JOHNS HOPKINS UNIVERSITY

Reinforcement Learning

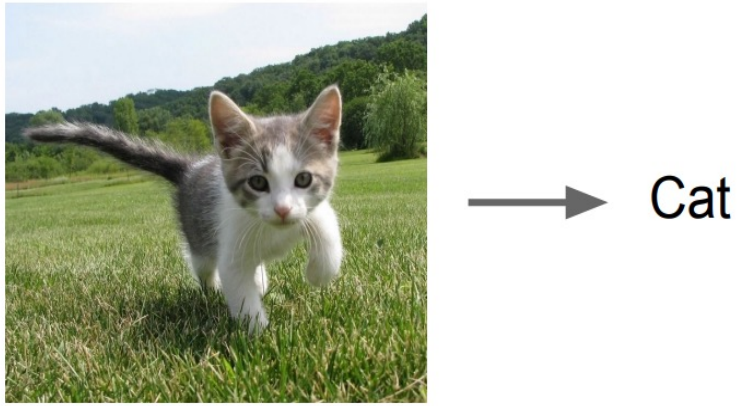
Thanh Nguyen-Tang*

EN.601.475/675 Machine Learning

*Many of the slides are credited to Doina Precup at MLSS'20, Fei-Fei Li & Justin Johnson & Serena Yeung at CS231n (Stanford '17), and Rich Sutton's book

So far … supervised learning

- $x \in \mathcal{X}$ is data input, $y \in \mathcal{Y}$: label
- **Goal**: Learn a function that maps from $\mathcal{X}$ to $\mathcal{Y}$
- **Examples**: classification, regression, object detection, image captioning, image segmentation

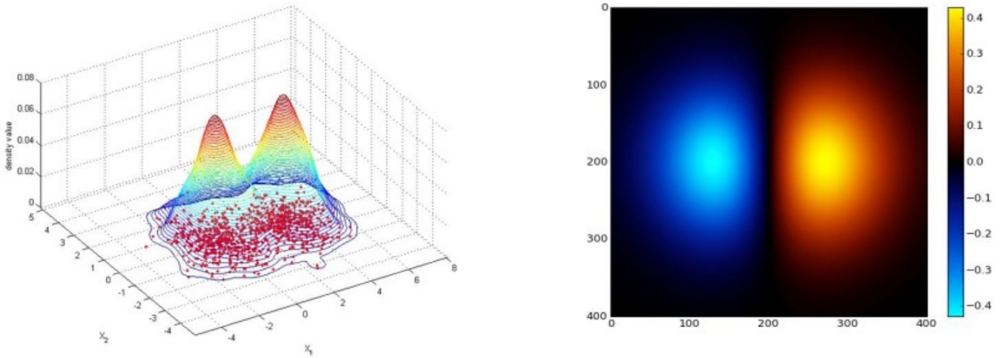


Classification

Unsupervised learning

- **Data**: x, no label
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**: clustering, dimension reduction, feature learning, density estimation

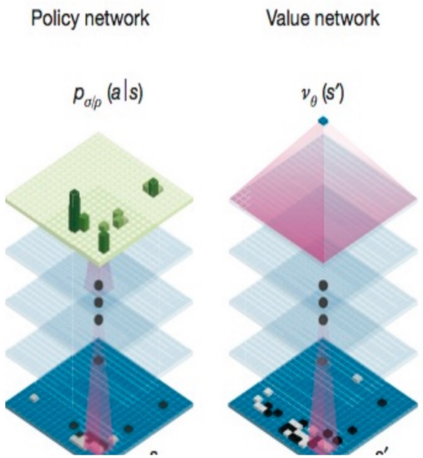
1-d density estimation



2-d density estimation

Today: Reinforcement Learning

- **Problems** involve learning from interacting with an environment, which provides reward signals for each action



- **Goal**: Learn how to take a sequence of actions to maximize the total rewards

Key features of reinforcement learning

- The learner is not told what actions to take; instead, it needs to find out by **trial-and-error search**

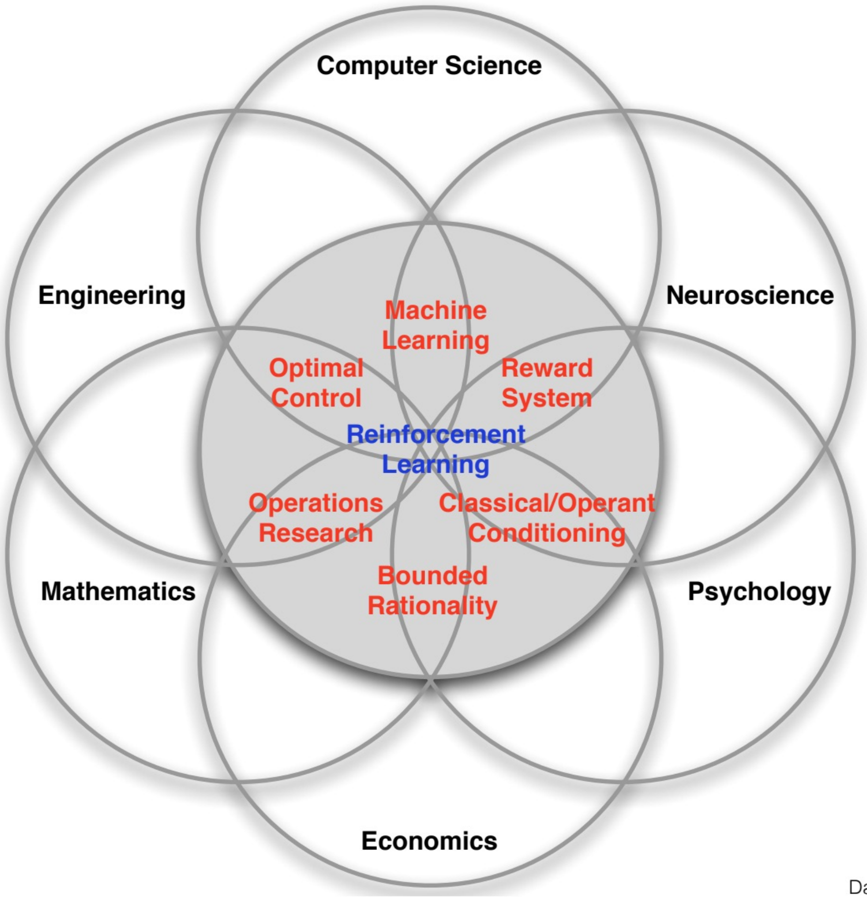
e.g., Players trained by playing thousands of simulated games, with no expert inputs on what are good moves and bad moves

- The environment is **stochastic**
- The reward may be **delayed**

e.g., Player might get rewards only at the end of the game, and needs to assign credit to moves along the way (**credit assignment problem**)

- Learner needs to balance the need to **explore** its environment and **exploit** its current knowledge

e.g., Player might need to try new strategies but also need to win the game

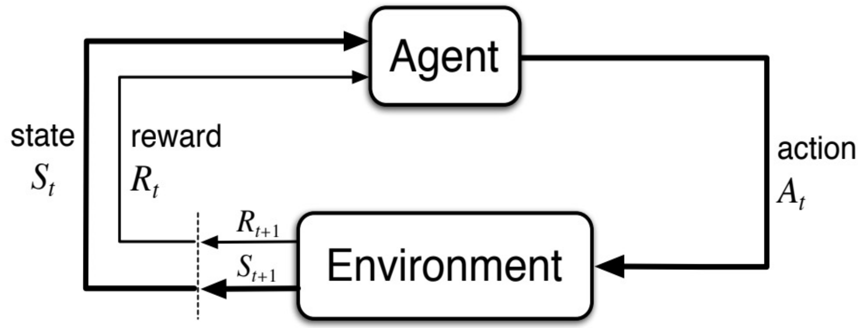


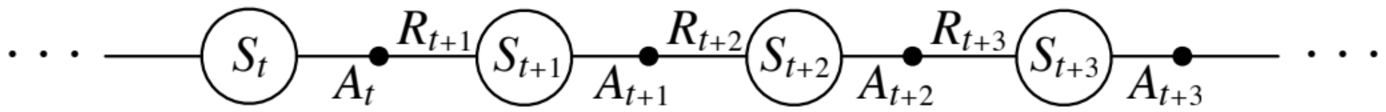
David Silver 2015

Agent-Environment Interaction Protocol

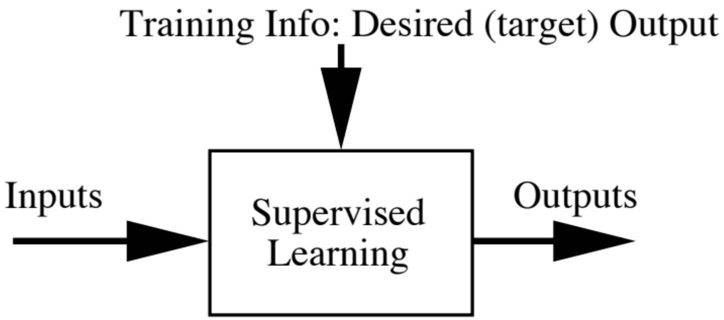
Agent and environment interact at discrete time steps $t = 0, 1, \ldots,$

- Agent observes state: $s_t \in \mathcal{S}$
- Agent chooses action: $a_t \in \mathcal{A}$
- Agent gets reward: $r_t \in \mathbb{R}$
- Environment transitions into next state $s_{t+1} \in \mathcal{S}$

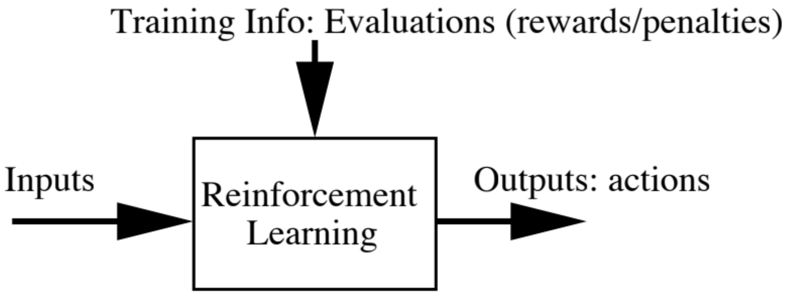




Supervised learning vs. reinforcement learning



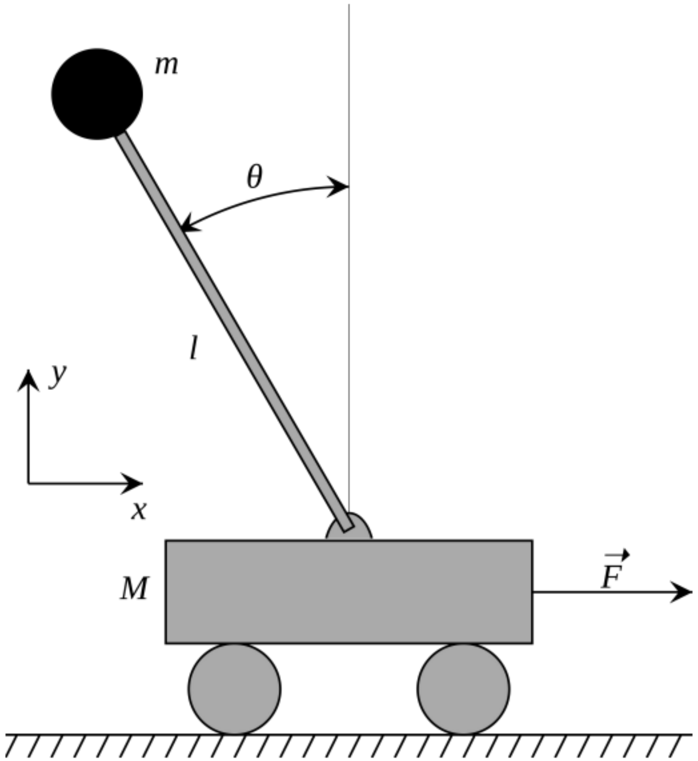
Error = (target output - actual output)



Objective: Get as much total reward as possible

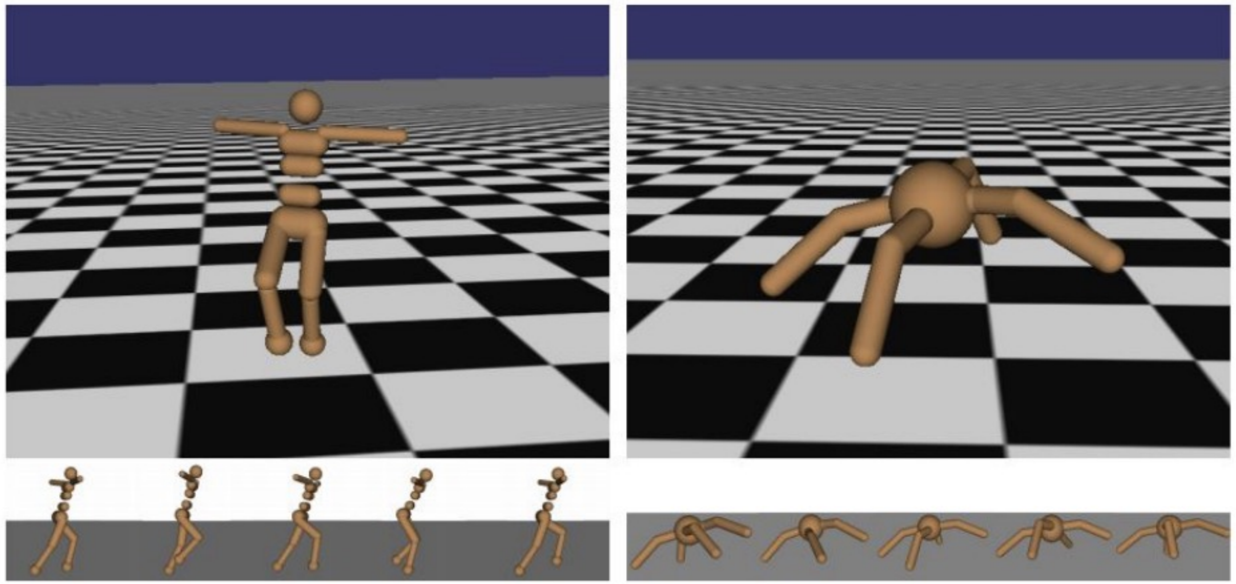
Example: Cart-Pole problem

- **Objective**: Balance a pole on top of a movable cart
- **State**: angle, angular speed, position, horizontal velocity
- **Action**: horizontal force applied to the cart
- **Reward**: 1 at every step if the pole is upright



Robot locomotion

- **Objective**: Make the robot move forward
- **State**: Angle and position of the joints
- **Action**: Torques applied on joints
- **Reward**: 1 at each time upright + moving forward

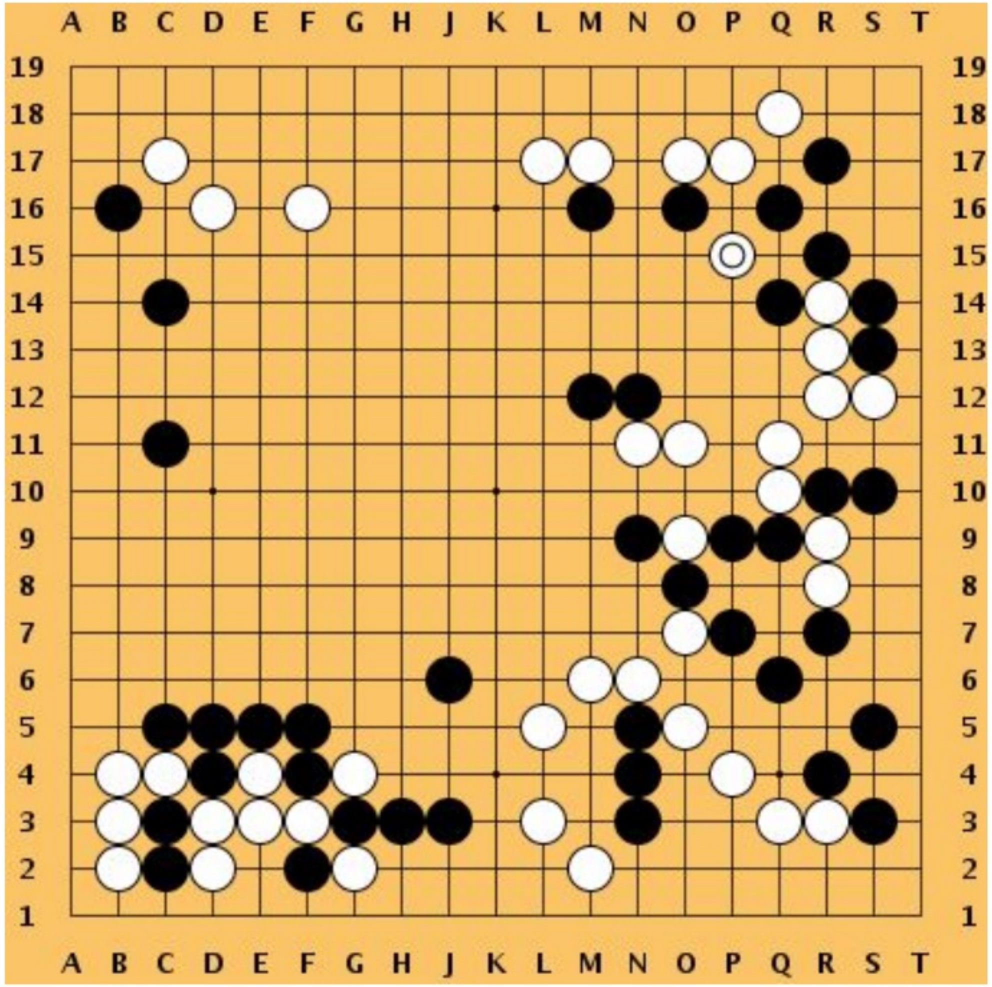


Atari games

- **Objective**: play the game with highest scores
- **State**: Raw pixel input of game states
- **Action**: Game control, e.g., left, right, up, down
- **Reward**: Score increase/decease at each time step

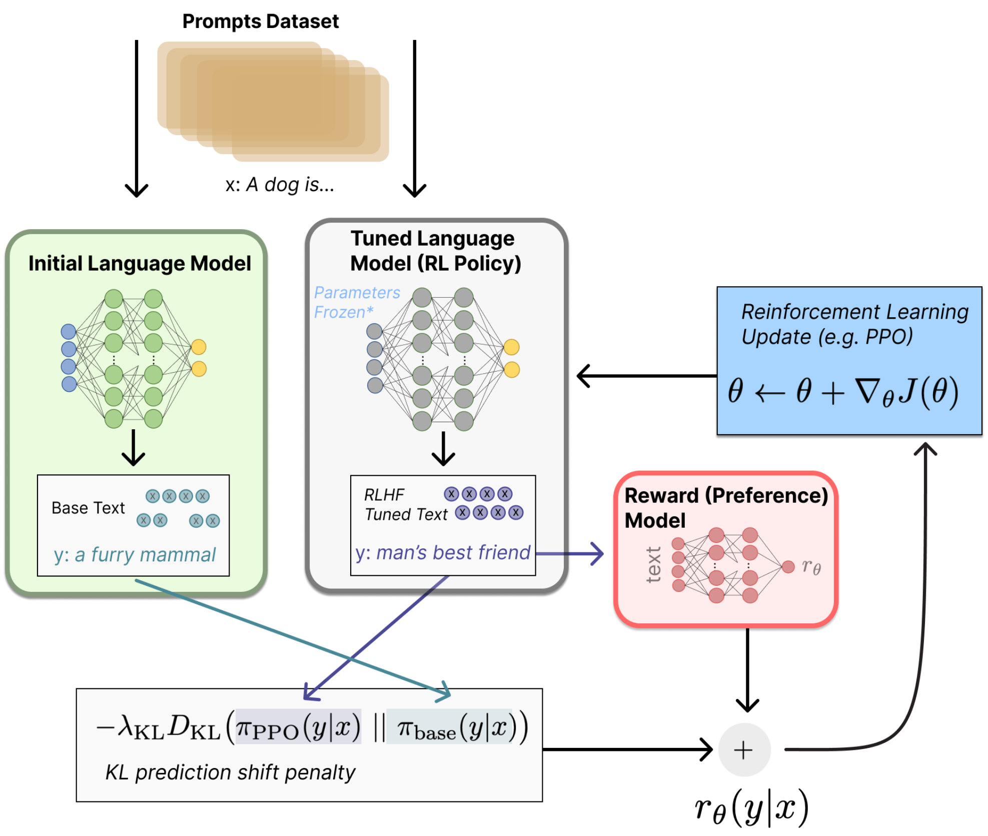
Go Game

- **Objective**: win the game
- **State**: Position of all pieces
- **Action**: where to put the next piece down
- **Reward**: 1 if win at the end of the game, 0 otherwise



RL from human feedback (RLHF): Fine-tuning ChatGPT

- **Objective**: Fine-tune ChatGPT to make its response well-aligned with human feedback
- **State**: input prompt
- **Action**: ChatGPT's generated text
- **Reward**: high reward if the generated text is ranked high by a reward model that is learned from human feedback



Policy (strategy, way of behaving)

- Execute actions in environment, observe rewards, and learn policy (strategy, way of behaving) $\pi: \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$

$$\pi(a|s) = \Pr(a_t = a|s_t = s)$$

- The policy can be deterministic, $\pi: \mathcal{S} \rightarrow \mathcal{A}$, with $\pi(s) = a$ giving action chosen in state s

Return

- Suppose the sequence of rewards after step t is

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

We want to compute (in policy evaluation task) or maximize (in control task) the expected **return** $\mathbb{E}\, G_t$ on each step t

- **Total rewards**: G_t= sum of all future rewards in the episode
- **Discounted rewards**: G_t= sum of all future discounted rewards
- **Average rewards**: G_t= average reward per time step

Policy evaluation task vs control task

- **Policy evaluation**: Given a policy π, compute the "value" of π

$$V^{\pi}(s) := \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \mid s_t = s\right]$$

$$= \mathbb{E}_{\pi}\left[r_{t+1} + \gamma r_{t+2} + \gamma^{2} r_{t+3} + \cdots \mid s_t = s\right]$$

- **Control task**: Find a policy that maximizes the expected total (discounted) rewards $\mathbb{E}\, G_t$ on each step t

Episodic tasks

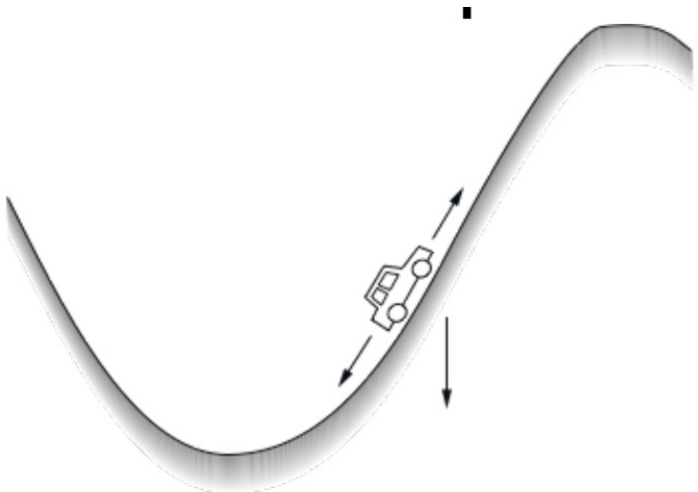
- **Episodic tasks**: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze

- In episodic tasks, we almost always use total rewards

$$G_t = r_{t+1} + r_{t+2} + \cdots + r_T$$

where T is a final step at which a terminal state is reached

Example: Mountain car



Get to the top of the hill
as quickly as possible.

reward = –1 for each step where **not** at top of hill

$\Rightarrow$ return = – number of steps before reaching top of hill

Return is maximized by minimizing
number of steps to reach the top of the hill.

Continuing tasks

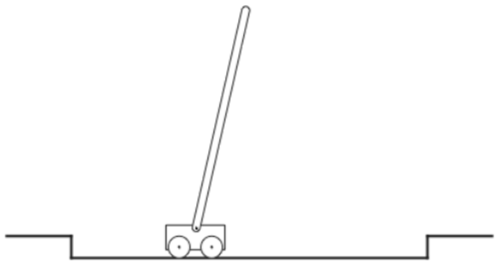
- **Continuing tasks**: interaction does not have natural episodes, but just going on and on

- In this task class, we use discounted return

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where $\gamma \in [0,1]$ is the discount rate.

Example: Pole balancing



Avoid **failure:** the pole falling beyond a critical angle or the cart hitting end of track

As an **episodic task** where episode ends upon failure:

reward = +1 for each step before failure

$\Rightarrow$ return = number of steps before failure

As a **continuing task** with discounted return:

reward = –1 upon failure; 0 otherwise

$\Rightarrow$ return = $-\gamma^k$, for k steps before failure

In either case, return is maximized by avoiding failure for as long as possible.

Markov decision process

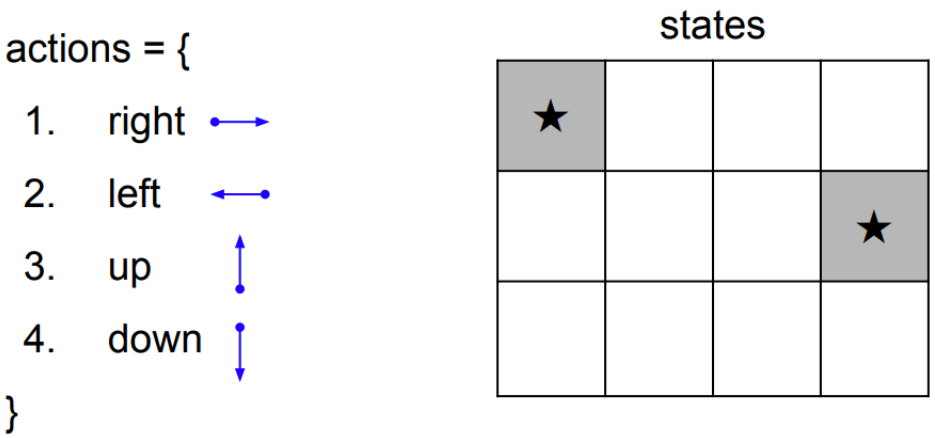
- A mathematical formulation of RL problems
- **Markov property**: current state completely characterizes the state of the world and next state depends only on the current state and the action.
- Defined by tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$
  - S: set of possible states
  - $\mathcal{A}$: set of possible actions
  - R: distribution of reward per each (state, action) pair
  - P: transition probability, i.e., probability of next state given current state, action
  - γ: discount factor

Markov decision process (con't)

- At time $t = 0$, environment samples initial state $s_0 \sim p(s_0)$
- From time $t = 0$ until done
  - Agent selects action a_t
  - Environment samples reward $r_t \sim R(s_t, a_t)$
  - Environment samples next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$
  - Agent receives reward r_t and observes next state s_{t+1}
- A policy $\pi: \mathcal{S} \to \Delta(\mathcal{A})$ maps a state into a distribution over action space
- Discounted return: $G_t := \sum_{i=0}^{\infty} \gamma^i \, r_{t+i+1}$
- **Objective**: Find an optimal policy π^* that maximizes G_t on each step t

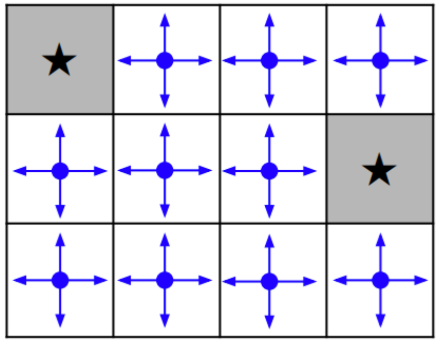
Example: A simple grid world

Control objective: reach one of the terminal states (gray cells) in least number of actions

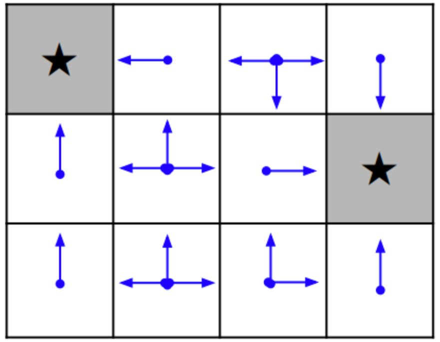


Set a negative "reward" for each transition (e.g. r = -1)

Simple Grid World



Random Policy



Optimal Policy

Optimal policy π^*, formally

- **Optimal policy**: any policy that maximizes the expected cumulative reward

$$\pi^* \in \operatorname{argmax}_\pi \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t \,|\, \pi\right]$$

$$s_0 \sim p_0(s_0), a_t \sim \pi(\cdot\,|s_t), r_t \sim R(s_t, a_t), s_{t+1} \sim P(\cdot\,|s_t, a_t)$$

Value functions and Q-value functions

- State-value function

$$V^{\pi}(s) \coloneqq \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right]$$

  - The expected cumulative reward when following policy π, starting from state s

- Q-value function

$$Q^{\pi}(s, a) \coloneqq \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid (s_t, a_t) = (s, a)\right]$$

  - The expected cumulative reward when following policy π, starting from state-action pair (s, a)

Bellman equation

Theorem:

$$Q^{\pi}(s,a) = \sum_{r,s'} \sum_{a'} \pi(a'|s') p(r,s'|s,a) \left(r + \gamma Q^{\pi}(s',a') \right), \forall (s,a)$$

Optimal value functions

- Optimal state-value function:

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

- Optimal action-value function

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \ \ \forall s, a$$

Optimality Bellman equation

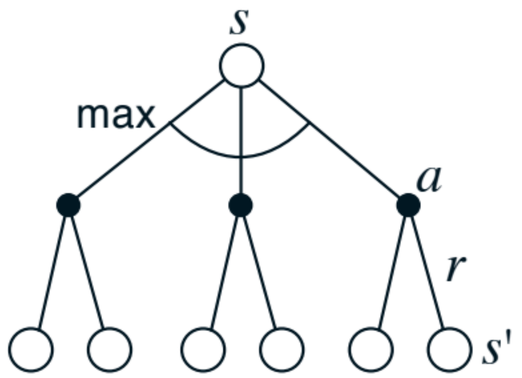
Theorem:

$$Q^*(s,a) = \mathbb{E}_{r\sim R(s,a),s'\sim P(\cdot|s,a)}\left[r + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a')\right], \forall(s,a)$$

Bellman optimality equation for V^*

- The value of a state under an optimal policy must equal the expected return for the best action from that state

$$
\begin{aligned}
V^*(s) &= \max_a Q^*(s,a) \\
&= \max_a \mathbb{E}[r + \gamma V^*(s') \mid s, a] \\
&= \max_a \sum_{r,s'} p(r, s' | s, a)(r + \gamma V^*(s'))
\end{aligned}
$$

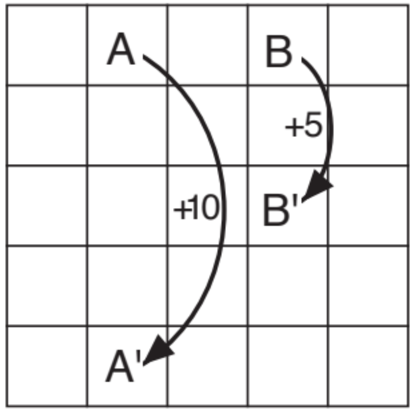


- V^* is the unique solution of this system of equations

Why Optimal State-Value Functions are useful?

- Any policy that is greedy with respect to V^* is an optimal policy
- Therefore, given V^* , one-step-ahead search produces the long-term optimal action

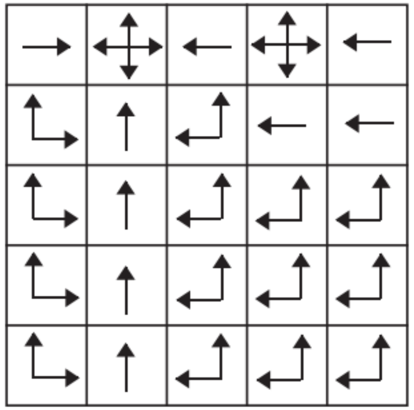
E.g., back to the gridworld:



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) v_*



c) π_*

What about optimal action-value functions?

- Given Q^*, the agent does not even need to do one-step-ahead search

$$\pi_*(s) \in \mathrm{argmax}_a Q^*(s, a)$$

Dynamic programming

- Policy Evaluation
- Policy Iteration
- Value Iteration

Policy evaluation

How to compute the state-value function V^{π} for an arbitrary policy π?

- **Key idea**: Use Bellman equation
- **Procedure**:
  - Initialize any function Q_1
  - Iteratively compute

$$Q_{t+1}(s,a) \leftarrow \sum_{r,s'} \sum_{a'} \pi(a'|s')p(r,s'|s,a)\left(r + \gamma Q_t(s',a')\right)$$

- **Result**: Q_t converges to Q^{π} when $t \rightarrow \infty$

Policy improvement

Given a policy π, how can we get a new policy π' that has **better value** than π?

- We greedify with respect to a given the value function Q^{π}
- Let π' be a greedy policy with respect to Q^{π}, i.e.,

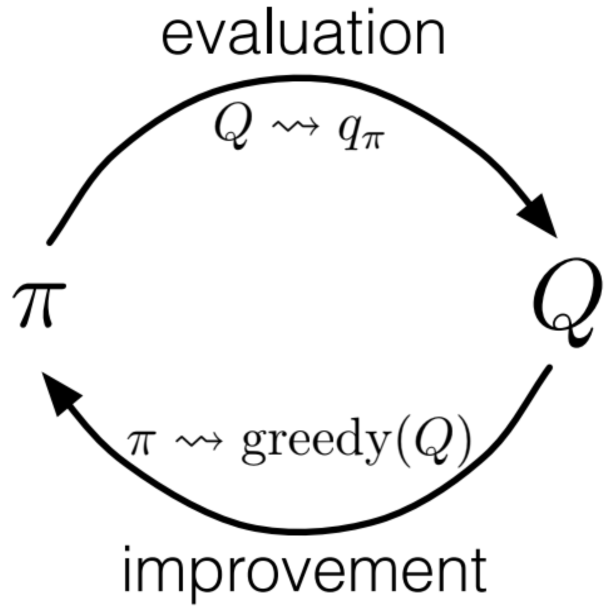
$$\pi'(\mathrm{argmax}_a Q^{\pi}(s,a)|s) = 1$$

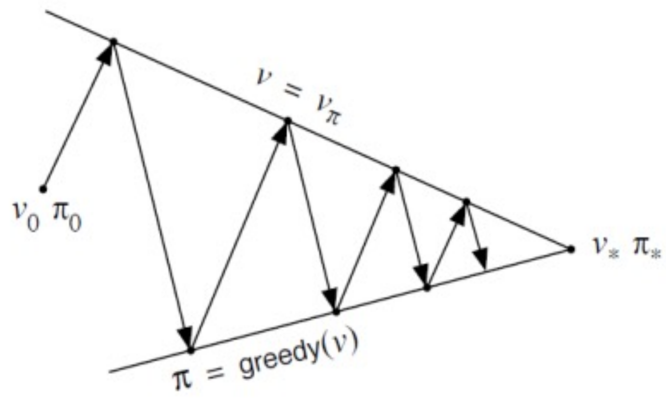
- Then we have

$$V^{\pi'}(s) \geq V^{\pi}(s), \forall s$$

Policy Iteration for control task

- Compute an optimal policy using **Policy Evaluation** and **Policy Improvement**





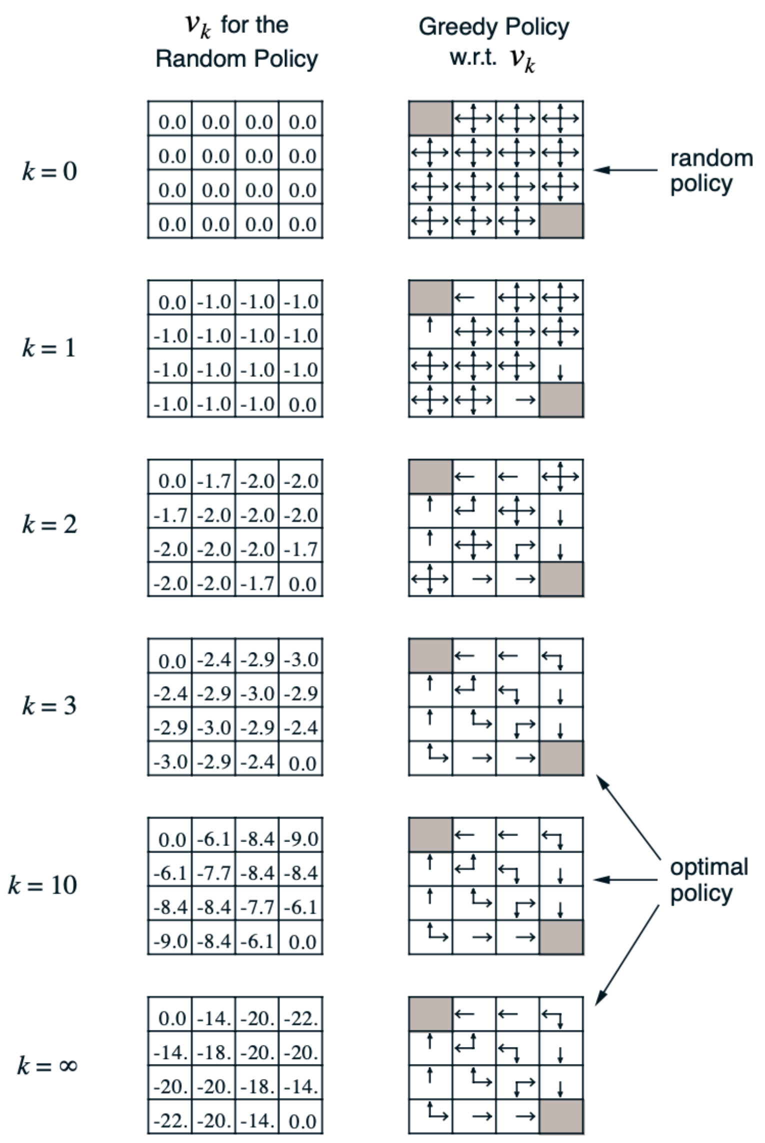
- One **drawback** of policy iteration is that each of its iteration involves policy evaluation

Value Iteration for control task

- Value iteration algorithm: Use Bellman equation as an iterative update

$$Q_{i+1}(s,a) \leftarrow \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a')\right]$$

- **Result**: Q_i converges to Q^* when $\mathrm{i} \rightarrow \infty$



Drawback of Dynamic Programming

- DP require the full knowledge about the MDP

- In learning setting, we don’t know MDP and must learn from experience
  - Monte Carlo methods
  - TD (temporal-difference) learning