AA203
Optimal and Learning-based Control

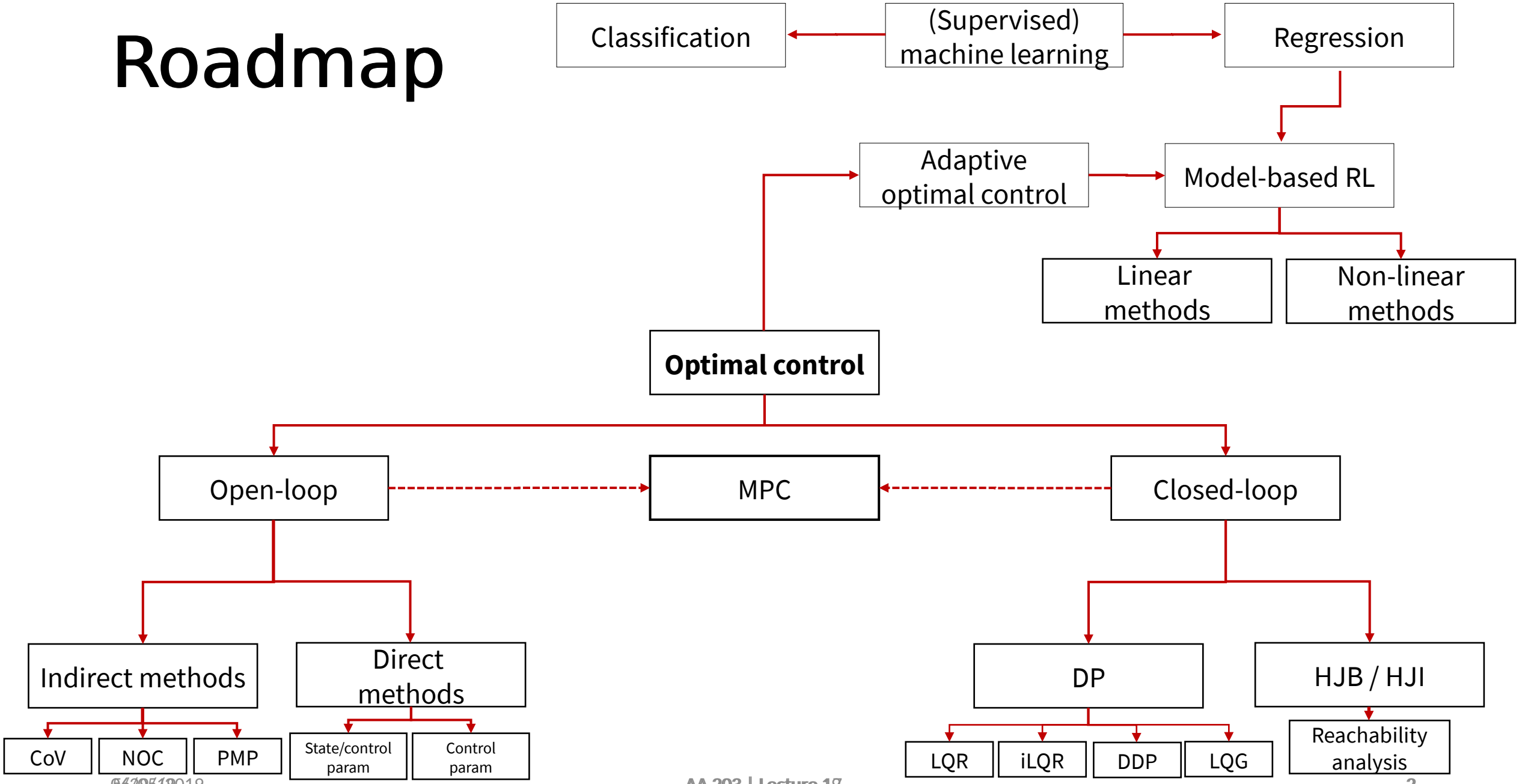
Intro to model-free RL, connections to model-based RL *

Stanford University

* Slides prepared by Roberto Calandra (rcalandra@fb.com)

Autonomous Systems Lab

Roadmap


Classification
(Supervised) machine learning
Regression
Adaptive optimal control
Model-based RL
Linear methods
Non-linear methods
Optimal control
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
CoV
NOC
PMP
State/control param
Control param
DP
HJB / HJI
LQR
iLQR
DDP
LQG
Reachability analysis

Today Lecture

- In previous lectures, we studied model-based RL.
- Today, we will look at model-free RL, and hybrid model-based/model-free algorithms
  - Introduction to model-free RL
  - Case study 1: Q-learning
  - Case study 2: Policy Gradient
  - Hybrid model-based/model-free algorithms
- After that, Marco's Course Recap

Model-free RL

- Aim at finding an optimal policy $\pi^* \quad \text{s.t.} \quad \pi^* = \arg\min_\pi \mathbb{E}[\sum_{t=0}^{T} c(\boldsymbol{x}_t, \boldsymbol{u}_t)]$
- Unlike model-based RL, it does **not** rely on using a dynamics model
- By far the most used type of RL
- Many different approaches within model-free RL:
  - Value-based
  - Actor-critic
  - Policy Search
  - …

Notation

- Notation in the RL literature:
  - Reward $r = -c$ (to be maximized)
  - State $x = s$
  - Action $u = a$
  - Induced trajectory τ
  - Future discounted reward $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$
- State value function $V(x)$
  in practice, it depends on the policy used $V^\pi(x) = E_\pi[R_t | x_t = x]$
- (State-)Action value function (more commonly, Q-function) $Q(x, u)$
  (Also depending on the policy) $Q^\pi(x, u) = E_\pi[R_t | x_t = x, u_t = u]$

Bellman Equations

- The importance of the Bellman equations is that they let us express values of states as values of other states.

$$x' = x_{t+1}\,, \quad u' = u_{t+1}$$

$$P^u_{xx'} = Pr(x_{t+1} = x' | x_t = x, u_t = u)$$

$$R^u_{xx'} = \mathbb{E}[r_{t+1} | x_t = x, x_{t+1} = x', u_t = u]$$

$$V^\pi(x) = \mathbb{E}_\pi[r_{t+1} + \gamma \sum_{k=0} \gamma^k r_| x_t = x]$$

$$Q^\pi(x,u) = \sum_{u'} P^u_{xx'}[R^u_{xx'} + \gamma \sum_{u'} \pi(x',u') Q^\pi(x',u')]$$

Q-learning

- Intuition:
  Value of an action = Immediate value + sum of all optimal future actions
- Optimal action-value function induced by the optimal policy π^*

  $Q^*(x_t, u_t) = E_\pi[R_t | x_t = x, u_t = u, \pi^*]$
- Iteratively update the value estimation of an action based on the reward we got and the reward we expect next (until $Q \to Q*$)

  $Q(x_t, u_t) = Q(x_t, u_t) + \alpha[r_{t+1} + \gamma \max_u Q(x_{t+1}, u) - Q(x_t, u_t)]$
- Does not explicitly learn a policy!
  Instead we optimize $\pi(x_t) = \arg\max_u Q(x, u)$
  (plus, usually, some exploration term)

Limitations and improvements of Q-learning

- (basic) Q-learning assumes discrete states and actions
- For continuous problems, we would need to discretize
  → this is typically impractical as it suffer severe scaling issues
- An alternative is to use a function approximator to estimate the Q-function

$$Q^*(\boldsymbol{x}, \boldsymbol{u}) \approx Q(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\theta})$$

- For example, when using a NN as function approximator, we obtain Deep Q-networks (DQN)
- Variations of Q-learning are very used in practice

Policy Search

- Consider an existing parametrized policy $\pi(\theta)$
- Directly optimize the parameters of the policy w.r.t. the cost

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} J^{\pi}(\boldsymbol{\theta})$$

- In theory, this can be done with your optimizer of choice
- Commonly zero-order or first-order optimizers
- Using first-order optimizer take the name of “Policy Gradient”

Policy Gradient

- Given $\theta^* = \arg\max_{\boldsymbol{\theta}} J^\pi(\boldsymbol{\theta})$ we want to compute the gradients $\nabla_{\boldsymbol{\theta}} J^\pi(\boldsymbol{\theta})$
- How do we compute them?
- Trivial solution is by finite-difference
  → Non-trivial in practice -- especially for stochastic systems
- Alternative are the so-called Likelihood Ratio Methods
  - Mega-trick $\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \nabla_\theta \log p_\theta(\tau)$
  - As a result

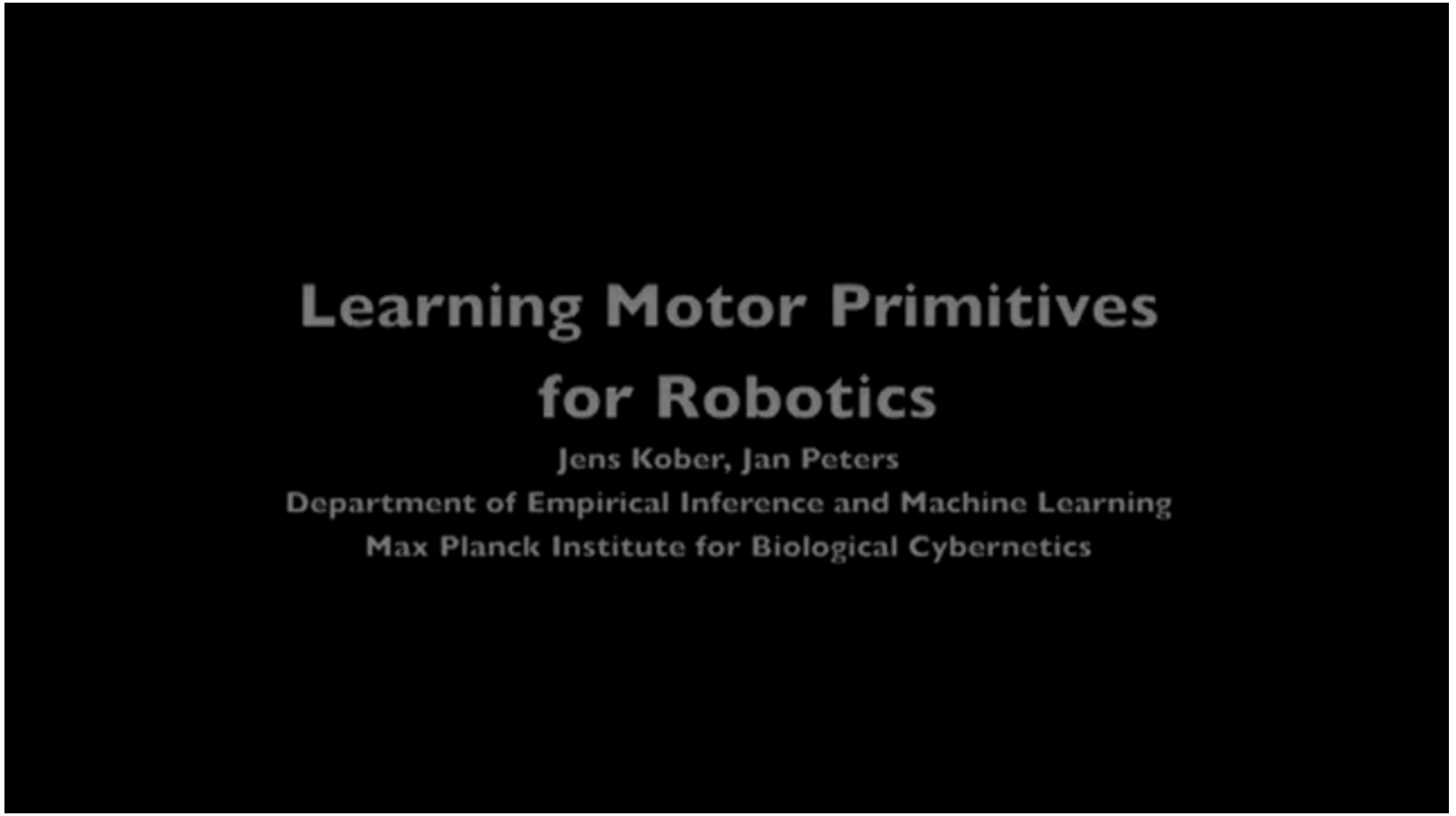
$$\nabla_\theta J^\pi(\theta) = \int_T \nabla_\theta p_\theta(\tau) c(\tau) d\tau = \int_T p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) c(\tau) d\tau = \mathbb{E}[\nabla_\theta \log p_\theta(\tau) c(\tau)]$$

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t|x_t) \text{ since } p_\theta(\tau) = p(x_0) \prod_{t=0}^{H} p(x_{t+1}|x_t, u_t) \pi_\theta(u_t|x_t)$$

REINFORCE algorithm

- Run the policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)$ and obtain the trajectories $\{\tau^i\}$
- Compute the gradients $\nabla_{\boldsymbol{\theta}} J^{\pi}(\boldsymbol{\theta}) \approx \sum_i (\sum_t \nabla_\theta \log \pi_\theta(u_t^i|x_t^i))(\sum_t c(x_t^i, u_t^i))$
- Update the parameters $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Policy Gradient



Strengths and limitations of Policy Gradient

- Effectively it is a first-order optimization problem
  - Can use techniques from optimization (e.g., natural gradients)
  - Local convergence guarantees
- Often more data-efficient than value-based methods
- Only local convergence
- Might still need a significant amount of trials

Strengths and limitations of model-free RL

- Simple to implement
- Computationally light
- Often works well in practice
- Does not generalize (to new tasks)
- Data-inefficient (might need millions of trials)
- Recent research results often difficult to reproduce
  (See talk from Joelle Pineau on "Reproducible, Reusable, and Robust Reinforcement Learning")

Integrate Model-based and Model-free

- Can we try to get the best of both world?
- Evidence from neuroscience that humans use both approaches!
- Multiple approaches to do this:
  - Use model-based first, later switch to model-free
  - Alternatively, can we use models to capture mode structure? If so, what to model?
- Currently a very active field of research.

How to choose which RL algorithm to use?

- Many important considerations:
  - Sample complexity
  - Implementation complexity
  - Computational cost
  - Deterministic/stochastic environment
  - Continuous/discrete state/actions space
  - on-policy/off-policy
  - …
- No silver bullet (No free lunch theorem), each algorithm comes with different trade-offs
- As designers, it is our role to chose the most appropriate one
  (plus most of these algorithms comes with hyper-parameters that need to be manually tuned…)

Summary

- Introduced the formalism of model-free RL and policy search
- Presented two popular model-free approaches:
  - Q-learning
  - Policy gradient
- Briefly discussed hybrid model-based/model-free approaches

Further Reading

- Sutton, R. S. & Barto, A. G. Reinforcement learning: An introduction MIT press, 2018, second edition
- Deisenroth, M. P.; Neumann, G. & Peters, J. A Survey on Policy Search for Robotics. Foundations and Trends in Robotics, 2013, 2, 1-142
- Peters, J. & Schaal, S. Reinforcement learning of motor skills with policy gradients Neural networks, Elsevier, 2008, 21, 682-697
- Gläscher, J.; Daw, N.; Dayan, P. & O'Doherty, J. P. States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning Neuron, Elsevier, 2010, 66, 585-595
- Nagabandi, A.; Kahn, G.; Fearing, R. S. & Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning IEEE International Conference on Robotics and Automation (ICRA), 2018, 7559-7566
- Gu, S.; Lillicrap, T.; Sutskever, I. & Levine, S. Continuous deep q-learning with model-based acceleration International Conference on Machine Learning, 2016, 2829-2838
- Wilson, A.; Fern, A. & Tadepalli, P. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning Journal of Machine Learning Research (JMLR), 2014, 15, 253-282

Marco’s Course Recap