Principles of Robot Autonomy I

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
• RL in Robotics
Markov Decision Process (MDP)

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

Action: \( u \in U \) \quad (often \ a \in 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: \( \gamma \)

Horizon: \( H \)

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

MDP: \[ M = (X, U, T, R, \gamma, H) \]

Policy: \[ u_t = \pi(x_t) \]

Goal: Choose policy that maximizes cumulative reward.

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

If you know the model, use dynamic programming
• Value Iteration / Policy Iteration

RL: Learning from interaction
• Model-Based
• Model-free
  • Value based
  • Policy based
Dynamic Programming in MDPs

Define a policy’s **value function** as the expected cumulative discounted reward when acting according to the policy from a given state.

\[
V_k^\pi(x) = E \left[ \sum_{t=0}^{k} \gamma^t R(x_t, \pi(x_t)) | x_0 = x \right]
\]

Value with \( k \) steps to go

\[
V_k^\pi(x) = R(x, \pi(x)) + \gamma \sum_{x' \in X} T(x' | x, \pi(x)) V_{k-1}^\pi(x')
\]
Optimality in MDPs

The optimal policy \( \pi^* \) is a policy that has the highest value.

\[
\pi_t^*(x) = \arg \max_{\pi} V_t^\pi(x)
\]

There exists a unique optimal value function:

\[
V_t^*(x) = V_t^{\pi^*}(x)
\]
Gridworld Example

- Reward: -1 at each timestep
- Actions: N/S/E/W
- State: 2D location
Gridworld Example

Optimal Policy

Optimal Value
Value Iteration

• Dynamic programming for MDPs
• Initialize $V_0^*(x) = 0$ for all states $x$
• Loop until finite horizon / convergence:

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

Another related function in MDPs is the Q function, which is a function of state and action, and corresponds to the value of taking a given action and then acting according to the given policy:

\[ Q^\pi_k (x, u) = R(x, u) + \gamma \sum_{x' \in X} T(x'|x, u) V^\pi_{k-1}(x') \]

Similarly, we can define the optimal Q function:

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

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

\[ V_{k+1}^*(x) = \max_u Q_{k+1}^*(x, u) \]
Policy Iteration

Suppose we have a policy $\pi_k(x)$

We can use DP to compute $Q^{\pi_k}(x, u)$

Define $\pi_{k+1}(x) = \arg\max_u Q^{\pi_k}(x, u)$

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

For simplicity, let’s assume $H = \infty$, so optimal value and policy don’t depend on time. Why?

Optimal Q function satisfies

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

So, in expectation,

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

Temporal Difference (TD) error
Q Learning

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

Let $\pi(x)$ be an $\epsilon$-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 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(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

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_\theta \log p(\tau; \theta)] \]

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

\[ = \sum_{t \geq 0} \log T(x_{t+1} | x_t, u_t) + \log \pi_\theta(u_t | x_t) \]

\[ \nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \log \pi_\theta(u_t | x_t) \]

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.

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

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"
Results in Robotics

*Levine et al., “End-to-End Training of Deep Visuomotor Policies”*
Results in Robotics

OpenAI, “Solving Rubik's Cube with a Robot Hand”
Challenges in RL for Robotics

- Data-efficiency
- Sim-to-real
- Exploration
- Reward design
Further Reading

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
Demo Day Tomorrow

Thanks for a great quarter!