Principles of Robot Autonomy II

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*

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: \( x \in X \) (often \( s \in S \))

Action: \( u \in U \) (often \( a \in A \))

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

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

Discount Factor: \( \gamma \)

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

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

• The optimal cost $V^*(x)$ satisfies Bellman’s equation

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

• For any stationary policy $\pi$, the costs $V_\pi(x)$ are the unique solution to the equation

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

• Initialize $V^*_0(x) = 0$ for all states $x$
• Loop until finite horizon / convergence:

$$V^*_{k+1}(x) = \max_u \left( R(x, u) + \gamma \sum_{x' \in \mathcal{X}} T(x'|x, u) V^*_k(x') \right)$$
Q functions

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

\[ V^*(x) = \max_u Q^*(x, u) \]

- VI 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

Suppose we have a policy $\pi_k(x)$
We can use VI to compute $V_{\pi_k}(x)$
Define $\pi_{k+1}(x) = \arg\max_u R(x, u) + \gamma \sum_{x' \in X} T(x' | x, u) V_{\pi_k}(x')$

**Proposition:**
$V_{\pi_{k+1}}(x) \geq V_{\pi_k}(x) \ \forall \ x \in X$
Inequality is strict if $\pi$ 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

• Require 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:

• $\tau = (x_0, u_0, r_0, x_1, ..., u_{H-1}, r_{H-1}, x_H)$
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

$\epsilon$-greedy exploration:
- With probability $\epsilon$, take a random action; otherwise take the most promising action
Model-free, value based: Q Learning

Optimal Q function satisfies

\[ Q^*(x, u) = R(x, u) + \gamma \sum_{x' \in X} T(x' | x, u) \max_u Q^*(x', u') \]

Therefore, the optimal Q function satisfies

\[ E \left[ (r_t + \gamma \max_{u'} Q^*(x_{t+1}, u') - Q^*(x_t, u_t)) \right] = 0 \]

Temporal Difference (TD) error
Q Learning

Initialize $Q(x, u)$ for all states and actions
Gather trajectory data $\tau = (x_0, u_0, r_0, x_1, ..., u_{H-1}, r_{H-1}, x_H)$
Loop:
   Take a piece of experience $(x_t, u_t, r_t, x_{t+1})$
   Update $Q$ to correct TD error:
   $$Q(x_t, u_t) \leftarrow Q(x_t, u_t) + \alpha \left( r_t + \max_u Q(x_{t+1}, u) - Q(x_t, u_t) \right)$$
   $$t = t + 1$$
Fitted Q Learning

Large / continuous action space?
Use parametric model for Q function: $Q_\theta(x, u)$

Gradient ascent on $\theta$:

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

learning rate \quad \frac{d\text{(Squared TD Error)}}{dQ} \quad \frac{dQ}{d\theta}$
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 $\pi_\theta$ where $\theta$ are the parameters of the policy

Can we learn the optimal $\theta$ from interaction?

**Goal:** use trajectories to estimate a gradient of policy performance w.r.t parameters $\theta$
Policy Gradient

A particular value of $\theta$ 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 $\pi_\theta$ 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
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
Actor Critic

Actor: Learned Policy, $\pi_\theta$
Critic: Estimated Q function of Actor, $V_\phi$
Critic helps **reduce variance** in gradient estimates for the actor

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left[ r(\tau) - V_\phi(x_0) \right] \nabla_\theta \log \pi_\theta(u_t|x_t)$$

Learn $\phi$ by minimizing TD error, as before

**Result:** learning is more data-efficient
Deep Reinforcement Learning

- Deep Q learning:
  - Use neural network as Q function
  - Works in nonlinear, continuous state space domains

- Deep Policy Gradient:
  - Parameterize policy as deep neural network
  - Policy can act on high dimensional input, e.g. directly from visual feedback
Results in simulation

Heess et al., “Emergence of Locomotion Behaviours in Rich Environments”
Levine et al., “End-to-End Training of Deep Visuomotor Policies”
Challenges in RL for Robotics

Data-efficiency

Sim-to-real

Exploration

Reward design
Next time