Lecture 15: Model-based Reinforcement Learning

Bolei Zhou

The Chinese University of Hong Kong

bzhou@ie.cuhk.edu.hk

March 24, 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

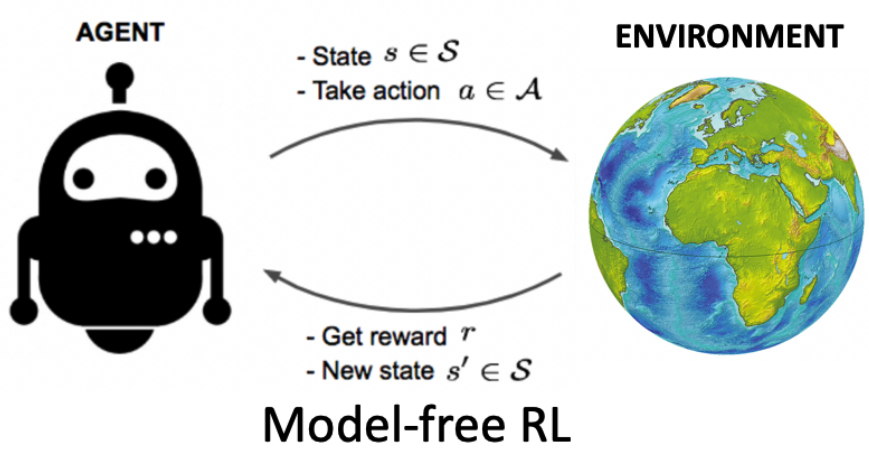
1. Previous lectures on model-free RL
   1. Learn policy directly from experience through policy gradient
   2. Learn value function through MC or TD
2. This lecture will be on model-based RL
   1. learn model of the environment from experience

Model-based and Model-free RL

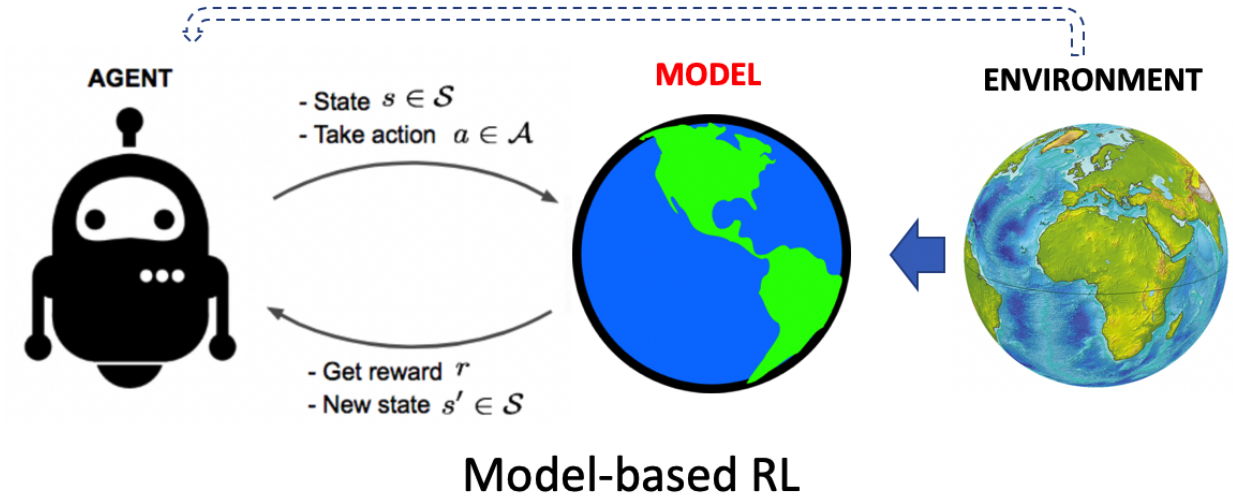
1. Model-free RL
   1. No model
   2. Learn value/policy functions from experience
2. Model-based RL
   1. Besides learn policy function or value function from the experience, also learn a model from experience
   2. Plan value/policy functions from model

Building a Model of the Environment

1. Diagram of model-free reinforcement learning



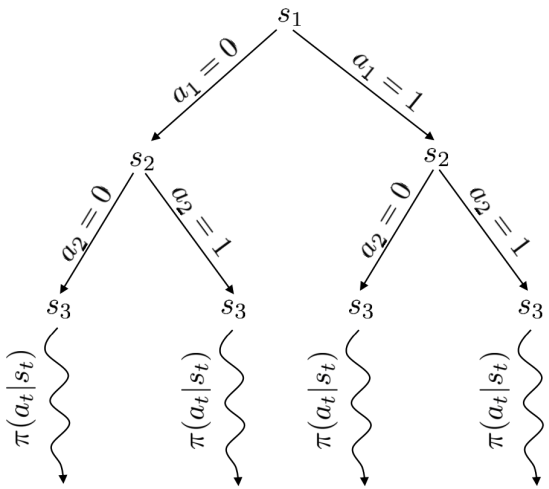
2. Diagram of model-based reinforcement learning



Modeling the Environment for Planning

1. Plan to better interact with the real environment

s_t a_t



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

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

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 RL methods share a common structure

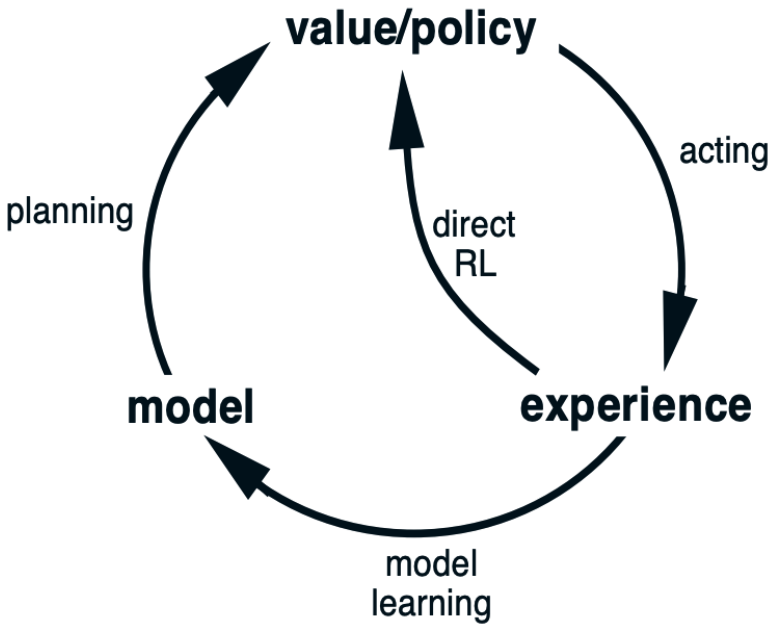
$$\text{model} \rightarrow \text{simulated environment} \xrightarrow{\textit{backups}} \text{values} \rightarrow \text{policy}$$

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

$$\text{model} \rightarrow \text{policy}$$

Structure of the Model-based RL

1. Relationships among learning, planning and acting

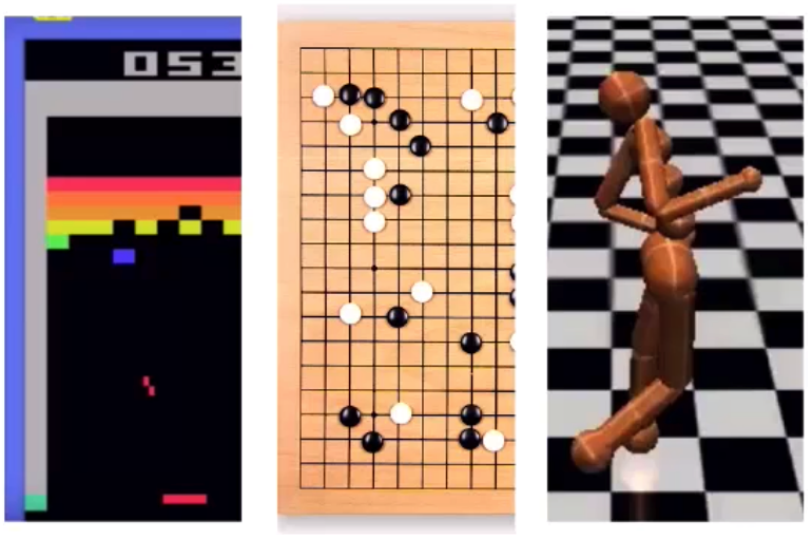


2. Two roles of the real experience:
   1. Improve the value and policy directly using previously methods
   2. Improve the model to match the real environment more accurately (predictive model on the environment): $p(s_{t+1}|s_t, a_t)$, $R(s_t, a_t)$)

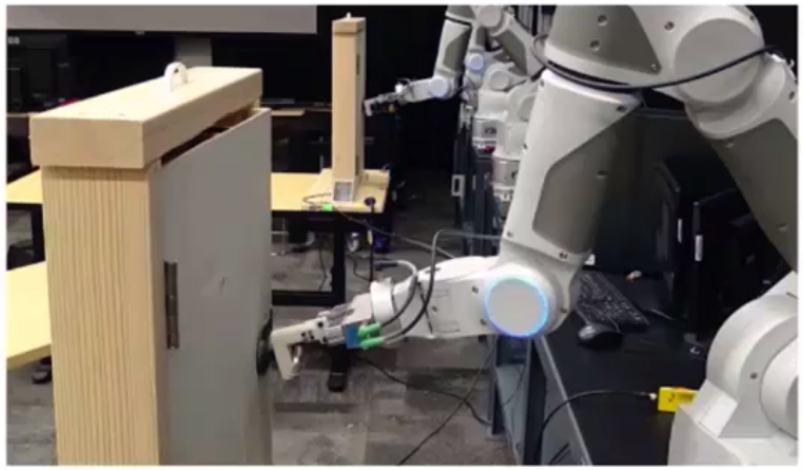
Advantage of Model-based RL

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

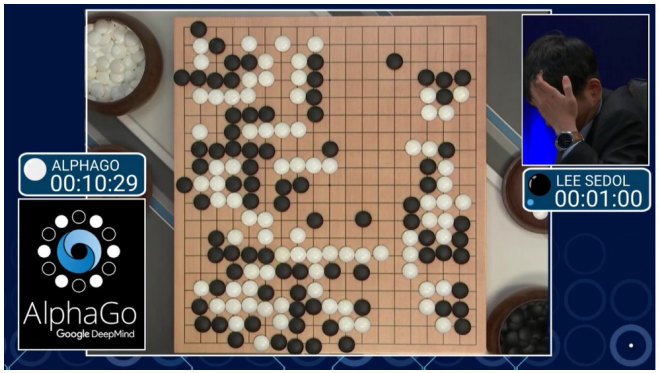
$$
\begin{aligned}
S_{t+1} &\sim \mathcal{P}_\eta(S_{t+1}|S_t, A_t) \\
R_{t+1} &= \mathcal{R}_\eta(R_{t+1}|S_t, A_t)
\end{aligned}
$$

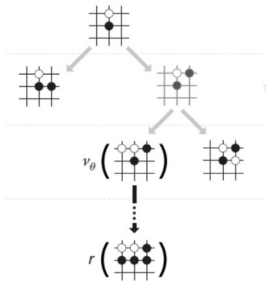
3. Typically we assume conditional independence between state transitions and rewards

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

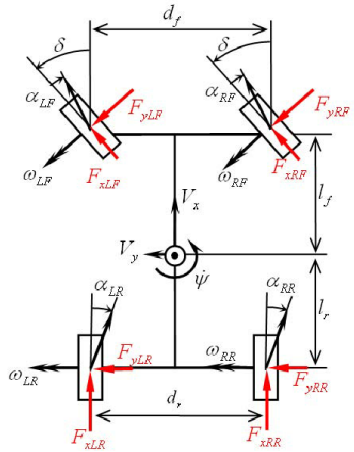
Sometimes it is easy to access the model

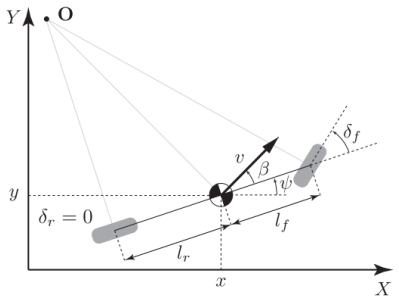
1. Known models: Game of Go: the rule of the game is the model





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 $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, ..., S_T\}$
   1. So consider it as a supervised learning problem

$$
\begin{aligned}
S_1, A_1 &\rightarrow R_2, S_2 \\
S_1, A_1 &\rightarrow R_2, S_2 \\
&\vdots \\
S_1, A_1 &\rightarrow R_2, S_2
\end{aligned}
$$

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

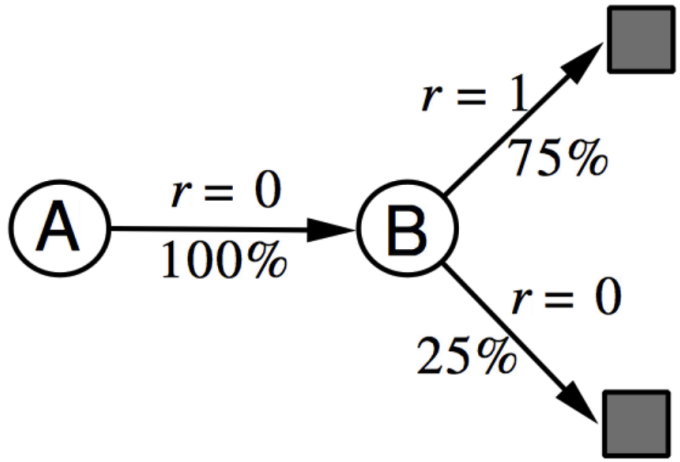
1. Model is an explicit MDP, $\hat{\mathcal{P}}$ and $\hat{\mathcal{R}}$
2. Count visits N(s,a) to each state action pair

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

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

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

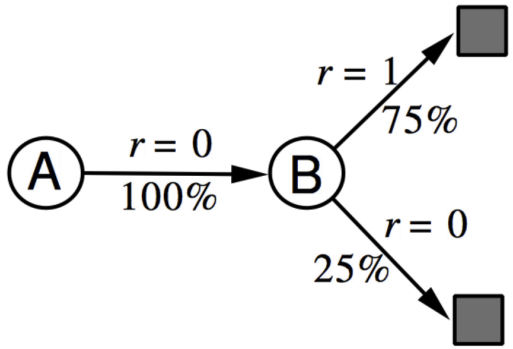
1. A simple but sample-efficient approach to planning
2. Use the model only to generate samples
3. General procedure:
   1. Sample experience from the model

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

   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 model is wrong, use model-free RL
    2. Reason explicitly about the model uncertainty (how confident we are for the estimated state)

Real and Simulated Experience

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

$$S' \sim \mathcal{P}^a_{s,s'}$$
$$R = \mathcal{R}^a_s$$

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

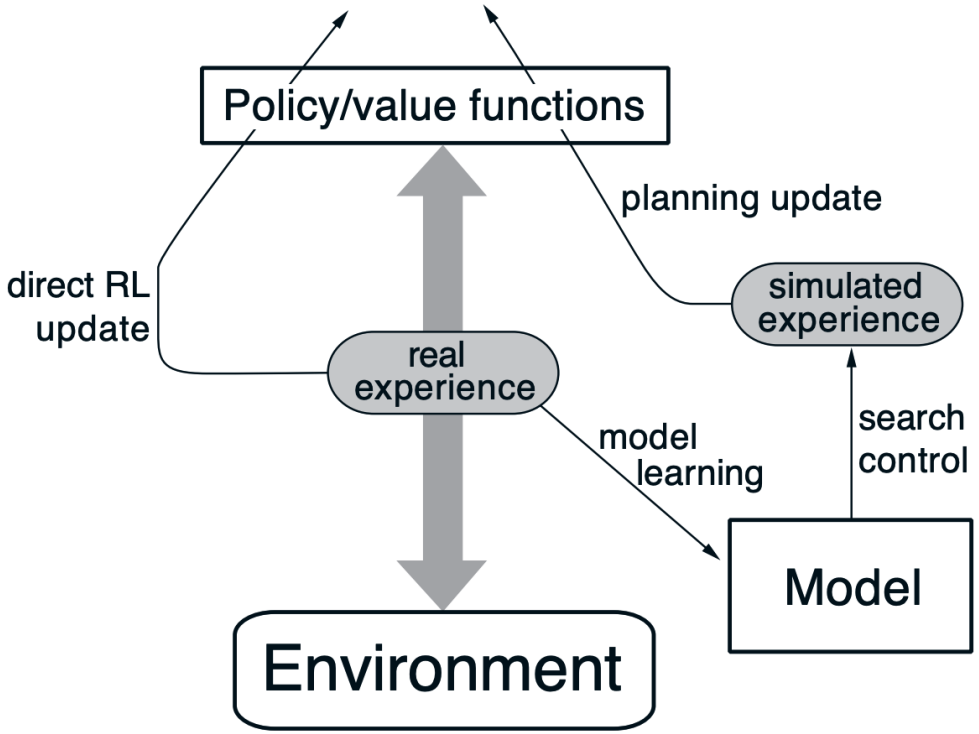
$$S' \sim \mathcal{P}_\eta(S'|S, A)$$
$$R = \mathcal{R}_\eta(R|S, A)$$

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

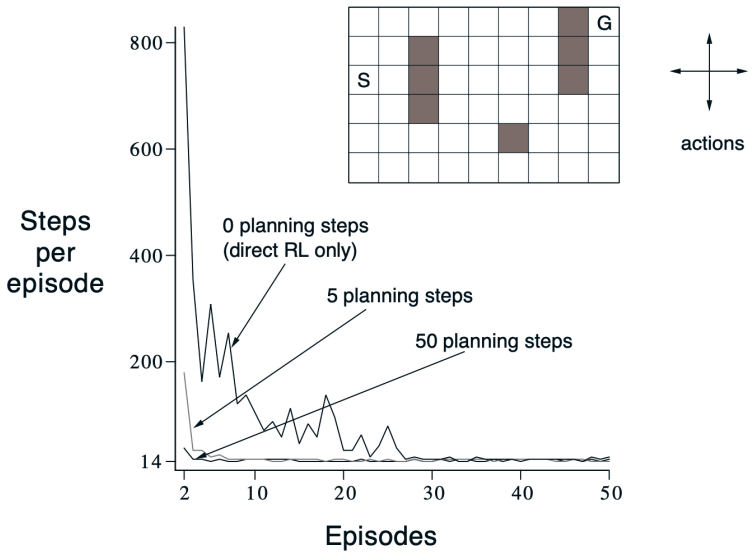
Tabular Dyna-Q

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) $A \leftarrow \epsilon$-greedy(S,Q)
- (c) Execute action A; observe resultant reward, R, and state, S'
- (d) $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$
- (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
  - $R, S' \leftarrow Model(S,A)$
  - $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma \max_a Q(S',a) - Q(S,A)\big]$

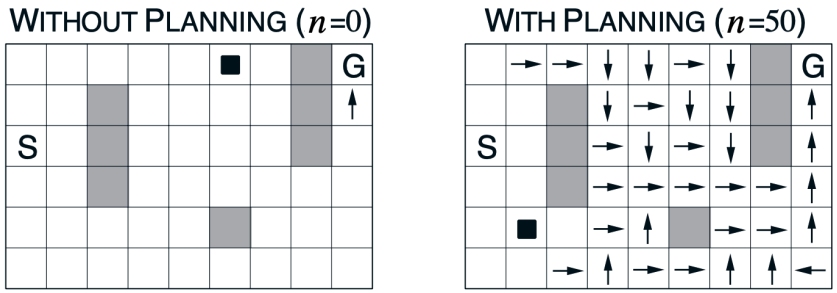
1. Combing direct RL, model learning, and planning together

Result of Dyna

1. A simple maze environment: travel fro 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:

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

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

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

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

$$\tau = \{s_1, a_1, s_2, a_2 ..., s_T, a_T\} \sim \pi_\theta(a_t|s_t)$$
$$\arg\max_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t \gamma^t r(s_t, a_t)\right]$$

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

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

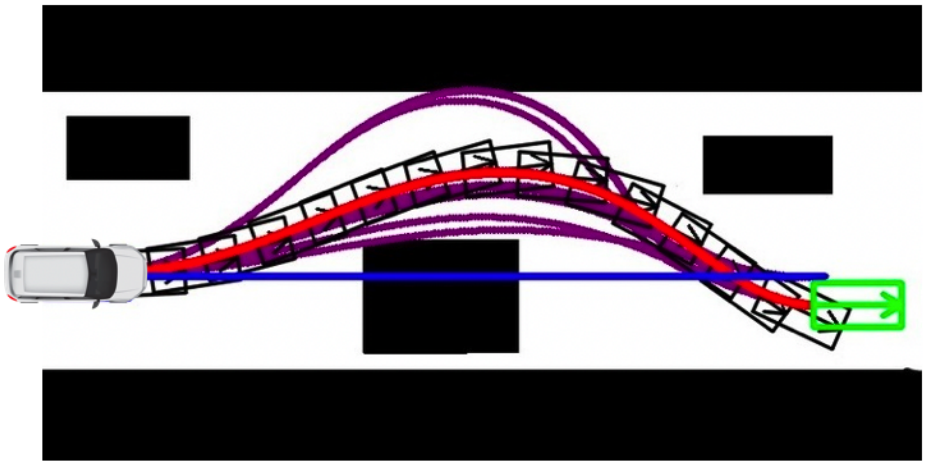
3. But can we do better if we know the model or able to learn the model?

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 (the system dynamics $s_t = f(s_{t-1}, a_{t-1})$ to decide the optimal controls for a trajectory to minimize the cost:

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

Optimal Control for Trajectory Optimization



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

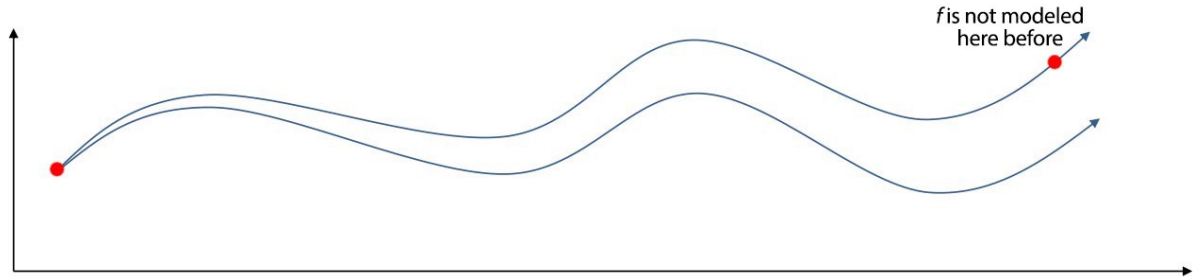
1. If the dynamics is known it becomes the optimal control problem
2. Cost function is the negative reward of the RL problem
3. The optimal solution can be solved by Linear-Quadratic Regulator (LQR) and iterative LQR (iLQR) under some simplified assumptions

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
3. Step 2 is supervised learning to train a model to minimize the least square error from the sampled data
4. 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

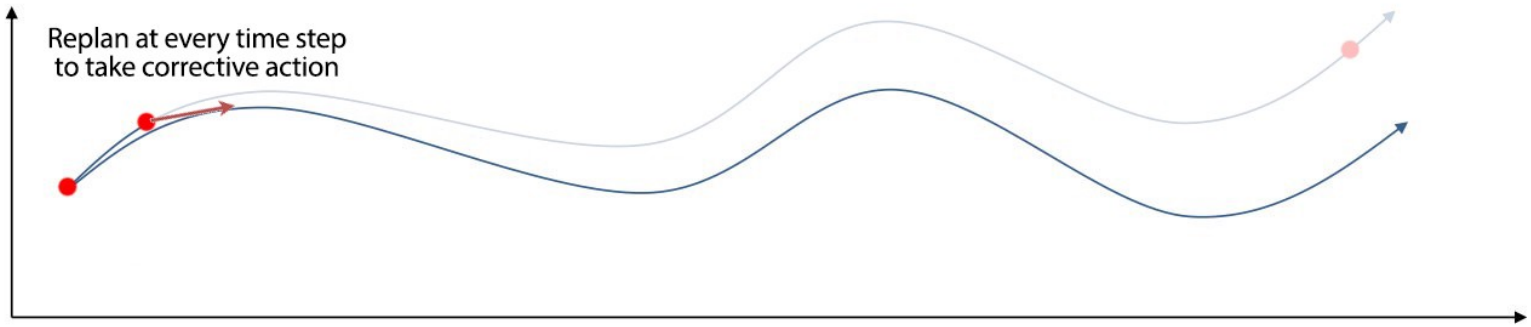
1. The previous solution is vulnerable to drifting, a tiny error accumulates fast along the trajectory
2. We may also land in areas where the model has not been learned yet



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

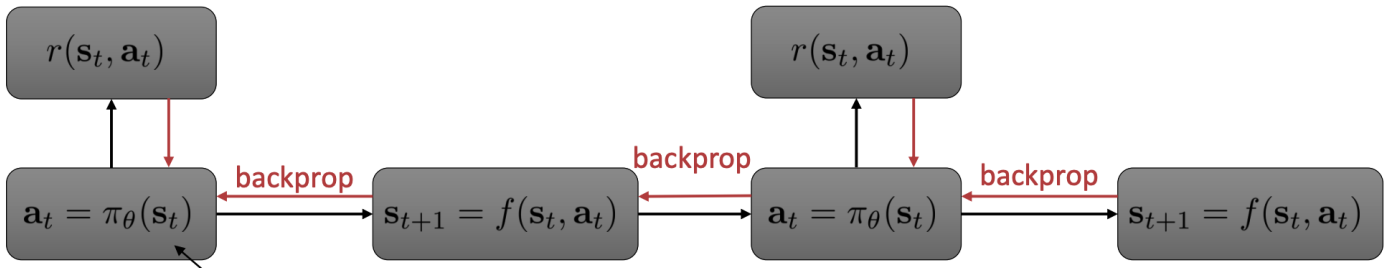


Model Learning for Trajectory Optimization: Algorithm 3

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 with Learning 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 $A_t = \frac{df}{dx_t}$ and $B_t = \frac{df}{du_t}$

Global Model versus Local Model

1. Local model as time-varying linear-Gaussian

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

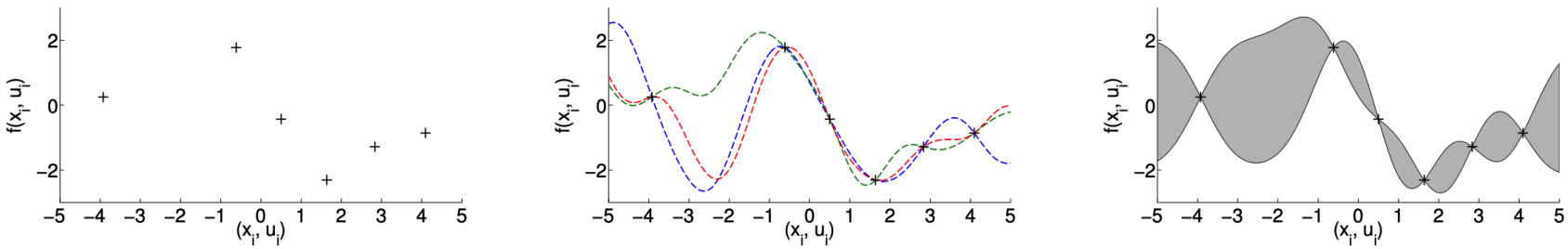


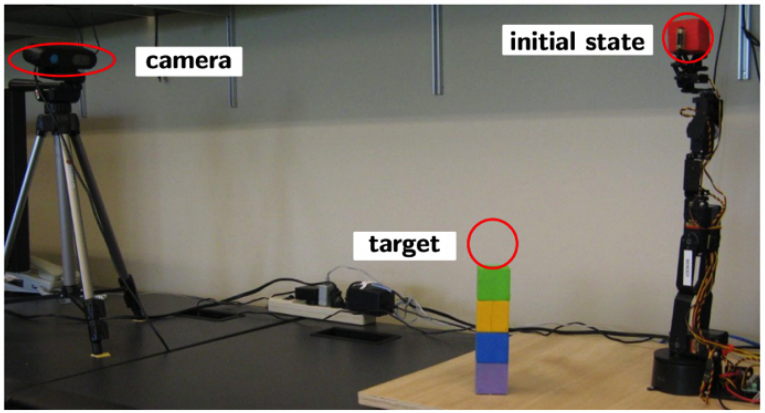
Figure 1. Small data set of observed transitions (left), multiple plausible deterministic function approximators (center), probabilistic function approximator (right). The probabilistic approximator models uncertainty about the latent function.

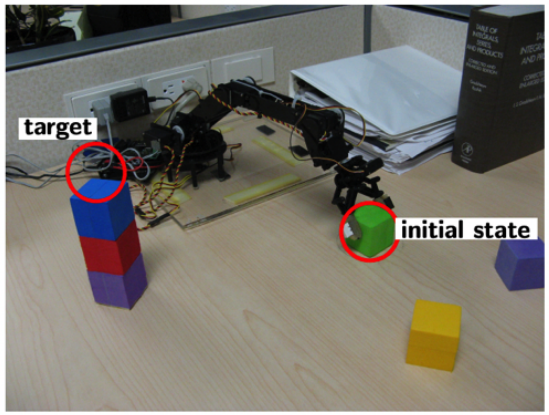
Today's Plan

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-degree Arm(\$370)+ 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-prarametric Gaussian process GP

Algorithm 1 PILCO

init: Set controller parameters ψ to random.
Apply random control signals and record data.
repeat
  Learn probabilistic GP dynamics model using all data
  repeat ▷ Model-based policy search
    Approx. inference for policy evaluation: get $J^\pi(\psi)$
    Gradients $\mathrm{d}J^\pi(\psi)/\mathrm{d}\psi$ for policy improvement
    Update parameters ψ (e.g., CG or L-BFGS).
  until convergence; **return** ψ^*
  Set $\pi^* \leftarrow \pi(\psi^*)$.
  Apply π^* to robot (single trial/episode); record data.
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

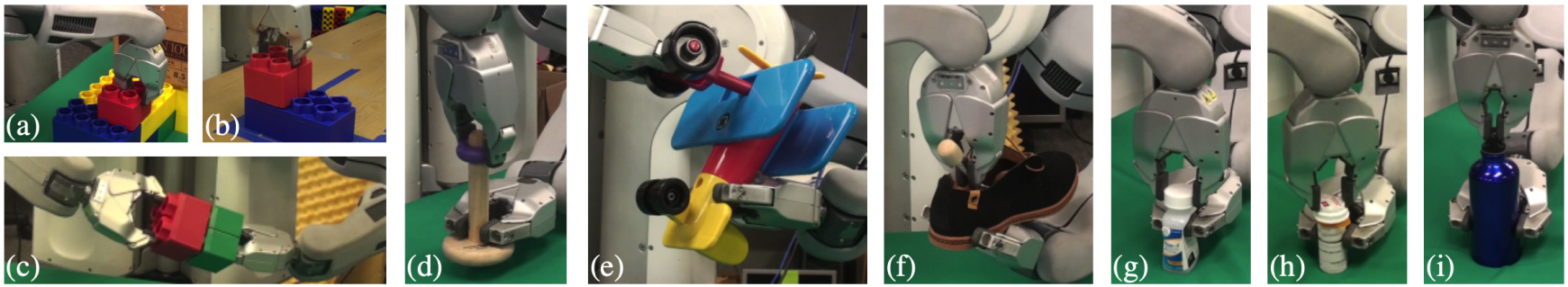


Fig. 2: Tasks in our experiments: (a) stacking large lego blocks on a fixed base, (b) onto a free-standing block, (c) held in both gripper; (d) threading wooden rings onto a tight-fitting peg; (e) assembling a toy airplane by inserting the wheels into a slot; (f) inserting a shoe tree into a shoe; (g,h) 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`.

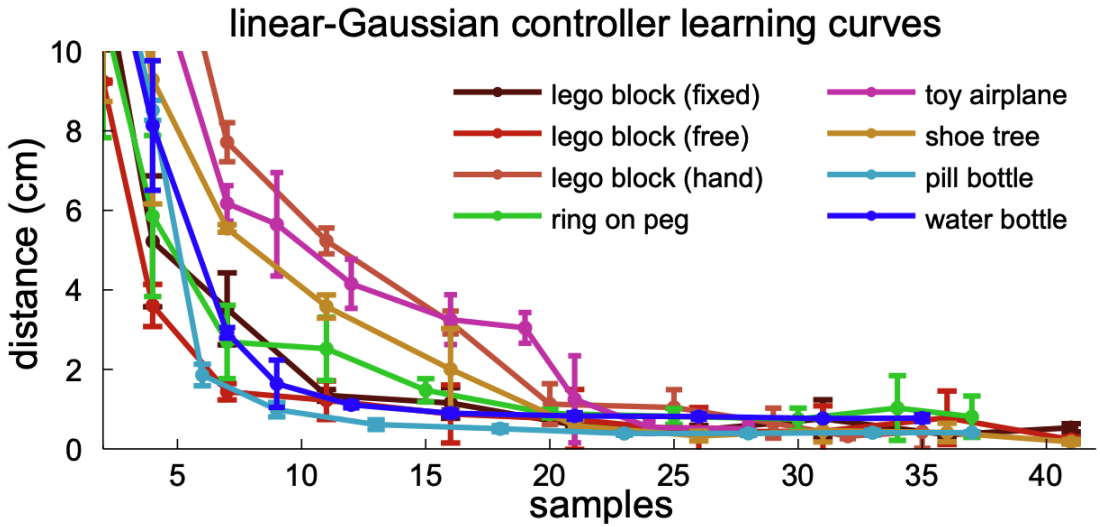
Case Study 2: Model-based Robotic Object Manipulation

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: $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}x_t + f_{\mathbf{u}t}\mathbf{u}_t, \mathbf{F}_t)$
   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 $\sum_{i,t}\lambda_{i,t}D_{\mathrm{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|p_i(\mathbf{x}_t, \mathbf{u}_t))$ with respect to θ 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 $\alpha D_{\mathrm{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|p_i(\mathbf{x}_t, \mathbf{u}_t))$
6: **end for**
7: **return** optimized policy parameters θ

Case Study 2: Model-based Robotic Object Manipulation



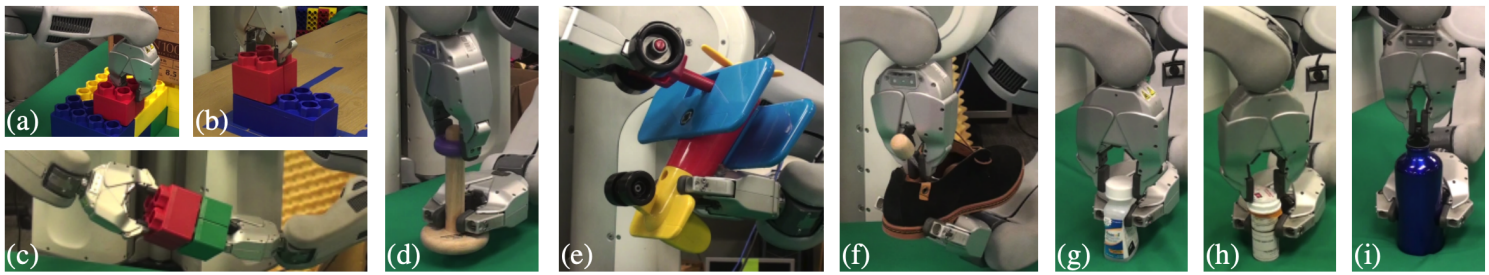
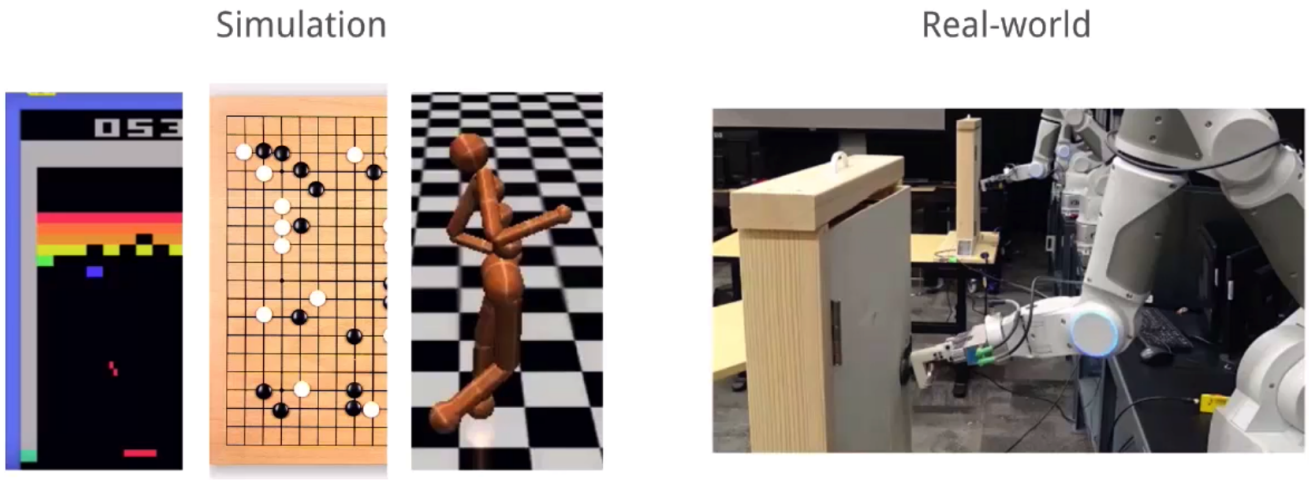


Fig. 2: Tasks in our experiments: (a) stacking large lego blocks on a fixed base, (b) onto a free-standing block, (c) held in both gripper; (d) threading wooden rings onto a tight-fitting peg; (e) assembling a toy airplane by inserting the wheels into a slot; (f) inserting a shoe tree into a shoe; (g,h) 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`.

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 a huge sample efficiency, which is crucial for robotic applications