Markov Chain: Robot Navigation Example

Scenario: A robot moves in a straight hallway with 3 rooms: A, B, and C.

- The robot moves randomly between rooms.
- Its next position depends only on its current location.

Markov Chain: Robot Navigation Example

Scenario: A robot moves in a straight hallway with 3 rooms: A, B, and C.

- The robot moves randomly between rooms.
- Its next position depends only on its current location.

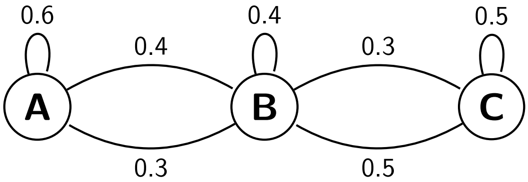
States: $S = \{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$

Transition Matrix:

$$P = \begin{bmatrix} 0.6 & 0.4 & 0.0 \\ 0.3 & 0.4 & 0.3 \\ 0.0 & 0.5 & 0.5 \end{bmatrix}$$

- From A: 60% stay in A, 40% move to B
- From B: 30% to A, 30% to C, 40% stay
- From C: 50% stay, 50% go to B

State Diagram: Robot Movement



Why is this a Markov Chain?

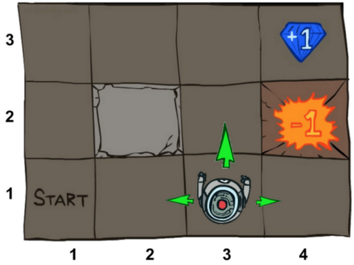
- The robot's next position depends only on where it is now — not where it was before.
- This satisfies the **Markov Property**.
- There are no decisions or rewards — only probabilistic state transitions.
- Later, we will add:
  - **Actions** the robot can choose
  - **Rewards** for moving to certain rooms
- This leads us from a Markov Chain to a **Markov Decision Process (MDP)**.

Recap: The Goal of Reinforcement Learning

- **Finite horizon case:** T is finite.
- **Infinite horizon case:** $T = \infty$.
- The total reward is often **discounted**:

$$\sum_t \gamma^t r(s_t, a_t), \quad \text{where } 0 < \gamma \leq 1$$

- **Goal:** Find a policy $\pi_\theta(a \mid s)$ that maximizes expected cumulative reward.
- Example shown: *Grid World environment*

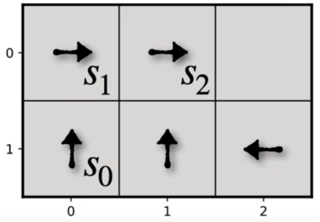


Intro to the Bellman Equation

- The **value function** $v^\pi(s_0)$ gives the expected cumulative reward starting from state s_0, following policy π.
- It's defined as:

$$v^\pi(s_0) = \mathbb{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots\right]$$

- $\gamma \in (0,1)$ is the **discount factor** that reduces the weight of future rewards.
- This sum grows as the trajectory gets longer $\rightarrow$ computing it exactly can become tedious!



Simplifying the Value Function with the Bellman Equation

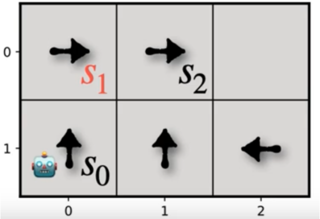
- We can rewrite the value function recursively using the Bellman equation:

$$v^{\pi}(s_0) = \mathbb{E}[R(s_0)] + \gamma\mathbb{E}[R(s_1) + \gamma R(s_2) + \dots]$$

- That second part is just the value of the next state!

$$v^{\pi}(s_0) = R(s_0) + \gamma v^{\pi}(s_1)$$

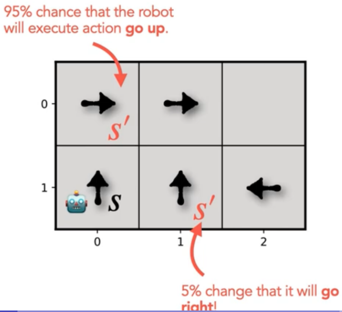
- This is the foundation of dynamic programming in RL — break long-term values into simpler recursive steps.



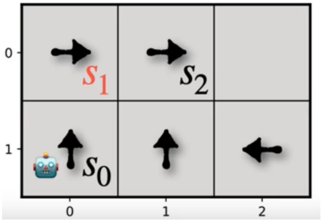
Generalizing the Bellman Equation

- So far: we assumed that the next state is deterministic.
- But in many environments, the next state is **stochastic**.
- Example: Robot at state s tries to go up...
  - 95% chance it goes up $\rightarrow$ state s_1
  - 5% chance it slips right $\rightarrow$ state s_2
- We must now average over all possible next states:

$$v^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s))\, v^{\pi}(s')$$



Bellman Equations for a Simple Environment



$$
\begin{cases}
v^\pi((0,0)) = R((0,0)) + \gamma v^\pi((0,1)) \\
v^\pi((0,1)) = R((0,1)) + \gamma v^\pi((0,2)) \\
v^\pi((0,2)) = R((0,2)) \\
v^\pi((1,0)) = R((1,0)) + \gamma v^\pi((0,0)) \\
v^\pi((1,1)) = R((1,1)) + \gamma v^\pi((0,1)) \\
v^\pi((1,2)) = R((1,2)) + \gamma v^\pi((1,1))
\end{cases}
$$

Solve the Problem with Python

- Solve the equations with python:

$$\gamma = 0.9$$

$$\begin{cases} v^{\pi}((0,0)) = R((0,0)) + \gamma v^{\pi}((0,1)) \\ v^{\pi}((0,1)) = R((0,1)) + \gamma v^{\pi}((0,2)) \\ v^{\pi}((0,2)) = R((0,2)) \\ v^{\pi}((1,0)) = R((1,0)) + \gamma v^{\pi}((0,0)) \\ v^{\pi}((1,1)) = R((1,1)) + \gamma v^{\pi}((0,1)) \\ v^{\pi}((1,2)) = R((1,2)) + \gamma v^{\pi}((1,1)) \end{cases}$$

Solve the Problem with Python

- Consider a 4x6 map
- Write a function which can generate a random policy
- The robot will stop at the goal position
- When the robot reach the boundary, it will go back
- Write a script to get the value function of the random policy
- The most import thing is to create A and b

Value Functions

- **Return:** G_0 is the total discounted reward from time step 0 onward:

$$G_0 = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots = \sum_{t=0}^{\infty} \gamma^t R(s_t)$$

- **State-value function** $v^\pi(s)$:

$$v^\pi(s) = \mathbb{E}_\pi \left[G_0 \mid s_0 = s \right]$$

- **Action-value function** $q^\pi(s, a)$:

$$q^\pi(s, a) = \mathbb{E}_\pi \left[G_0 \mid s_0 = s, a_0 = a \right]$$

- **Why do we need both?**
- **What does it mean when** $q^\pi(s, a) > v^\pi(s)$

Optimal Value Functions

- So far, we evaluated value functions under a given policy π.
- Now, we define the **optimal value functions**, which represent the best possible return:

$$v^*(s) = \max_\pi v^\pi(s)$$

$$q^*(s,a) = \max_\pi q^\pi(s,a)$$

- These functions represent the maximum expected return achievable from:
  - $v^*(s)$: starting at state s and acting optimally
  - $q^*(s,a)$: starting at state s, taking action a, and acting optimally afterward

Relationship Between v^* and q^*

- Once we know the optimal action-value function $q^*(s,a)$, we can recover the optimal state-value function:

$$v^*(s) = \max_a q^*(s,a)$$

- And the **optimal policy** is to choose the action that maximizes q^*:

$$\pi^*(s) = \arg\max_a q^*(s,a)$$

- This gives us a way to derive the best behavior from the learned value functions.

From $v^*(s)$ to $q^*(s,a)$ and Optimal Policy

- If we already know the optimal state-value function $v^*(s)$, we can compute:

$$q^*(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a)\, v^*(s')$$

- Then, we derive the optimal policy by choosing the action that maximizes q^*:

$$\pi^*(s) = \arg\max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s,a)\, v^*(s') \right]$$

- **Summary:**
  - $v^*(s)$ gives us the best possible state values.
  - Using the environment's transition model P, we can compute the best actions.

The Bellman Operator

- The **Bellman operator** T^π applies to a value function v under policy π:

$$(T^\pi v)(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s))\, v(s')$$

- This operator returns a new value function, one step closer to the true v^π.
- Similarly, for the optimal case, we define the **optimal Bellman operator** T^*:

$$(T^* v)(s) = \max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a)\, v(s') \right]$$

- Fixed points:
  - v^π is a fixed point of T^π
  - v^* is a fixed point of T^*

Contraction Property of the Bellman Operator

- The Bellman operator is a **contraction mapping** under the max norm:

$$\|Tv_1 - Tv_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$$

  where $0 < \gamma < 1$
- This means:
  - Repeatedly applying the Bellman operator brings value functions closer to the fixed point.
  - Value iteration converges to v^*!
- **Banach Fixed-Point Theorem:**
  - Every contraction mapping has a unique fixed point.
  - So the Bellman operator guarantees convergence to v^*.

Goal: Prove Bellman Operator is a Contraction

- Let v_1 and v_2 be two value functions.
- Define the Bellman operator for any policy π:

$$(T^\pi v)(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s))\, v(s')$$

- We want to show:

$$\|T^\pi v_1 - T^\pi v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$$

- This proves that T^π is a **contraction mapping** with factor $\gamma < 1$.

Proof: Bellman Operator is a Contraction

- Consider:

$$|(T^\pi v_1)(s) - (T^\pi v_2)(s)| = \left|\gamma \sum_{s'} P(s' \mid s, \pi(s)) \left(v_1(s') - v_2(s')\right)\right|$$

- By triangle inequality and properties of probabilities:

$$\leq \gamma \sum_{s'} P(s' \mid s, \pi(s)) \left|v_1(s') - v_2(s')\right|$$

- Since probabilities sum to 1 and max norm bounds all components:

$$\leq \gamma \|v_1 - v_2\|_\infty$$

- Taking the max over all s:

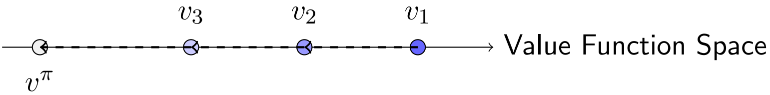
$$\|T^\pi v_1 - T^\pi v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$$

- **Conclusion:** T^π is a contraction mapping.

Visual Intuition: Bellman Operator as a Contraction

- Each time we apply T^π, the value function moves closer to the true v^π.
- This is guaranteed by the contraction property:

$$\|T^\pi v - v^\pi\|_\infty \leq \gamma \|v - v^\pi\|_\infty$$



- The Bellman operator "pulls" any value function toward the fixed point v^π.
- Repeated application: $v_{k+1} = T^\pi v_k$

Policy Evaluation

- Given a policy π, we compute its value function $v^{\pi}(s)$.
- This is done by solving the Bellman equation for v^{π}:

$$v^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) v^{\pi}(s')$$

- This gives us an accurate estimate of how good each state is under the current policy.
- In practice, we often compute v^{π} iteratively:

$$v_{k+1}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) v_k(s')$$

Policy Improvement

- Once we have v^π, we can improve the policy by choosing better actions:

$$\pi'(s) = \arg\max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) v^\pi(s') \right]$$

- This gives us a new policy π' that is at least as good as the old one:

$$v^{\pi'}(s) \geq v^\pi(s) \quad \forall s$$

- Repeating evaluation and improvement leads to the **Policy Iteration** algorithm:
  1. Evaluate v^π
  2. Improve policy $\rightarrow$ get π'
  3. Repeat until policy stops changing

Optimal Bellman Operator

- Define the **optimal Bellman operator** T^*:

$$(T^* v)(s) = \max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) v(s') \right]$$

- This operator selects the action that yields the highest expected return from state s.
- Goal: Prove that T^* is a **contraction mapping** under the max norm:

$$\|T^* v_1 - T^* v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$$

- If true, then T^* has a unique fixed point v^*, and value iteration converges to it!

Proof: Expand and Bound

- Start by expanding the difference at each state:

$$|(T^* v_1)(s) - (T^* v_2)(s)|$$

- Use the definition of T^*:

$$\begin{aligned} &|(T^* v_1)(s) - (T^* v_2)(s)| \\ &= \left| \max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) v_1(s') \right] \right. \\ &\quad \left. - \max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) v_2(s') \right] \right| \end{aligned}$$

- Apply the inequality:

$$|\max f_1 - \max f_2| \leq \max |f_1 - f_2|$$

- So:

$$\leq \max_a \left| \gamma \sum_{s'} P(s' \mid s, a)(v_1(s') - v_2(s')) \right|$$

Proof: Apply Norm and Finish

- Pull out γ, apply triangle inequality:

$$\leq \gamma \max_a \sum_{s'} P(s' \mid s, a) |v_1(s') - v_2(s')|$$

- Use the max norm definition:

$$|v_1(s') - v_2(s')| \leq \|v_1 - v_2\|_\infty$$

- Since probabilities sum to 1:

$$\sum_{s'} P(s' \mid s, a) \leq 1 \Rightarrow \leq \gamma \|v_1 - v_2\|_\infty$$

- Take max over s:

$$\|T^* v_1 - T^* v_2\|_\infty \leq \gamma \|v_1 - v_2\|_\infty$$

- **Conclusion:** T^* is a contraction mapping!

Value Iteration

- Instead of evaluating a policy until convergence, we can directly compute v^* using:

$$v_{k+1}(s) = \max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) v_k(s') \right]$$

- This update uses the **optimal Bellman operator** T^*:

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

- Repeat until convergence:

$$\|v_{k+1} - v_k\|_\infty < \epsilon$$

- Once v^* is computed, extract the optimal policy:

$$\pi^*(s) = \arg\max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) v^*(s') \right]$$

Q-Value Iteration

- Q-value iteration updates the action-value function $q(s,a)$ directly:

$$q_{k+1}(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_{a'} q_k(s',a')$$

- This avoids computing $v(s)$ explicitly.
- Once $q^*(s,a)$ converges, the optimal policy is:

$$\pi^*(s) = \arg\max_{a} q^*(s,a)$$

- Advantages:
  - Works directly with actions
  - Useful when optimal value of actions is more important than states

What is Q-Learning?

- **Q-Learning** is a model-free reinforcement learning algorithm.
- It learns the optimal Q-function $q^*(s, a)$ directly by interacting with the environment.
- It is **off-policy** — it learns about the optimal policy even if actions are chosen randomly (exploration).
- No need for transition probabilities $P(s' \mid s, a)$.

Goal

Learn $q^*(s, a)$ such that:

$$\pi^*(s) = \arg\max_a q^*(s, a)$$

Q-Learning Update Rule

- At each step, observe transition: (s, a, r, s')
- Then update:

$$q(s,a) \leftarrow q(s,a) + \alpha \left[r + \gamma \max_{a'} q(s', a') - q(s,a) \right]$$

where:

  - α: learning rate
  - γ: discount factor

- This is a stochastic approximation of value iteration.
- Over time, with sufficient exploration, $q(s,a) \to q^*(s,a)$

Q-Learning: Summary Key Properties

- **Type:** Off-policy, model-free
- **Learns:** Optimal Q-function $q^*(s, a)$
- **Exploration:** Requires exploration (e.g., -greedy)
- **Convergence:** Proven to converge under certain conditions:
  - All state-action pairs are visited infinitely often
  - Learning rate α_t decays appropriately
- **Advantages:**
  - Simple and effective
  - Widely used in practice and theory

Why Fitted Q Iteration?

- Classical Q-iteration requires a table of all state-action pairs — not feasible in large or continuous spaces.
- **Fitted Q Iteration (FQI)** addresses this by:
  - Using samples: (s, a, r, s')
  - Approximating the Q-function with regression models (e.g. neural nets, decision trees)
- FQI is an example of:
  - **Batch RL**: Learns from a fixed dataset (no environment interaction)
  - **Value-based method**: Focused on learning $q(s, a)$

Fitted Q Iteration: Algorithm Overview

- Given a dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^N$
- Initialize $\hat{q}_0(s,a) = 0$ or a random regressor
- For $k = 0, 1, 2, \ldots$
  1. For each sample:
  $$y_i = r_i + \gamma \max_{a'} \hat{q}_k(s_i', a')$$
  2. Fit a regression model:
  $$\hat{q}_{k+1} \leftarrow \text{Regress } (s_i, a_i) \mapsto y_i$$
- Repeat until convergence

Why Not Fit $v(s)$ in Fitted Iteration?

- In theory, we could try to approximate $v(s)$ using:

$$v(s) = \mathbb{E}_{a \sim \pi(s)} \left[R(s) + \gamma \mathbb{E}_{s'} [v(s')] \right]$$

- We need to know or estimate the transition model $P(s' \mid s, a)$

$$\pi^*(s) = \arg\max_a \left[R(s) + \gamma \sum_{s'} P(s' \mid s, a) v^*(s') \right]$$

Why Deep Q-Learning (DQN)?

- Q-learning stores a table $q(s, a)$ — this doesn't scale to:
  - Large or continuous state spaces (e.g., raw pixels)
  - High-dimensional action spaces
- **Solution:** Use a deep neural network to approximate $q(s, a; \theta)$
- **Deep Q-Learning (DQN):**
  - Introduced by DeepMind (2015)
  - Combines Q-learning with deep learning
  - First algorithm to learn control policies directly from raw images

Deep Q-Learning (DQN): Algorithm

- Maintain a neural network $q(s, a; \theta)$
- At each time step:
  1. Observe (s, a, r, s')
  2. Compute target:

$$y = r + \gamma \max_{a'} q(s', a'; \theta^-) \quad \text{(use target network)}$$

  3. Minimize the loss:

$$\mathcal{L}(\theta) = [y - q(s, a; \theta)]^2$$

- Update θ with gradient descent
- Every few steps, update target network:

$$\theta^- \leftarrow \theta$$

Key Innovations in DQN

- **Function approximation:** Use neural nets to approximate $q(s, a)$
- **Experience Replay:**
  - Store transitions in a replay buffer
  - Sample mini-batches for training
  - Breaks correlation between samples
- **Target Network:**
  - Use a separate, slowly updated network $q(s, a; \theta^-)$ for stable targets
  - Prevents divergence during training
- **Exploration:** Often uses -greedy policy with decaying ε