18-661 Introduction to Machine Learning

Reinforcement Learning

Spring 2025

ECE – Carnegie Mellon University

Please complete course evaluations

- One bonus point for completing Faculty Course Evaluations (FCEs) and uploading the confirmation page or email to Gradescope. We appreciate your feedback on the following aspects of the class so that we can continue improving it in future offerings
  - Course content/structure
  - Our teaching
  - Homeworks and Pytorch problems
  - Any other constructive feedback or suggestions
- DO NOT upload your responses. Those should remain confidential and anonymous

Final Exam Logistics

- Final Exam Friday of next week, May 2, 1 pm-4 pm ET (10am-1pm PT, 7pm-10pm CAT) in HOA 160 (Pitt), CMR F205 (Kigali), B23 118 (SV). This is not the usual Pittsburgh lecture room.
- All topics are included, with a focus on post-midterm topics
- Mix of true/false, multiple choice, and descriptive questions. No coding questions.
- 2 double-sided, US-letter or A4 sized handwritten cheat sheets allowed
- Calculators not permitted and not useful.
- Take the practice final exam, which will be posted on Piazza soon.

Office hours will be held as normal with exceptions noted on Piazza. Recitation will also give you a chance to practice for the exam, and ask questions about the material.

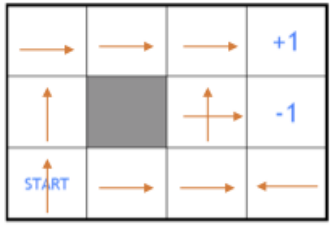
Outline

1. Online Learning Recap
2. Reinforcement Learning
3. Markov Decision Processes
4. Bellman Equations

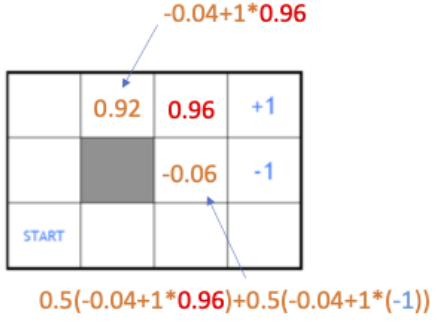
Online Learning Recap

Task 5: Reinforcement Learning

How to take the actions that maximize reward?



input:
state-action-reward trajectories



find: Value function or Q function

actions: UP, DOWN, LEFT, RIGHT

UP

80% move UP
10% move LEFT
10% move RIGHT

return: Optimal Policy π

Course Covers: Online learning and bandits, Bellman equation, Policy Evaluation, Q-learning

What Is Online Learning?

Online learning occurs when we do not have access to our entire training dataset when we start training. We consider a sequence of data and update the predictor based on the latest sample(s) in the sequence.

- Stochastic gradient descent
- Multi-armed bandits

Online learning has practical advantages:

- Helps us handle large datasets.
- Automatically adapts models to changes in the underlying process over time (e.g., house prices' relationship to square footage).

Evaluating Online Learning Models

How do we evaluate online learning models?

- Previously, we measured the loss of our model on test data–but the model changes over time! For model parameters β: $\beta_1 \to \beta_2 \to \ldots \beta_t \to \ldots$
- Define regret as the cumulative difference in loss compared to the best model in hindsight. For a sequence of data (x_t, y_t):

$$R(T) = \sum_{t=1}^{T} [l(x_t, y_t, \beta_t) - l(x_t, y_t, \beta^*)]$$

  Evaluate the loss of each sample using the model parameters at that time, compared to the best parameters in hindsight.
  - Usually, we want the regret to decay sub-linearly over time.
  - Implies that the regret per iteration decays to zero: eventually, you will recover the optimal-in-hindsight model.

Bandit Formulation

We can play multiple rounds $t = 1, 2, \ldots, T$. In each round, we select an arm i_t from a fixed set $i = 1, 2, \ldots, n$; and observe the reward $r(i_t)$ that the arm gives.

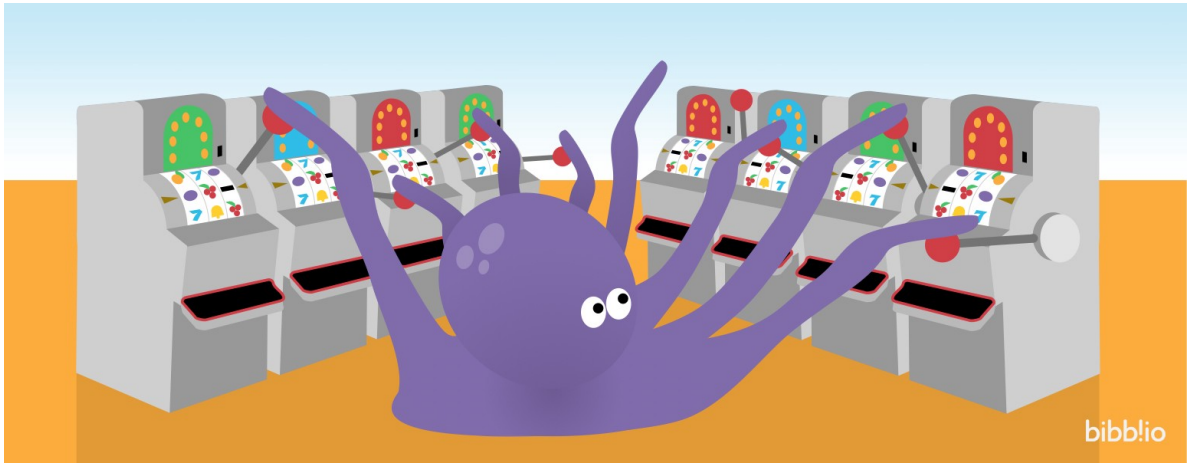
Arm 1

Arm 2

Arm 3

Objective: Maximize the total reward over time, or equivalently minimize the regret compared to the best arm in hindsight.

Exploration vs. Exploitation Tradeoff



Which arm should I play?

- Best arm observed so far? (exploitation)
- Or should I look around to try and find a better arm? (exploration)

We need both in order to maximize the total reward.

The ϵ-Greedy Algorithm

Very simple solution that is easy to implement. The idea is to exploit the best arm, but explore a random arm ϵ fraction of the time.

- Try each arm and observe the reward.
- Calculate the empirical average reward for each arm i:

$$\overline{r(i)} = \frac{\text{total reward from pulling this arm in the past}}{\text{number of times I pulled this arm}} = \frac{\sum_{t:j(t)=i} r_t}{T_i},$$

  where $j(t) = i$ indicates that arm i was played at time t, r_t is the reward, and T_i is the number of times arm i has been played.
- With probability $1 - \epsilon$, play the arm with highest $\overline{r(i)}$. Otherwise, choose an arm at random.
- Observe the reward and re-calculate $\overline{r(i)}$ for that arm.
- Repeat steps 2-4 for T rounds.

The UCB1 Algorithm

Very simple policy from Auer, Cesa-Bianchi, and Fisher (2002) with $O(\log T)$ regret. The idea is to try the best arm, where "best" includes exploration and exploitation.

- Try each arm and observe the reward.
- Calculate the UCB (upper confidence bound) index for each arm i:

$$UCB_i = \overline{r(i)} + \sqrt{\frac{2 \log T}{T_i}},$$

  where $\overline{r(i)}$ is the empirical average reward for arm i and T_i is the number of times arm i has been played.
- Play the arm with the highest UCB index.
- Observe the reward and re-calculate the UCB index for each arm.
- Repeat steps 2-4 for T rounds.

Thompson Sampling

Which arm should you pick next?

- ϵ-greedy: Best arm so far (exploit) with probability $1 - \epsilon$, otherwise random (explore).
- UCB1: Arm with the highest UCB value.

Thompson sampling instead fits a distribution to the reward parameters. For example, if r_i is Bernoulli with probability p_i of being equal to 1:

1. Define a prior $p_i \sim$beta(α_i, β_i) distribution on p_i for each arm i.
2. Sample p_i from beta(α_i, β_i) for each arm i.
3. Play the arm with highest expected reward $p_i = \mathbb{E}[r_i|p_i]$.
4. Observe the reward and update the beta(α_i, β_i) distribution for the chosen arm (use MAP!)
5. Repeat steps 2 to 4.

Other Bandit Variations

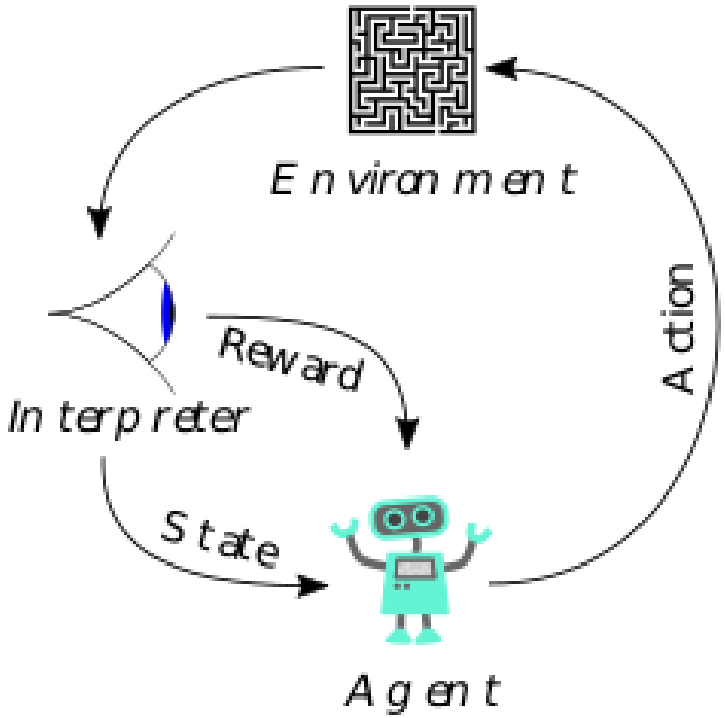
- Multi-player bandit: Many users are trying to play arms, and if multiple users play the same arm, they receive lower reward. Has important applications to wireless spectrum sharing.
- Cascading bandit: We can observe more than one arm. For instance, we can show multiple ads to a user and assume they click on the first appealing one.
- Combinatorial bandit: We need to choose a combination of arms, not a single arm. Often assumes semi-bandit feedback: we know the reward of each chosen arm, and the collective reward of their combination.
- Adversarial bandit: The rewards are chosen by an adversary who wants to fool us into making a bad decision.
- Contextual bandit: The distribution of rewards depends on some external, known context that may change over time. For instance, ads' appeal to users varies depending on the user demographics.

And many, many more...

Reinforcement Learning

Overview of Reinforcement Learning

In RL, an agent learns through interaction with the (unknown) environment.



RL is also sometimes called approximate dynamic programming. It can be viewed as a type of optimal control theory with no pre-defined model of the environment.

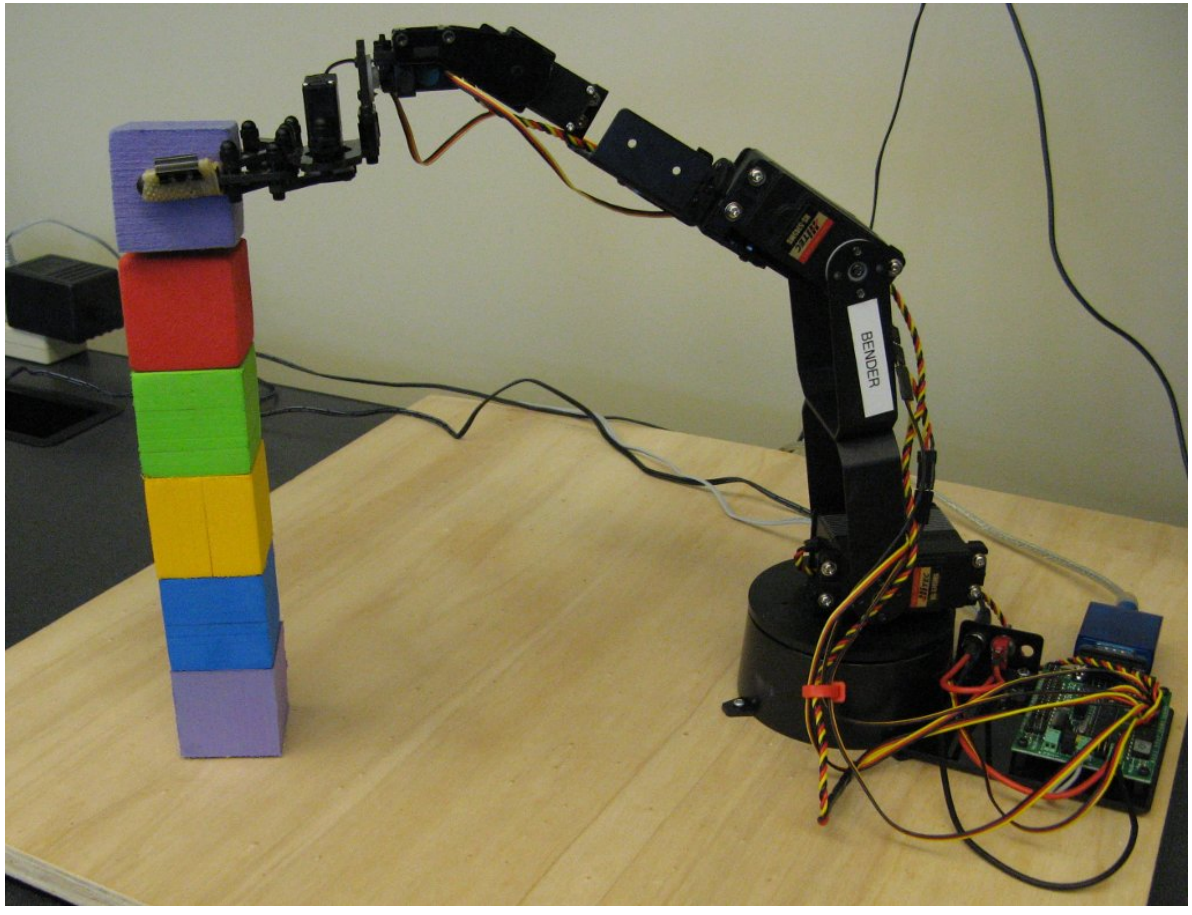
RL Applications

Reinforcement learning can be applied to many different areas.

- Robotics: in which direction and how fast should a robot arm move?
- Mobility: where should taxis go to pick up passengers?
- Transportation: when should traffic lights turn green?
- Recommendations: which news stories will users click on?
- System configuration: which parameter settings lead to the best allocation of resources?

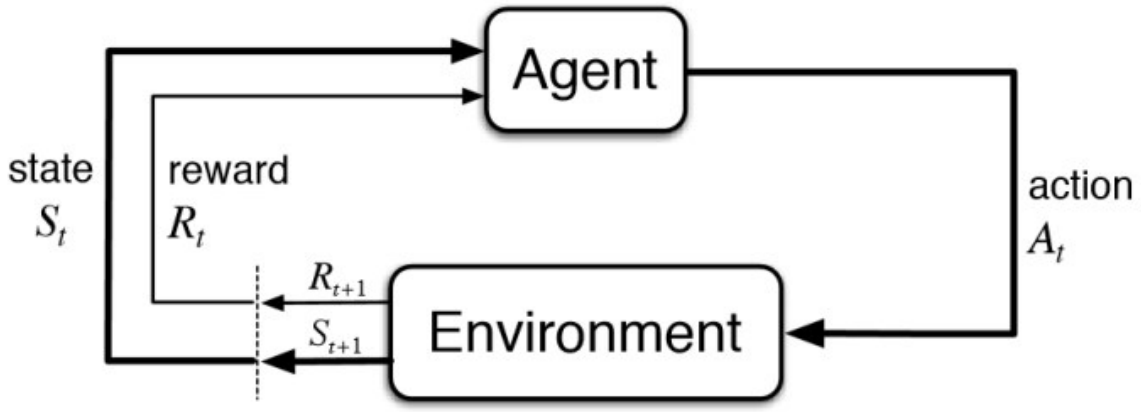
Similar to multi-armed bandits, but with a notion of state or context.

Grasping an Object



In reinforcement learning, the environment is changing in response to the agent's different actions and its current state.

RL: A Warm-up Formulation

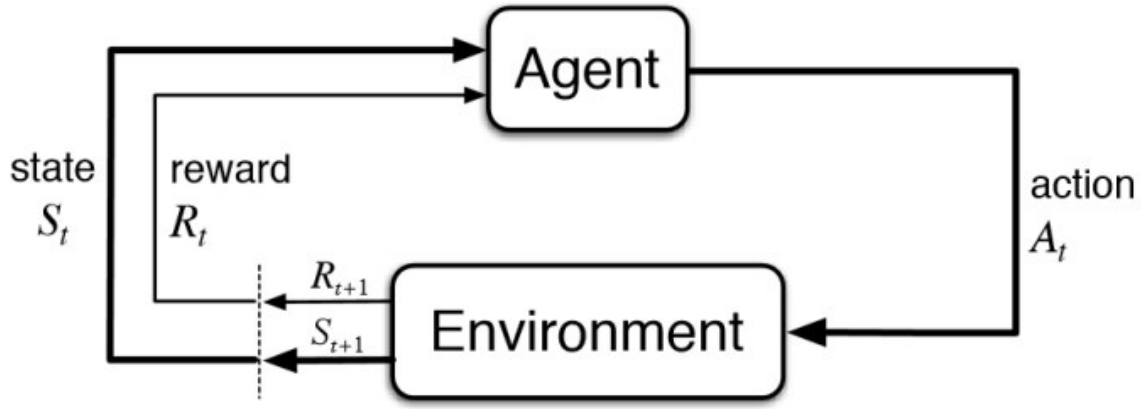


State space. Consider a sequence of time $t = 1, 2, \ldots, T$. At each time t, the agent experiences a state $s(t) \in S$ (which may be represented as a vector of real numbers). We sometimes call the state the "environment."

Examples:

- A snapshot of the current game.
- Position of objects that a robot wants to grasp.
- Number of passengers and taxis at different locations in a city.

RL: A Warm-up Formulation



Action space. At each time t, the agent takes an action $a(t)$, which is chosen from some feasible set $A(t)$.

- Direction in which a robot/taxi moves.

Reward. It then experiences a reward $r(a(t), s(t))$.

- Reward if an object is grasped/passenger is picked up.

The next state (at time $t+1$) is a (probabilistic) function of the current state and action taken: $s(t+1) \sim \sigma(a(t), s(t))$.

Objectives of RL

We choose the actions that maximize the expected total reward:

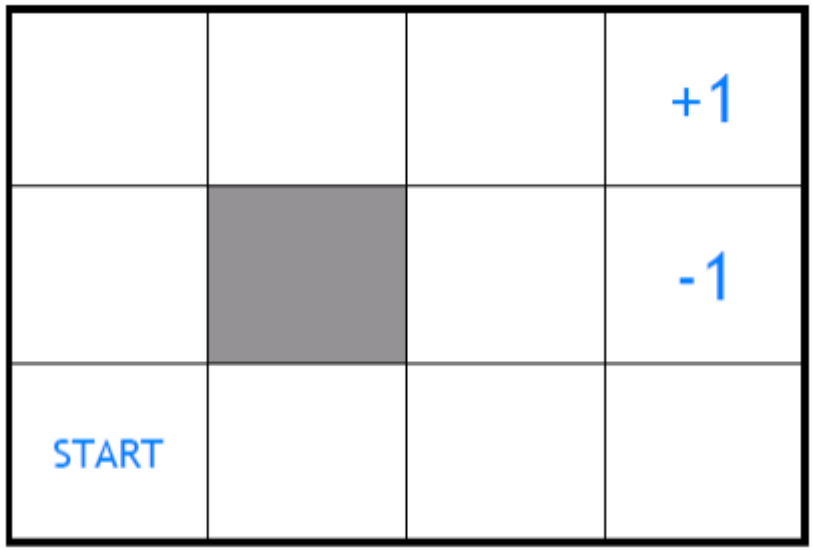
$$R(T) = \sum_{t=0}^{T} \mathbb{E}\left[r(a(t), s(t))\right], \quad R(\infty) = \sum_{t=0}^{\infty} \mathbb{E}\left[\gamma^t r(a(t), s(t))\right].$$

Infinite-horizon discounted reward. We *discount* the reward at future times by $\gamma < 1$ to ensure convergence when $T = \infty$. The expectation is taken over the probabilistic evolution of the state, and possibly the probabilistic reward function.

A policy tells us which action to take, given the current state.

- Deterministic policy: $\pi : S \to A$ maps each state s to an action a.
- Stochastic policy: $\pi(a|s)$ specifies a probability of taking each action $a \in A$ given state s. We draw an action from this probability distribution whenever we encounter state s.

Example: Robot Movements



- Actions: UP, DOWN, LEFT, RIGHT
- Reward of + 1 if we reach [4,3] and -1 if we reach [4,2]; -0.04 for taking each action
- What action should we take at state [3,3]? RIGHT

The policy is a mapping from state (i.e. where you are) to action (which direction to move).

(A Few) Key Challenges of Reinforcement Learning

- **Unknown dynamics.** The relationship of future states to past states and actions, $s(t+1) \sim \sigma(a(t), s(t))$, must be learned.
- **Partial information feedback.** The reward feedback $r(a(t), s(t))$ only applies to the action taken $a(t)$, and may itself be stochastic.
- **Delayed reward.** Since actions affect future states, they should be chosen so as to maximize the total future reward $\sum_{t=0}^{\infty} \gamma^t r(a(t), s(t))$, not just the current reward.
- **Exploration-exploitation tradeoff.** Since we have partial information, we need to take actions so as to explore their resulting rewards.

We can address these challenges by formulating RL using Markov decision processes (focusing on discrete state/action spaces).

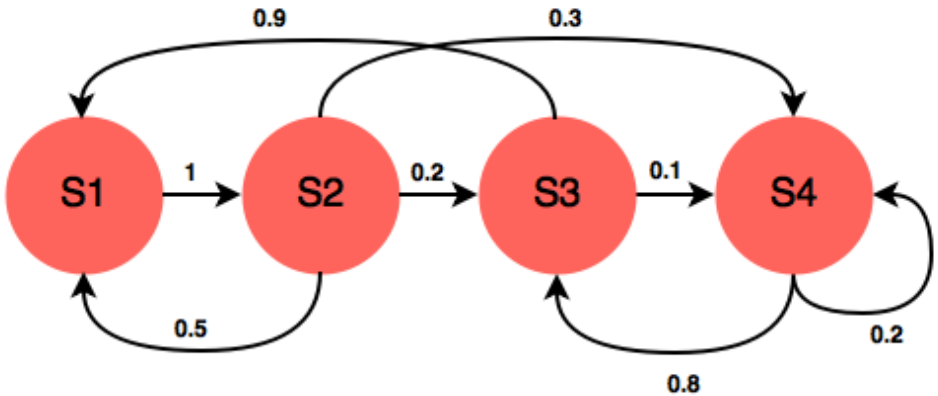
Markov Decision Processes

Finite-State, Discrete-Time Markov Chains

Suppose that $S = \{s_1, s_2, \ldots, s_n\}$: these are the possible state values. Then if $s(t) = s_i$, we can represent the probability distribution of $s(t+1)$ as:

$$\mathbb{P}\left(s(t+1) = s_j | s(t) = s_i\right) = p_{i,j},$$

where the probabilities $p_{i,j}$ are called transition probabilities.



The Markov property says that the state evolution is memoryless: the probability distribution of future states depends only on the value of the current state, not the values of previous states.

Motivating the Markov Property

The memorylessness of the Markov property significantly simplifies our predictions of future states (and thus, future rewards).

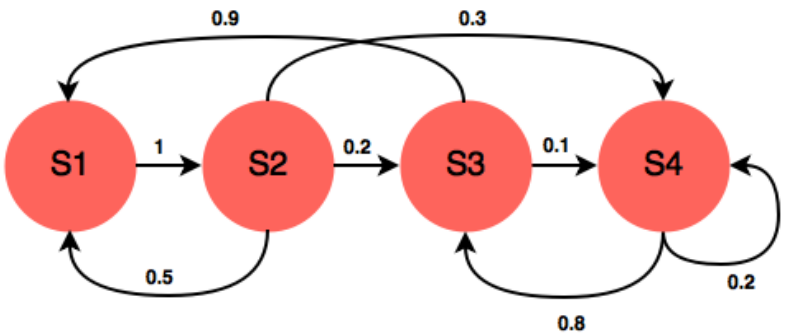
- Robotic arms: What is the probability that a block moves from point A to point B in the next 5 minutes? Does it matter where the block was 5 minutes ago? 10 minutes ago?
- Drawing Balls from an Urn: Suppose an urn has 10 red and 10 blue balls, and each day you draw 2 balls from the urn and don't replace them. What is the probability of drawing at least 1 red ball on day 5? Does this depend on the number of red/blue balls left after day 4? day 3? day 1?

Transition Matrices

A Markov chain can be represented with a transition matrix:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \ddots & \vdots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{bmatrix}$$

Each entry (i, j) is the probability of transitioning from state s_i to state s_j.



$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.2 & 0.3 \\ 0.9 & 0 & 0 & 0.1 \\ 0 & 0 & 0.8 & 0.2 \end{bmatrix}$$

Markov Reward Processes

In a Markov Reward Process, the agent receives a reward for entering each state s_i:

$$r(s_i) \in \mathbb{R}$$

The state-value function of an MRP is the expected return starting from state s:

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s(t)\right) | s(0) = s\right].$$

where the discount factor $\gamma \in [0, 1]$ represents the present value of future rewards.

- γ close to 0 leads to "myopic" evaluation
- γ close to 1 leads to "far-sighted" evaluation

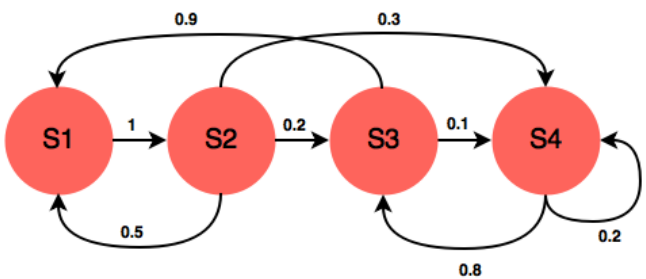
Introducing Decision Variables

Markov decision processes make state transitions dependent on user actions

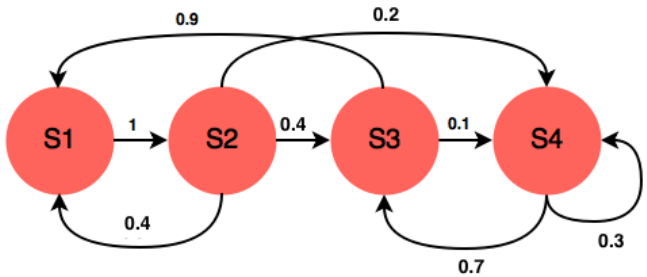
Markov processes represent transitions from state $s(t)$ to $s(t+1)$, but they do not include actions taken by the user. We now need to introduce a set of transition matrices:

$$p_{i,j} \rightarrow p^a_{i,j} = P(s(t+1) = s_j | s(t) = s_i, a(t) = a),$$

where a is a given action.



Action 1



Action 2

Markov Decision Processes in RL

- At each time t, the agent experiences a state $s(t)$. We sometimes call the state the "environment."
- At each time t, the agent takes an action $a(t)$, which is chosen from some feasible set A. It then experiences a reward $r(a(t), s(t))$.
- The next state (at time $t+1$) is a (probabilistic) function of the current state and action taken: $s(t+1) \sim \sigma(a(t), s(t))$.

The state-action relationships are just a Markov decision process!
$\sigma(a(t), s(t))$ is given by the transition probabilities $p^{a(t)}_{s(t),j}$ where j are the possible states.

Evaluating a Policy: The State-Value Function

The state-value function of a given policy $\pi : S \to A$ gives its expected future reward when starting at state s:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(s),P,r}\left[\sum_{t=0}^{\infty}\gamma^t r\left(a(t), s(t)\right)|s(0) = s\right].$$

The expectation may be taken over a stochastic policy and reward as well as the Markov decision process (MDP) of the state transitions.

- The action $a(t)$ at any time t is determined by the policy, $\pi(s(t))$.
- Due to the Markov property of the underlying MDP, the optimal policy at any time is only a function of the last observed state.

Goal: Finding an Optimal Policy

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r\left(\pi(s(t)), s(t)\right) | s(0) = s\right].$$

We want to find the optimal policy π^* that maximizes the state-value function.

From Dynamic Programming to Reinforcement Learning:

- "Planning" if we knew the underlying MDP, this falls into the realm of dynamic programming;
- "Reinforcement Learning" if we don't know the underlying MDP, this falls into the realm of reinforcement learning or approximate dynamic programming.

The Action-Value Function

Consider a variant of $V_\pi(s)$, the Q function

$$Q_\pi(a,s) = \mathbb{E}\left[r(a,s) + \underbrace{\sum_{t=1}^{\infty} \gamma^t r\left(\pi(s(t)), s(t)\right)}_{\text{use policy } \pi} | s(0) = s, a(0) = a \right]$$

The Q function gives the expected value of taking an action a at time 0, and then following a given policy π.

How is this function related to learning the optimal policy π^*?

Optimizing the Action-Value Function

- Suppose we know the "optimal" Q-function, $Q^*(a, s)$, for each action a and state s.
- Then, we maximize the state-value function by choosing the action $a^*(s)$ at state s so as to maximize $Q^*(a, s)$.

$$V^*(s) = \max_a Q^*(a, s)$$

But this gives us π^*!

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

$$V^*(s) = V_{\pi^*}(s)$$

greedification of Q^* yields the optimal policy!

- For tabular MDPs, there is always a *deterministic* optimal policy (not necessarily unique). In this lecture, we will focus on deterministic policies.

Policy Search Methods

Given the above insight, it suffices to learn either the optimal policy π^* (policy-based) or the optimal action-value function Q^* (value-based).

When we know the MDP of the state transitions, these approaches are called policy iteration and value iteration.

Two canonical problems:

- Policy evaluation (for prediction): Bellman equation
- Find the optimal policy (for control): Bellman's equation of optimality

Bellman Equations

Policy Evaluation

Evaluating a deterministic policy π boils down to a Markov reward process, where the state transition is

$$P(s(t+1) = s_j | s(t) = s_i) = P(s_j | s_i, a = \pi(s_i))$$

i.e. taking a slice of the transition kernel based on the action w.r.t. π, and similarly, the reward function is

$$r(s_j) = r(a = \pi(s_j), s_j).$$

(In fact, it is straightforward to generalize to stationary policies, too)

How do we evaluate the state-value function of a MRP?

Evaluating the State-Value Function

Let the sequence of received rewards be $r_0, r_1, \ldots$ following policy π:

$$
\begin{aligned}
V(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right] \\
&= \mathbb{E}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \ldots | s_0 = s\right] \\
&= \mathbb{E}\left[r_0 | s_0 = s\right] + \gamma \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | s_0 = s\right] \\
&= \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | s_0 = s, s_1 = s'\right] \\
&= \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots | s_1 = s'\right] \\
&= \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} V(s')
\end{aligned}
$$

The value function can be decomposed into two parts

- immediate reward $\mathbb{E}\left[r(s)\right]$
- discounted value of at the successor state $\gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} V(s')$

this defines a recursive relationship between the state values!

Bellman Equation

$$\forall s: \; V(s) = \mathbb{E}\left[r(s)\right] + \gamma \mathbb{E}_{s' \sim P(s_1 | s_0 = s)} V(s')$$

Put this in a matrix form using the probability transition matrix $\boldsymbol{P}$ induced by the policy, and

$$\boldsymbol{V} = \begin{bmatrix} V(s_1) \\ V(s_2) \\ \vdots \\ V(s_n) \end{bmatrix}, \qquad \boldsymbol{r} = \begin{bmatrix} \mathbb{E}\left[r(s_1)\right] \\ \mathbb{E}\left[r(s_2)\right] \\ \vdots \\ \mathbb{E}\left[r(s_n)\right] \end{bmatrix}$$

we have the Bellman equation:

$$\boldsymbol{V} = \boldsymbol{r} + \gamma \boldsymbol{P} \boldsymbol{V}$$

Bellman Equation

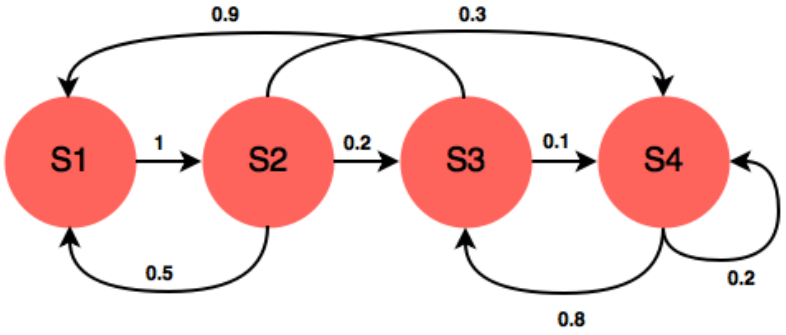
$$\boldsymbol{V} = \boldsymbol{r} + \gamma \boldsymbol{P}\boldsymbol{V}$$

- We can solve the Bellman equation directly by matrix inversion:

$$\begin{aligned}(\boldsymbol{I} - \gamma \boldsymbol{P})\boldsymbol{V} &= \boldsymbol{r} \\ \boldsymbol{V} &= (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1}\boldsymbol{r}\end{aligned}$$

- Computational complexity is $O(n^3)$ for n states
- Try iterative methods for large MRPs, e.g. dynamic programming, monte-Carlo evaluation, temporal-difference learning.

Example



$$\boldsymbol{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.2 & 0.3 \\ 0.9 & 0 & 0 & 0.1 \\ 0 & 0 & 0.8 & 0.2 \end{bmatrix}$$

Assume $r(s_1) = 1$, $r(s_2) = 1$, $r(s_3) = 1$, $r(s_4) = 10$, and $\gamma = 0.9$, then the value function is

$$\boldsymbol{V} = (\boldsymbol{I} - \gamma \boldsymbol{P})^{-1} \boldsymbol{r} = \begin{bmatrix} 21.5896 \\ 22.8773 \\ 21.2656 \\ 30.8674 \end{bmatrix}$$

Verify: $V(s_3) = r(s_3) + 0.9[0.9V(s_1) + 0.1V(s_4)]$

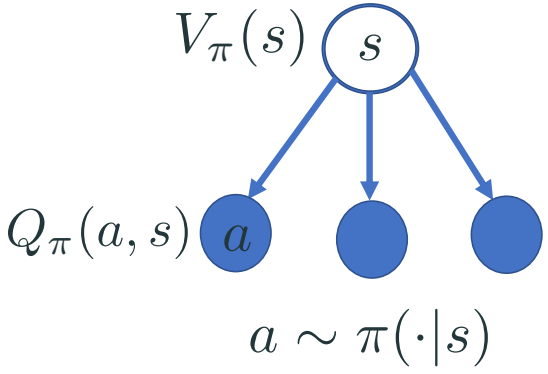
Bellman Equations for MDP

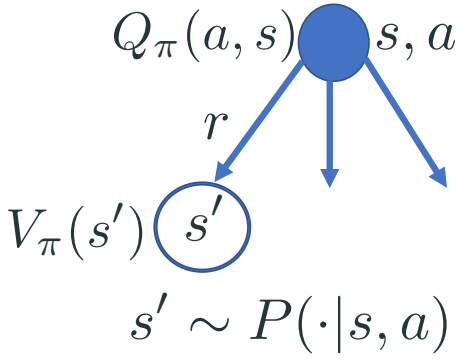
Bellman equation for state-value function:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(a, s),$$

Bellman equation for action-value function:

$$Q_\pi(a, s) = \mathbb{E}[r(a, s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s'),$$





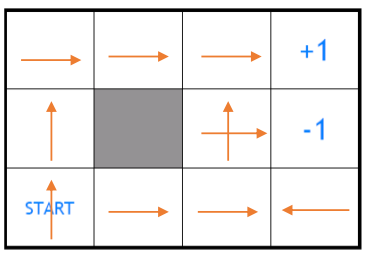
Bellman equation for state-value function:

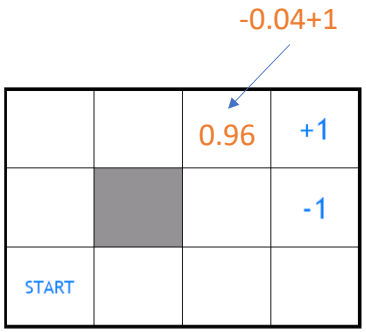
$$V_\pi(s) = \mathbb{E}[r(a, s)] + \gamma \sum \pi(a|s) \sum P(s'|s, a) V_\pi(s')$$

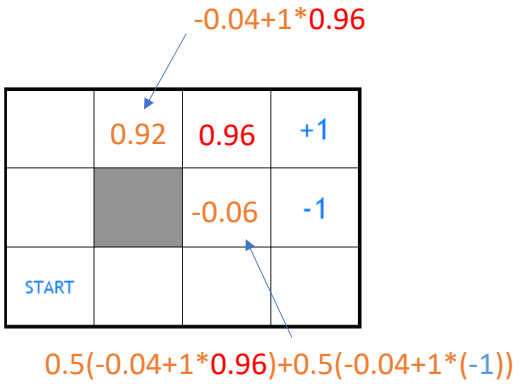
Example

Consider the same maze example as earlier, where $\gamma = 1$ and

- we receive a reward of $+1$ if we reach [4,3] and -1 if we reach [4,2], which terminate the epoch; -0.04 for taking an action;
- The policy is plotted in arrows; where if there are two arrows in a cell, both directions take equal probability.
- One can certainly use the Bellman equation to solve the value function, but for this example, we can also find it by "dynamic programming" from the end state.







Summary

You should know:

- Basic concepts regarding RL.
- What a Markov decision process is (action and state variables, transition probabilities).
- What the action-value and state-value functions are.
- Bellman equations and policy evaluation