Principles of Robot Autonomy II

Intro to Reinforcement Learning

Stanford University

Autonomous Systems Lab

Today's lecture

- Aim
  - Provide intro to RL

References:

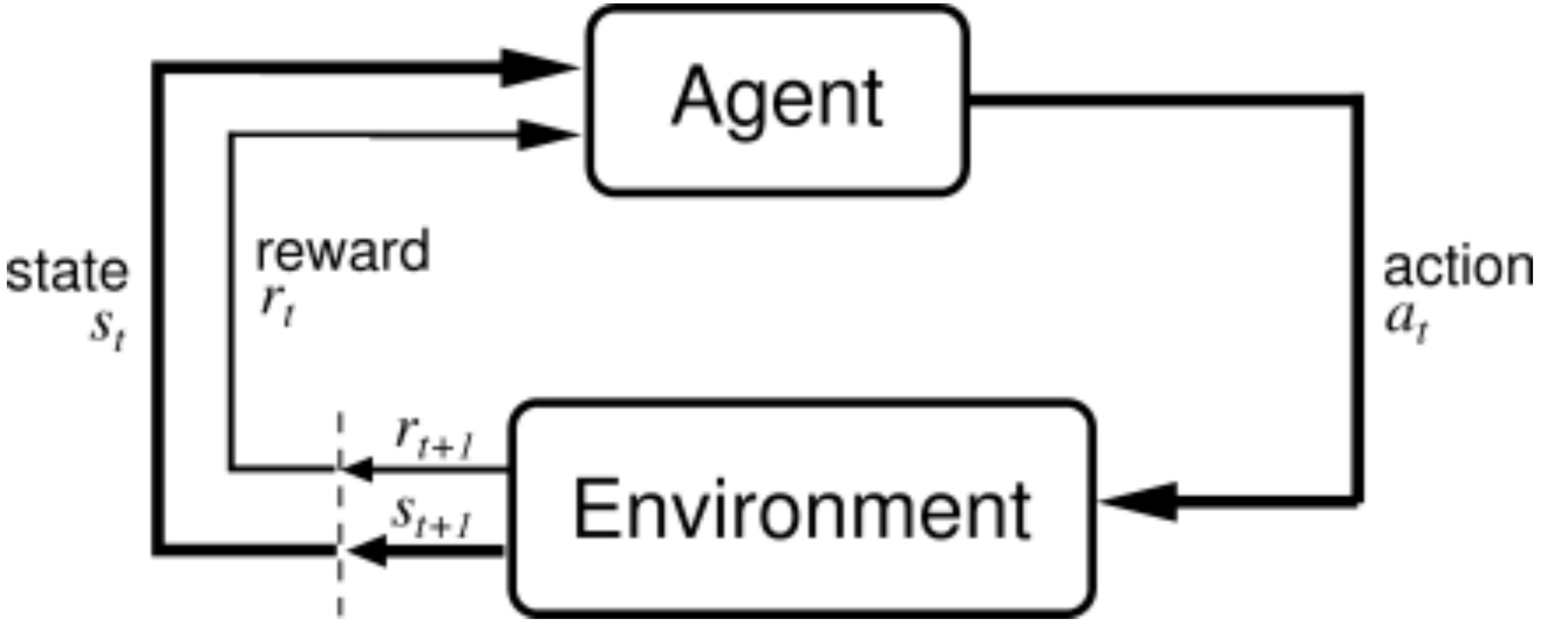
- Sutton and Barto, *Reinforcement Learning: an Introduction*
- Bertsekas, *Reinforcement Learning and Optimal Control*

Courses at Stanford:

- CS 234 Reinforcement Learning
- CS 332 Advanced Survey of Reinforcement Learning
- MS&E 338 Reinforcement Learning

What is Reinforcement Learning?

Learning how to make good decisions by interaction



Why Reinforcement Learning

- Only need to specify a **reward function**. Agent learns everything else!
- Successes in
  - Helicopter acrobatics
  - Superhuman Gameplay: Backgammon, Go, Atari
  - Investment portfolio management
  - Making a humanoid robot walk

Why Reinforcement Learning?

- Only need to specify a **reward function**. Agent learns everything else!
- Successes in
  - Helicopter acrobatics
    - positive for following desired traj, negative for crashing
  - Superhuman Gameplay: Backgammon, Go, Atari
    - positive/negative for winning/losing the game
  - Investment portfolio management
    - positive reward for $$$
  - Making a humanoid robot walk
    - positive for forward motion, negative for falling

Infinite Horizon MDPs

State: $x \in \mathcal{X}$ (often $s \in \mathcal{S}$)

Action: $u \in \mathcal{U}$ (often $a \in \mathcal{A}$)

Transition Function: $T(x_t \mid x_{t-1}, u_{t-1}) = p(x_t \mid x_{t-1}, u_{t-1})$

Reward Function: $r_t = R(x_t, u_t)$

Discount Factor: γ

MDP (<u>stationary model</u>): $\mathcal{M} = (\mathcal{X}, \mathcal{U}, T, R, \gamma)$

Infinite Horizon MDPs

MDP**:** $\mathcal{M} = (\mathcal{X}, \mathcal{U}, T, R, \gamma)$

<u>Stationary</u> policy: $u_t = \pi(x_t)$

Goal: Choose policy that **maximizes cumulative (discounted) reward**

$$V^* = \max_{\pi} E\left[\sum_{t \geq 0} \gamma^t R\big(x_t, \pi(x_t)\big)\right];$$

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t \geq 0} \gamma^t R\big(x_t, \pi(x_t)\big)\right]$$

Infinite Horizon MDPs

- The optimal value function $V^*(x)$ satisfies Bellman's equation

$$V^*(x) = \max_u \left(R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,u)\, V^*(x') \right)$$

- For any stationary policy π, the values $V_\pi(x) \coloneqq E\left[\sum_{t \geq 0} \gamma^t R\left(x_t, \pi(x_t)\right) \middle| \, x_0 = x\right]$ are the unique solution to the equation

$$V_\pi(x) = R(x, \pi(x)) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x, \pi(x))\, V_\pi(x')$$

State-action value functions (Q functions)

- The expected cumulative discounted reward starting from x, applying u, and following the optimal policy thereafter

$$V^*(x) = \max_u \Big(\underbrace{R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,u)\, V^*(x')}_{Q^*(x,u)} \Big)$$

- The optimal Q function, $Q^*(x,u)$, satisfies Bellman's equation

$$Q^*(x,u) = R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,u) \max_{u'} Q^*(x',u')$$

- For any stationary policy π, the corresponding Q function satisfies

$$Q_\pi(x,u) = R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x,u)\, Q_\pi(x',\pi(x'))$$

Solving infinite-horizon MDPs

If you know the model (i.e., the transition function T and reward function R), use ideas from dynamic programming

- Value Iteration / Policy Iteration

Reinforcement Learning: learning from interaction

- Model-based
- Model-free

Value Iteration

- Initialize $V_0(x) = 0$ for all states x
- Loop until finite horizon / convergence:

$$V_{k+1}(x) = \max_u \left(R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x' | x, u)\, V_k(x') \right)$$

- Value iteration for Q functions

$$Q_{k+1}(x,u) = R(x,u) + \gamma \sum_{x' \in \mathcal{X}} T(x' | x, u) \max_{u'} Q_k(x', u')$$

Policy Iteration

Starting with a policy $\pi_k(x)$, alternate two steps:

1. <u>Policy Evaluation</u>
   Compute $V_{\pi_k}(x)$ as the solution of
   $$V_{\pi_k}(x) = R(x, \pi_k(x)) + \gamma \sum_{x' \in \mathcal{X}} T(x' | x, \pi(x))\, V_{\pi_k}(x')$$
2. <u>Policy Improvement</u>
   Define $\pi_{k+1}(x) = \arg\max_u \left(R(x, u) + \gamma \sum_{x' \in \mathcal{X}} T(x' | x, u)\, V_{\pi_k}(x') \right)$

Proposition: $V_{\pi_{k+1}}(x) \geq V_{\pi_k}(x)\ \forall\ x \in \mathcal{X}$

Inequality is strict if π_k is suboptimal

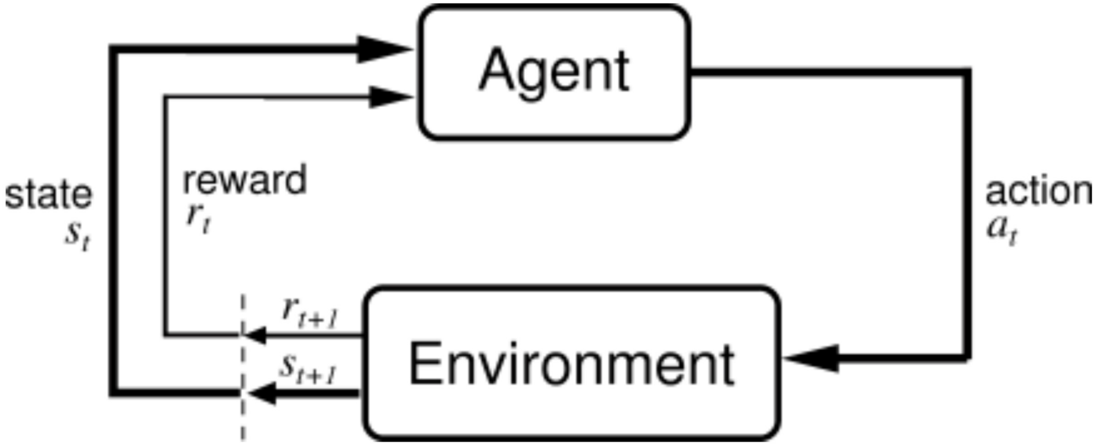
Use this procedure to iteratively improve policy until convergence

Recap

- Value Iteration
  - Estimate optimal value function
  - Compute optimal policy from optimal value function
- Policy Iteration
  - Start with random policy
  - Iteratively improve it until convergence to optimal policy
- Requires **model of MDP** to work!

Learning from Experience

- Without access to the model, agent needs to optimize a policy from interaction with an MDP
- Only have access to trajectories in MDP:
- $\tau = (x_0, u_0, r_0, x_1, \ldots, u_{H-1}, r_{H-1}, x_H)$



Learning from Experience

How to use trajectory data?

- Model based approach: estimate $T(x'|x, u)$, then use model to plan
- Model free:
  - Value based approach: estimate optimal value (or Q) function from data
  - Policy based approach: use data to determine how to improve policy
  - Actor Critic approach: learn both a policy and a value/Q function

Learning from Experience

How to use trajectory data?

- Model based approach: estimate $T(x'|x, u)$, then use model to plan
- Model free:
  - Value based approach: estimate optimal value (or Q) function from data
  - Policy based approach: use data to determine how to improve policy
  - Actor Critic approach: learn both a policy and a value/Q function

Temporal difference (TD) learning

- Main idea: use *bootstrapped* Bellman equation to update value estimates
- *Bootstrapping*: use learned value for next state to update value at current state
  - aims to enforce consistency with respect to Bellman's equation:

$$\mathrm{E}[\underbrace{Q_\pi(x_k, u_k) - (r_k + \gamma Q_\pi(x_{k+1}, u_{k+1})}_{\text{Temporal Difference (TD) error}})] = 0$$

TD policy evaluation

Suppose we have a policy π; we want to compute an estimate of Q_π.

With step size $\alpha \in (0,1)$, loop:

1. Sample (x_k, u_k, r_k, x_{k+1}) from MDP
2. $\hat{Q}(x_k, u_k) \leftarrow \hat{Q}(x_k, u_k) + \alpha\left(r_k + \gamma \hat{Q}(x_{k+1}, u_{k+1}) - \hat{Q}(x_k, u_k)\right)$

Notes:

- Can consider a decreasing sequence of step sizes to ensure convergence

Q-learning

Instead of estimating Q_π, try to estimate Q^* via

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha\left(r_k + \gamma \max_u Q(x_{k+1}, u) - Q(x_k, u_k)\right)$$

(using the TD error for the optimal policy π^*, instead of π)

Thus, we aim to estimate Q^* from a (possibly sub-optimal) demonstration policy π. This property is known as *off-policy* learning

Exploration vs. Exploitation

In contrast to standard machine learning on fixed data sets, in RL we **actively gather the data we use to learn**

- We can only learn about states we visit and actions we take
- Need to **explore** to ensure we get the data we need
- Efficient exploration is a fundamental challenge in RL!

Simple strategy: add noise to the policy.

ϵ-greedy exploration:

- With some small probability ϵ, take a random action; otherwise take the most promising action

Q-learning with ϵ-greedy exploration

Initialize $Q(x, u)$ for all states and actions.

Let $\pi(x)$ be an ϵ-greedy policy according to Q, i.e.,

$$\pi(x) = \begin{cases} \text{UniformRandom}(\mathcal{U}) & \text{with probability } \epsilon \\ \text{argmax}_u Q(x, u) & \text{with probability } (1 - \epsilon) \end{cases}$$

Loop:

1. Take action: $u_k \sim \pi(x_k)$.
2. Observe reward and next state: (r_k, x_{k+1}).
3. Update Q to minimize TD error:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha \left(r_k + \max_u Q(x_{k+1}, u) - Q(x_k, u_k) \right)$$

Fitted Q Learning

How to deal with large/continuous state/action spaces?

Use parametric model for Q function: $Q_\theta(x, u)$ (e.g., $Q_\theta(x, u) = \theta^T \phi(x, u)$)

Stochastic gradient descent on squared TD error to update θ:

$$\theta \leftarrow \theta + \alpha \left(\underbrace{r_k + \gamma \max_u Q_\theta(x_{k+1}, u) - Q_\theta(x_k, u_k)}_{\frac{d(\text{Squared TD Error})}{dQ}} \right) \underbrace{\nabla_\theta Q_\theta(x_k, u_k)}_{\frac{dQ}{d\theta}}$$

learning rate

Q Learning Recap

Pros:

- Can learn Q function from any interaction data, not just trajectories gathered using the current policy ("**off-policy" algorithm**)
- Relatively data-efficient (can reuse old interaction data)

Cons:

- Need to optimize over actions: hard to apply to continuous action spaces
- Optimal Q function can be complicated, hard to learn
- Optimal policy might be much simpler!

Other popular model-free, value-based approach: SARSA (on policy algorithm)

Next time

More on: