CSC 411: Introduction to Machine Learning

CSC 411 Lecture 21: Reinforcement Learning I

Mengye Ren and Matthew MacKay

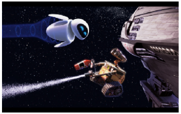
University of Toronto

Reinforcement Learning Problem

- In supervised learning, the problem is to predict an output t given an input x.
- But often the ultimate goal is not to predict, but to make decisions, i.e., take actions.
- And we need to take a sequence of actions.
- The actions have long-term consequences.

An agent

observes the world



takes an action and its states changes

with the goal of achieving long-term rewards.

Reinforcement Learning Problem: An agent continually interacts with the environment. How should it choose its actions so that its long-term rewards are maximized?

Playing Games: Atari

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Playing Games: Super Mario

https://www.youtube.com/watch?v=wfL4L_l4U9A

Making Pancakes!

https://www.youtube.com/watch?v=W_gxLKSsSIE

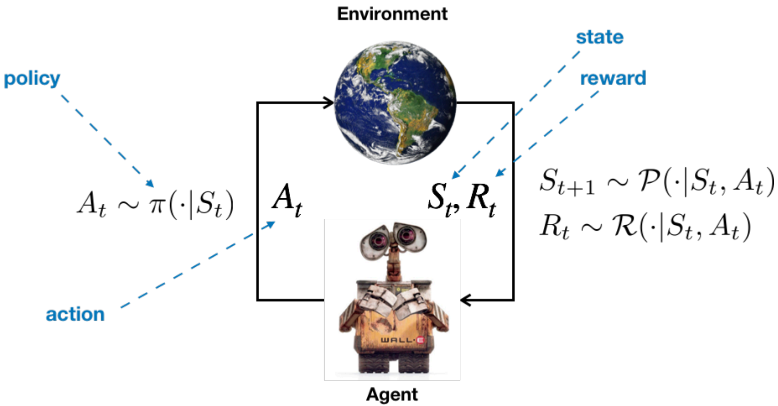
Reinforcement Learning Resources

- *Reinforcement Learning: An Introduction second edition*, Sutton & Barto Book (2018)
- Video lectures by David Silver

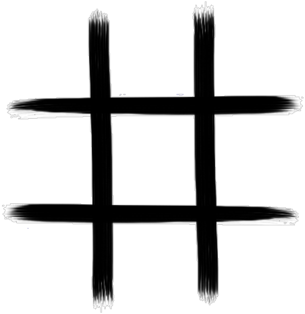
Reinforcement Learning

- Learning algorithms differ in the information available to learner
  - **Supervised**: correct outputs, e.g., class label
  - **Unsupervised**: no feedback, must construct measure of good output
  - **Reinforcement learning**: Reward (or cost)
- More realistic learning scenario:
  - Continuous stream of input information, and actions
  - Effects of action depend on state of the world
  - Obtain reward that depends on world state and actions
    - You know the reward for your action, not other actions.
    - Could be a delay between action and reward.

Reinforcement Learning

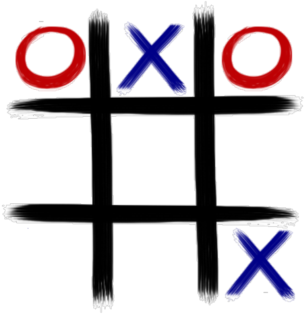


Example: Tic Tac Toe, Notation



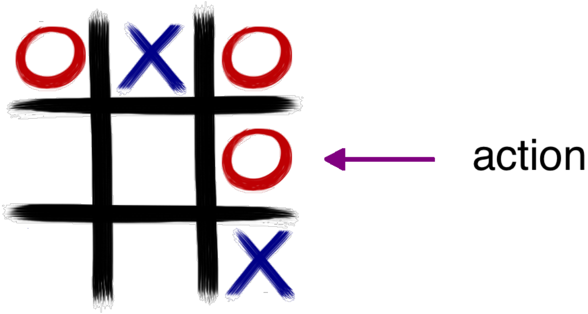
environment

Example: Tic Tac Toe, Notation

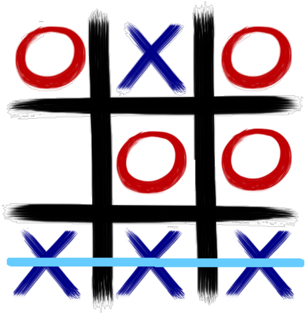


(current)
state

Example: Tic Tac Toe, Notation



Example: Tic Tac Toe, Notation



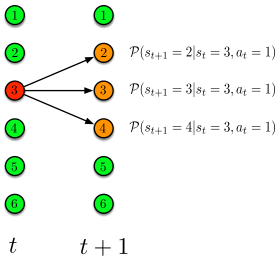
reward
(here: -1)

Formalizing Reinforcement Learning Problems

- Markov Decision Process (MDP) is the mathematical framework to describe RL problems
- A discounted 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 $(0 \leq \gamma < 1)$

Formalizing Reinforcement Learning Problems

- The agent has a **state** $s \in \mathcal{S}$ in the environment, e.g., the location of X and O in tic-tac-toc, or the location of a robot in a room.
- At every time step $t = 0, 1, \dots,$ the agent is at state s_t.
  - Takes an **action** a_t
  - Moves into a new state s_{t+1}, according to the dynamics of the environment and the selected action, i.e., $s_{t+1} \sim \mathcal{P}(\cdot | s_t, a_t)$
  - Receives some **reward** $r_{t+1} \sim \mathcal{R}(\cdot | s_t, a_t, s_{t+1})$



Formulating Reinforcement Learning

- The action selection mechanism is described by a **policy** π
  - Policy π is a mapping from states to actions, i.e., $a_t = \pi(s_t)$
- The goal is to find a policy π such that **long-term rewards** of the agent is maximized.
- Different notations of long-term reward:
  - Average reward:

$$r_t + r_{t+1} + r_{t+2} + \dots$$

  - Sometimes a future reward is discounted by γ^{k-1}, where k is the number of time-steps in the future when it is received:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

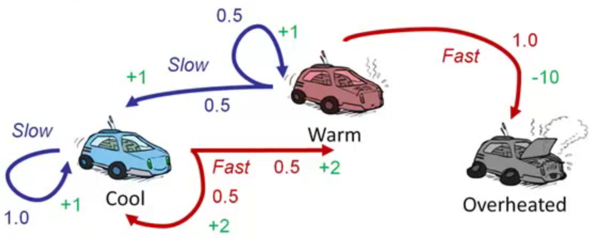
    - If γ close to 1, rewards further in the future count more, and we say that the agent is "farsighted"
    - γ is less than 1 because there is usually a time limit to the sequence of actions needed to solve a task (we prefer rewards sooner rather than

Transition Probability (or Dynamics)

- The transition probability describes the changes in the state of the agent when it chooses actions

$$\mathcal{P}(s_{t+1} = s', r_{t+1} = r' | s_t = s, a_t = a)$$

- This model has **Markov property**: the future depends on the past only through the current state



Policy

- A **policy** is the action selection mechanism of the agent, and describes its behaviour.
- Policy can be deterministic or stochastic:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $A \sim \pi(\cdot|s)$

Value Function

- **Value function** is the expected future reward, and is used to evaluate the desirability of states.
- State-value function V^π (or simply value function) for policy π is a function defined as

$$V^\pi(s) \triangleq \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t R_t \mid S_0 = s \right].$$

  It describes the expected discounted reward if the agent starts from state s and follows policy π.
- The action-value function Q^π for policy π is

$$Q^\pi(s, a) \triangleq \mathbb{E}_\pi \left[\sum_{t \geq 0} \gamma^t R_t \mid S_0 = s, A_0 = a \right].$$

  It describes the expected discounted reward if the agent starts from state s, takes action a, and afterwards follows policy π.

Value Function

- Our aim will be to find a policy π that maximizes the value function (the total reward we receive over time): find the policy with the highest expected reward
- Optimal value function:

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

- Given Q^*, the optimal policy can be obtained as

$$\pi^*(s) \leftarrow \operatorname*{argmax}_a Q^*(s, a)$$

- The goal of an RL agent is to find a policy π that is close to optimal, i.e., $Q^\pi \approx Q^*$.

Bellman Equation

The value function satisfies the following recursive relationship:

$$\begin{aligned}
Q^\pi(s,a) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a\right] \\
&= \mathbb{E}\left[R(S_0, A_0) + \gamma \sum_{t=0}^{\infty} \gamma^t R_{t+1} | s_0 = s, a_0 = a\right] \\
&= \mathbb{E}\left[R(S_0, A_0) + \gamma Q^\pi(S_1, \pi(S_1)) | S_0 = s, A_0 = a\right] \\
&= \underbrace{r(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(\mathrm{d}s'|s,a) Q^\pi(s', \pi(s'))}_{\triangleq (T^\pi Q^\pi)(s,a)}
\end{aligned}$$

This is called the Bellman equation and $T^\pi : B(\mathcal{S} \times \mathcal{A}) \to B(\mathcal{S} \times \mathcal{A})$ is the Bellman operator. Similarly, we define the Bellman *optimality* operator:

$$(T^* Q)(s,a) \triangleq r(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(\mathrm{d}s'|s,a) \max_{a' \in \mathcal{A}} Q(s', a')$$

Bellman Equation

- Key observation:

$$Q^{\pi} = T^{\pi} Q^{\pi}$$
$$Q^{*} = T^{*} Q^{*}$$

- Value-based approaches try to find a $\hat{Q}$ such that

$$\hat{Q} \approx T^{*} \hat{Q}$$

- The greedy policy of $\hat{Q}$ is close to the optimal policy:

$$Q^{\pi(x; \hat{Q})} \approx Q^{*}$$

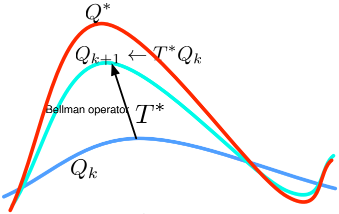
where the greedy policy is defined as

$$\pi(s; \hat{Q}) \leftarrow \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}(s, a)$$

Finding the Optimal Value Function: Value Iteration

- Assume that we know the model $\mathcal{P}$ and $\mathcal{R}$. How can we find the optimal value function?
- This is the problem of **Planning**.
- We can benefit from the Bellman optimality equation and use a method called **Value Iteration**

$$Q_{k+1} \leftarrow T^* Q_k$$

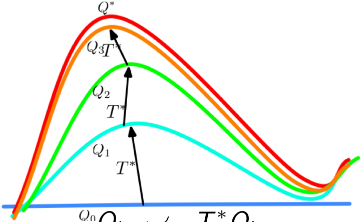


$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(\mathrm{d}s'|s,a) \max_{a' \in \mathring{A}} Q_k(s',a')$$

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathring{A}} Q_k(s',a')$$

Value Iteration

- The Value Iteration converges to the optimal value function.
- This is because of the contraction property of the Bellman (optimality) operator, i.e., $\|T^*Q_1 - T^*Q_2\|_\infty \leq \gamma \|Q_1 - Q_2\|_\infty$.

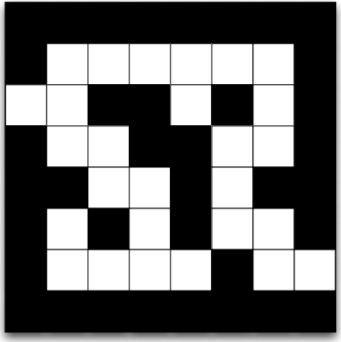


$$Q_{k+1} \leftarrow T^* Q_k$$

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \int_{\mathcal{S}} \mathcal{P}(\mathrm{d}s'|s,a) \max_{a' \in \mathring{A}} Q_k(s',a')$$

$$Q_{k+1}(s,a) \leftarrow r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) \max_{a' \in \mathring{A}} Q_k(s',a')$$

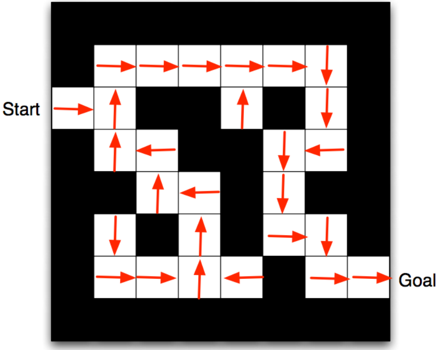
Maze Example



- Rewards: -1 per time-step
- Actions: N, E, S, W
- States: Agent's location

[Slide credit: D. Silver]

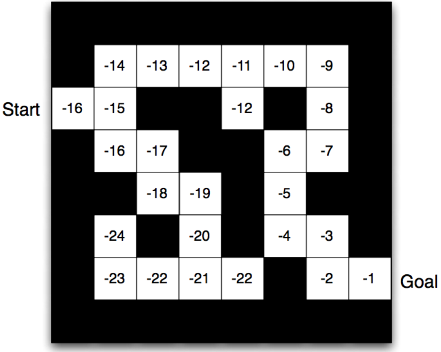
Maze Example



- Arrows represent policy $\pi(s)$ for each state s

[Slide credit: D. Silver]

Maze Example



- Numbers represent value $V^{\pi}(s)$ of each state s

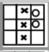
[Slide credit: D. Silver]

Example: Tic-Tac-Toe

- Consider the game tic-tac-toe:
  - **reward**: win/lose/tie the game $(+1/-1/0)$ [only at final move in given game]
  - **state**: positions of X's and O's on the board
  - **policy**: mapping from states to actions
    - based on rules of game: choice of one open position
  - **value function**: prediction of reward in future, based on current state
- In tic-tac-toe, since state space is tractable, we can use a table to represent value function

RL & Tic-Tac-Toe

- Each board position (taking into account symmetry) has some probability

State	Probability of a win (Computer plays "o")
	0.5
	0.5
	1.0
	0.0
	0.5
etc	

- Simple learning process:
  - start with all values $= 0.5$
  - **policy**: choose move with highest probability of winning given current legal moves from current state
  - update entries in table based on outcome of each game
  - After many games value function will represent true probability of winning from each state
- Can try alternative policy: sometimes select moves randomly (exploration)