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 HDP setting assumes that one has knowledge of the dynamics of the world in response to any actions. This is clearly unrealistic but offers 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 problems.

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

\* \(S\) is a finite set of states, \(S_t \in S\) is the state at time \(t\).
\* \(A\) is a finite set of actions, \(A_t \in A\) is the action at time \(t\).
\* \(P\) is a probability transition kernel: \(P_{st} = P[S_{t+1} = s' | S_t = s, A_t = a]\)
\* \(R\) is a reward function: \(R^a_s = E_t [R_{t+1} | S_t = s, A_t = a]\)
\* \(\gamma\) is the discount factor involved in the return:

\[G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad 0 < \gamma < 1\]

Remark: Other settings are possible, e.g., \(G_t = \sum_{k=0}^{\text{termination time}} R_{t+k+1}\) with finite expectation.
Bellman equations

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

$$v_\pi(s) = E_\pi[G_{t+1} | S_t = s] = E_\pi\left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$$

can be computed by solving Bellman equation:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) (R^a_s + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')) \quad \text{\cite{8E}}$$

Introducing the action-value function $q_\pi(s,a) = E_\pi[G_{t+1} | S_t = s, A_t = a]$ allows one to obtain the Bellman optimality equation characterizing the (unique) optimal value function $v^* = \max_\pi v_\pi$, where we consider the order $v_\pi \gg v_\pi \gg v_\pi$ for all $s$.

1. $v^*_t(s) = \max_a q^*_t(s, a)$ is most beneficial action (not necessarily unique)

2. $q^*_t(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v^*_t(s')$ is value of following an optimal policy after taking action $a$ at $s$.

Taking 1 and 2 together yields

$$v^*_t(s) = \max_a \left( R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} v^*_t(s') \right) \quad \text{\cite{BO}}$$

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 $\pi^*$ (that can be chosen to be deterministic) such that $v_{\pi^*} > v_{\pi}$ for all $\pi$.

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:

\[ T_\pi : \mathbb{R}^{\mathcal{S}} \rightarrow \mathbb{R}^{\mathcal{S}} \]
\[ \sigma \mapsto \{ s \mapsto T_\pi \sigma(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left( R_s^a + \gamma \sum_{s'} P_{ss'}^a \sigma(s') \right) \} \]

\[ T : \mathbb{R}^{\mathcal{S}} \rightarrow \mathbb{R}^{\mathcal{S}} \]
\[ \sigma \mapsto \{ s \mapsto T \sigma(s) = \max_{a \in \mathcal{A}} \left( R_s^a + \gamma \sum_{s'} P_{ss'}^a \sigma(s') \right) \} \]

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

More importantly, computationally, these value functions can be computed as \( \sigma_\pi = \lim_{n \rightarrow +\infty} T_\pi^n(\sigma_0) \) for some initial \( \sigma_0 \).

\[ \sigma^* = \lim_{n \rightarrow +\infty} T^n(\sigma_0) \]

This observation justifies considering various computational approaches, namely value iteration and policy iteration algorithm, which are related to dynamic programming.

Applying \( T_\pi \) and \( T \) is commonly referred to performing a back up call and represented diagrammatically as follows:

\[ T_\pi(\sigma_k) = \sigma_{k+1} ; \quad \sigma_{k+1}(s) \leftarrow \min_a \{ \sigma_k(s) ; s \rightarrow s' \} \]

\[ T(\sigma_k^*) = \sigma_{k+1}^* ; \quad \sigma_{k+1}^*(s) \leftarrow \max_a \{ \sigma_k^*(s) ; s \rightarrow s' \} \]

Understate maximum operation

Decide
Iterative steps

Algorithms solving MDP will involve two types of steps:

1. **Policy evaluation steps** = applying operator $T_\pi$ to the current guess for the sought after policy $\pi$.

   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: $||T_\pi \sigma - \sigma||_\infty$.

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

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

* If $\pi(s) = \arg \max_a \quad q_\pi(s,a)$ then $\pi^*_\pi(s) = \max_a q_\pi(s,a)$ for all $s$ and $\pi^*_\pi$ satisfies $(R0)$, thus $\pi^*_\pi = \sigma^*_\pi$.

* Otherwise, define the greedy strategy $\pi^*_\pi(s) = \arg \max_a q_\pi(s,a)$:

  We have $\pi^*_\pi \geq \pi^*_\pi$. Indeed for all $s$, we have $\pi^*_\pi(s) \leq q_\pi(s, \pi^*_\pi(s))$, thus

  \[
  \pi^*_\pi(s) \leq q_\pi(s, \pi^*_\pi(s)) = \mathbb{E}_\pi \left[ R_t + \gamma \pi^*_\pi(S_{t+1}) \mid S_t = s \right] \\
  \leq \mathbb{E}_\pi \left[ R_t + \gamma \pi(S_{t+1}, \pi(S_{t+1})) \mid S_t = s \right] \\
  = \mathbb{E}_\pi \left[ R_t + \gamma \left( R_{t+1} + \pi^*_\pi(S_{t+2}) \right) \mid S_t = s \right] \\
  = \mathbb{E}_\pi \left[ R_t + \gamma R_{t+1} + \gamma^2 \pi^*_\pi(S_{t+2}) \mid S_t = s \right] \\
  \vdots \\
  \leq \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] = \pi^*_\pi(s)
  \]
Policy Iteration

1. Let $T_i$ to be any policy

2.1 At each iteration: given $T_k$, compute $V_{T_k} = V_k$ (evaluation)

* Compute the greedy policy (improvement)

$$ T_{k+1}(s) = \text{arg max}_a \left[ R_s^a + \gamma \sum_{s'} P_{ss'}^a V(s') \right] $$

3. Return last policy $T_k$, (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 $V_k = \pi^*$ and the algorithm converges in a finite number of steps.

Proof: At step k, we have $V_k = T_k \pi^* \geq T_k \pi_k$, $T_k = T_{T_k}$

By definition of the greedy policy $T_{k+1}$, we also have $T_{k+1} \pi_k = T_{k+1} \pi_k$.

Moreover, because of the max operation, we have

$$ V_k = T_k \pi_k \leq T_{k+1} \pi_k = T_{k+1} \pi_k $$

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

$$ V_k \leq \lim_{n \to \infty} (T_{k+1})^n \pi_k = \pi_{k+1} $$

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 \( \mathbb{R}^n \).

2. At each iteration, compute: \( v_k(s) = T v_k(s) = \max_a \left( R_s^a + y \sum_{s'} P_{ss'}^a v_k(s') \right) \)

3. Return last policy \( \pi_k \) (K smallest number satisfying some halting conditions) with \( \pi_k(s) = \arg \max_a \left( R_s^a + y \sum_{s'} P_{ss'}^a v_k(s') \right) \)

Asymptotic convergence guaranteed by contraction argument of \( T \)

L is each iteration is computationallly efficient but only asymptotic convergence.

Comment: Themethods 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 \( v(s) \):
   \[
   v(s) \leftarrow \max_a \left( R_s^a + y \sum_{s'} P_{ss'}^a v(s') \right)
   \]

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

3. Real-time dynamic programming. Simulate an agent and only update visited states:
   \[
   v(S_t) \leftarrow \max_a \left( R_{S_t}^a + y \sum_{S_t'} P_{S_tS_t'}^a v(S_t') \right)
   \]
   visit at time \( t \)
   visit at time \( t \)
Dynamic programming

Optimization for sequential problem with full knowledge/mode 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 times to build the full solution.

Type of "divide-and-conquer" approach.

**Typical example**: shortest path distance in graph?

A dynamic programming solution **Dijkstra's algorithm** (1956)

```
1  2  3  4  5  6
```

Number estimate of the number of candidate paths given that no edges is visited twice

# possible path = 151!

1) **Optimal substructure**: if 1-5-2-6 is the shortest path between 1 and 6 than 1-5-2 and 5-2-6 are shortest path.

2) **Overlapping subproblem**: only shortest path of size < L can feature as subpaths of shortest path of size L.

**Algorithm**:

1) create a set of unvisited vertex: \( S = V \setminus \{ s \} \) and reference vertex \( d_s = +\infty \), \( s \in S \), \( d_i = 0 \).

2) while \( S \) is non empty, pick \( u = \arg \min \{ d(v) : v \in S \} \).

3) set \( S = S \setminus \{ u \} \).

4) for all \( v \in Q, \) \( v \) neighbor of \( u \).

```
update of the value function
```

5) if \( d(v) + \| u,v \| < d(v) \) then \( d(v) \leftarrow d(v) + \| u,v \| \) distance \( d(u) \).

6) return \( d(u) \)
**Recursive optimality principle**

**Shortest path problem:**

\[ d(i,j) = \min_{p \in S_{i,j}} \left[ \min_{u \neq i} \left( \sum_{p \in S_{u,i}} L(p) \right) \right] \]

Paths from \( i \) to \( j \)  
Length of path  
Paths from \( u \) to \( i \)

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:

* \( S_{t+1} = \mathbf{F}_t (S_t, \Pi_t(S_t), z_t) \) — state dynamics
  
  \[ \uparrow \text{control} \quad \uparrow \text{noise perturbation} \]

* \( J_t = E_j \left\{ q_t(S_t) + \sum_{k=0}^{T-1} g_k(S_k, \Pi_k(S_k), z_k) \right\} \)
  
  \[ \uparrow \text{terminal cost} \quad \uparrow \text{intermediate cost} \]

* \( J^*_i(S_i) = \min_{\Pi_i} E_j \left\{ q_i(S_i, \Pi_i, z_i) + J^*_{i+1}[F_i(S_i, \Pi_i, z_i)] \right\} \)

**Bellman optimality principle for dynamic programming**

\[ J^*_i(S_i) = \min_{\Pi_i} E_j \left\{ q_i(S_i, \Pi_i, z_i) + J^*_{i+1}[F_i(S_i, \Pi_i, z_i)] \right\} \]

\[ J^*_T(S_T) = q_T(S_T) \]