Autonomous Robotics

Reinforcement Learning I

Josiah Hanna
University of Wisconsin — Madison

Announcements

Hope you had a good spring break.

Thanks to those who completed the midterm evaluation.

Looking ahead:

- We have finished the classical robotics part of the course.

- Final five weeks will cover advanced topics.

Learning Outcomes

After today's lecture, you will:

- Understand the motivation for reinforcement learning (RL) in robotics.
- Understand when RL is (and is not) a good tool in robotics.
- Understand how to define an RL problem.
- Be able to identify key classes of RL methods for robot control problems.

What is Reinforcement Learning?

- Type of machine learning that focuses on learning from rewards and trial and error interaction.
- The learning agent takes actions, receives rewards, and over time learns to take actions that lead to the most reward.
- Think: training a dog to do tricks.

Why (and why not) RL in robotics?

- Opportunities:
  - Well-suited for tasks where success can be defined but the correct actions to achieve success are unclear.
  - Well-suited for addressing unknown environments.
  - Well-suited for changing environments.
- Challenges:
  - May require long training times.
  - In some cases, we already have good existing controllers (e.g., basic inverse kinematics)
  - Success may be difficult to define.

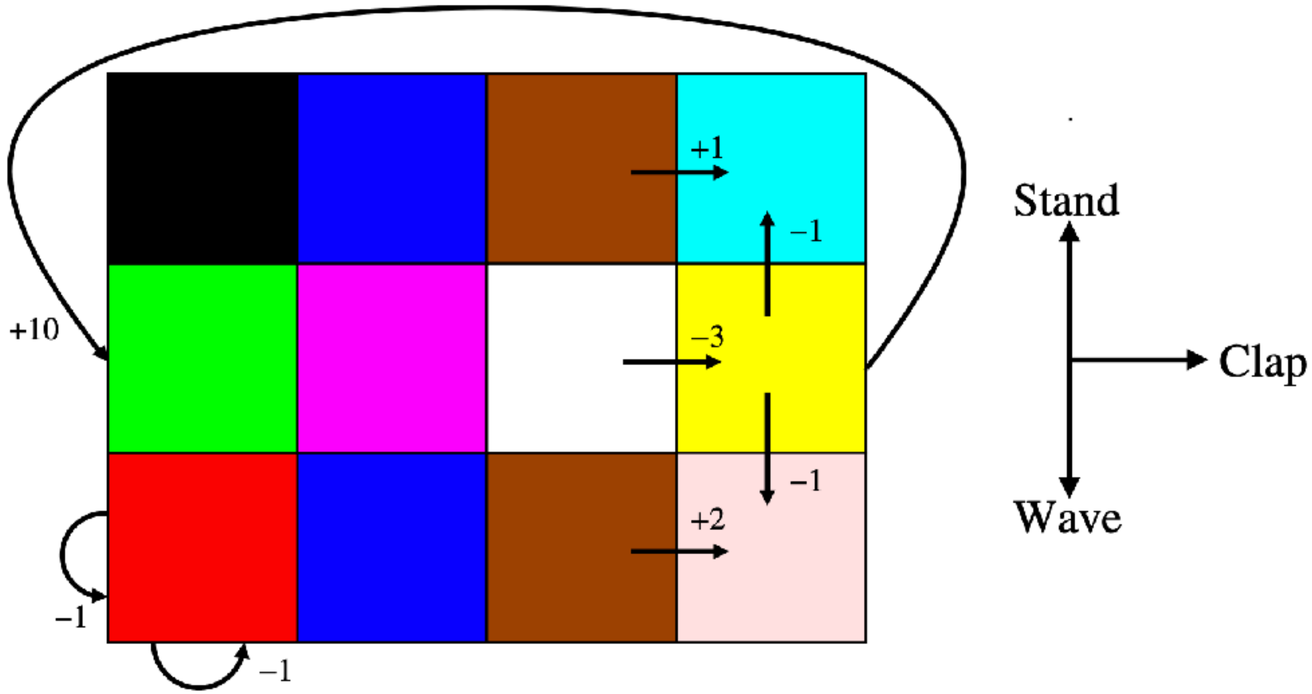
Be an RL Agent*

- You (as a class) are the learning agent.
- Three actions: stand, clap, or wave
- Observations: colors $\in \{$red, blue, orange, pink$\}$
- Rewards: depends on color you see and action you take.
- Goal: find the optimal policy.
  - Policy: mapping from colors to actions.
  - Optimal policy: policy that gives you the most reward.

*** Activity credit to Peter Stone.**

Be an RL Agent

- How did you learn?
- What structure does the world have?



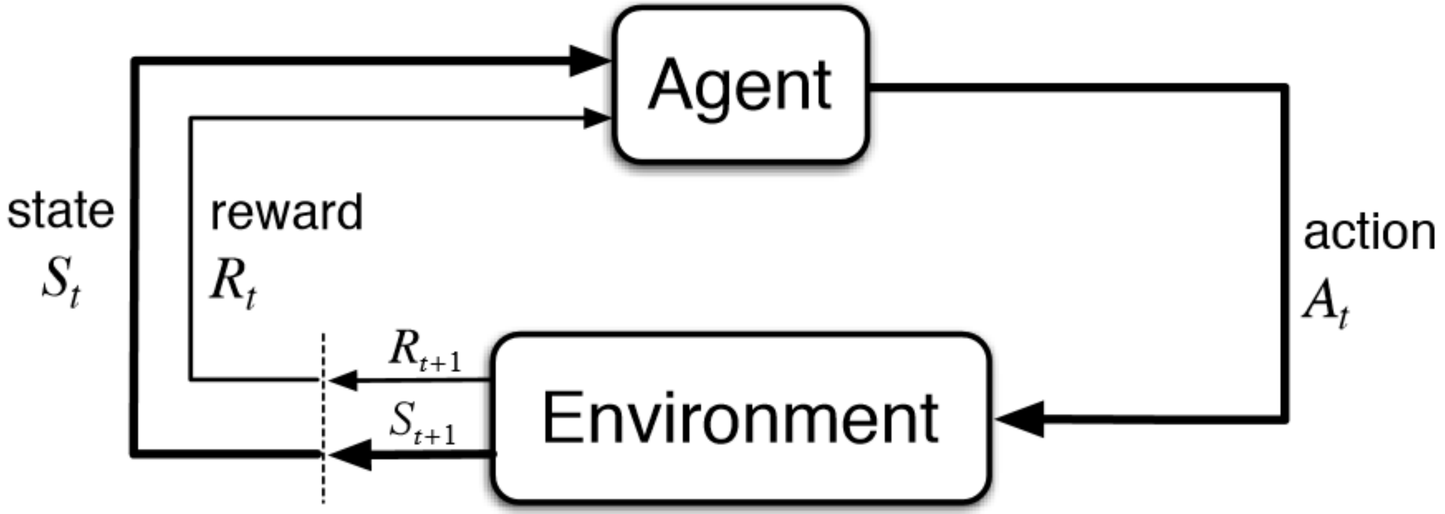
Reinforcement Learning Problems

- States: 3x4 grid
- Observations: colors
  - For our discussion, states and observations will be treated the same.
- Actions: stand, clap, wave
- Rewards: +1, +2, -1, or +10
- State transitions dependent on action chosen.

General Reinforcement Learning

- States: $s \in \mathcal{S}$
- Actions: $a \in \mathcal{A}$
- Rewards: $R \sim r(s, a)$
- State transitions: $S \sim p(\cdot \mid s, a)$ Markov!
- Goal: Find a policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, that maximizes cumulative reward.

General Reinforcement Learning



$$\ldots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \ldots$$

$$S_{t+1}, R_{t+1} \sim p(\cdot \mid S_t, A_t)$$

$$A_{t+1} \leftarrow \pi(S_{t+1})$$

Returns and Episodes

- The **return** is the discounted sum of future rewards:
  $G_t := R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+2} + \dots.$
- Recursive definition: $G_t = R_{t+1} + \gamma G_{t+1}$.
- **Episodes** are subsequences of interaction that begin in some initial state and end in a special terminal state.
- The initial state of one episode is independent of interaction in the preceding episode.

Value functions

- State transitions and rewards are stochastic so we define the utility of states and actions in terms of **expected return.**

- The expected return from a state, $v_\pi(s)$, is only well-defined with respect to a particular policy, π. (Why?)

- State-value and action-value functions are always defined in terms of some policy.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \,|\, S_t = s] = \mathbb{E}_\pi[\sum_{k=0} \gamma^k R_{t+k+1} \,|\, S_t = s]$$

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0} \gamma^k R_{t+k+1} \,|\, S_t = s, A_t = a]$$

Bellman Equations: Recursive Relationship of State Values

$$v_\pi(s) := \mathbb{E}_\pi[G_t \,|\, S_t = s]$$

Definition of return

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \,|\, S_t = s]$$

Definition of state value

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \,|\, S_t = s]$$

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi(S_{t+1})) \,|\, S_t = s, A_t = a]$$

Optimality

- Agent's objective: find the policy that maximizes $v_\pi(s)$ for all s.
- The optimal policy is the policy that has maximal value in all states. $\pi^\star \geq \pi$ if $v_{\pi^\star}(s) \geq v_\pi(s)$ for all states and possible policies.
- Possibly multiple, but always at least one optimal policy in a finite MDP.
  - Also, deterministic and Markovian, i.e., action selection only depends on the current state.
- $\pi^\star(s) = \arg\max_a q_{\pi^\star}(s, a)$ $\quad q_{\pi^\star}(s, a) = \mathbb{E}[R_{t+1} + \gamma v_{\pi^\star}(S_{t+1}) \,|\, S_t = s, A_t = a]$

Dynamic Programming in RL

- Goal: compute value functions and then use them to compute optimal policies.
- Turn Bellman equations into value function updates.
  - $\forall s$, initialize $v_0(s)$, e.g., $v_0(s) \leftarrow 0$.
  - Loop over states and make the update:

$$v_{k+1}(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s'} \sum_r p(s', r \mid s, a)[r + \gamma v_k(s')]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})]$$

Policy Iteration

- Use dynamic programming to compute $v_\pi(s)$ for the current policy π. How can we improve π?
- Alternate:
  - Set $\pi'(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')]$
  - Then repeat dynamic programming to compute $v_{\pi'}$.
- Procedure monotonically improves the policy and converges to $\pi^\star$.

S A R S A

- Key limitation of dynamic programming: must know the transition and reward function.
- Goal: want to learn from $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$ transition tuples.
- Temporal-difference learning of action-values:

$$q_{k+1}(S_t, A_t) \leftarrow q_k(S_t, A_t) + \alpha[R_{t+1} + \gamma q_k(S_{t+1}, A_{t+1}) - q_k(S_t, A_t)]$$

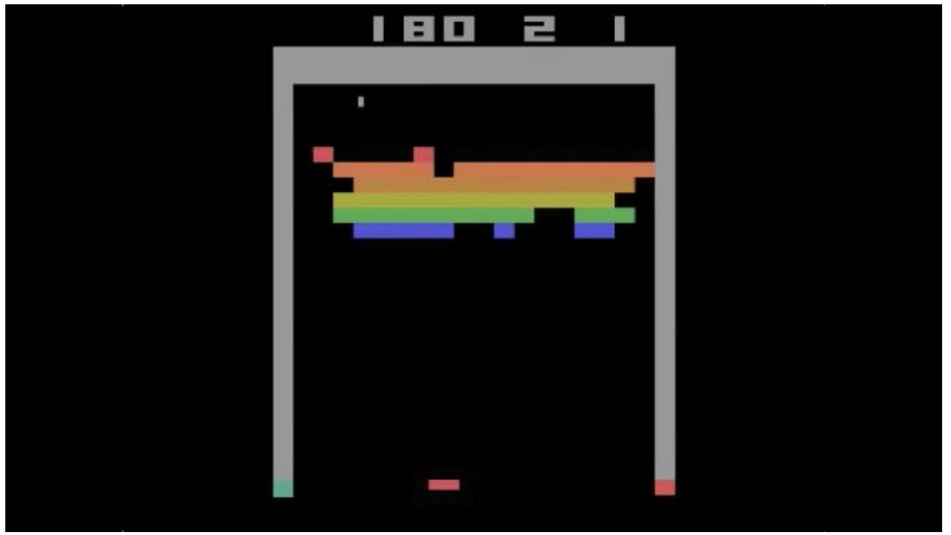
  - Essentially, approximating the exact dynamic programming computation.
- At a high level, it's the same policy iteration scheme from the previous slide.
  - Evaluate π_k using transition tuples obtained by running π_k in the agent's environment.
  - Make π_{k+1} ϵ**-greedy** with respect to q_k. Why?
- It only converges to $q^\star$ if exploration is reduced.

Q-Learning

- Q-learning: an alternative to SARSA:

$$q_{k+1}(S_t, A_t) \leftarrow q_k(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') - q_k(S_t, A_t)]$$

- Converges to $q^\star$ for any sufficiently exploratory exploration policy.
- The underlying algorithm for Deep Q-networks, which was a landmark result in the history of RL.



Derivative algorithms: DQN, Rainbow, BBF, QT-Opt

Q-Learning for Continuous Actions

$$q_{k+1}(S_t, A_t) \leftarrow q_k(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') - q_k(S_t, A_t)]$$

- The max over the action space is difficult to compute when actions take on real-values. Why?
- One idea: use optimization to find the best action.
  - Examples: cross-entropy method, gradient ascent
  - But can be slow.
- Another idea: discretize the action space.
  - Sometimes works but loses precision in control.

Deterministic Actor-Critic

$$q_{k+1}(S_t, A_t) \leftarrow q_k(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} q_k(S_{t+1}, a') - q_k(S_t, A_t)]$$

- Final idea: learn a policy that outputs the maximizing action.
- $\mu_\theta(s) \rightarrow a$.
  - Learn θ such that $q_k(s, \mu_\theta(s)) \approx \max_a q_k(s, a)$.
  - $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta q_k(s, \mu_\theta(s))$
- Actor: the policy μ_θ.
- Critic: the action-value function, trained with SARSA to estimate q_{μ_θ}.

Derivative algorithms: DDPG, TD3, Soft actor-critic

Stochastic Policy Gradient RL

- So far we have only considered deterministic policies.
- Policy gradient methods use a differentiable and stochastic policy (e.g., a neural network) and learn policy parameters with gradient ascent.
- $\pi_\theta(a \mid s) = \Pr(A_t = a \mid S_t = s; \theta)$
- $J(\theta) = v_{\pi_\theta}(s_0)$ for some special start state s_0.
- $\theta_{k+1} \leftarrow \theta_k + \alpha \nabla_\theta J(\theta_k)$

Stochastic Policy Gradient Theorem

- $J(\theta) := v_{\pi_\theta}(s_0) = \sum_a \pi_\theta(a \mid s_0) \sum_{s',r} p(s', r \mid s_0, a)[r + \gamma v_{\pi_\theta}(s')]$
- $\nabla_\theta J(\theta) \propto \sum_s \sum_a \mu_\theta(s) q_{\pi_\theta}(s, a) \nabla_\theta \pi_\theta(a \mid s)$
- The direction in which an infinitesimally small change to θ produces the maximum increase in $J(\theta)$.
- $\nabla_\theta J(\theta)$ does not depend on any gradients of the state transition function, p.

REINFORCE

- $\nabla_\theta J(\theta)$ can only be estimated.
- $\nabla_\theta J(\theta) \propto \mathbf{E}[\sum_a \nabla_\theta \pi(a \mid S_t) q_\pi(S_t, a)] = \mathbf{E}[\sum_a \pi_\theta(a \mid S_t) \frac{\nabla_\theta \pi_\theta(a \mid S_t)}{\pi_\theta(a \mid S_t)} q_\pi(S_t, a)]$
- Finally, replace $q_\pi(s, a)$ with G_t.
- $\theta_{k+1} \leftarrow \theta_k + \alpha G_t \nabla_\theta \ln \pi_\theta(A_t \mid S_t)$

Derivative algorithms: TRPO, PPO, GRPO

REINFORCE

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Usually dropped in practice

Is the policy gradient a gradient? Nota and Thomas. 2020.
Bias in Natural Actor-Critic Algorithms. Thomas. 2014.

Actor-Critic Methods

- REINFORCE uses the return following an action to determine which actions are reinforced.
- Actor-critic methods use learned value functions to drive policy changes.
  - Actor: the policy.
  - Critic: value function.
  - $\theta_{t+1} \leftarrow \theta_t + \alpha \delta_t \nabla_\theta \ln \pi(A_t \mid S_t)$ $\quad\quad$ $\delta_t \leftarrow R_{t+1} + \gamma \hat{v}(S_{t+1}) - \hat{v}(S_t)$

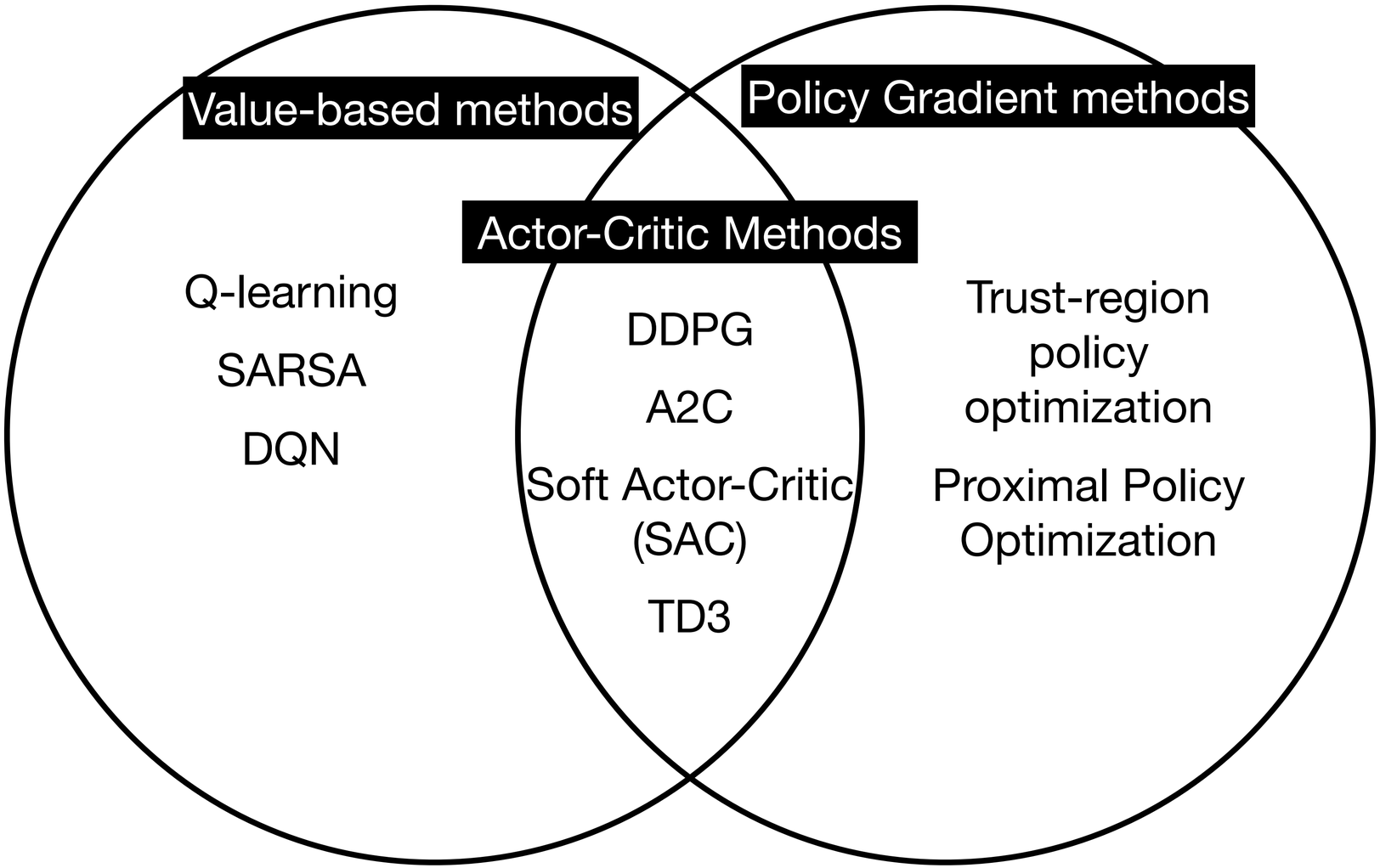
Derivative algorithms: A2C, soft actor-critic, PPO

Actor-Critic Methods

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    ~~$I \leftarrow \gamma I$~~
    $S \leftarrow S'$

Model-Free RL



Summary

Today we covered:

1. The reinforcement learning problem definition.

2. Basic algorithms for RL.

3. Discussion of advantages and challenges with using RL.

Action Items

Complete homework 4

Begin robot learning reading