CS-566 Deep Reinforcement Learning

Markov Decision Process

Nazar Khan
Department of Computer Science
University of the Punjab

RL in Daily Life
Finding a Supermarket

- New city, no map, no phone.
- You explore randomly and find a supermarket.
- You note the route, and retrace your steps home.
- Next time:
  - **Exploit:** follow the known route.
  - **Explore:** try new routes, maybe shorter.

RL Concepts in the Supermarket Story

- **Agent:** you
- **Environment:** the city
- **States:** your location at each step
- **Actions:** move left, right, forward, back
- **Trajectories:** routes you tried
- **Policy:** rule for choosing next action
- **Reward/Cost:** distance or time taken
- **Exploration vs. Exploitation:** try new vs. repeat old routes
- **Transition model:** your notebook map

RL in Daily Life
Supermarket Shopping

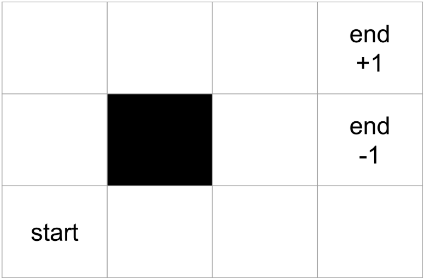
- **Agent:** The shopper.
- **Environment:** Supermarket layout.
- **State:** Items already in cart, location in store.
- **Actions:** Move to aisle, pick/skip item.
- **Reward:** Healthy, affordable, and complete shopping basket.

Sequential Decision Problems

- RL is used to solve **sequential decision problems**.
- Agent must make a **sequence of decisions** to maximize overall reward.
- Each problem involves:
  - **Agent** = solver
  - **Environment** = world/problem
- Goal: Find the **optimal policy** (sequence of actions).

Example: Grid World

- Simple environment for RL experiments.
- Start state → Goal state.
- Actions: **Up, Down, Left, Right**.
- Variations:
  - Loss squares (negative reward).
  - Wall squares (impenetrable).
- By exploring the grid, taking different actions, and recording the reward, the agent can find a route.
- When it has a route, it can try to find a shorter route to the goal.

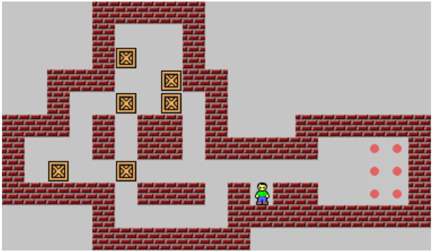


From Grids to Mazes

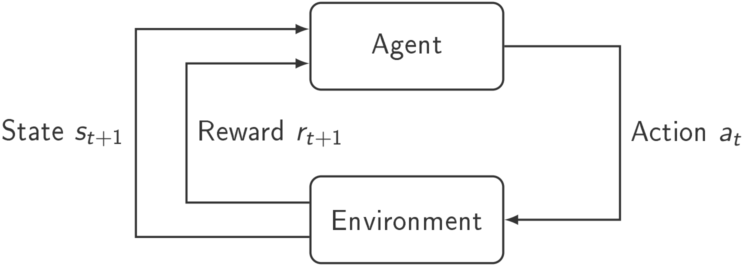
- Grid worlds are simple.
- Mazes introduce **walls and complexity**.
- Used for path-finding in:
  - Robotics trajectory planning
  - AI path-finding problems

Box-Pushing Puzzles: Sokoban

- Classic planning + learning benchmark.
- Rules:
  - Boxes can only be **pushed**, not pulled.
  - Wrong moves create dead-ends.
- Hardness:
  - Small instances solvable exactly.
  - Larger instances are NP-hard/PSPACE-hard.



Agent and Environment



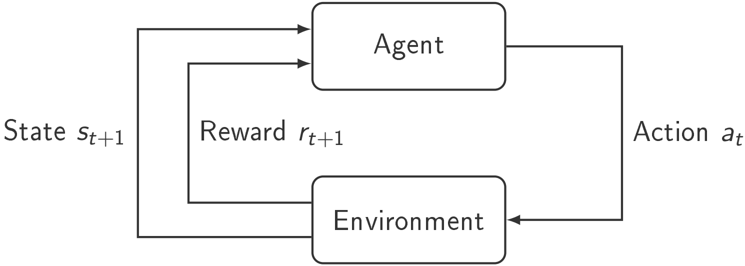
Agent and environment

- **Agent:** Learner/decision maker.
- **Environment:** Provides states, rewards, transitions.
- Agent interacts $\rightarrow$ learns optimal policy.

Tabular Value-Based RL

- Reinforcement learning finds the best **policy** to operate in an environment
- Key idea: **Agent** interacts with an **Environment**
- Environment provides feedback for agent's actions
- Feedback can in the form of positive or negative reward.
- Goal: learn a policy that maximizes long-term reward

Agent and Environment



Agent–Environment Interaction

- Environment has a state s_t
- Agent chooses an action a_t
- Transition: $s_t \to s_{t+1}$
- Reward r_{t+1} received
- Goal: find **optimal policy function** $\pi^\star(s) : s \to a$ that gives in each state s the best action a to take in that state.

Learning the Policy

- By trying different actions, agent accumulates rewards
- Learns which actions are best for each state
- Environment only provides a number (reward), not instructions
- Advantage: can generate as much experience as needed (no labeled dataset!)
- Optimal policy is learned from repeated interaction with the environment

Markov Decision Processes (MDPs)

- Framework for dealing with sequential decision problems
- Next state s_{t+1} depends only on:
  - Current state s_t
  - Current action a_t
- No dependence on history (*Markov property*)
- Enables reasoning about future using **only** present information

Formal Definition of MDP

An MDP is a 5-tuple (S, A, T_a, R_a, γ):

- S is the set of states (environment configurations)
- A is the set of actions available
- $T_a(s, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the probability that action a in state s at time t will *transition* to state s' at time $t+1$ in the environment
- $R_a(s, s')$ is the reward for transition $s \rightarrow s'$ because of action a
- $\gamma \in [0, 1]$ is a *discount* factor representing the distinction between immediate and long-term reward

State S

- Basis of every MDP: the **state** s_t at time t
- State s uniquely represents the configuration of the environment
- Examples:
  - Supermarket: current street corner
  - Chess: full board configuration
  - Robotics: joint angles and limb positions
  - Atari: all screen pixels

Deterministic vs. Stochastic Environments

- **Deterministic:** each action leads to exactly one new state
  - Gridworld, Sokoban, Chess
- **Stochastic:** the same action can lead to multiple possible outcomes
  - Robot pours water: success or spillage
  - Outcomes depend on unknown factors in environment

Action A

- In state s, the agent chooses an action a (based on policy $\pi(a|s)$)
- Action irreversibly changes the environment
- Examples:
  - Supermarket: walk East
  - Sokoban: push a box
- Possible actions differ by state (e.g., walls may block moves)

Discrete vs. Continuous Actions

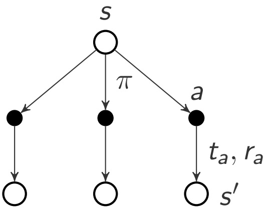
- **Discrete:** finite set of actions
  - Board games, grid navigation
- **Continuous:** actions span a range of values
  - Robot arm movements
  - Bet sizes in games
- Two types of RL algorithms:
  - *Value-based algorithms* work well for discrete action spaces
  - *Policy-based algorithms* work well for both discrete an continuous action spaces

Transition Function T_a

- Transition function $T_a(s, s')$: defines how states change after action a
- Every environment has its own transition function T_a
- Two kinds of RL:
  - **Model-free:** agent does not know T_a; learns by interaction
  - **Model-based:** agent learns its own approximation of the environment's T_a

Graph View of the State Space

- Dynamics of an MDP are modelled by transition function $T_a(\cdot)$ and reward function $R_a(\cdot)$
- The imaginary space of *all possible states* is called the *state space*
- States and actions can be seen as nodes in a *transition graph*

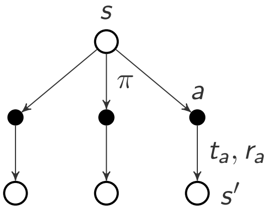


1-level transition graph for an MDP

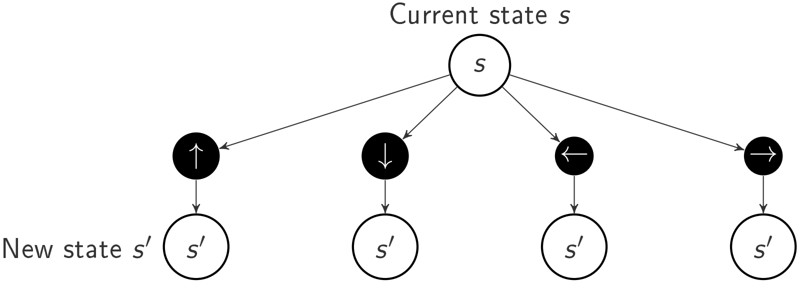
- Edges represent transitions $s \to a \to s'$
- Reward r_a is associated with each transition t_a

Graph View of the State Space

- RL is also known as learning by *trial end error*.
- *Trial:* moving **down** the tree (selecting actions)
- *Error:* propagating rewards **up** the tree (learning)

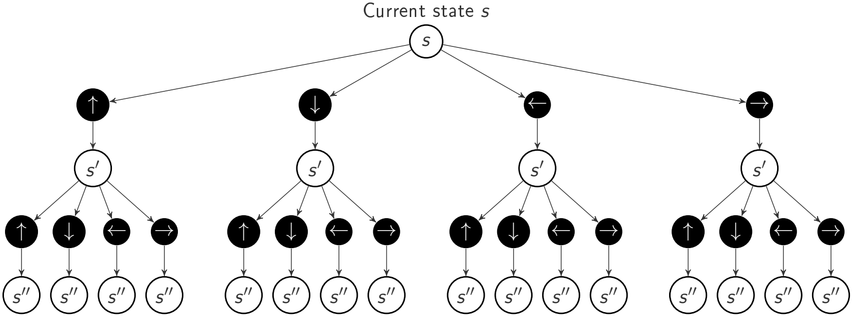


Transition Graph for Grid World



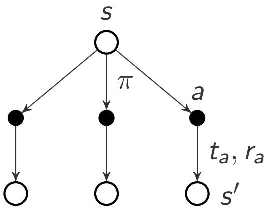
1-level transition graph for an MDP representing the Grid World

Transition Graph for Grid World

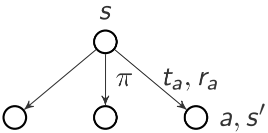


2-level transition graph for an MDP representing the Grid World

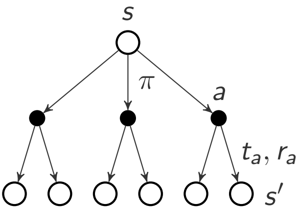
Stochastic vs. Deterministic State Spaces



Deterministic



Deterministic



Stochastic

Reward R_a

- Reward is a measure of the *quality* of a state (good or bad outcome)
- Important: we care about **sequences** of rewards
- *Return:* total cumulative reward of a sequence
- *Value function $V^\pi(s)$:* expected cumulative reward from s under policy π

Discount Factor γ

- Balances present vs. future rewards
- $\gamma < 1$: future rewards are discounted for *continuous*, never-ending tasks
- $\gamma = 1$: no discounting for *episodic* tasks that end, e.g., chess
- Most RL tasks in this course: episodic, so $\gamma = 1$

Policy π

- Policy π: rule for choosing actions
- $\pi(a|s)$: probability of taking action a in state s
- Example: tabular stochastic policy (probabilities for each action)
- Deterministic policy: $\pi(s) \rightarrow a$

Example: Stochastic vs Deterministic Policy

Deterministic Policy

s	$\pi(s)$
1	down
2	right
3	up

$\pi(s) \to a$

Stochastic Policy (table)

s	up	down	left	right
1	0.2	0.8	0.0	0.0
2	0.0	0.0	0.0	1.0
3	0.7	0.0	0.3	0.0

$\pi(a|s) =$ probability of action a in state s