Markov decision process (MDP)

Setting: An agent can bias the evolution of a randomly fluctuating environment by taking "actions", in the hope of increasing the future "reward" it will collect.

Goal: Learning the optimal set of environment-dependent decisions or action ensuring maximum reward over the horizon of the process.

Assumption: The MDP setting assumes that one has knowledge of the dynamics of the world in response to any actions. This is clearly unrealistic but offer a well-posed framework. A realistic learning method should only rely on past reward / action / state experience to improve decision policies. This will be the topic of next class. This class is about the computational role of dynamic programming in solving MDP problem.

Recall: A stationary MDP is given by $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$

* $\mathcal{S}$ is a finite set of states, $S_t \in \mathcal{S}$ is the state at time t.
* $\mathcal{A}$ is a finite set of actions, $A_t \in \mathcal{A}$ is the action at time t.
* P is a probability transition kernel: $P^a_{ss'} = \mathbb{P}[S_{t+1}=s' \mid S_t=s, A_t=a]$
* R is a reward function: $R^a_s = \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a]$
* γ is the discount factor involved in the return:

$$G_t = \sum_{k \geq 0} \gamma^k R_{t+k+1}, \quad 0 < \gamma < 1$$

Remark: Other settings are possible, e.g., $G_t = \sum_{k=0}^{T} R_{t+k+1}$ ← T: termination time, with finite expectation

Bellman equations

At fixed policy $\pi(a|s) = \mathbb{P}[A_t = a \mid S_t = s]$, the sequence (S_t, A_t) defines a Markov chain and the expected return conditioned to being in some state s, i.e., the value function

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\Big[\sum_{k\geq 0} \gamma^k R_{t+k+1} \mid S_t = s\Big]$$

can be computed by solving Bellman equation:

$$\boxed{v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\Big(\mathcal{R}_s^a + \gamma \sum_{s'} P_{ss'}^a \cdot v_\pi(s)\Big) \leftarrow (BE)}$$

Introducing the action-value function $q_\pi(s,a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$ allows one to obtain the Bellman optimality equation characterizing the (unique) optimal value function $v^* = \max_\pi v_\pi$, where we consider the order $v_\pi \gg v_{\pi'} \iff v_\pi(s) \geq v_{\pi'}(s)$ for all s.

1) $v_*(s) = \max_a q_*(s,a)$ ← most beneficial action (not necessarily unique)

2) $q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} P_{ss'}^a \cdot v_*(s')$ ← value of following an optimal policy after taking action a at t.

Taking 1) and 2) together yields

$$\boxed{v_*(s) = \max_a \Big(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} P_{ss'}^a \cdot v_*(s')\Big) \leftarrow (BO)}$$

In writing the above equation, we use the fact that 1) justifies that there is always a deterministic policy by picking a single action among the potentially many best ones.

Theorem: For all MDP, there is an optimal policy π_* (that can be chosen to be deterministic) such that $v_{\pi_*} \geq v_\pi$ for all π. All optimal policies achieve the same optimal value function $v_* = v_{\pi_*}$ and the same optimal action-value function $q_* = q_{\pi_*}$.

Bellman operators

Interpreting both Bellman equations in term of fixed point equations suggests considering the two Bellman operators:

* $T_\pi : \mathbb{R}^{|\mathcal{S}|} \longrightarrow \mathbb{R}^{|\mathcal{S}|}$

$$v \longmapsto \left\{ s \longmapsto T_\pi v(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s'} P_{ss'}^a v(s') \right) \right\}$$

* $T : \mathbb{R}^{|\mathcal{S}|} \longrightarrow \mathbb{R}^{|\mathcal{S}|}$

$$v \longmapsto \left\{ s \longmapsto T v(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s'} P_{ss'}^a v(s') \right) \right\}$$

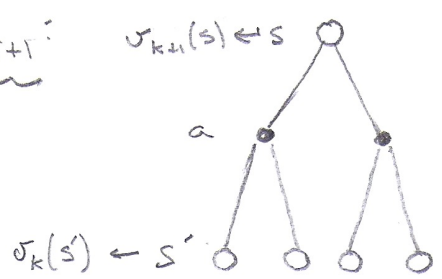
Both T_π and T are contraction on $\mathbb{R}^{|\mathcal{S}|}, \|\cdot\|_\infty$. By the Banach fixed point theorem, there are unique value functions v_π and v_* such that $T_\pi(v_\pi) = v_\pi$ and $T(v_*) = v_*$.

More importantly computationwise, these value functions can be computed as $v_\pi = \lim_{n \to +\infty} T_\pi^n(v_0)$ for some initial v_0.

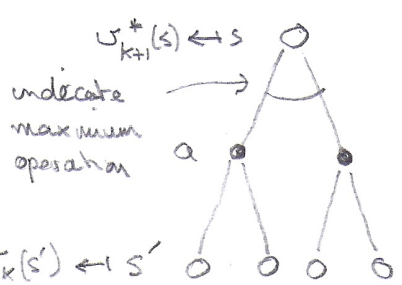
$$v_* = \lim_{n \to +\infty} T^n(v_0)$$

- This observation justifies considering various computational approaches, namely value iteration and policy iteration algorithm, which are related to dynamic programming.
- Applying T_π and T is commonly referred to performing a backup call and represented diagrammatically as follows:

$T_\pi(v_k) = v_{k+1}$:



$T(v_k^*) = v_{k+1}^*$:



Iterative steps

Algorithms solving MDP will involve two types of steps:

1) Policy evaluation steps = applying operator T_π to the current guess for the sought after policy π.

Full policy evaluation involves solving (BE) by matrix inversion or running the iterative steps until fixed point convergence. In practice, we only have guaranteed asymptotic convergence. There is need for an halting criterion typically involving the Bellman error: $\|T_\pi v - v\|_\infty$.

2) Policy improvement steps = modifying current policy by adopting the greedy strategy, i.e, the strategy that optimize immediate reward with no consideration for future reward

We can restrict ourselves to deterministic policies: $a = \pi(s)$.

* IF $\pi(s) = \arg\max_a q_\pi(s,a)$ then $v_\pi(s) = \max_a q_\pi(s,a)$ for all s and v_π satisfies (BO), thus $v_\pi = v_*$.
* Otherwise, define the greedy strategy $\pi'(s) = \arg\max_a q_\pi(s,a)$: We have $\pi' \geq \pi$. Indeed for all s, we have $v_\pi(s) \leq q_\pi(s,\pi'(s))$, thus

$$
\begin{aligned}
v_\pi(s) \leq q_\pi(s,\pi'(s)) &= \mathbb{E}_{\pi'}\left[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_t + \gamma\, q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s\right] \\
&= \mathbb{E}_{\pi'}\left[R_t + \gamma\, \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+2})\right] \mid S_t = s\right] \\
&= \mathbb{E}_{\pi'}\left[R_t + \gamma R_{t+1} + \gamma^2 v_\pi(S_{t+2}) \mid S_t = s\right] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}\left[\sum_{k\geq 0} \gamma^k R_{t+k+1} \mid S_t = s\right] = v_{\pi'}(s) \quad \checkmark
\end{aligned}
$$

Policy iteration

1] Let π_0 be any policy

2] At each iteration: * given π_k compute $v_{\pi_k} = v_k$. (evaluation)

* compute the greedy policy (improvement)

$$\pi_{k+1}(s) = \arg\max_a \left[R_s^a + \gamma \sum_{s'} P_{ss'}^a v(s') \right]$$

3] Return last policy π_K. (K smallest such that $v_K = v_{K-1}$ if policy evaluation is exact. Halting condition otherwise).

> Policy iteration yields a sequence of policies with non decreasing value function: $v_{k+1} \geq v_k$. Moreover if $v_{k+1} = v_k$ then $v_k = v_*$ and the algorithm converges in a finite number of steps.

Proof: At step k, we have $v_k = T_k v_k$, $T_k = T_{\pi_k}$.

By definition of the greedy policy π_{k+1}, we also have $Tv_k = T_{k+1} v_k$

Moreover, because of the max operation, we have

$$v_k = T_k v_k \leq T v_k = T_{k+1} v_k$$

Finally by monotonicity of T_{k+1}, we obtain $(T_{k+1})^n v_k \leq (T_{k+1})^{n+1} v_k$ for all $n \in \mathbb{N}$. Thus:

$$v_k \leq \lim_{n \to +\infty} (T_{k+1})^n v_k = v_{k+1}$$

i.e., v_k is a non decreasing sequence. Finiteness of the steps follows from the finiteness of deterministic policies.

Comment

* Exact policy evaluation is not always possible
* Even when exact policy evaluation is possible, convergence may be computationally prohibitive

Value iteration

1] Let v_0 be any vector in $\mathbb{R}^{|\mathcal{S}|}$

2] At each iteration; compute $v_{k+1}(s) = \mathcal{T} v_k(s) = \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a v_k(s') \right)$

3] Return last policy π_k (k smallest number satisfying some halting conditions) with $\pi_k(s) = \arg\max_a \left(\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a v_k(s') \right)$

Asymptotic convergence guaranteed by contraction argument of $\mathcal{T}$

↳ each iteration is computationally efficient but only asymptotic convergence.

Comment: The methods described so far involved synchronous backups, i.e., all state value are updated in one sweep. Asynchronous backups are possible whereby states are individually backed up. This can substantially reduce computations.

Examples: 1] In-place value iteration stores only one copy $v(s)$:

$$v(s) \leftarrow \max_a \left(\mathcal{R}_s^a + \gamma \sum_{s'} \mathcal{P}_{ss'}^a v(s') \right)$$

2] In-place value iteration with prioritized ordering: visit to states follow an order prescribed by decreasing Bellman error: $|\mathcal{T} v(s) - v(s)|$.

3] Real-time dynamic programming. Simulate an agent and only update the visited state.

$$v(S_t) \leftarrow \max_a \left(\mathcal{R}_{S_t}^a + \gamma \sum_{s'} \mathcal{P}_{S_t s'}^a v(s') \right)$$

$v(S_t)$: visit at time t; $\mathcal{R}_{S_t}^a$, $\mathcal{P}_{S_t s'}^a$: visit at time t

Dynamic programming

Optimization for sequential problem with full knowledge / model of the state / action space.

Dynamic programming generally applies if:

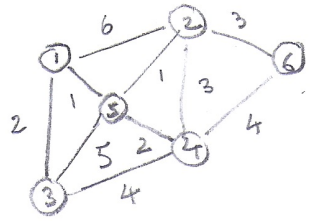
1] Optimal substructure: solution may be decomposed in collections of subproblems.

2] Overlapping subproblem: solution to the same subproblem is used many time to build the full solution

Type of "divide-and-conquer" approach.

Typical example: shortest path distance in graph ?

↳ dynamic programming solution Dijkstra's algorithm (1956)



Naive estimate of the number of candidate path given that no edges is visited twice

possible path = |E|!

1] Optimal substructure: if 1-5-2-6 is the shortest path between 1 and 6 then 1-5-2 and 5-2-6 are shortest path.

2] Overlapping subproblem: only shortest paths of size $< L$ can feature as subpaths of shortest path of size L.

Algorithm: 1] create a set of unvisited vertex: $S = V \setminus \{i\}$ ← reference vertex
and initialize $d_j = +\infty$, $j \in S$, $d_i = 0$.

2] while S is non empty, pick $u = \underset{v \in S}{\operatorname{argmin}}\, d(v)$.

a] Set $S = S \setminus \{u\}$.

b] for all $v \in Q$, v neighbor of u.

→ if $d(u) + l_{uv} < d(v)$ then $d(v) \leftarrow d(u) + l_{uv}$

"update of the" value function distance $d(u)$.

3] return $d(u)$

Recursive optimality principle

edge length

Shortest path problem:

$$d(i,j) = \min_{p \in \mathcal{P}_{ij}} L(p) = \min_{v \neq i} \left[l_{iv} + \min_{p \in \mathcal{P}_{vj}} L(p) \right]$$

- $\mathcal{P}_{ij}$: paths from i to j
- $L(p)$: length of path
- $\mathcal{P}_{vj}$: paths from v to j

Same recursive structure as Bellman equation.

In the context of optimal control theory dynamic programming applies generally to optimization of additive cost functions:

For instance:

* $S_{t+1} = F_t(S_t, \Pi_t(S_t), \xi_t)$ ← state dynamics
  - Π_t: control
  - ξ_t: noise perturbation

* $J_\Pi = \mathbb{E}_\xi \left\{ g_T(S_T) + \sum_{k=0}^{T-1} g_k(S_k, \Pi_k(S_k), \xi_k) \right\}$
  - $g_T(S_T)$: terminal cost
  - $g_k(S_k, \Pi_k(S_k), \xi_k)$: intermediary cost

* $J_i^*(S_i) = \min_\Pi \mathbb{E}_\xi \left\{ g_T(S_T) + \sum_{k=i}^{T-1} g_k(S_k, \Pi_k(S_k), \xi_k) \right\}$

Bellman optimality principle for dynamic programming

$$J_i^*(S_i) = \min_{\Pi_i} \mathbb{E}_{\xi_i} \left\{ g_i(S_i, \Pi_i, \xi_i) + J_{i+1}^*[F_i(S_i, \Pi_i, \xi_i)] \right\}$$

$$J_T^*(S_T) = g_T(S_T).$$