Markov Decision Processes and Reinforcement Learning

Day 1 — Definition of MDP and Examples

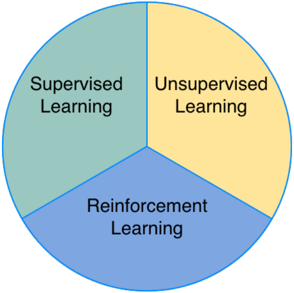
Luiz Felipe Martins

Department of Mathematics and Statistics
Cleveland State University

January 26, 2026

Outline

The Three ML Paradigms



- **Supervised Learning**: Model is given input–output pairs and learns to predict the correct output for new, unseen inputs.
- **Unsupervised Learning**: Model discovers structure or patterns without predefined target values.
- **Reinforcement Learning**: A model for sequential decision making, where the agent learns how to perform optimally in an environment.

What is Reinforcement Learning?

- In a Reinforcement Learning (RL) model, an agent acts on an environment.
- Depending on an action chosen by the agent, the state of the environment changes and the agent receives a numerical reward.
- The agent's goal is to choose actions in a way that long-term rewards are maximized.
- Historically, RL was independently developed in different areas. It is also known as Dynamic Programming, Markov Decision Processes (MDPs) and Sequential Decision Models.

Areas of Application

Application Area	Concrete Examples
Robotics & Control	Robotic grasping and assembly; drone navigation; quadruped locomotion; industrial pick-and-place systems.
Games & Decision Making	AlphaGo/AlphaZero (Go, Chess); Atari agents (DQN); StarCraft II and Dota 2 self-play systems.
Recommendation & Personalization	YouTube video ranking; e-commerce product ordering; personalized news feeds; adaptive educational tutors.
Autonomous Systems	Self-driving vehicles; warehouse AGVs; traffic light coordination; spacecraft docking.
Operations & Resource Allocation	Cloud job scheduling; dynamic pricing; energy grid control; supply chain routing and inventory management.
Finance & Trading	Portfolio optimization; algorithmic trading; market making; sequential fraud detection.
Healthcare & Medicine	Optimal drug dosing; radiotherapy planning; clinical decision support; robotic rehabilitation systems.

Workshop Goals

Goals:

- Understand the mathematical framework used by RL models.
- Learn about both classical and modern solution methods.
- Understand how to set up an RL model in an applied problem.
- Learn how to translate a mathematical RL model into a computational model.
- Introduce current libraries for solving RL problems.

Intuitive Description of a Markov Chain

- A **Markov Chain** is a probabilistic model for a process that evolves in time. It is a particular kind of discrete-time stochastic process.
- A key feature of a Markov Chain is that, at any time, the future evolution of the process depends only on the current state, and not on previous states visited by the chain.
- This seems to be restrictive, but it is a common assumption in mathematical modeling. It can usually be achieved by enlarging the state space to include all information needed to predict the future of the process.
- My favorite example: Isaac Newton realized that, to completely predict the motion of a particle in a force field, it is necessary to know both its initial *position* and *velocity*.

Definition of Markov Chain

- Let $\mathscr{S} = \{s_1, s_2, \ldots, s_N\}$ be a finite set. We call this set the **state space** of the chain, or **environment**.
- A **Markov Chain** on $\mathscr{S}$ is a stochastic sequence $\{S_t\}_{t\in\mathbb{N}}$ on $\mathscr{S}$ such that:

$$\mathbb{P}(S_{t+1} = s_j \mid S_t = s_i, S_{t-1}, S_{t-2}, \ldots, S_0) = \mathbb{P}(S_t = s_j \mid S_{t-1} = s_i)$$

  for all $t \geq 0$. This is called the **Markov Property**.
- The values:

$$\mathbb{P}(S_t = s_j \mid S_{t-1} = s_i)$$

  are called **transition probabilities**.

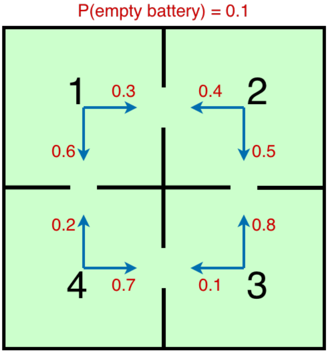
Transition Probability Matrix

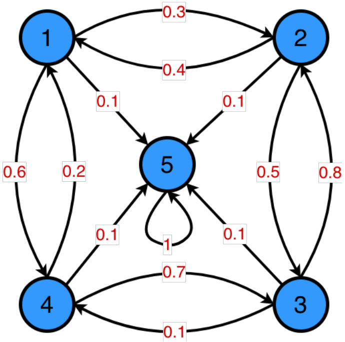
- We assume that our Markov Chain is **time-invariant**, meaning that the transition probabilities are independent of t.
- The **transition probability matrix** P of a Markov Chain $\{S_t\}_{t\in\mathbb{N}}$ is defined by:

$$P_{ij} = \mathbb{P}(S_{t+1} = s_j \mid S_t = s_i)$$

- Everything we need to compute for a Markov Chain can be expressed in terms of the matrix P.

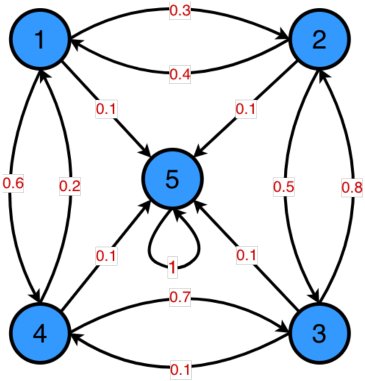
Markov Chain Example





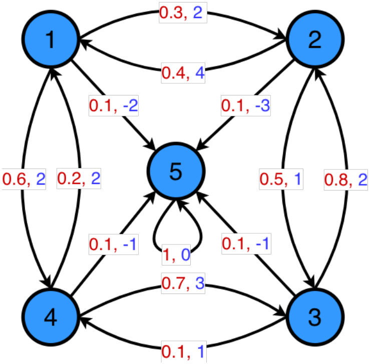
- Robot operates in a 4-room environment, in each step it transitions to another room with probabilities as given in the figure.
- Each step there is a probability of 0.1 that the robot's battery runs out, and the process terminates.

Transition Probability Matrix



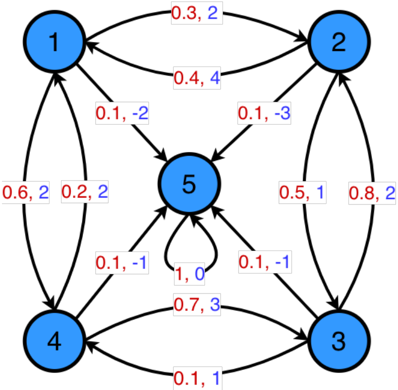
$$P = \begin{bmatrix} 0 & 0.3 & 0 & 0.6 & 0.1 \\ 0.4 & 0 & 0.5 & 0 & 0.1 \\ 0 & 0.8 & 0 & 0.1 & 0.1 \\ 0.2 & 0 & 0.7 & 0 & 0.1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Running Rewards



- A transition from s to s' yields a **reward** $r(s, s')$.
- A negative reward represents a penalty.
- Once the battery empties, the process stops.

Rewards Matrix



$$R = \begin{bmatrix} - & 2 & - & 2 & -2 \\ 4 & - & 1 & - & -3 \\ - & 2 & - & 1 & -1 \\ 2 & - & 1 & - & -1 \\ - & - & - & - & 0 \end{bmatrix}$$

- If the probability of a transition from s to s' is zero, the corresponding entry in the rewards matrix is irrelevant, and is marked “–”.
- Usually, these are just set to 0.

Markov Chain Example — Expected Total Reward

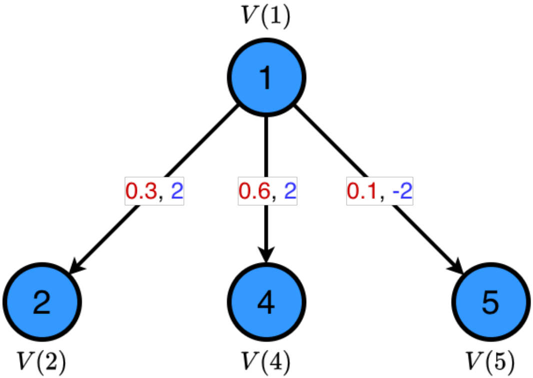
- Let $R_{t+1} = r(S_t, S_{t+1})$.
- The **expected total reward** is defined by:

$$\mathbb{E}\left[\sum_{t=0}^{T} R_{t+1}\right]$$

- T represents the termination time. Notice that $P(T < \infty) = 1$.
- The **state value function** is defined as the expected total reward:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{T} R_{t+1} \,\middle|\, S_0 = s\right]$$

One-step Analysis



$$V(1) = 0.3(2 + V(2)) + 0.6(2 + V(4)) + 0.1(-2 + V(5))$$

Computation of the Value Function

V is the solution of the *linear system*:

$$
\begin{aligned}
V(1) &= 0.3(2+V(2)) + 0.6(2+V(4)) + 0.1(-2+V(5)) \\
V(2) &= 0.4(4+V(1)) + 0.5(1+V(3)) + 0.1(-3+V(5)) \\
V(3) &= 0.8(2+V(2)) + 0.1(1+V(4)) + 0.1(-1+V(5)) \\
V(4) &= 0.2(2+V(1)) + 0.7(1+V(3)) + 0.1(-1+V(5)) \\
V(5) &= 0
\end{aligned}
$$

Solution:

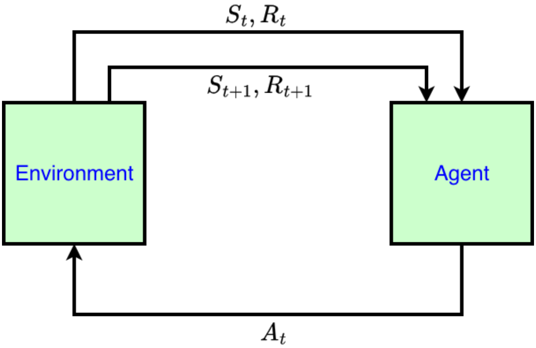
$$V(1) = 15.512 \quad V(2) = 15.943 \quad V(3) = 15.876$$

$$V(4) = 15.215 \quad V(5) = 0$$

The Components of an RL Model

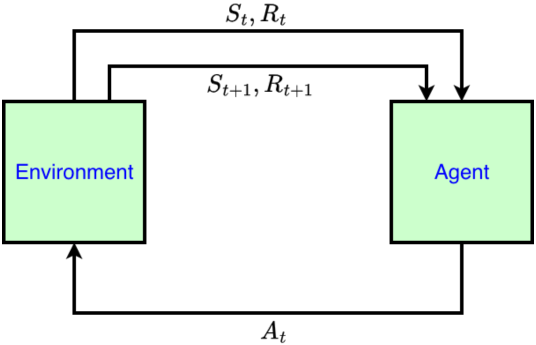
- The **environment** represents all information that is relevant for the optimization problem. It keeps a record of its *current state*, and provide the agent with information about *observations* and *rewards*.
- The **agent** interacts with the environment, obtaining *observations* (partial representations of the environment) and *rewards*. The rewards can be negative, representing penalties.
- The agent chooses **actions**. The actions determine both the probabilistic evolution of the environment and the rewards obtained by the agent.
- The agent's goal is to **maximize expected rewards** on a run of interactions with the environment.

The Agent-Environment Interaction



- The **observes** the environment, and chooses an **action**
- As a result of the action, the environment changes **state**.
- The agent receives a **reward**, dependent on the state of the environment and the action chosen.

A simplification



- We assume that, when choosing an action, the agent has *complete information about the state of the environment*.
- In the general case, the observation can be a subset of the state.

Definition of Markov Decision Process

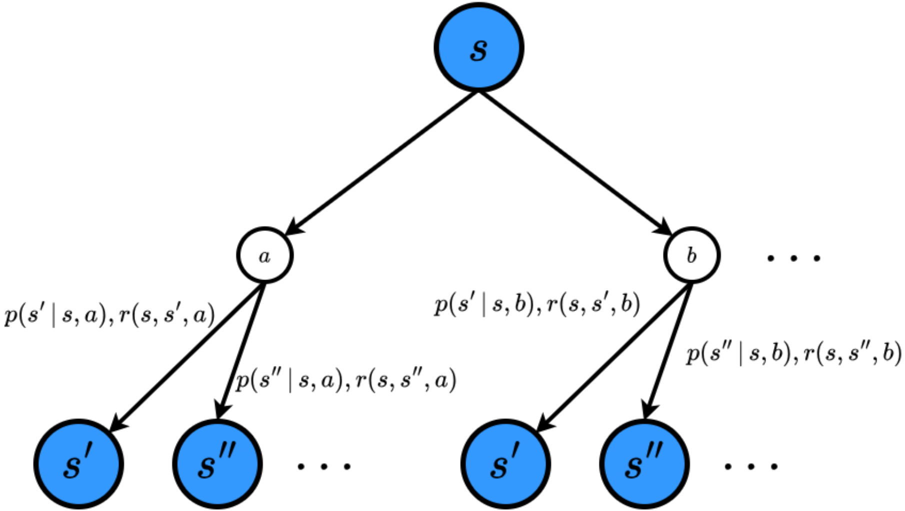
A **Markov Decision Process** (MDP) consists of:

- A finite set $\mathscr{S} = \{s_1, s_2, \ldots, s_N\}$, the set of states the environment can be in.
- For each $s \in \mathscr{S}$, a finite set $\mathscr{A}(s)$, the set actions available to the agent when the environment is in state s.
- Two functions of three variables:

$p(s' \mid s, a)$ representing *transition probabilities*

$r(s, s', a)$ representing *running rewards*

Graphical Representation



Transition probabilities

The **transition probability matrix** $P^{(a)}$ associated to action a is defined as follows:

$$P_{ij}^{(a)} = p(s_j \mid s_i, a)$$

We require:

$$0 \leq P_{ij}^{(a)} \leq 1, \quad \sum_{j=1}^{N} P_{ij}^{(a)} = 1$$

Note: $P_{ij}^{(a)}$ is defined arbitrarily if action a is not available for state s_i.

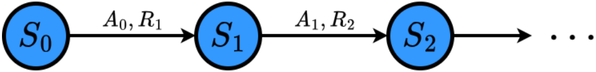
Running Rewards

- We assume that the reward received for a transision from s to s' under action a is given by a *deterministic function*:

$$r(s, s', a)$$

- Computational frameworks allow for randomized rewards.
- The **reward matrix** $R^{(a)}$ associated with action a is defined as:

$$R_{ij}^{(a)} = r(s_i, s_j, a)$$



A **trajectory** of a MDP is represented by three sequences of random variables:

- $S_0, S_1, S_2, \ldots$: the successive *states* of the environment.
- $A_0, A_1, A_2, \ldots$, the successive *actions* taken by the agent.
- $R_0, R_1, R_2, \ldots$, the successive *rewards* obtained by the agent.

These sequences are related as follows:

$$\mathbb{P}(S_{t+1} = s' \mid S_t = s) = p(s' \mid s, A_t)$$

$$R_{t+1} = r(S_t, S_{t+1}, A_t)$$

Curse of Dimensionality

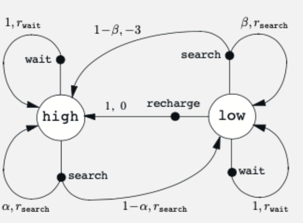
- The number of possible states is usually astronomically large. This phenomenon is known as the **curse of dimensionality**.
- Example: the game **2048**

- Each cell can be empty or contain a power of two, between 2 and $2048 = 2^{11}$. Thus, the number of possible states the environment can be in is:

$$12^{16} = 184852952865954416.$$

- In practical applications, it is impossible to tabulate all values of transition probabilities.

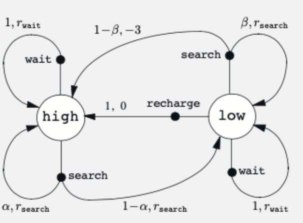
Example 1: Recycling Robot



- $\mathscr{S} = \{\texttt{high}, \texttt{low}\}$ representing the battery levels of the robot.
- $\mathscr{A}(\texttt{high}) = \{\texttt{wait}, \texttt{search}\}$. When in the high state, the robot can do nothing or search for a soda can.
- $\mathscr{A}(\texttt{low}) = \{\texttt{wait}, \texttt{search}, \texttt{recharge}\}$. When in the low state, the robot, besides waiting and searching, can go to a recharging station.

(From R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*)

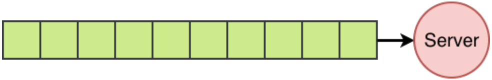
Transition Probabilities and Reward Vectors



$$P^{\texttt{search}} = \begin{bmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{bmatrix} \quad P^{\texttt{wait}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad P^{\texttt{recharge}} = \begin{bmatrix} - & - \\ 1 & 0 \end{bmatrix}$$

$$R^{\texttt{search}} = \begin{bmatrix} r_{\texttt{search}} & r_{\texttt{search}} \\ -3 & r_{\texttt{search}} \end{bmatrix} \quad R^{\texttt{wait}} = \begin{bmatrix} r_{\texttt{wait}} & - \\ - & r_{\texttt{wait}} \end{bmatrix} \quad R^{\texttt{recharge}} = \begin{bmatrix} - & - \\ 0 & - \end{bmatrix}$$

Example 2 — Server Optimization: System Evolution

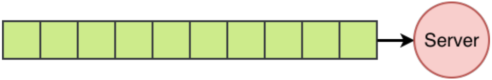


- A computer server receives service requests. Requests are queued in the order they arrive.
- Each second, there is a probability p that a new request arrives. The probability that two or more requests arrive in a second is negligible.
- The server has three modes of operation, `high`, `normal` and `low`. Every second, the probabilities that a request is completed are:

$$q_{\texttt{high}} > q_{\texttt{normal}} > q_{\texttt{low}} > 0$$

- The system can hold at most M requests. Requests that arrive when the system is full are rejected.

Example 2 — Server Optimization: Costs



Each second, following running costs occur:

- For each service mode, there is an *operation cost*, where:

$$c_{\texttt{high}} > c_{\texttt{normal}} > c_{\texttt{low}} > 0$$

- If there are N items in the system (being served or waiting for service), there is a *holding cost* $h(N)$. Function h is a positive, increasing function.
- Whenever an item is rejected due to a full system, a *service loss cost* $c_{\texttt{loss}}$ is incurred.

Transition Probabilities

Each instant in time, if there are i items in the system, the only possible transitions are to i, $i+1$ and $i-1$. Let $\texttt{mode} \in \{\texttt{high}, \texttt{normal}, \texttt{low}\}$:

- If $1 \leq i \leq M$:

$$p(i+1 \,|\, i, \texttt{mode}) = p(1 - q_{\texttt{mode}})$$
$$p(i \,|\, i, \texttt{mode}) = pq_{\texttt{mode}} + (1-p)(1 - q_{\texttt{mode}})$$
$$p(i-1 \,|\, i, \texttt{mode}) = (1-p)q_{\texttt{mode}}$$

- For an empty system:

$$p(1 \,|\, 0, \texttt{mode}) = p, \quad p(0 \,|\, 0, \texttt{mode}) = 1 - p$$

- For a full system:

$$P(M-1 \,|\, M, \texttt{mode}) = (1-p)q_{\texttt{mode}}, \quad P(M \,|\, \texttt{mode}) = 1 - (1-p)q_{\texttt{mode}}$$

Rewards

Suppose that there are i items in the system and mode of operation $\texttt{mode}$ is selected.

- If $0 \leq i \leq M$:

$$r(i, \texttt{mode}) = -(h(i) + c_{\texttt{mode}})$$

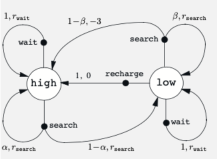
- For a full system:

$$r(M, \texttt{mode}) = -(h(M) + c_{\texttt{mode}} + p(1 - q_{\texttt{mode}})c_{\texttt{loss}}).$$

Episodic and Continuing Tasks

- In an **episodic task**, runs terminate after a finite number of steps with probability 1.

- In a **continuing task**, runs never terminate, and the agent operates in the environment forever.



Terminal States

- In an episodic task, a **terminal state** is a state s such that:

$$p(s \mid s, a) = 1 \text{ and } r(s, s, a) = 0$$

  for every action a.
- For any sequence of actions, a terminal state is eventually reached with probability one.
- We let T represent the (random) time a terminal state is first reached.

Total Return for Episodic Tasks

- The **total return** accumulated in a run of an episodic task is:

$$\sum_{t=0}^{T} R_{t+1}$$

- This does not work for continuing tasks, since the sum would be infinite!

Total Return for Continuing Tasks

- The total reward for a continuing task is defined as:

$$\sum_{t=0}^{\infty} \gamma^t R_{t+1}$$

- The number γ is called **discount factor**, and we assume $0 < \gamma < 1$. This definition is motivated by the concept of *net present value* from economics.
- Under assumption of finiteness of states and actions, rewards are uniformly bounded, so the series always converges.

Unified Notation for Episodic and Continuing Tasks

We will use, both for episodic and continuing tasks, the following notation for total returns:

$$\sum_{t=0}^{T} \gamma^t R_{t+1}$$

- For *episodic tasks*, we require $P(T < \infty) = 1$ and $0 < \gamma \leq 1$.
- For *continuing tasks*, we assume $P(T = \infty) = 1$ and $0 < \gamma < 1$.

Policies

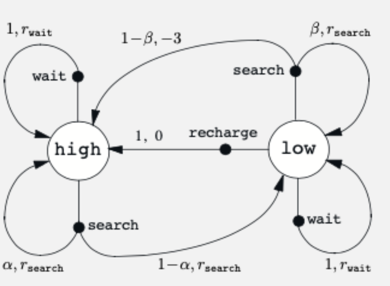
- A **policy** is a function that maps every state s to a probability distribution on the set of allowed actions $\mathscr{A}(s)$.
- We denote by $\pi(a \mid s)$ the probability that action $a \in \mathscr{A}(s)$ is selected when in state s:

$$\mathbb{P}(A_t = a \mid S_t = s) = \pi(a \mid s)$$

- The state sequence $S_0, S_1, \ldots$ is a Markov Chain with transition probability matrix

$$P_{ij}^{(\pi)} = \sum_{a \in \mathscr{A}(s_i)} \pi(a \mid s_i) p(s_j \mid s_i, a)$$

Example Policy for the Recycling Robot



Policy:

$$\pi(\texttt{search}\,|\,\texttt{high}) = 2/3$$
$$\pi(\texttt{wait}\,|\,\texttt{high}) = 1/3$$

$$\pi(\texttt{search}\,|\,\texttt{low}) = 1/5$$
$$\pi(\texttt{wait}\,|\,\texttt{low}) = 1/5$$
$$\pi(\texttt{recharge}\,|\,\texttt{low}) = 3/5$$

$$\mathbb{P}(S_{t+1} = \texttt{high}\,|\,S_t = \texttt{low}) = \frac{1}{5}(1-\beta) + \frac{1}{5}\cdot 0 + \frac{3}{5}\cdot 1$$

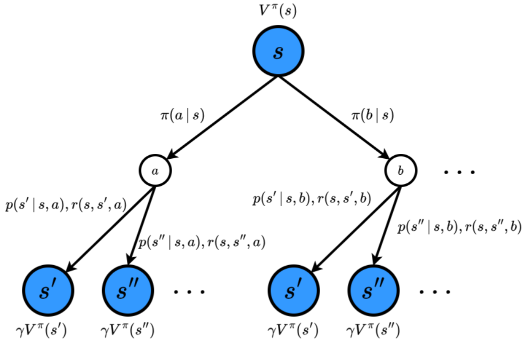
Value Function Associated to a Policy

- Suppose a policy π is chosen.
- Then, the stochatic evolution of the random sequence $\{(S_t, A_t, R_t)\}$ is completely determined (once the distribution of S_0 is chosen).
- We use the symbols $\mathbb{P}_\pi$ and $\mathbb{E}_\pi$ to denote, respectively, the probability measure and expected value operator associated with policy π.
- Then, the **state value function** associated to π is defined as:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_{t+1} \,|\, S_0 = s \right]$$

Bellman Equation for a Policy

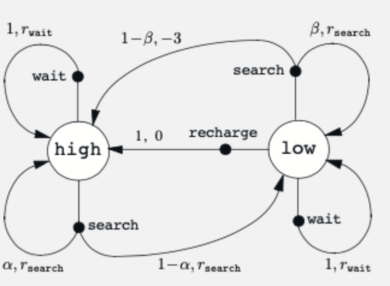
Backup diagram:



$$V^\pi(s) = \sum_{s' \in \mathscr{S}} \sum_{a \in \mathscr{A}(s)} \pi(a|s) p(s'|s,a) (r(s,s',a) + \gamma V^\pi(s'))$$

For episodic tasks, set $V(s) = 0$ if s is a terminal state.

Value Function Computation Example



Policy:

$$\pi(\texttt{search} \mid \texttt{high}) = 2/3$$
$$\pi(\texttt{wait} \mid \texttt{high}) = 1/3$$

$$\pi(\texttt{search} \mid \texttt{low}) = 1/5$$
$$\pi(\texttt{wait} \mid \texttt{low}) = 1/5$$
$$\pi(\texttt{recharge} \mid \texttt{low}) = 3/5$$

$$V^\pi(\texttt{high}) = \frac{2}{3}\left[\alpha(r_{\texttt{search}} + \gamma V^\pi(\texttt{high})) + (1-\alpha)(r_{\texttt{search}} + \gamma V^\pi(\texttt{low})\right] + \frac{1}{3}(r_{\texttt{wait}} + \gamma V^\pi(\texttt{high}))$$

Linear System for the Value Function

$$
\begin{aligned}
V(\mathtt{h}) &= \frac{2}{3} r_{\mathtt{s}} + \frac{1}{3} r_{\mathtt{w}} + \gamma \left[\left(\frac{2}{3}\alpha + \frac{1}{3} \right) V(\mathtt{h}) + \frac{2}{3}(1-\alpha) V(\mathtt{l}) \right] \\
V(\mathtt{l}) &= -\frac{3}{5}(1-\beta) + \frac{1}{5}\beta r_{\mathtt{s}} + \frac{1}{5} r_{\mathtt{w}} \\
&\quad + \gamma \left[\left(\frac{1}{5}(1-\beta) + \frac{3}{5} \right) V(\mathtt{h}) + \left(\frac{1}{5}\beta + \frac{1}{5} \right) V(\mathtt{l}) \right]
\end{aligned}
$$

This system has the form:

$$V = b + \gamma Q V$$

where Q is a transition probability matrix (entries are non-negative and rows add to 1).

Solution of the Linear System

$$V = b + \gamma QV$$

- **Linear algebra solution**. Write the system in the form:

  $$(I - \gamma Q)V = b$$

  Solve it using numerical linear algebra. For example, use an LU decomposition.
- **Gauss-Jacobi Iteration**. Choose V_0 arbitrarily and iterate:

  $$V_{n+1} = b + \gamma QV_n$$

  Then,

  $$V = \lim_{n\to\infty} V_n$$

  exists and is the solution of the system.

Jacobi Iteration — Numerical Example

Parameters:

$$\alpha = 0.8, \quad \beta = 0.3, \quad r_{\texttt{search}} = 15, \quad r_{\texttt{wait}} = 10, \quad \gamma = 0.9$$

$$b = \begin{bmatrix} 13.33 \\ 2.48 \end{bmatrix} \qquad Q = \begin{bmatrix} 0.87 & 0.13 \\ 0.74 & 0.26 \end{bmatrix}$$

Gauss-Jacobi Iteration:

n	$V_n(\texttt{high})$	$V_n(\texttt{low})$
0	0	0
30	113.68382504	101.43401316
60	118.42373411	106.17392222
90	118.62466434	106.37485246
120	118.63318201	106.38337012
150	118.63354308	106.38373119
180	118.63355839	106.38374650

Gauss-Seidel Iteration

$$V = b + \gamma Q V$$

Computational implementation:

```
for i in range(n):
    Vnew[i] = b[i] + gamma*sum(Q[i,j]*V[j] for j in range(n))
```

It is *very tempting* to write this code as:

```
for i in range(n):
    V[i] = b[i] + gamma*sum(Q[i,j]*V[j] for j in range(n))
```

The second version

- Saves memory.
- Uses already updated values of V in each iteration.

Gauss-Seidel Iteration

Parameters:

$$\alpha = 0.8, \quad \beta = 0.3, \quad r_{\texttt{search}} = 15, \quad r_{\texttt{wait}} = 10, \quad \gamma = 0.9$$

$$b = \begin{bmatrix} 13.33 \\ 2.48 \end{bmatrix} \qquad Q = \begin{bmatrix} 0.87 & 0.13 \\ 0.74 & 0.26 \end{bmatrix}$$

Gauss-Jacobi Iteration:

n	$V_n(\texttt{high})$	$V_n(\texttt{low})$
0	0	0
30	115.21947040	103.29702236
60	118.53517952	106.29480087
90	118.63072419	106.38118412
120	118.63347738	106.38367332
150	118.63355671	106.38374505
180	118.63355900	106.38374712

Computation of Value Functions

- When computing value functions, we prefer iterative algorithms to "exact" linear algebra methods, such as LU decomposition.
- Iterative methods are easier to code and verify, and save memory.
- In general, *we only need approximate solutions*. In many cases, we do just a few iterations of Gauss-Jacobi or Gauss-Seidel.
- All things being equal, we prefer Gauss-Seidel, since it is easier to code, saves memory and converges faster.
- *However*, in parallel architectures, Gauss-Jacobi may be an efficient alternative.

Optimization

- The goal of RL is to figure out how the agent should choose actions to maximize the expected reward.
- We define the **optimal state value function** by:

$$V^\star(s) = \max_\pi V^\pi(s)$$

  where π ranges over all admissible policies.
- A policy $\pi^\star$ is *optimal* if:

$$V^{\pi^\star}(s) = V^\star(s) \quad \text{for all } s \in \mathscr{S}.$$

  or, equivalently:

$$V^{\pi^\star}(s) \leq V^\pi \quad \text{for all } s \in \mathscr{S},$$

  for all admissible policies π.

- In the next session, we will start to learn methods to find the optimal value function and optimal policy.
- In realistic cases, the best we can hope is to find approximations to $V^\star$ and $\pi^\star$. These are called *nearly optimal policies*.