AA203
Optimal and Learning-based Control

Intro to reinforcement learning

Today’s lecture

- Aim
  - Provide intro to RL

References:

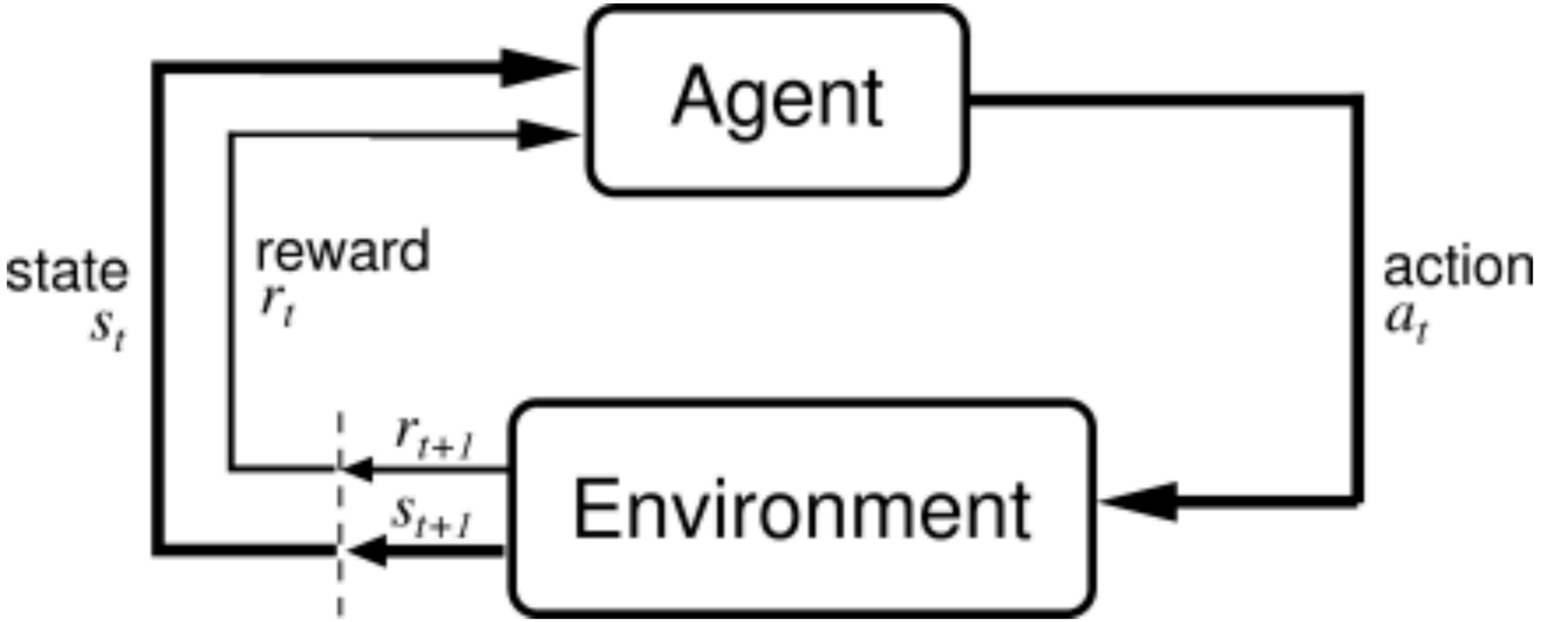
- Sutton and Barto, *Reinforcement Learning: an Introduction*
- Bertsekas, *Reinforcement Learning and Optimal Control*
- Course notes

Courses at Stanford:

- CS 234 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

Outline

- Formalisms
- Algorithms
- Deep Reinforcement Learning
- Overview of RL content in this course

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

Infinite Horizon MDPs

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

Stationary policy: $u_t = \pi(x_t)$

Goal: Choose policy that **maximizes cumulative reward.**

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

Infinite Horizon MDPs

- The optimal cost $V^*(x)$ 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 costs $V_\pi(x)$ are the unique solution to the equation

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

Solving infinite-horizon MDPs

If you know the model, use DP-ideas

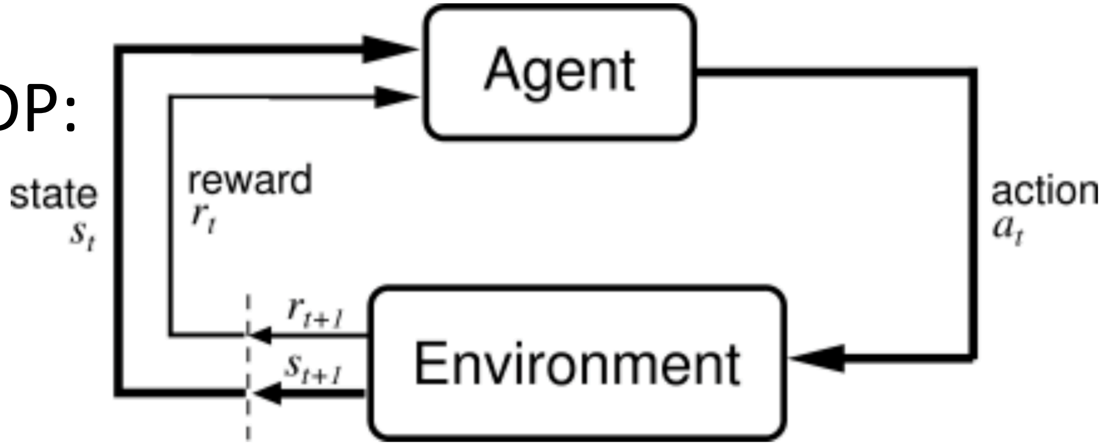
- Value Iteration / Policy Iteration (Covered in lecture 4)

RL: Learning from interaction

- Model-Based
- Model-free
    - Value based
    - Policy based

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

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 probability ϵ, take a random action; otherwise take the most promising action

Model-free, value based: Q Learning

Optimal Q function satisfies

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

So, in expectation,

$$\mathrm{E}\left[\underbrace{Q^*(x_t,u_t) - \left(r_t + \gamma \max_{u'} Q^*(x_{t+1},u')\right)}_{\text{Temporal Difference (TD) error}}\right] = 0$$

Temporal difference learning

- Main idea: use *bootstrapped* Bellman equation to update value estimates
- *Bootstrapping*: use learned value for next state to estimate value at current state
  - Combines Monte Carlo and dynamic programming

Q Learning

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

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

Loop:

Take action: $u_t \sim \pi(x_t)$.

Observe reward and next state: (r_t, x_{t+1}).

Update Q to minimize TD error:

$$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left(r + \max_u Q(x_{t+1}, u) - Q(x_t, u_t) \right)$$

$$t = t + 1$$

Fitted Q Learning

Large / Continuous Action Space?

Use parametric model for Q function: $Q_\theta(x, u)$

Gradient descent on TD error to update θ:

$$\theta \leftarrow \theta + \alpha\left(r_t + \gamma \max_u Q_\theta(x_{t+1}, u) - Q_\theta(x_t, u_t)\right)\nabla_\theta Q_\theta(x_t, u_t)$$

learning rate

$$\boldsymbol{\frac{d(Squared\ TD\ Error)}{dQ}}$$

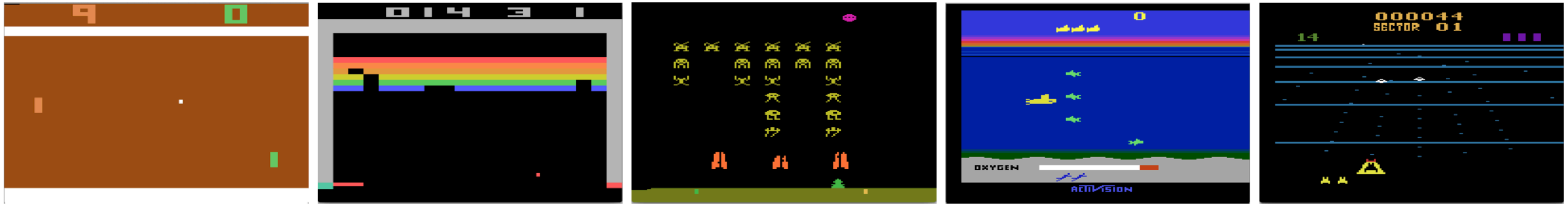
$$\boldsymbol{\frac{dQ}{d\theta}}$$

Deep Q Learning

- Many possible function approximators for Q
  - Linear, nearest neighbors, aggregation
- Recent success: neural networks with loss function

$$\left(r_t + \gamma \max_u Q_{\theta'}(x_{t+1}, u) - Q_\theta(x_t, u_t)\right)^2$$

- Deep Q Network (DQN; Mnih et al. 2013)
  - Experience replay



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!

Model-free, policy based: Policy Gradient

Instead of learning the Q function, learn the policy directly!

Define a class of policies π_θ where θ are the parameters of the policy.

Can we learn the optimal θ from interaction?

Goal: use trajectories to estimate a gradient of policy performance w.r.t parameters θ

Policy Gradient

A particular value of θ induces a distribution of possible trajectories.

Objective function:

$$J(\theta) = E_{\tau \sim p(\tau;\theta)}[r(\tau)]$$

$$J(\theta) = \int_{\tau} r(\tau)p(\tau;\theta)d\tau$$

where $r(\tau)$ is the total discounted cumulative reward of a trajectory.

Policy Gradient

Gradient of objective w.r.t. parameters:

$$\nabla_\theta J(\theta) = \int_\tau r(\tau)\nabla_\theta p(\tau;\theta)d\tau$$

Trick: $\nabla_\theta p(\tau;\theta) = p(\tau;\theta)\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta)\nabla_\theta \log p(\tau;\theta)$

$$\nabla_\theta J(\theta) = \int_\tau (r(\tau)\nabla_\theta \log p(\tau;\theta))p(\tau;\theta)\, d\tau$$

$$\nabla_\theta J(\theta) = E_{\tau\sim p(\tau;\theta)}[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

Policy Gradient

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau;\theta)}[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

$$\log p(\tau;\theta) = \log\left(\prod_{t\geq 0} T(x_{t+1}|x_t,u_t)\pi_\theta(u_t|x_t)\right)$$
$$= \sum_{t\geq 0} \log T(x_{t+1}|x_t,u_t) + \log \pi_\theta(u_t|x_t)$$
$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(u_t|x_t)$$

We don't need to know the transition model to compute this gradient!

Policy Gradient

If we use π_θ to sample a trajectory, we can approximate the gradient:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(u_t | x_t)$$

Intuition: adjust theta to:

- Boost probability of actions taken if reward is high
- Lower probability of actions taken if reward is low

Learning by trial and error

Example

$$\nabla_\theta J(\theta) = E_{\tau \sim p(\tau;\theta)}\left[\sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(u_t|x_t)\right]$$

Policy Gradient Recap

Pros:

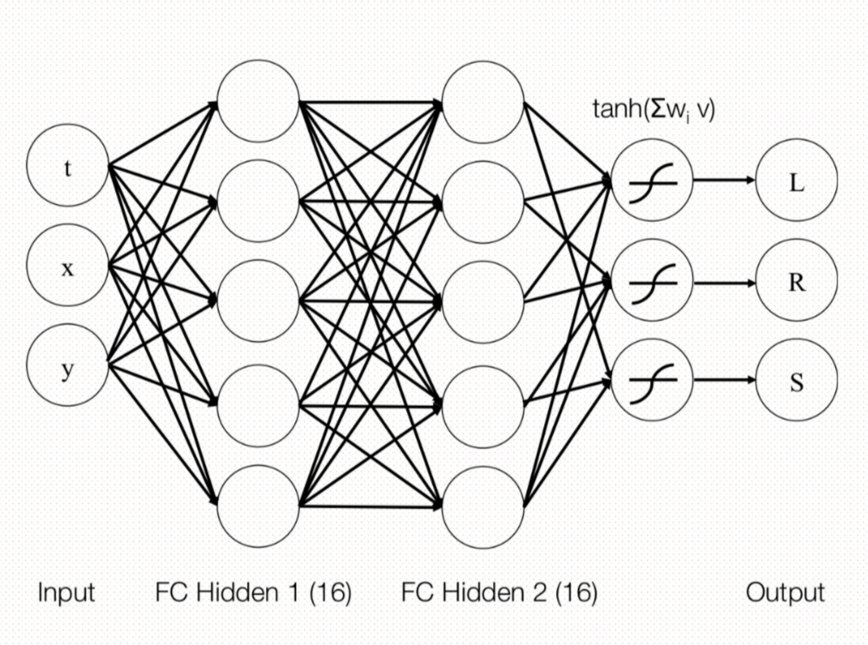
- Learns policy directly – often more stable
- Works for continuous action spaces
- Converges to local maximum of $J(\theta)$

Cons:

- Needs data from current policy to compute gradient – data inefficient
- Gradient estimates can be very noisy

Deep policy gradient

- Parametrize policy as deep neural network
- In practice, very unstable
  - Need to combine with value estimate: *actor-critic*



Summary

- Model-based RL
  - Learn model from interacting with environment
- Model-free RL
  - Value-based methods: learn via minimizing bootstrapped TD error
  - Policy-based methods: directly optimize policy

Later in this class

- Optimal adaptive control
  - How to learn online
  - How to optimal explore
- Model-based RL
  - Model learning for continuous state spaces
  - Combining optimal control with learned models
- Modern model-free RL
  - Especially actor-critic methods
- Combining model-free and model-based RL

Next time

- Dynamic programming in continuous time: HJB and HJI