

Week 9: Model-based Reinforcement Learning

Bolei Zhou

The Chinese University of Hong Kong

November 2, 2020

Today's Plan

- ① Introduction to model-based reinforcement learning
- ② Model-based value optimization
- ③ Model-based policy optimization
- ④ Case study on robot object manipulation

Model-based Reinforcement Learning

- ① Previous lectures on model-free RL
 - ① Learn policy directly from experience through policy gradient
 - ② Learn value function through MC or TD
- ② This lecture will be on model-based RL
 - ① learn model of the environment from experience

Model-based and Model-free RL

① Model-free RL

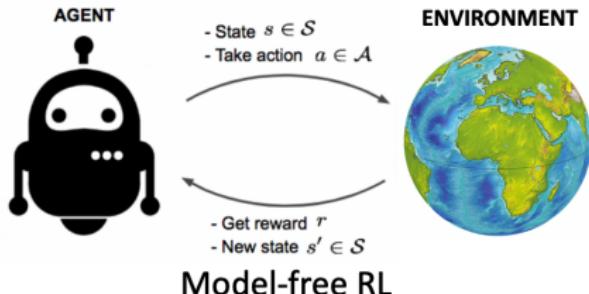
- ① No model
- ② Learn value/policy functions from experience

② Model-based RL

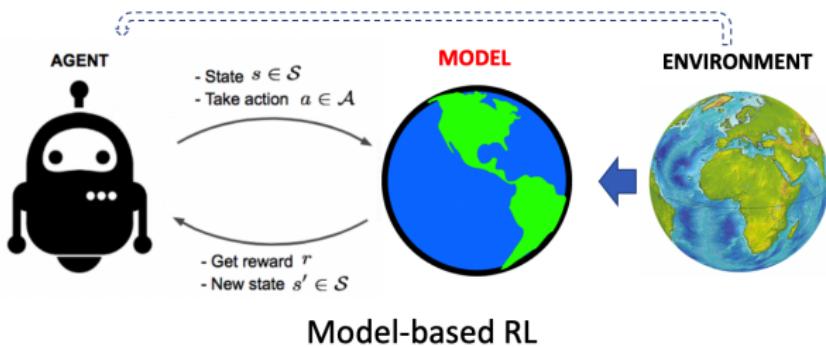
- ① Besides learn policy function or value function from the experience, also learn a model from experience
- ② Plan value/policy functions from model

Building a Model of the Environment

① Diagram of model-free reinforcement learning

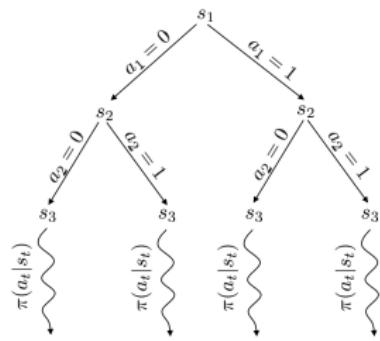


② Diagram of model-based reinforcement learning



Modeling the Environment for Planning

- ① Plan to better interact with the real environment



Modeling the Environment for Planning

- ① Planning is the computational process that takes a model as input and produces or improves a policy by interacting with the modeled environment

experience $\xrightarrow{\text{learning}}$ model $\xrightarrow{\text{planning}}$ better policy

- ② State-space planning: search through the state space for an optimal policy or an optimal path to a goal
- ③ Model-based **value optimization** methods share a common structure

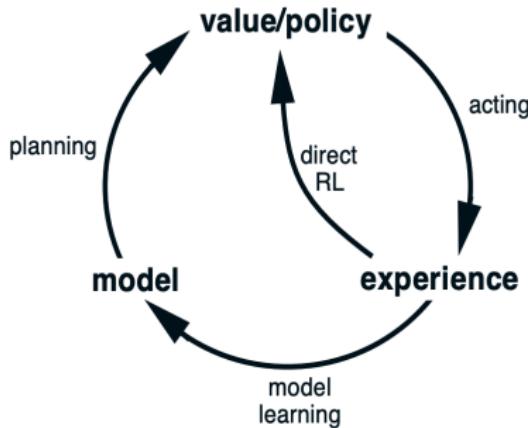
model \rightarrow simulated trajectories $\xrightarrow{\text{backups}}$ values \rightarrow policy

- ④ Model-based **policy optimization** methods have a simpler structure as

model \rightarrow policy

Structure of the Model-based RL

- ① Relationships among learning, planning and acting



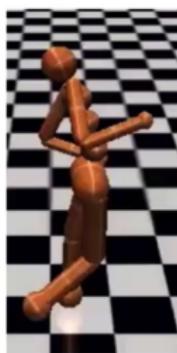
- ② Two roles of the real experience:

- ① Improve the value and policy directly using previously methods
- ② Improve the model to match the real environment more accurately (predictive model on the environment): $p(s_{t+1}|s_t, a_t)$, $R(s_t, a_t)$

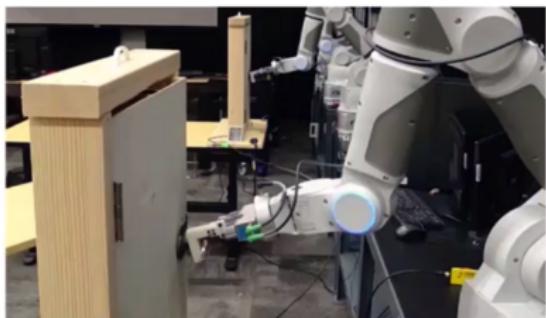
Advantage of Model-based RL

- ① Pros: Better sample efficiency

Simulation



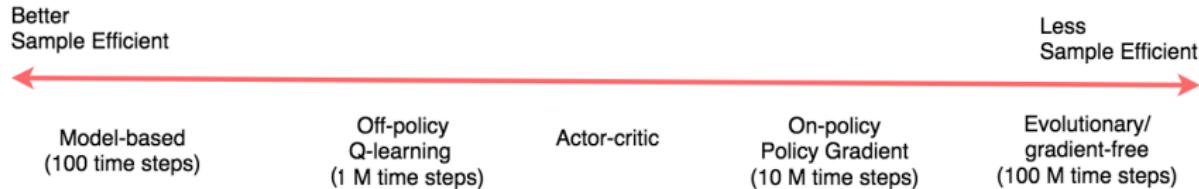
Real-world



- ① Sample-efficient learning is crucial for real-world RL applications such as robotics
DARPA robotics failure
- ② Model can be learned efficiently by supervised learning methods

Advantage of Model-based RL

① Better sample efficiency



② Cons:

- ① First learning a model then constructing a value function or policy function leads to two sources of approximation error
- ② Difficult to come up with guarantee of convergence

What is a Model

- ① A model \mathcal{M} is a representation of an MDP parameterized by η
- ② Usually a model $\mathcal{M} = (\mathcal{P}, \mathcal{R})$ represents state transitions and rewards

$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1}|S_t, A_t)$$

$$R_{t+1} = \mathcal{R}_\eta(R_{t+1}|S_t, A_t)$$

- ③ Typically we assume conditional independence between state transitions and rewards

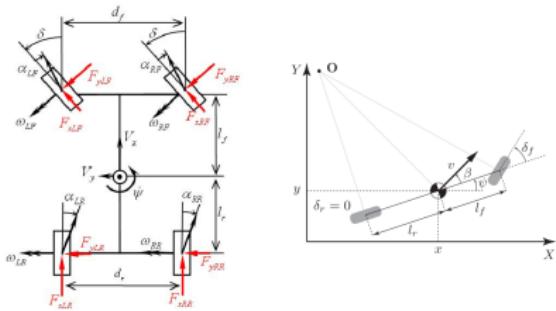
$$P(S_{t+1}, R_{t+1}|S_t, A_t) = P(S_{t+1}|S_t, A_t)P(R_{t+1}|S_t, A_t)$$

Sometimes it is easy to access the model

- ① Known models: Game of Go: the rule of the game is the model



- ② Physics models: Vehicle dynamics model and kinematics bicycle model



Today's Plan

- ① Intro on model-based reinforcement learning
- ② Model-based value optimization
- ③ Model-based policy optimization
- ④ Case study on robot object manipulation

Learning the Model

- ① Goal: learn model \mathcal{M}_η from experience $\{S_1, A_1, R_2, \dots, S_T\}$
 - ① So consider it as a supervised learning problem

$$S_1, A_1 \rightarrow R_2, S_2$$

$$S_1, A_1 \rightarrow R_2, S_2$$

⋮

$$S_1, A_1 \rightarrow R_2, S_2$$

- ② Learning $s, a \rightarrow r$ is a regression problem
- ③ Learning $s, a \rightarrow s'$ is a density estimation problem
- ④ Pick a loss function, e.g., mean-squared error, KL divergence then optimize η that minimize the empirical loss

Examples of Models

- ① Table Lookup Model
- ② Linear Expectation Model
- ③ Linear Gaussian Model
- ④ Gaussian Process Model
- ⑤ Deep Belief Network Model ...

Table Lookup Model

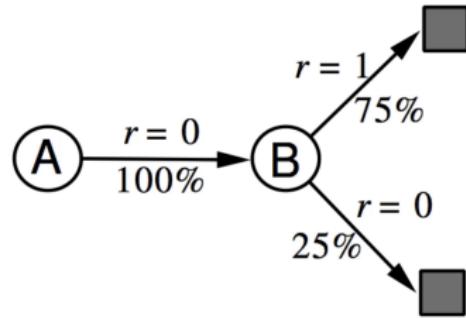
- ① Model is an explicit MDP, $\hat{\mathcal{P}}$ and $\hat{\mathcal{R}}$
- ② Count visits $N(s,a)$ to each state action pair

$$\hat{\mathcal{P}}_{s,s'}^a = \frac{1}{N(s, a)} \sum_{t=1}^T \mathbf{1}(S_t = s, A_t = a, S_{t+1} = s')$$

$$\hat{\mathcal{R}}_s^a = \frac{1}{N(s, a)} \sum_{t=1}^T \sum_{t=1}^T \mathbf{1}(S_t = s, A_t = a) R_t$$

Example of AB

- ① Two states A and B; no discounting;
- ② Observed 8 episodes of experience:
 - ① (State, Reward, Next State, Next Reward...)
 - ② (A, 0, B, 0), (B, 1), (B, 1), (B, 1), (B, 1), (B, 1), (B, 1), (B, 0)
- ③ So the estimated a table lookup model from the experience as follows



Sample-Based Planning

- ① A simple but sample-efficient approach to planning
- ② Use the model only to generate samples
- ③ General procedure:
 - ① Sample experience from the model

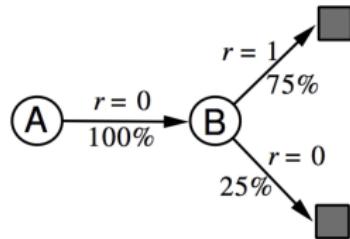
$$S_{t+1} \sim \mathcal{P}_\eta(S_{t+1}|S_t, A_t)$$

$$R_{t+1} = \mathcal{R}_\eta(R_{t+1}|S_t, A_t)$$

- ② Apply model-free RL to sampled experiences:
 - ① Monte-Carlo control
 - ② Sarsa
 - ③ Q-learning

Sample-Based Planning for AB Example

- ① Observed 8 episodes of experience in the format of (State, Reward, Next State, Next Reward...)
 - ① (A, 0, B, 0), (B, 1), (B, 1), (B, 1), (B, 1), (B, 1), (B, 1), (B, 0)
- ② Construct the model



- ③ Sample experience from the model
 - ① (B, 1), (B, 0), (B, 1), (A, 0, B, 1), (B, 1), (A, 0, B, 1), (B, 1), (B, 0)
- ④ Monte-Carlo Learning on the sampled experience
 - ① $V(A) = 1, V(B) = 0.75$

Planning with an Inaccurate Model

- ① Given an imperfect model $\langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle \neq \langle \mathcal{P}, \mathcal{R} \rangle$
- ② Performance of model-based RL is limited to the optimal policy for approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
 - ① Model-based RL is only as good as the estimated model
- ③ When the model is inaccurate, planning process will compute a suboptimal policy
- ④ Possible solutions:
 - ① When the accuracy of the model is low, use model-free RL
 - ② Reason explicitly about the model uncertainty (how confident we are for the estimated state): Use probabilistic model such as Bayesian and Gaussian Process

Real and Simulated Experience

- ① We now have two sources of experience
- ② **Real experience:** sampled from the environment (true MDP)

$$S', S \sim \mathcal{P}_{s,s'}^a$$

$$R = \mathcal{R}_s^a$$

- ③ **Simulated experience:** sampled from the model (approximate MDP)

$$\hat{S}', \hat{S} \sim \mathcal{P}_\eta(S'|S, A)$$

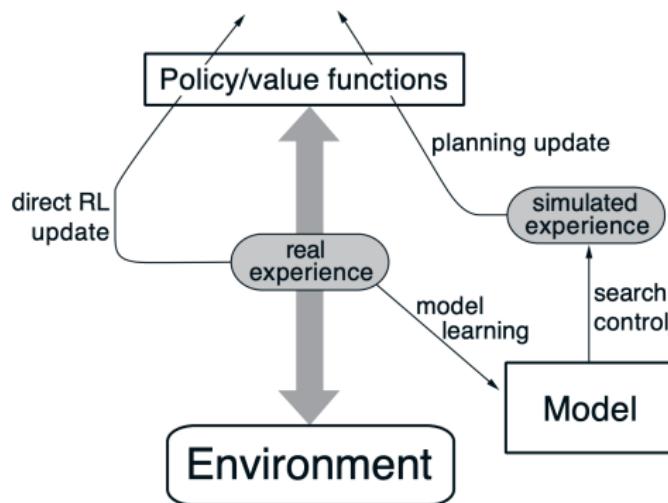
$$\hat{R} = \mathcal{R}_\eta(R|S, A)$$

Integrating Learning and Planning

- ① Model-free RL
 - ① No model
 - ② Learn value function (and/or policy) from real experience
- ② Model-based RL (using Sample-based Planning)
 - ① Learn a model from real experience
 - ② Plan value function (and/or policy) from simulated experience
- ③ Dyna
 - ① Learn a model from real experience
 - ② **Learn and plan value** function (and/or policy) from **both** real and simulated experience

Dyna for Integrating Learning, Planning, and Reacting

① Architecture of Dyna



- ② By Richard Sutton. ACM SIGART Bulletin 1991
- ③ Chapter 8 of the Textbook

Algorithm of Dyna

Tabular Dyna-Q

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

Do forever:

- (a) $S \leftarrow$ current (nonterminal) state
- (b) $A \leftarrow \epsilon\text{-greedy}(S, Q)$
- (c) Execute action A ; observe resultant reward, R , and state, S'
- (d) $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
- (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
- (f) Repeat n times:

$S \leftarrow$ random previously observed state

$A \leftarrow$ random action previously taken in S

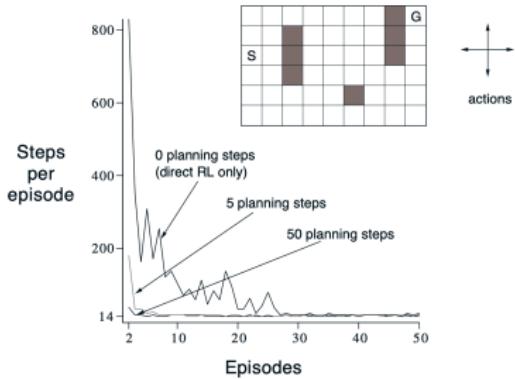
$R, S' \leftarrow Model(S, A)$

$Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$

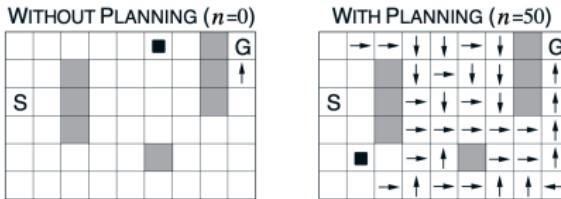
① Combining direct RL, model learning, and planning together

Result of Dyna

- ① A simple maze environment: travel from S to G as quickly as possible
- ② learning curves varying the number of planning steps per real step



- ③ Policies found by planning and nonplanning Dyna-Q agents



Today's Plan

- ① Intro on model-based reinforcement learning
- ② Model-based value optimization
- ③ Model-based policy optimization
- ④ Case study on robot object manipulation

Policy Optimization with Model-based RL

- ① Previous model-based value-based RL:

model → simulated trajectories $\xrightarrow{\text{backups}}$ values → policy

- ② Can we optimize the policy and learn the model directly, without estimating the value?

model $\xrightarrow{\text{improves}}$ policy

Model-based Policy Optimization in RL

- ① Policy gradient, as a model-free RL, only cares about the policy $\pi_\theta(a_t|s_t)$ and expected return

$$\tau = \{s_1, a_1, s_2, a_2, \dots, s_T, a_T\} \sim \pi_\theta(a_t|s_t)$$

$$\arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_t \gamma^t r(s_t, a_t) \right]$$

- ② In policy gradient, no $p(s_{t+1}|s_t, a_t)$ is needed (no matter it is known or unknown)

$$p(s_1, a_1, \dots, s_T, a_T) = p(s_1) \prod_{t=1}^T \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$$

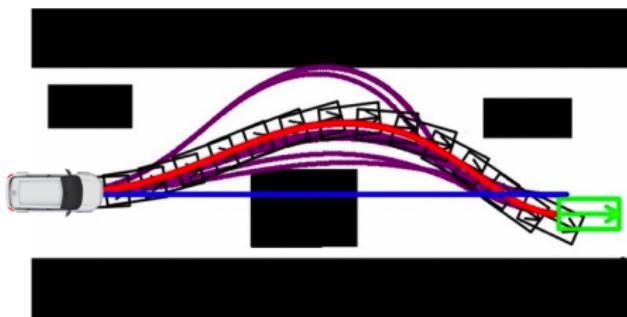
- ③ But can we do better if we know the model or are able to learn the model?

Model-based Policy Optimization in RL

- ① Model-based policy optimization in RL has a strong influence from the control theory, that optimizes a controller
- ② The controller uses the model, also termed as the system dynamics $s_t = f(s_{t-1}, a_{t-1})$, to decide the optimal controls for a trajectory to minimize the cost:

$$\arg \min_{a_1, \dots, a_T} \sum_{t=1}^T c(s_t, a_t) \text{ subject to } s_t = f(s_{t-1}, a_{t-1})$$

Optimal Control for Trajectory Optimization



$$\min_{a_1, \dots, a_T} \sum_{t=1}^T c(s_t, a_t) \text{ subject to } s_t = f(s_{t-1}, a_{t-1})$$

- ① If the dynamics is known it becomes the optimal control problem
- ② Cost function is the negative reward of the RL problem
- ③ The optimal solution can be solved by Linear-Quadratic Regulator (LQR) and iterative LQR (iLQR) under some simplified assumptions

Model Learning for Trajectory Optimization: Algorithm 1

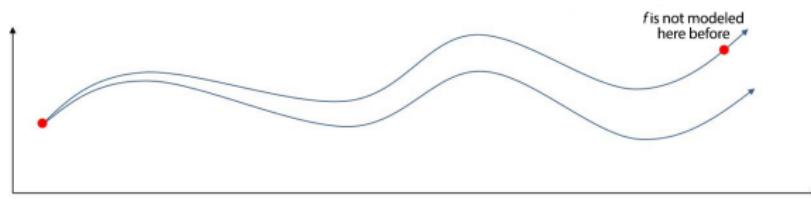
- ① If the dynamics model is unknown, we can combine model learning and trajectory optimization

② Algorithm 1

- ① run base policy $\pi_0(a_t|s_t)$ (random policy) to collect $\mathcal{D} = \{(s, a, s')\}_i$
- ② learn dynamics model $s' = f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s'_i\|^2$
- ③ plan through $f(s, a)$ to choose actions
- ④ Step 2 is supervised learning to train a model to minimize the least square error from the sampled data
- ⑤ Step 3 can be solved by Linear Quadratic Regulator (LQR), to calculate the optimal trajectory using the model and a cost function

Model Learning for Trajectory Optimization: Algorithm 2

- ① The previous solution is vulnerable to drifting, a tiny error accumulates fast along the trajectory
- ② We may also land in areas where the model has not been learned yet



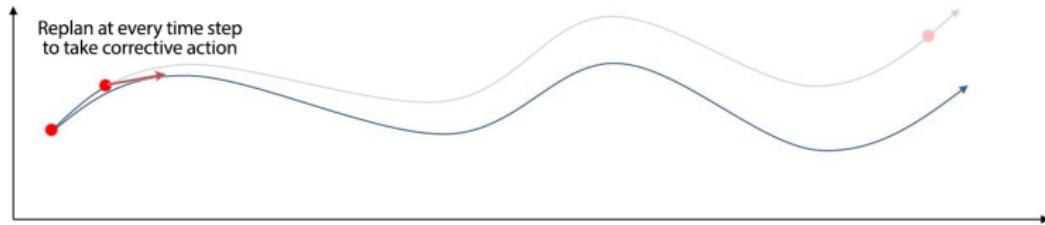
- ③ So we have the following improved algorithm with learning the model *iteratively*

④ Algorithm 2

- ① run base policy $\pi_0(a_t|s_t)$ (random policy) to collect $\mathcal{D} = \{(s, a, s')_i\}$
- ② Loop
 - ① learn dynamics model $s' = f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s'_i\|^2$
 - ② plan through $f(s, a)$ to choose actions
 - ③ execute those actions and add the resulting data $\{(s, a, s')_i\}$ to \mathcal{D}

Model Learning for Trajectory Optimization: Algorithm 3

- ① Nevertheless, the previous method executes all planned actions before fitting the model again. We may be off-grid too far already
- ② So we can use Model Predictive Control (MPC) that we optimize the whole trajectory but we take the first action only, then we observe and replan again
- ③ In MPC, we optimize the whole trajectory but we take the first action only. We observe and replan again. The replan gives us a chance to take corrective action after observed the current state again



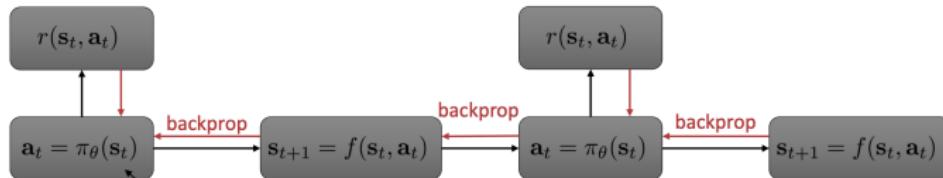
Model Learning for Trajectory Optimization: Algorithm 3

① Algorithm 3 with MPC

- ① run base policy $\pi_0(a_t|s_t)$ to collect $\mathcal{D} = \{(s, a, s')_i\}$
- ② Loop every N steps
 - ① learn dynamics model $s' = p(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s'_i\|^2$
 - ② Loop each step
 - ① plan through $f(s, a)$ to choose actions
 - ② execute the first planned action and observe the resulting state s' (MPC)
 - ③ append (s, a, s') to dataset \mathcal{D}

Model Learning for Trajectory Optimization: Algorithm 4

- ① Finally we can plug the policy learning along with model learning and optimal control



② Algorithm 4: Learning Model and Policy Together

- ① run base policy $\pi_0(a_t|s_t)$ (random policy) to collect $\mathcal{D} = \{(s, a, s')\}_i\}$
- ② Loop

- ① learn dynamics model $f(s, a)$ to minimize $\sum_i \|f(s_i, a_i) - s'_i\|^2$
- ② backpropagate through $f(s, a)$ into the policy to optimize $\pi_\theta(a_t|s_t)$
- ③ run $\pi_\theta(a_t|s_t)$, appending the visited (s, a, s') to \mathcal{D}

Parameterizing the Model

What function is used to parameterize the dynamics?

- ① Global model: $s_{t+1} = f(s_t, a_t)$ is represented by a big neural network
 - ① Pro: very expressive and can use lots of data to fit
 - ② Con: not so great in low data regimes, and cannot express model uncertainty
- ② Local model: model the transition as time-varying linear-Gaussian dynamics
 - ① Pro: very data-efficient and can express model uncertainty
 - ② Con: not great with non-smooth dynamics
 - ③ Con: very slow when dataset is big
- ③ Local model as time-varying linear-Gaussian dynamics

$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t))$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t$$

- ① All we needed are the local gradients $A_t = \frac{df}{d\mathbf{x}_t}$ and $B_t = \frac{df}{d\mathbf{u}_t}$

Global Model versus Local Model

① Local model as time-varying linear-Gaussian

$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t))$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t$$

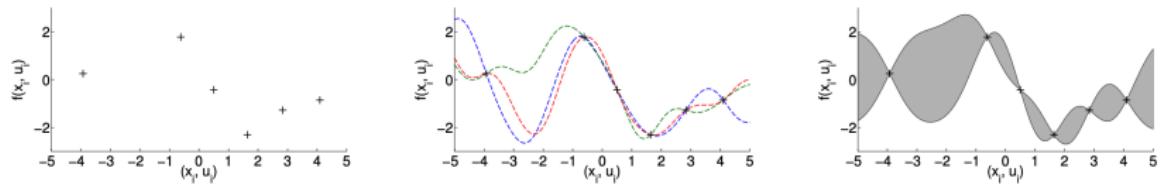


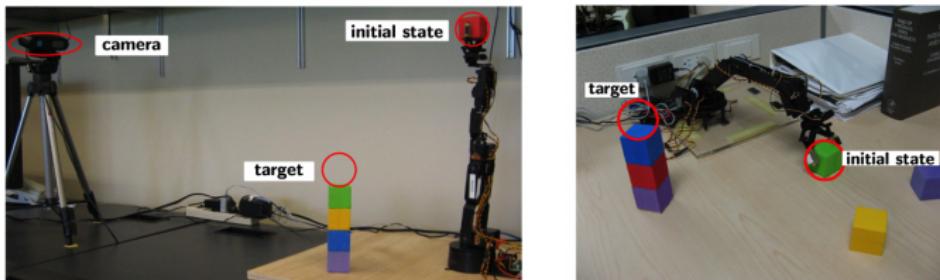
Figure 1. Small data set of observed transitions (left), multiple plausible deterministic function approximators (center), probabilistic function approximator (right). The probabilistic approximator models uncertainty about the latent function.

Today's Plan

- ① Intro on model-based reinforcement learning
- ② Model-based value optimization
- ③ Model-based policy optimization
- ④ Case study on robot object manipulation

Case Study 1: Model-based Robotic Object Manipulation

- ① Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning. RSS 2011



- ② No pose feedback, visual feedback from a Kinetics-type depth camera
- ③ Total cost: $\$500 = 6\text{-degree Arm}(\$370) + \text{Kinetics}(\$130)$
- ④ System setup:
 - ① Control signal $u \in R^4$: Pulse widths for the first four motors
 - ② State $x \in R^3$: 3D center of the object
 - ③ Policy $\pi : R^3 \rightarrow R^4$
 - ④ Expected return $J^\pi = \sum_{t=0}^T \mathbb{E}_{x_t}[c(x_t)]$ where $c = -\exp(-d^2/\sigma_c^2)$

Case Study 1: Model-based Robotic Object Manipulation

- ① Model the system dynamics as probabilistic non-parametric Gaussian process GP

Algorithm 1 PILCO

```
1: init: Set controller parameters  $\psi$  to random.  
2: Apply random control signals and record data.  
3: repeat  
4:   Learn probabilistic GP dynamics model using all data  
5:   repeat           ▷ Model-based policy search  
6:     Approx. inference for policy evaluation: get  $J^\pi(\psi)$   
7:     Gradients  $dJ^\pi(\psi)/d\psi$  for policy improvement  
8:     Update parameters  $\psi$  (e.g., CG or L-BFGS).  
9:   until convergence; return  $\psi^*$   
10:  Set  $\pi^* \leftarrow \pi(\psi^*)$ .  
11:  Apply  $\pi^*$  to robot (single trial/episode); record data.  
12: until task learned
```

- ② PILCO: A model-based and data-efficient approach to policy search.
Deisenroth and Rasmussen. ICML 2011
- ③ Demo link: <http://mlg.eng.cam.ac.uk/pilco/>

Case Study 2: Model-based Robotic Object Manipulation

- ① Learning Contact-Rich Manipulation Skills with Guided Policy Search.
Sergey Levine and Pieter Abbeel. The best Robotics Manipulation Paper award at ICRA 2015
- ② One of Sergey Levine's representative works

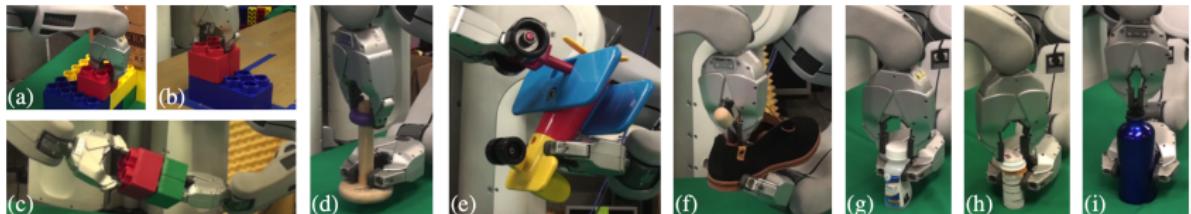


Fig. 2: Tasks in our experiments: (a) stacking large lego blocks on a fixed base, (b) onto a free-standing block, (c) held in both gripper; (d) threading wooden rings onto a tight-fitting peg; (e) assembling a toy airplane by inserting the wheels into a slot; (f) inserting a shoe tree into a shoe; (g,h) screwing caps onto pill bottles and (i) onto a water bottle. Videos are included with the supplementary material and at <http://rll.berkeley.edu/icra2015gps/index.htm>.

Case Study 2: Model-based Robotic Object Manipulation

① Local models + Iterative LQR

- ① Linear-Gaussian controller: $p(\mathbf{u}_t | \mathbf{x}_t) = \mathcal{N}(K_t \mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$
- ② Time-varying linear-Gaussian dynamics:
$$p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t} \mathbf{x}_t + f_{\mathbf{u}t} \mathbf{u}_t, \mathbf{F}_t)$$
- ③ Can be solved as linear-quadratic-Gaussian (LQG) problem using optimal control

② Guided policy search for global model:

- ① policy model: π_θ
- ② supervised learning of neural network using the guide of the linear-Gaussian controller

Algorithm 1 Guided policy search with unknown dynamics

```
1: for iteration  $k = 1$  to  $K$  do
2:   Generate samples  $\{\tau_i^j\}$  from each linear Gaussian
      controller  $p_i(\tau)$  by running it on the robot
3:   Minimize  $\sum_{i,t} \lambda_{i,t} D_{\text{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$ 
      with respect to  $\theta$  using samples  $\{\tau_i^j\}$ 
4:   Update  $p_i(\mathbf{u}_t|\mathbf{x}_t)$  using the LQG-like method
5:   Increment each of the dual variables  $\lambda_{i,t}$  by
       $\alpha D_{\text{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$ 
6: end for
7: return optimized policy parameters  $\theta$ 
```

Case Study 2: Model-based Robotic Object Manipulation

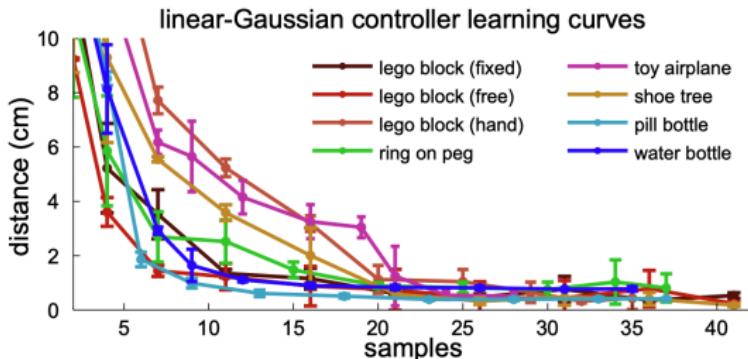


Fig. 2: Tasks in our experiments: (a) stacking large lego blocks on a fixed base, (b) onto a free-standing block, (c) held in both gripper; (d) threading wooden rings onto a tight-fitting peg; (e) assembling a toy airplane by inserting the wheels into a slot; (f) inserting a shoe into a shoe; (g,h) screwing caps onto pill bottles and (i) onto a water bottle. Videos are included with the supplementary material and at <http://rll.berkeley.edu/icra2015gps/index.htm>.

- 1 Demo link: <https://www.youtube.com/embed/mSzEyKaJTSU>

Summary of Model-based RL



- ① Instead of fitting a policy or a value function, we develop a model to predict the system dynamics
- ② Model-based RL has much higher sample efficiency, which is crucial for real-world applications such as robotic manipulation

