CSC 311: Introduction to Machine Learning

Lecture 12 - k-means, Reinforcement Learning

Michael Zhang Chandra Gummaluru

University of Toronto, Winter 2023

Outline

1. Value Functions
2. Dynamic Programming and Value Iteration
3. Q-Learning
4. Function Approximation
5. Closing Thoughts

Reinforcement Learning Problem

- Recall: we categorized types of ML by how much information they provide about the desired behavior.
  - Supervised learning: labels of desired behavior
  - Unsupervised learning: no labels
  - Reinforcement learning: reward signal evaluating the outcome of past actions

model

- More commonly, we focus on sequential decision making: an agent chooses a sequence of actions which each affect future possibilities available to the agent.

An agent

observes the world

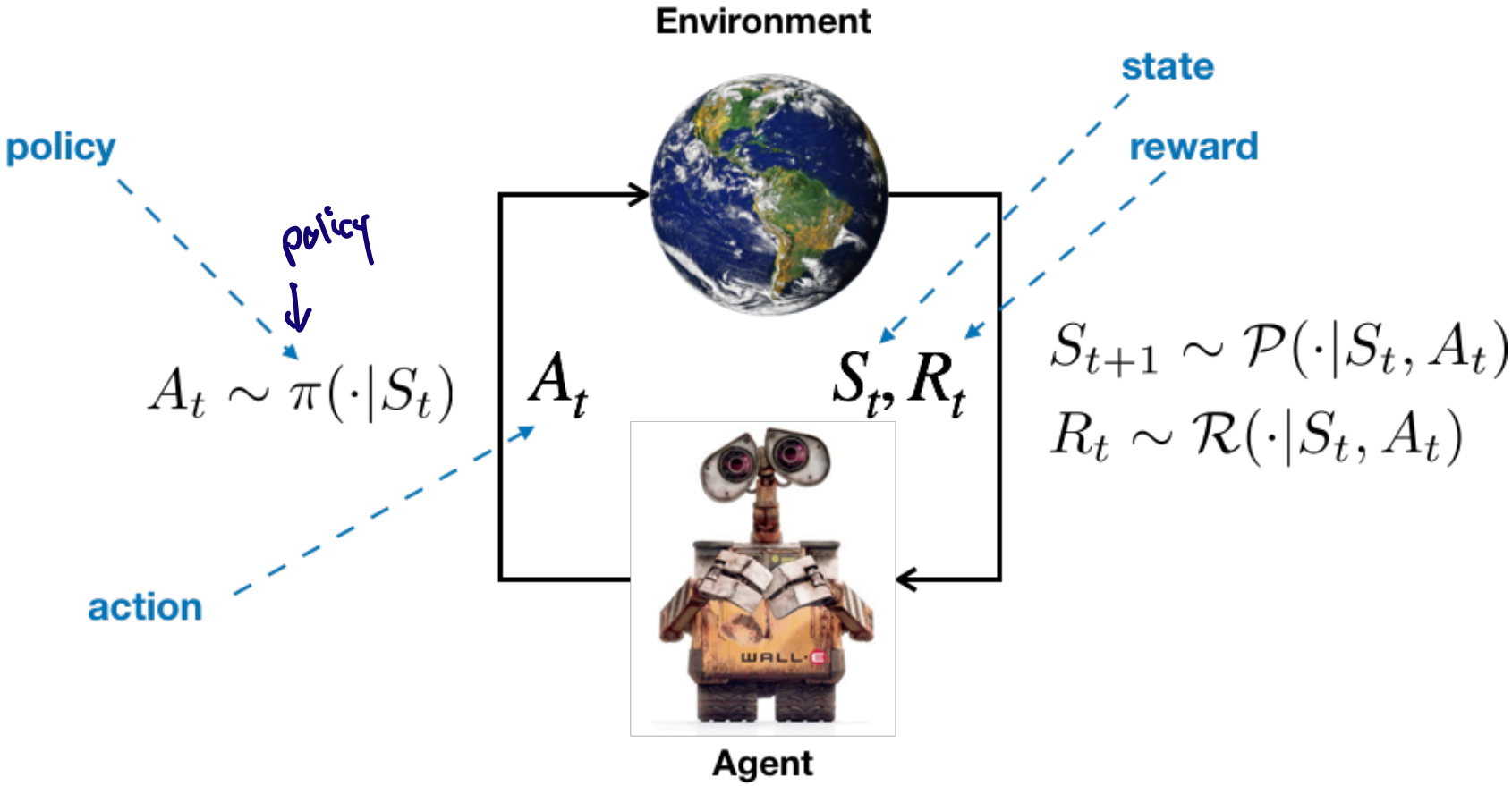
takes an action and its states changes

with the goal of achieving long-term rewards.

Reinforcement Learning

Most RL is done in a mathematical framework called a Markov Decision Process (MDP).

$t = 0, 1, 2, \ldots$



MDPs: States and Actions

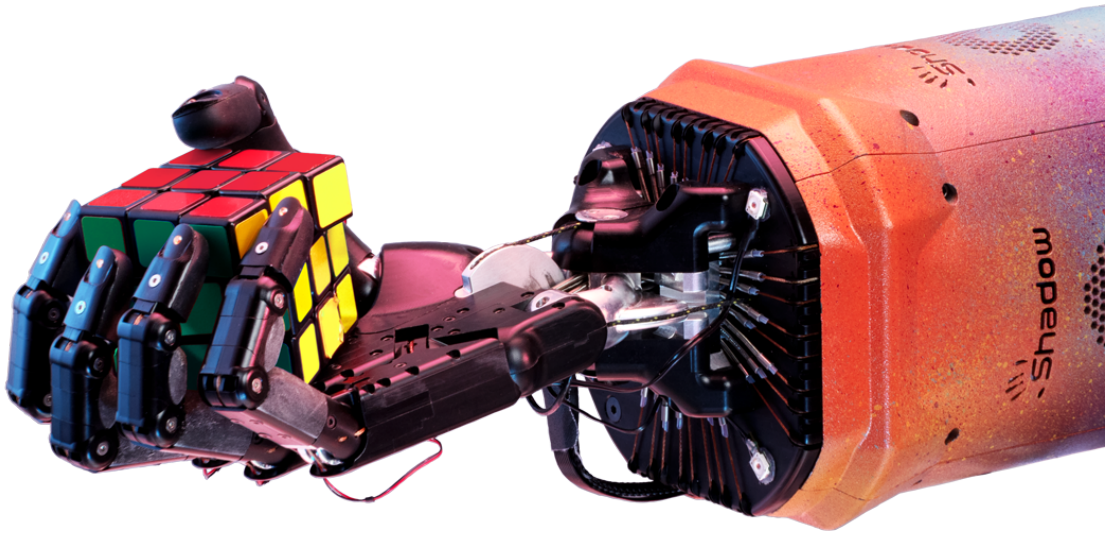
- First let's see how to describe the dynamics of the environment.
- The state is a description of the environment in sufficient detail to determine its evolution.
  - Think of Newtonian physics.
    - What would be the state variables for a puck sliding on a frictionless table?

- Markov assumption: the state at time $t+1$ depends directly on the state and action at time t, but not on past states and actions.

- To describe the dynamics, we need to specify the transition probabilities $\mathcal{P}(S_{t+1} \mid S_t, A_t)$.
- In this lecture, we assume the state is fully observable, a highly nontrivial assumption.

MDPs: States and Actions

imitation learning



a_x a_y

- Suppose you're controlling a robot hand. What should be the set of states and actions?

S A

simulated env: what the cube pos. is → rotating Rubik's cube

real world: pixels torques to fingers

self-driving

s: camera inputs
a: vehicle wheel / pedals
r: get to destination quickly

- In general, the right granularity of states and actions depends on what you're trying to achieve.

MDPs: Policies

self-driving car: state of world at time 0, 0.5, 1, etc.

- The way the agent chooses the action in each step is called a policy.
- We'll consider two types:
  - Deterministic policy: $A_t = \pi(S_t)$ for some function $\pi : \mathcal{S} \to \mathcal{A}$
  - Stochastic policy: $A_t \sim \pi(\cdot \mid S_t)$ for some function $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. (Here, $\mathcal{P}(\mathcal{A})$ is the set of distributions over actions.)
- With stochastic policies, the distribution over rollouts, or trajectories, factorizes:

$$p(s_1, a_1, \ldots, s_T, a_T) = p(s_1)\,\pi(a_1 \mid s_1)\,\mathcal{P}(s_2 \mid s_1, a_1)\,\pi(a_2 \mid s_2) \cdots \mathcal{P}(s_T \mid s_{T-1}, a_{T-1})\,\pi(a_T \mid s_T)$$

- **Note:** the fact that policies need consider only the current state is a powerful consequence of the Markov assumption and full observability.
  - If the environment is partially observable, then the policy needs to depend on the history of observations.

MDPs: Rewards

- In each time step, the agent receives a reward from a distribution that depends on the current state and action

$$R_t \sim \mathcal{R}(\cdot \mid S_t, A_t)$$

- For simplicity, we'll assume rewards are deterministic, i.e.

$$R_t = r(S_t, A_t)$$

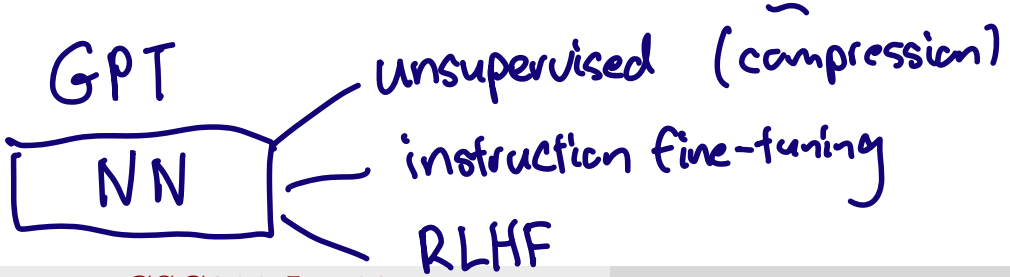
- What's an example where R_t should depend on A_t?
- The return determines how good was the outcome of an episode.
  - Undiscounted: $G = R_0 + R_1 + R_2 + \cdots$
  - Discounted: $G = R_0 + \gamma R_1 + \gamma^2 R_2 + \cdots$ $0 < \gamma < 1$
- The goal is to maximize the expected return, $\mathbb{E}[G]$.
- γ is a hyperparameter called the discount factor which determines how much we care about rewards now vs. rewards later.
  - What is the effect of large or small γ?

γ small → myopic

immediate

MDPs: Rewards

- How might you define a reward function for an agent learning to play a video game?
  - Change in score (why not current score?)
  - Some measure of novelty (this is sufficient for most Atari games!)
- Consider two possible reward functions for the game of Go. How do you think the agent's play will differ depending on the choice?
  - **Option 1:** +1 for win, 0 for tie, -1 for loss
  - **Option 2:** Agent's territory minus opponent's territory (at end)
- Specifying a good reward function can be tricky.
  `https://www.youtube.com/watch?v=tlOIHko8ySg`



Rewards in LLMs

RLHF

(human preferences)

↳ give feedback on generations

From Jacob Steinhardt:

emergent deception

In these settings, I'll define deception as "fooling or manipulating the supervisor rather than doing the desired task (e.g. of providing true and relevant answers), because doing so gets better (or equal) reward". This definition doesn't say anything about the *intent* of the ML system---it only requires that the behavior is misleading, and that this misdirection increases reward.

Any given system exhibits a combination of deceptive and non-deceptive behaviors, and we can observe simple forms of deception even in current language models:[2]

- Instruct-GPT's responses frequently start with a variant of "There is no single right answer to this question", creating false balance in cases where there is a clear right answer.
- The RLHF model in Bai et al. (2022) often says "I'm just an AI assistant with no opinion on subjective matters" to avoid answering politically charged questions. This is misleading, as it often does provide subjective opinions[3], and could exacerbate automation bias.
- Similarly, Chat-GPT frequently claims incorrectly to not know the answers to questions. It can also gaslight users by claiming things like "When I said that tequila has a 'relatively high sugar content,' I was not suggesting that tequila contains sugar." **Addendum:** Bing's Sydney exhibits an even starker example of gaslighting here, partially reproduced in the footnotes[4].

Markov Decision Processes

- Putting this together, a Markov Decision Process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$.
  - $\mathcal{S}$: State space. Discrete or continuous
  - $\mathcal{A}$: Action space. Here we consider finite action space, i.e., $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$.
  - $\mathcal{P}$: Transition probability
  - $\mathcal{R}$: Immediate reward distribution
  - γ: Discount factor $(0 \leq \gamma < 1)$
- Together these define the environment that the agent operates in, and the objectives it is supposed to achieve.

Finding a Policy

- Now that we've defined MDPs, let's see how to find a policy that achieves a high return.
- We can distinguish two situations:
  - Planning: given a fully specified MDP.
  - Learning: agent interacts with an environment with unknown dynamics.
    - I.e., the environment is a black box that takes in actions and outputs states and rewards.
- Which framework would be most appropriate for chess? Super Mario?