.

(1) In-place DP

Only store one copy of value fun^n

$$V(s) \leftarrow \max_{a \in A} \sum_{s'} P^a_{ss'} \left(r(s,a,s') + \gamma V(s') \right)$$

(2) Real-time DP

Only update/ backup states that are ~~backup~~ relevant to the agent. Use agent's exp. to guide selection of states.

(3) Prioritised sweeping

Use magnitude of Bellman error to guide state selection.

Eg.

$$\left| \max_{a \in A} \left(\sum_{s'} P^a_{ss'} \left(r(s,a,s') + \gamma V(s') \right) \right) - V(s) \right|$$