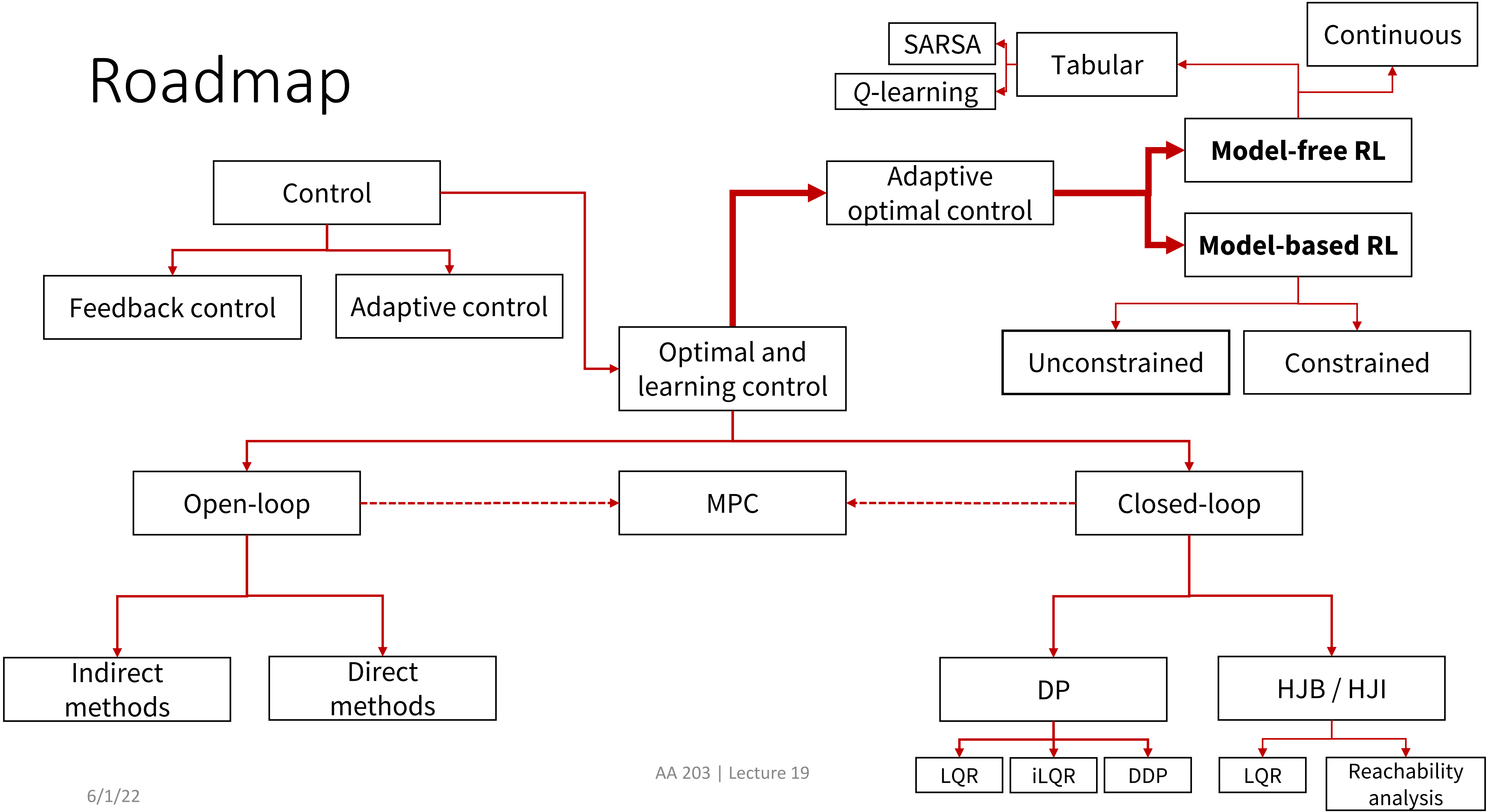
AA203
Optimal and Learning-based Control

Combining model and policy learning

Stanford University

Autonomous Systems Lab

Roadmap



Control

Feedback control

Adaptive control

Optimal and learning control

Adaptive optimal control

Model-free RL

Tabular

SARSA

Q-learning

Continuous

Model-based RL

Unconstrained

Constrained

Open-loop

MPC

Closed-loop

Indirect methods

Direct methods

DP

LQR

iLQR

DDP

HJB / HJI

LQR

Reachability analysis

Combining MB and MF RL ideas

- Review model-based RL
- Combining model and policy learning in the tabular setting
- Combinations in the nonlinear setting

- Readings:
  - R. Sutton and A. Barto. *Reinforcement Learning: An Introduction,* 2018.
  - Several papers, referenced throughout.

Review: model-based RL

```
Choose initial policy $\pi_\theta$
Loop over episodes:
    Get initial state $x$
    Loop until end of episode:
        $u \leftarrow \pi_\theta(x)$
        Take action $u$ in environment, receive next state $x'$ and reward $r$
        Update model based on $x, u, x', r$
        Update policy $\pi_\theta$ based on updated model
        $x \leftarrow x'$
```

Dyna: combining model-free and model-based RL

(Tabular) Dyna-Q:

```
Init $Q(x,u), model(x,u)$ for all $x,u$; initialize state $x$
Loop forever:
    $u \leftarrow argmax_u Q(x,u)$   (possibly with exploration)
    Take action $u$ in environment, receive next state $x'$ and reward $r$
    $Q(x,u) \leftarrow Q(x,u) + \alpha[r + \gamma \max_{u'} Q(x',u') - Q(x,u)]$   (Q-learning with real data)
    $model(x,u) \leftarrow x', r$   (Learning a model)
    For $n = 1, \dots, N$:
        $x,u \leftarrow$ random previously observed state/action pair
        $x', r \leftarrow model(x,u)$
        $Q(x,u) \leftarrow Q(x,u) + \alpha[r + \gamma \max_{u'} Q(x',u') - Q(x,u)]$   (Q-learning with sim data)
```

Dyna: combining model-free and model-based RL

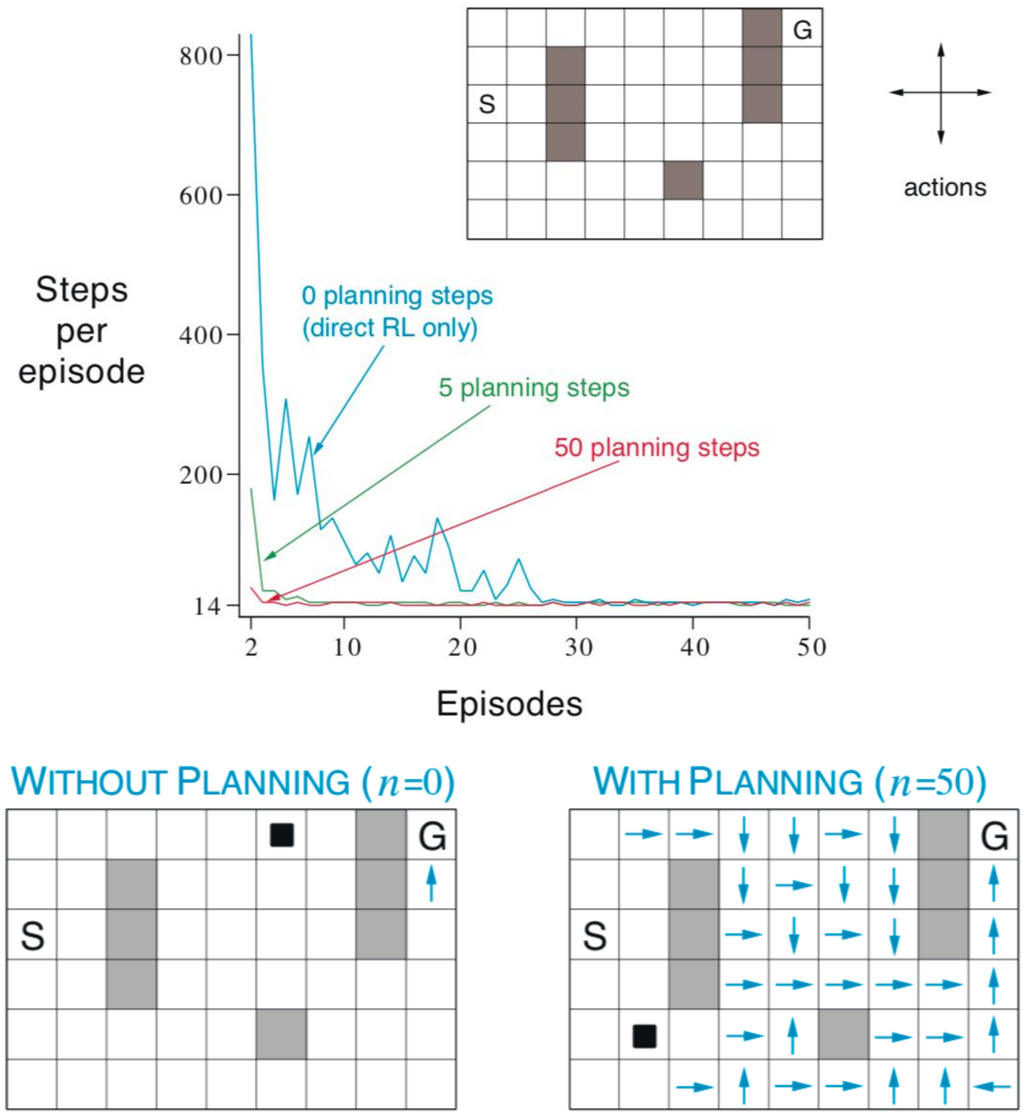
(Tabular) Dyna-Q:

```
Init $Q(x,u), model(x,u)$ for all $x, u$; initialize state $x$
Loop forever:
    $u \leftarrow argmax_u Q(x,u)$   (possibly with exploration)
    Take action $u$ in environment, receive next state $x'$ and reward $r$
    $Q(x,u) \leftarrow Q(x,u) + \alpha[r + \gamma \max_{u'} Q(x',u') - Q(x,u)]$        (Q-learning with real data)
    $model(x,u) \leftarrow x', r$        (Learning a model)
    For $n = 1, \ldots, N$:
        $x, u \leftarrow$ random previously observed state/action pair
        $x', r \leftarrow model(x,u)$
        $Q(x,u) \leftarrow Q(x,u) + \alpha[r + \gamma \max_{u'} Q(x',u') - Q(x,u)]$ (Q-learning with sim data)
```

(side note: this is not unlike experience replay; the hope is that a structured “model” might generalize better)

Dyna performance: deterministic maze

- Main idea of Dyna: interleave simulated and real experience in policy optimization.
    - Learned model allows you to propagate Q function updates back throughout state space, i.e., allows for planning
- Allows early model-based training acceleration, without performance limitations of model-based methods.
- Many “Dyna style” algorithms
    - MF policy optimization + learning a model + MB policy optimization



[Sutton & Barto, 2018.]

How to optimize policy?

Question: what should policy be?

	Tabular MDP	**Continuous MDP**
Limited horizon open loop	Monte Carlo tree search or search of finite horizon action sequence	Model predictive control
Closed-loop policy optimization	Dynamic programming: value iteration or policy iteration	**Main focus of today's lecture**

Why do limited search? Typically, if policy optimization is too expensive.

- Example: game of Go or other very large MDPs

Policy optimization with models

- Want to optimize π_θ via

$$\theta^* = \mathrm{argmax}_\theta \mathrm{E}_{x_0}[V^{\pi_\theta}(x_0)]$$

Approach: fit model $f_\phi(x, u)$, define value w.r.t. this model as

$$V^{\pi, f}(x) = \sum_t \mathrm{E}_{x_t \sim f, u_t \sim \pi}[r(x_t, u_t)]$$

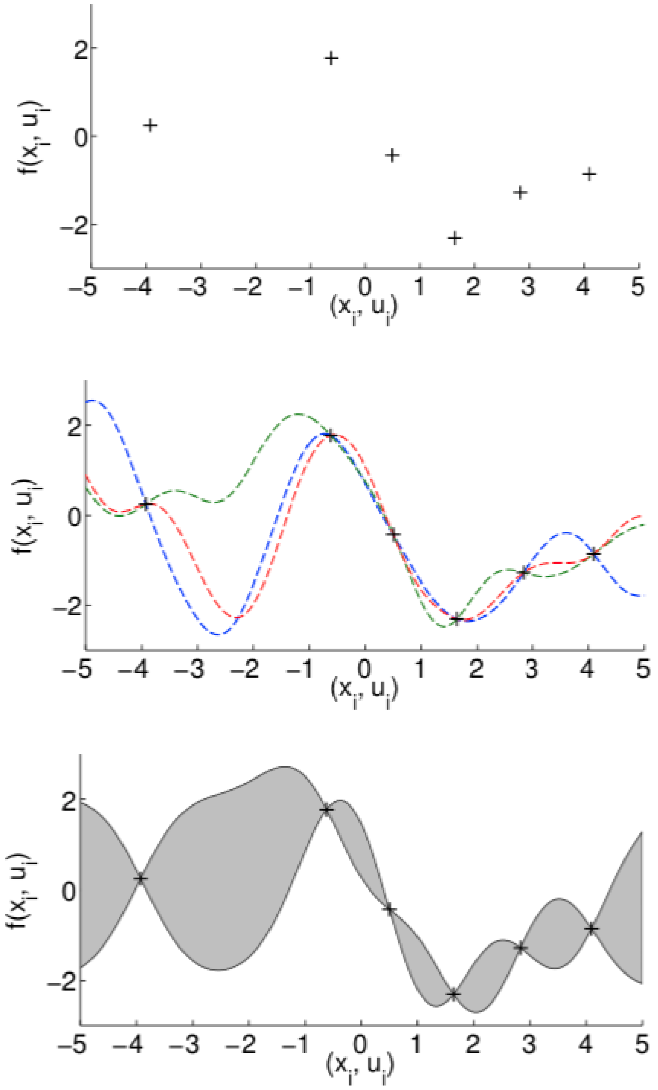
Want to compute gradient of this value w.r.t. policy parameters:

$$\theta \leftarrow \theta + \alpha \nabla_\theta V^{\pi_\theta, f_\phi}(x)$$

Case study: PILCO

Deisenroth and Rasmussen, *Probabilistic inference for learning control,* ICML 2011.

- Approach: use Gaussian process for dynamics model
    - Gives measure of *epistemic* uncertainty
    - Extremely sample efficient
- Pair with arbitrary (possibly nonlinear) policy
- By propagating the uncertainty in the transitions, capture the effect of small amount of data

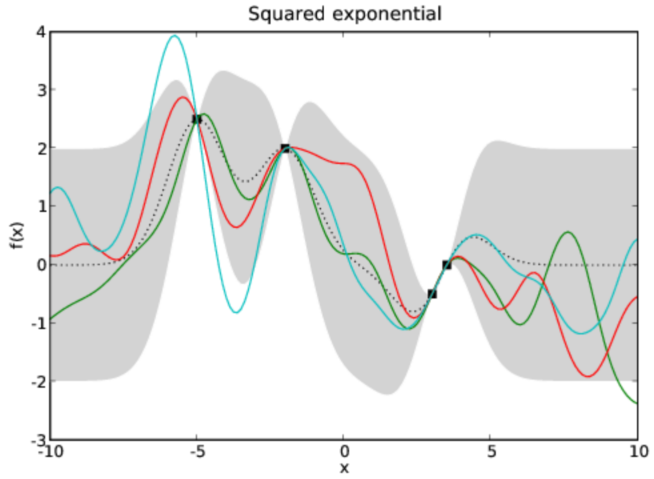


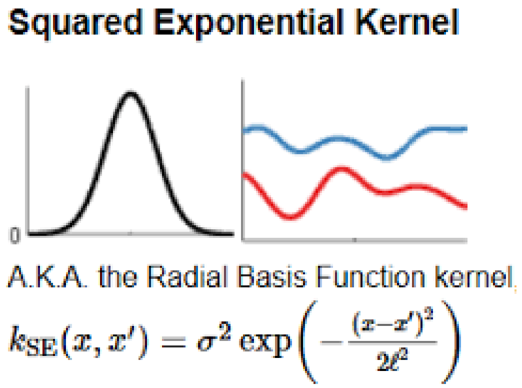
GP reminder

- Gaussian processes: Gaussian distributions over functions
- Typically, initialize with zero mean; behavior determined entirely by **kernel**

$$cov(x, x') = k(x, x')$$

- Standard kernel choice: squared exponential, used in PILCO
    - Has smooth interpolating behavior




Squared Exponential Kernel

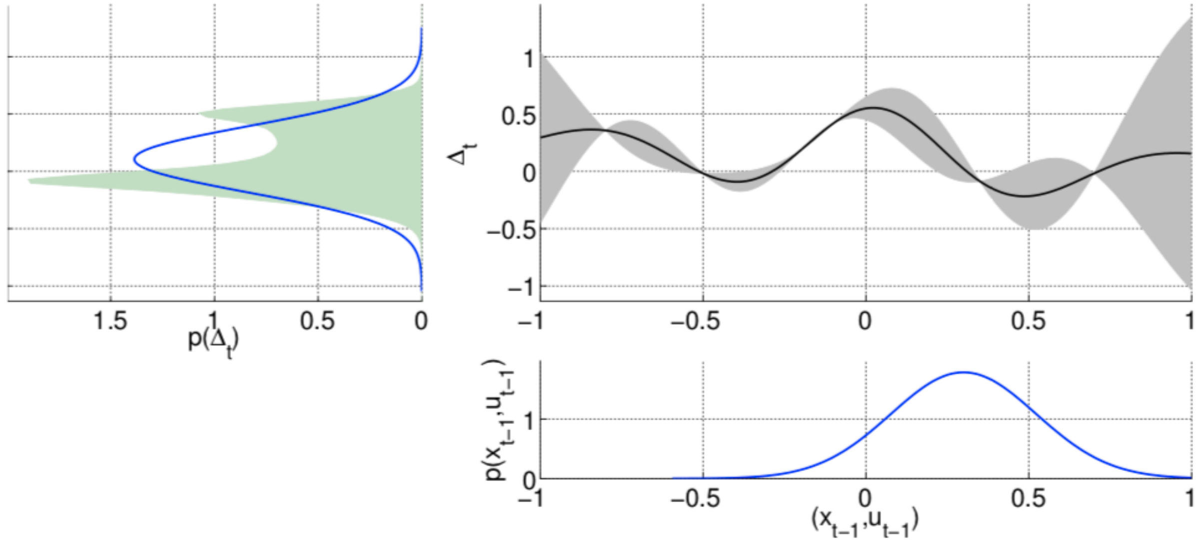
A.K.A. the Radial Basis Function kernel

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$$

Uncertainty propagation

- For GP conditioned on data, one step prediction is Gaussian
- But, need to make multistep predictions: so, need to derive multi-step predictive distribution
- Turn to approximating distribution at each time with a Gaussian via *moment matching*

$$p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{u}_{t-1}) = \mathcal{N}(\mathbf{x}_t \mid \mu_t, \boldsymbol{\Sigma}_t)\,,$$
$$\mu_t = \mathbf{x}_{t-1} + \mathbb{E}_f[\Delta_t]\,,$$
$$\boldsymbol{\Sigma}_t = \mathrm{var}_f[\Delta_t]\,.$$



Uncertainty propagation

All algorithm design choices made to ensure analytical tractability:

- Because of the squared exponential kernel, mean and variance can be computed in closed form
- Choose cost

$$c(\mathbf{x}) = 1 - \exp(-\|\mathbf{x} - \mathbf{x}_{\text{target}}\|^2/\sigma_c^2)$$

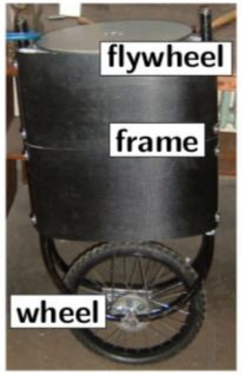
  which is similarly squared exponential; thus expected cost can be computed exactly, factoring in uncertainty.

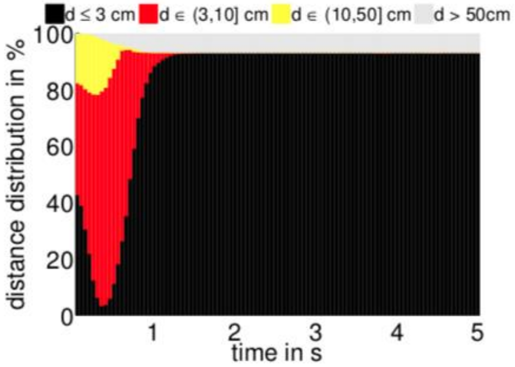
- Choose also radial basis function or linear policy, to enable analytical uncertainty propagation

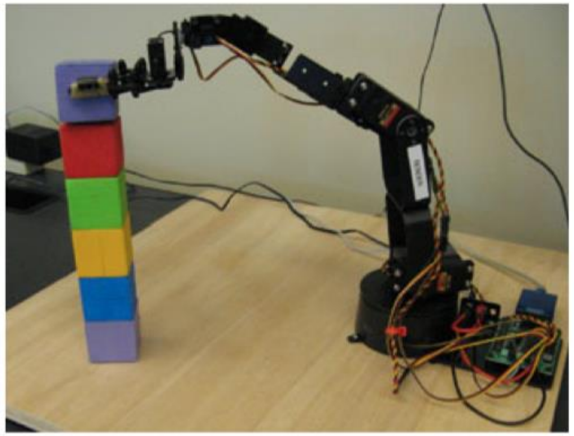
PILCO Summary

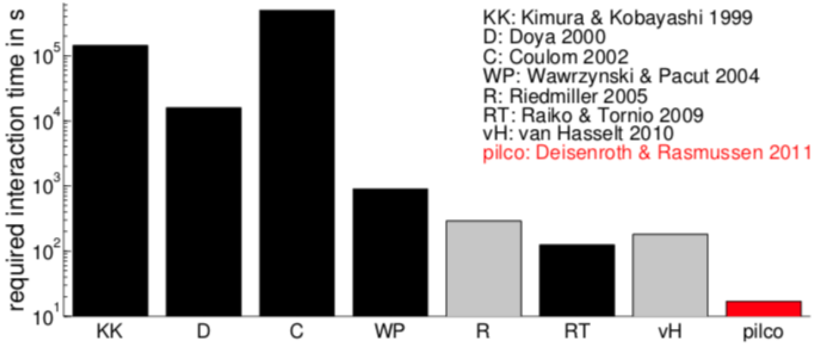
- Uncertainty prop: leverage specific functional forms to derive analytical expressions for mean and variance of trajectory under policy.
- Can use chain rule (aka backprop through time) to compute the gradient of expected total cost w.r.t. policy parameters
- Algorithm:
  - Roll out policy to get new measurements; update model
  - Compute (locally) optimal policy via gradient descent
    - This policy is “local” in the sense of the data we’ve given it, i.e., it’s tailored to the regions of state space it’s seen before; this is more general than “local” in the sense of linearization
  - Repeat

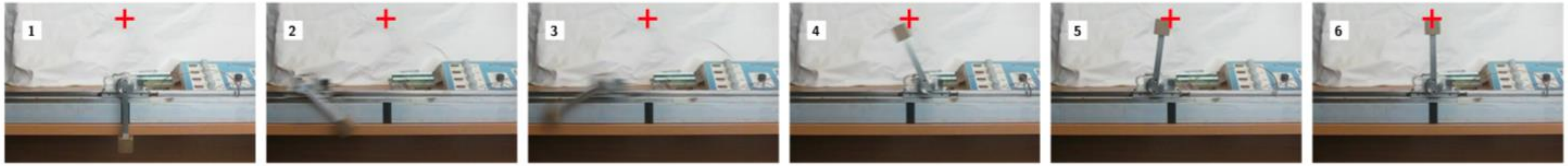
PILCO results







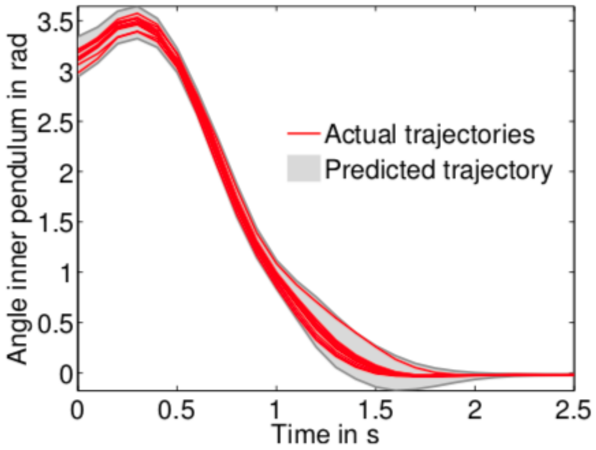


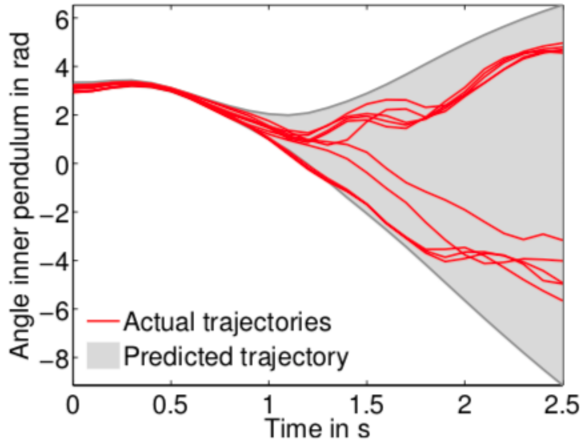


For more results and algorithm info: Deisenroth, Fox, and Rasmussen, *Gaussian Processes for Data-Efficient Learning in Robotics and Control*, TPAMI 2015.

PILCO limitations

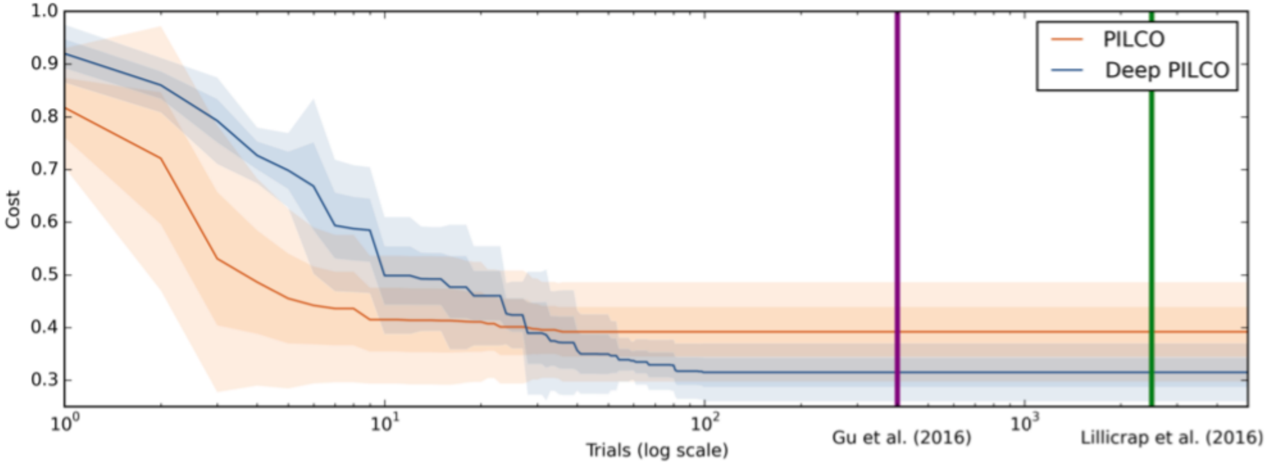
- Treatment of uncertainty
  - Propagates uncertainty via moment matching, so can't handle multi-modal outcomes
  - Limited in choice of kernel function
  - Doesn't capture temporal correlation
- Efficiency
  - GPs are extremely data efficient; however, *very* slow
  - Policy optimization (done after every rollout) can take on the order of ~1h





What about the same principles with neural network models?

- McHutchon, *Modelling nonlinear dynamical systems with Gaussian processes,* PhD thesis, 2014: particle propagation (alternative to moment matching) performs poorly.
- Gal, McAllister, Rasmussen, *Improving PILCO with Bayesian neural network dynamics models*, 2017.
    - Use a Bayesian network that provides samples from posterior
    - Again use moment matching; this time not necessary for analytical variance computation, but for performance – “Gaussianization” has a strong regularizing effect by decorrelating samples across time



For much deeper discussion of gradient estimation with particles, see:
Parmas, Rasmussen, Peters, Doya, *PIPPS: Flexible model-based policy search robust to the curse of chaos,* ICML 2018.

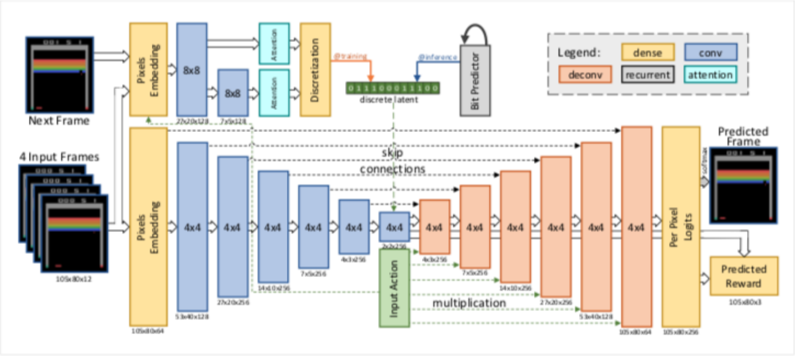
Policy optimization via backpropagation through neural network dynamics

Diving deeper on the challenges with sampling:

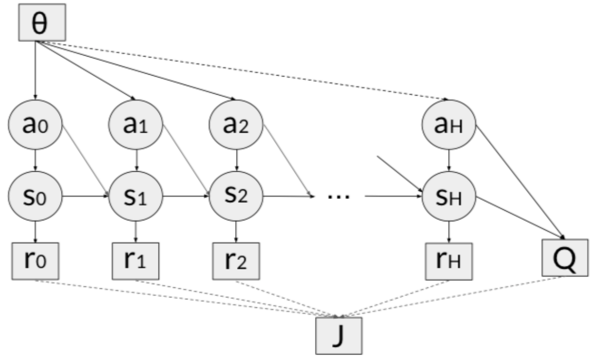
- Backpropagate through computation graph of dynamics and policy
- Same instability as shooting methods in trajectory optimization
  - However, in shooting methods, each time step is an independent action
- Here, the policy is the same at each time step: so very small changes in policy **dramatically** change trajectory
  - Accumulated gradients become very large as you backprop further
  - Similar to exploding/vanishing gradient problems in recurrent NNs

How to workaround this sensitivity problem?

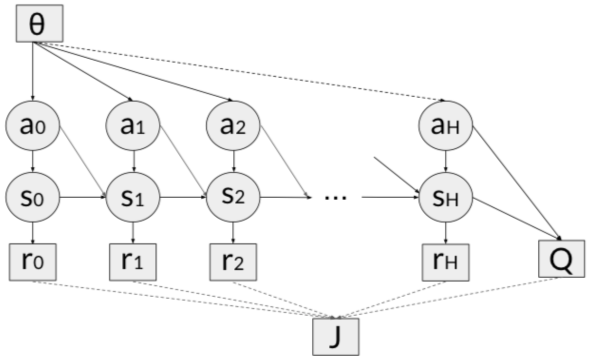
- Solution 1: use policy gradient from model-free RL
  - E.g., policy gradient algorithm such as A2C, TRPO, PPO, etc.
  - Doesn’t require multiplying many Jacobians, which leads to large gradient
  - Example: Kaiser, et al. “Model-Based RL for Atari,” ICLR 2020.
    - Uses video prediction model + PPO



- Solution 2: use value function for tail return
  - Value function now used not just for variance reduction, but sensitivity reduction as well
  - Example: Clavera, Fu, Abbeel, “Model-augmented actor critic: Backpropagating through paths,” ICLR 2020.
    - Stochastic policy and dynamics: estimate gradient via *pathwise derivative* (involves dynamics explicitly, unlike score function gradient estimator, i.e., REINFORCE)



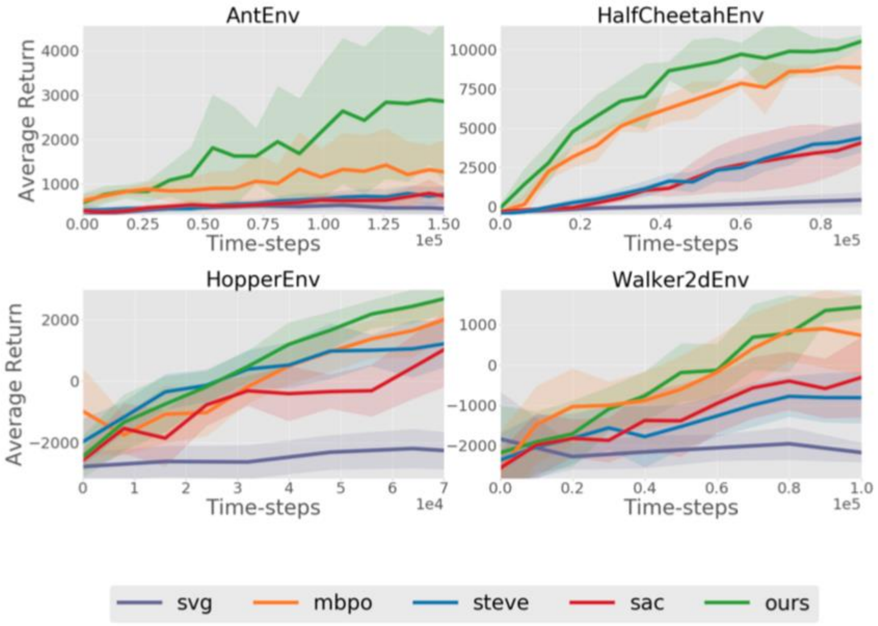
Solution 2: Use value function for tail return



- Clavera, Fu, Abbeel, *Model-augmented actor critic: Backpropagating through paths,* ICLR 2020.
- Stochastic policy and dynamics: compute gradient via pathwise derivative

$$J_\pi(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t) + \gamma^H \hat{Q}(s_H, a_H)\right]$$

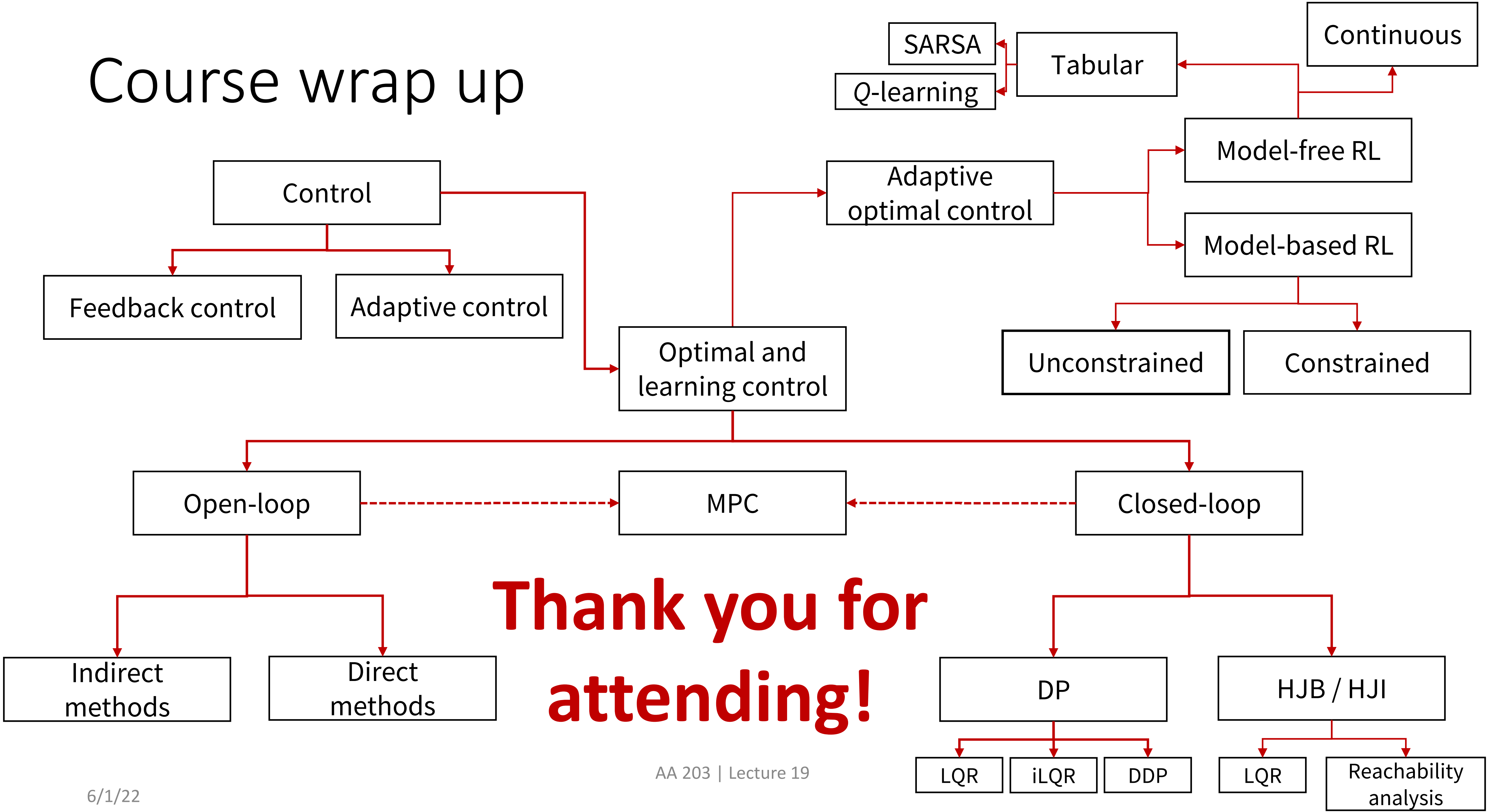
- Use ensemble of dynamics models, two Q functions, Dyna-style training



Combining model and policy learning

- Discussed two possible solutions; infinitely many more
- Very busy research direction! Many topics not covered here
  - Many possible combinations of planning/control, policies, values, and models
- Quite practical: model learning is data efficient and parameterized policy is cheap to evaluate at run time

Course wrap up



- Control
  - Feedback control
  - Adaptive control
  - Optimal and learning control
    - Adaptive optimal control
      - Model-free RL
        - Tabular
          - SARSA
          - *Q*-learning
        - Continuous
      - Model-based RL
        - Unconstrained
        - Constrained
    - Open-loop
      - Indirect methods
      - Direct methods
    - MPC
    - Closed-loop
      - DP
        - LQR
        - iLQR
        - DDP
      - HJB / HJI
        - LQR
        - Reachability analysis

Thank you for attending!