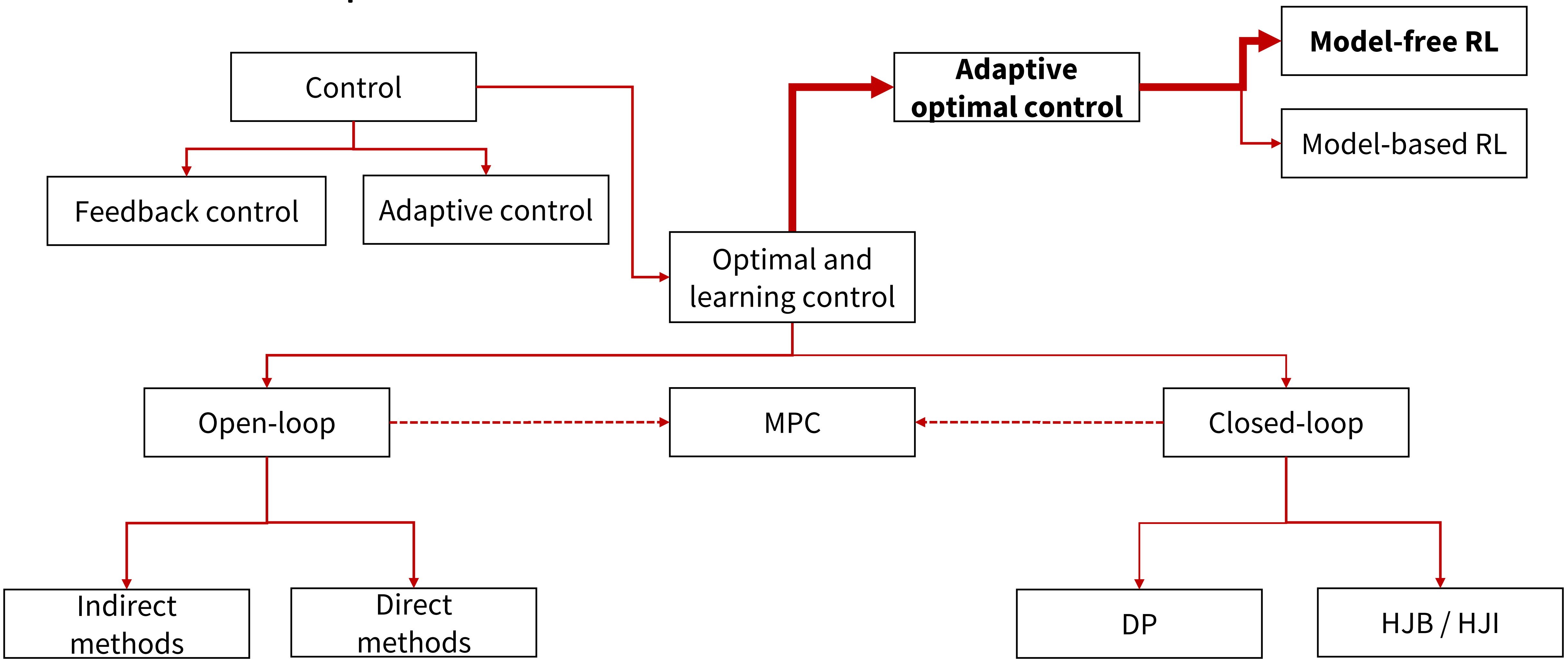
AA203
Optimal and Learning-based Control

Intro to reinforcement learning; dual control; LQG

Stanford University

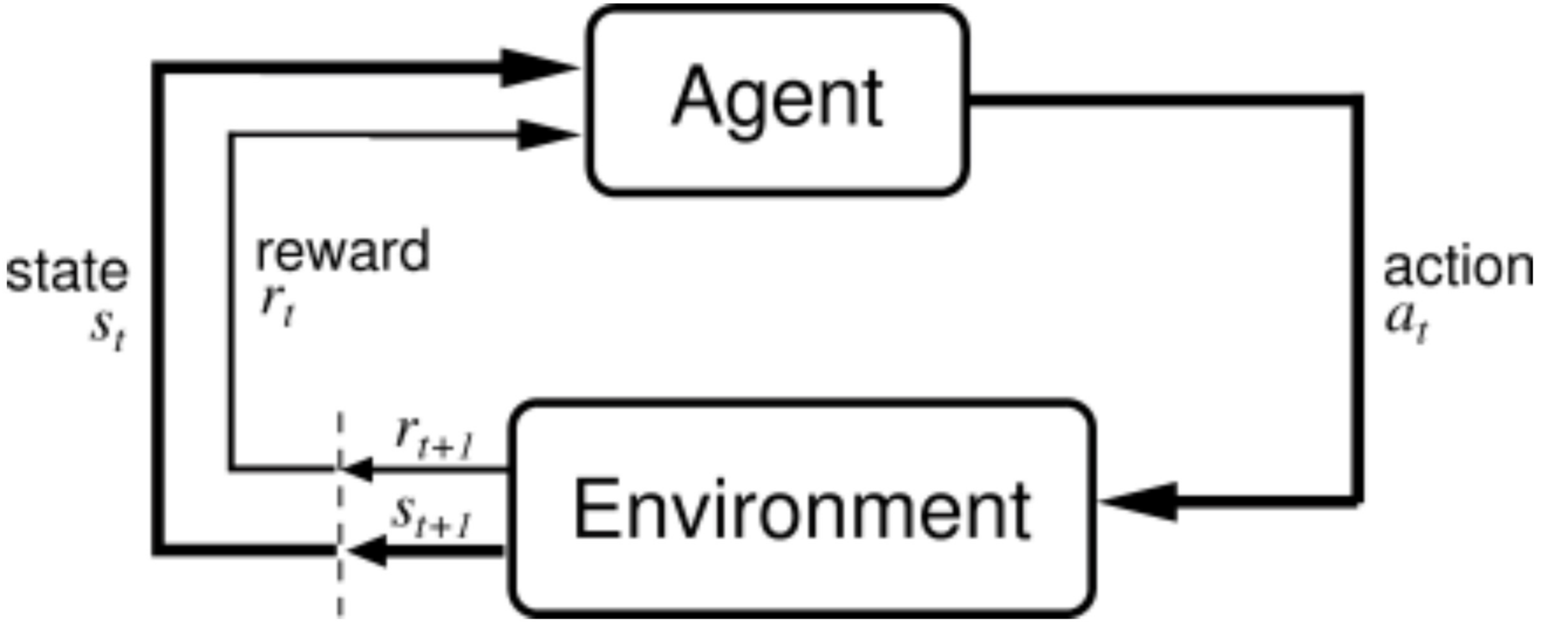
Autonomous Systems Lab

Roadmap


Control
Feedback control
Adaptive control
Optimal and learning control
Adaptive optimal control
Model-free RL
Model-based RL
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
DP
HJB / HJI

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:	$T(x_k \mid x_{k-1}, u_{k-1}) = p(x_k \mid x_{k-1}, u_{k-1})$	
Reward Function:	$r_t = R(x_k, u_k)$	
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: $u_k = \pi(x_k)$

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

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

Infinite Horizon MDPs

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

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

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

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

Solving infinite-horizon MDPs

If you know the model, use DP-ideas

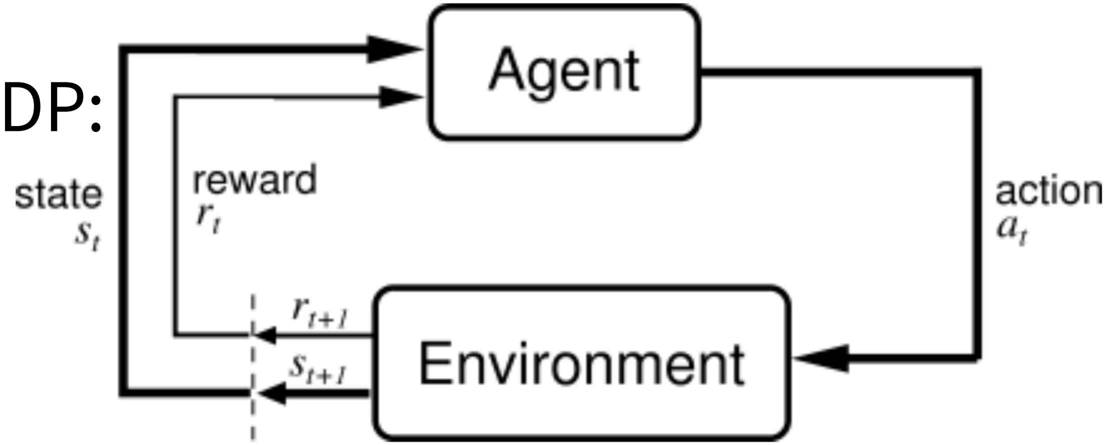
- Value Iteration / Policy Iteration (Covered in lecture 6)

RL: Learning from interaction

- Model-Based (related to system ID -- will see more later)
- Model-free
  - Value based (today)
  - 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:
- $\tau = (x_0, u_0, r_0, x_1, \dots, u_{H-1}, r_{H-1}, x_H)$



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

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

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

TD policy evaluation

Want to compute estimate of policy Q functions, Q_π

With $\alpha \in (0,1)$

- Sample (x_k, u_k, r_k, x_{k+1}) from MDP
- $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha\big(r_k + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)\big)$

Generalized policy iteration

Recall *generalized* policy iteration:

Loop:

- Perform *policy evaluation* step to estimate Q_π
- Perform *policy improvement* step using Q_π to yield π'
- Set $\pi \leftarrow \pi'$

SARSA

Combine TD policy evaluation step with

Policy improvement:

$$\pi'(x) = \ \operatorname{argmax}_u Q_\pi(x, u)$$

Greedy (with respect to Q function) policy improvement at each time step; thus will improve during online operation.

Q-learning

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

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

Thus, we aim to estimate Q^* from (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 probability ϵ, take a random action; otherwise take the most promising action

On-policy Q-learning algorithm

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

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

Loop:

Take action: $u_k \sim \pi(x_k)$.

Observe reward and next state: (r_k, x_{k+1}).

Update Q to minimize TD error:

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

$$k = k + 1$$

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:

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

learning rate

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

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

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!

Problems with imperfect state information

- Now the controller, instead of having perfect knowledge of the state, has access to observations $\boldsymbol{z}_k$ of the form
$$\boldsymbol{z}_0 = h_0(\boldsymbol{x}_0, \boldsymbol{v}_0), \qquad \boldsymbol{z}_k = h(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{v}_k)$$

- The random observation disturbance is characterized by a given probability distribution
$$P_{\boldsymbol{v}_k}(\cdot \mid \boldsymbol{x}_k, \dots, \boldsymbol{x}_0, \boldsymbol{u}_{k-1}, \dots, \boldsymbol{u}_0, \boldsymbol{w}_{k-1}, \dots, \boldsymbol{w}_0, \boldsymbol{v}_{k-1}, \dots, \boldsymbol{v}_0)$$

- The initial state $\boldsymbol{x}_0$ is also random and characterized by given $P_{\boldsymbol{x}_0}$

POMDP

- MDP with *observation model* $H(z|x,u)$
- Observations do not have Markov property: current observation does not provide same amount of info as history of all observations
- Includes systems with unknown parameters: often also called *Bayes-adaptive MDP*

Reduction to fully observed case

- Define the *information vector* as

$$\boldsymbol{I}_k = (\boldsymbol{z}_0, \ldots, \boldsymbol{z}_k, \boldsymbol{u}_0, \ldots, \boldsymbol{u}_{k-1}), \qquad \boldsymbol{I}_0 = \boldsymbol{z}_0$$

- Focus is now on *admissible* policies $\pi_k(\boldsymbol{I}_k) \in U_k$

- We want then to find an admissible policy that minimizes

$$J_\pi = E_{\substack{\boldsymbol{x}_0, \boldsymbol{w}_k, \boldsymbol{v}_k \\ k=0,\ldots,N-1}} \left[g_N(\boldsymbol{x}_N) + \sum_{k=0}^{N-1} g_k(\boldsymbol{x}_k, \pi_k(\boldsymbol{I}_k), \boldsymbol{w}_k) \right]$$

Solution strategies

1. Reformulation as a perfect state information problem (main idea: make the information vector the state of the system)
    - Main drawback: state has *expanding* dimension!
2. Reason in terms of sufficient statistics, i.e., quantities that ideally are smaller than $\boldsymbol{I}_k$ and yet summarize all its essential content
    - Main example: conditional probability distribution $P_{\boldsymbol{x}_k|\boldsymbol{I}_k}$ (assuming $\boldsymbol{v}_k \sim P_{\boldsymbol{v}_k}(\cdot\,|\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1}\,)$)
    - Condition probability distribution leads to a decomposition of the optimal controller in two parts:

$$\pi_k\left(P_{\boldsymbol{x}_k|\boldsymbol{I}_k}\right)$$

Actuator (π_k) — Estimator ($P_{\boldsymbol{x}_k|\boldsymbol{I}_k}$)

Dual control

- By performing DP in this “hyperstate”, one can find a controller that optimally probes/explores the system
- Practically, designing dual controllers is difficult, so sub-optimal exploration heuristics are used
- Active area of research: see Wittenmark, B. “Adaptive dual control,” (2008) for an introduction

Special case: LQG

Discrete LQG: find admissible control policy that minimizes

$$E\left[\boldsymbol{x}_N' Q \boldsymbol{x}_N + \sum_{k=0}^{N-1} (\boldsymbol{x}_k' Q \boldsymbol{x}_k + \boldsymbol{u}_k' R \boldsymbol{u}_k)\right]$$

subject to

- the dynamics $\boldsymbol{x}_{k+1} = A\boldsymbol{x}_k + B\boldsymbol{u}_k + \boldsymbol{w}_k$
- the measurement equation $\boldsymbol{z}_k = C\boldsymbol{x}_k + \boldsymbol{v}_k$

and with $\boldsymbol{x}_0, \{\boldsymbol{w}_k\}, \{\boldsymbol{v}_k\}$, independent and Gaussian vectors (and in addition $\{\boldsymbol{w}_k\}, \{\boldsymbol{v}_k\}$ zero mean)

LQG separation principle

LQG separation principle

LQG

- Have $x_k - E[x_k|I_k]$ independent of control actions $u_{0:k-1}$
- In fact, solution results in:
  - $\hat{x}_k = E[x_k|I_k]$ computed via Kalman filter
  - Optimal feedback $u_k = F_k\hat{x}_k$; F_k same as in LQR case
- We can design *state estimator* and *controller* independently
- Certainty-equivalent LQR control on estimated state is optimal dual controller---certainly not true in general!
- More proof details in lecture notes

Next time

- Nonlinearity: trajectory optimization, iterative LQR and DDP