Outline

- Finite Horizon MDPs
- Dynamic Programming
Finite Horizon MDPs - Value Functions

- The duration, or expected duration, of the process is finite
- Let's consider the following return functions:
  - The expected sum of rewards
    \[
    V^\pi(s) = \lim_{N \to \infty} E^\pi \left[ \sum_{t=1}^{N} r_t \mid s_0 = s \right]
    \]
    
  - The expected discounted sum of rewards
    \[
    V^\pi_\lambda(s) = \lim_{N \to \infty} E^\pi \left[ \sum_{t=1}^{N} \lambda^{t-1} r_t \mid s_0 = s \right]
    \]
    \[0 < \lambda < 1\]
    
a sufficient condition for the above to converge is \( r_t < r_{max} \)...
Return functions

- If \( r_t < r_{\text{max}} \) holds, then

\[
V_{\lambda}^\pi (s) \leq \sum_{t=1}^{N} \lambda^{t-1} r_{\text{max}} = \frac{r_{\text{max}}}{1 - \lambda}
\]

Note that this bound is very sensitive to the value of \( \lambda \)

- The expected average reward

\[
M^\pi (s) = \lim_{N \to \infty} \frac{1}{N} E_{\pi} \left[ \sum_{t=1}^{N} r_t \mid s_0 = s \right]
\]

\[
= \lim_{N \to \infty} \frac{1}{N} V_{N}^\pi (s)
\]

Note that the above limit does not always exist!
Consider a finite horizon problem where the horizon is random, i.e.

\[ V^\pi_N(s) = E_\pi E_N \left[ \sum_{t=1}^{N} r_t \mid s_0 = s \right] \]

Let's also assume that the final value for all states is zero.

Let \( N \) be geometrically distributed with parameter \( \lambda \), such that the probability of stopping at the \( N^{th} \) step is

\[ \Pr[N = n] = (1 - \lambda)\lambda^{n-1} \]

**Lemma:** we'll show that

\[ V^\pi_\lambda(s) = V^\pi_N(s) \]

under the assumption that \( |r_t| < r_{max} \)
Proof:

\[ V_N^\pi (s) = E_\pi \left\{ \sum_{n=1}^{\infty} \left( 1 - \lambda \right) \lambda^{n-1} \sum_{t=1}^{n} r_t \right\} \]

\[ \Delta = E_\pi \left[ \sum_{t=1}^{\infty} r_t (1 - \lambda) \sum_{n=t}^{\infty} \lambda^{n-1} \right] \]

\[ = E_\pi \left[ \sum_{t=1}^{\infty} r_t (1 - \lambda) \frac{\lambda^{t-1}}{1 - \lambda} \right] \]

\[ = E_\pi \left[ \sum_{t=1}^{\infty} \lambda^{t-1} r_t \right] \]

\[ = V_\lambda^\pi (s) \]
Outline

- Finite Horizon MDPs (cont.)
- Dynamic Programming
Example of a finite horizon MDP

Consider the following state diagram:

- State $S_1$ with actions $a_{11}$ and $a_{12}$, where $a_{11}$ has reward $(5, 0.5)$ and $a_{12}$ has reward $(10, 1)$.
- State $S_2$ with actions $a_{21}$ and $a_{22}$, where $a_{21}$ has reward $(-1, 1)$.

The diagram shows the transitions between states and the associated rewards for each action.
Why do we need DP techniques?

- Explicitly solving the Bellman Optimality equation is hard
  - Computing the optimal policy → solve the RL problem
- Relies on the following three assumptions
  - We have perfect knowledge of the dynamics of the environment
  - We have enough computational resources
  - The Markov property holds
- In reality, all three are problematic
- e.g. Backgammon game: first and last conditions are ok, but computational resources are insufficient
  - Approx. $10^{20}$ state
- In many cases we have to settle for approximate solutions (much more on that later ...)
During the next few weeks we'll talk about techniques for solving the RL problem

- **Dynamic programming** - well developed, mathematically, but requires an accurate model of the environment
- **Monte Carlo methods** - do not require a model, but are not suitable for step-by-step incremental computation
- **Temporal difference learning** - methods that do not need a model and are fully incremental
  - More complex to analyze
  - Launched the revisiting of RL as a pragmatic framework (1988)

The methods also differ in efficiency and speed of convergence to the optimal solution
Dynamic programming is the collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as an MDP.

- DP constitutes a theoretically optimal methodology.
- In reality often limited since DP is computationally expensive.
- Important to understand → reference to other models:
  - Do “just as well” as DP
  - Require less computations
  - Possibly require less memory
- Most schemes will strive to achieve the same effect as DP, without the computational complexity involved.
Dynamic Programming (cont.)

- We will assume finite MDPs (states and actions)
- The agent has knowledge of transition probabilities and expected immediate rewards, i.e.

\[
P^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}
\]

\[
R^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}
\]

- The key idea of DP (as in RL) is the use of value functions to derive optimal/good policies
- We’ll focus on the manner by which values are computed
- Reminder: an optimal policy is easy to derive once the optimal value function (or action-value function) is attained
Dynamic Programming (cont.)

Employing the Bellman equation to the optimal value/action-value function, yields

\[
V^*(s) = \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^*(s') \right]
\]

\[
Q^*(s, a) = \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right]
\]

DP algorithms are obtained by turning the Bellman equations into update rules.

These rules help improve the approximations of the desired value functions.

We will discuss two main approaches: policy iteration and value iteration.
Method #1: Policy Iteration

- Technique for obtaining the optimal policy
- Comprises of two complementing steps
  - **Policy evaluation** - updating the value function in view of current policy (which can be sub-optimal)
  - **Policy improvement** - updating the policy given the current value function (which can be sub-optimal)
- The process converges by “bouncing” between these two steps

![Diagram](image)
Policy Evaluation

- We’ll consider how to compute the state-value function for an arbitrary policy
- Recall that

\[
V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^{a} \left[ R_{ss'}^{a} + \gamma V^\pi(s') \right],
\]

(assumes that policy \( \pi \) is always followed)
- The existence of a unique solution is guaranteed as long as either \( \gamma < 1 \) or eventual termination is guaranteed from all states under the policy
- The Bellman equation translates into \(|S|\) simultaneous equations with \(|S|\) unknowns (the values)
- Assuming we have an initial guess, we can use the Bellman equation as an update rule
We can write

\[ V_{k+1}(s) = \sum_a \pi(s, a) \sum_{s'} P^{a}_{ss'} \left[ R^{a}_{ss'} + \gamma V_k(s') \right] \]

The sequence \( \{V_k\} \) converges to the correct value function as \( k \to \infty \).

In each iteration, all state-values are updated

- a.k.a. full backup

A similar method can be applied to state-action \((Q(s,a))\) functions.

An underlying assumption: all states are visited each time

- Scheme is computationally heavy
- Can be distributed - given sufficient resources (Q: How?)

"In-place" schemes - use a single array and update values based on new estimates

- Also converge to the correct solution
- Order in which states are backed up determines rate of convergence
Iterative Policy Evaluation algorithm

- A key consideration is the termination condition
- Typical stopping condition for iterative policy evaluation is

\[
\max_{s \in S} \left| V_{k+1}(s) - V_k(s) \right|
\]

Input \( \pi \), the policy to be evaluated
Initialize \( V(s) = 0 \), for all \( s \in S^+ \)
Repeat
\[
\Delta \leftarrow 0
\]
For each \( s \in S \):
\[
v \leftarrow V(s)
\]
\[
V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]
\]
\[
\Delta \leftarrow \max(\Delta, |v - V(s)|)
\]
until \( \Delta < \theta \) (a small positive number)
Output \( V \approx V^\pi \)
Policy Improvement

- Policy evaluation deals with finding the value function under a given policy.
- However, we don’t know if the policy (and hence the value function) is optimal.
- Policy improvement has to do with the above, and with updating the policy if non-optimal values are reached.
- Suppose that for some arbitrary policy, $\pi$, we’ve computed the value function (using policy evaluation).
- Let policy $\pi'$ be defined such that in each state $s$ it selects action $a$ that maximizes the first-step value, i.e.

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

- It can be shown that $\pi'$ is at least as good as $\pi$, and if they are equal they are both the optimal policy.
Consider a greedy policy, $\pi'$, that selects the action that would yield the highest expected single-step return:

$$\pi'(s) = \arg\max_a Q^\pi(s, a)$$

$$= \arg\max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$$

Then, by definition,

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s)$$

this is the condition for the policy improvement theorem.

The above states that following the new policy one step is enough to prove that it is a better policy, i.e. that

$$V^{\pi'}(s) \geq V^\pi(s)$$
Proof of the Policy Improvement Theorem

\[
V^\pi(s) \leq Q^\pi(s, \pi'(s)) \\
= E_{\pi'} \{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s \} \\
\leq E_{\pi'} \{ r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) \mid s_t = s \} \\
= E_{\pi'} \{ r_{t+1} + \gamma E_{\pi'} \{ r_{t+2} + \gamma V^\pi(s_{t+2}) \} \mid s_t = s \} \\
= E_{\pi'} \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) \mid s_t = s \} \\
\leq E_{\pi'} \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) \mid s_t = s \} \\
\vdots \\
\leq E_{\pi'} \{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \mid s_t = s \} \\
= V^{\pi'}(s).
\]
Policy Iteration

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

2. Policy Evaluation
   Repeat
   \[ \Delta \leftarrow 0 \]
   For each \( s \in S \):
   \[ v \leftarrow V(s) \]
   \[ V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[ \mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right] \]
   \[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]
   until \( \Delta < \theta \) (a small positive number)

3. Policy Improvement
   \( policy-stable \leftarrow true \)
   For each \( s \in S \):
   \[ b \leftarrow \pi(s) \]
   \[ \pi(s) \leftarrow \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a \left[ \mathcal{R}_{ss'}^a + \gamma V(s') \right] \]
   If \( b \neq \pi(s) \), then \( policy-stable \leftarrow false \)
   If \( policy-stable \), then stop; else go to 2