Model-based RL

Hao Su

(Contents from IERG5350 taught by Prof. Bolei Zhou.)

Spring, 2021

Agenda

click to jump to the section.

Recall: Learning Objective of RL

- Given an MDP, find a policy π to maximize the expected return induced by π
  - The conditional probability of trajectory τ given π is

$$\Pr(\tau|\pi) = \Pr(S_0 = s_0) \prod_t \pi(a_t|s_t) P(s_{t+1}|s_t, a_t) \Pr(r_{t+1}|s_t, a_t)$$

  - RL Objective: $\max_\pi J(\pi)$

$$\begin{aligned} J(\pi) &= \mathbb{E}_{\tau\sim\pi}[R_1 + \gamma R_2 + \dots] \\ &= \sum_\tau \Pr(\tau|\pi)(r_1 + \gamma r_2 + \dots) \end{aligned}$$

Concept of Model-based RL

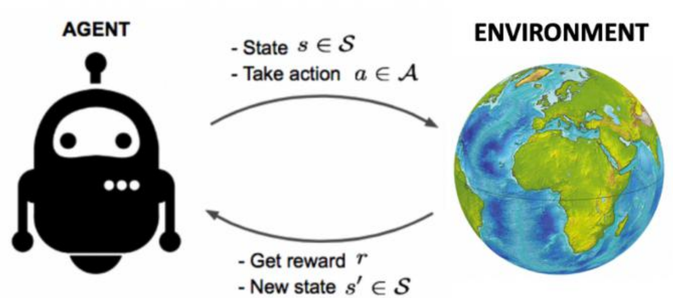
RL with Environment Model

- Model-free RL
  - Learn policy from collected experience through policy gradient
  - Learn value function through TD(DQN) or MC(PPO)
  - In model-free reinforcement learning, transition dynamics $P(s_{t+1}|s_t, a_t)$ and reward function $R(r_t|s_t, a_t)$ is unknown and we do not attempt to learn it

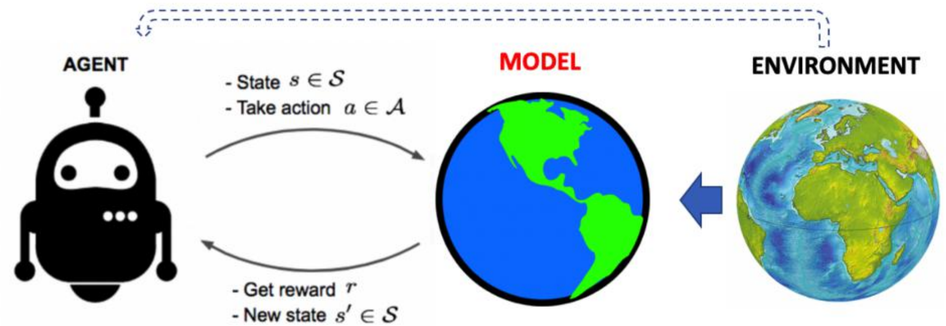
- Model-based RL
  - Learn/build model of environment from experience
  - Utilize the model of environment to get (or improve) policy(or value)

Model-free RL and Model-based RL

- Model-free RL



- Model-based RL



Concepts in Model-based RL

Concepts in Model-based RL

- **Rollout:**predict a short trajectory $s_1, s_2, \ldots, s_T$if we start at s_0 and execute $a_0, a_1, \ldots, a_{N-1}$ sequentially using environment dynamics

Concepts in Model-based RL

- **Rollout:**predict a short trajectory $s_1, s_2, \ldots, s_T$if we start at s_0 and execute $a_0, a_1, \ldots, a_{N-1}$ sequentially using environment dynamics
- **Search:** takes a model as input and produces or improves a policy by rollouts with the modeled environment

$$\text{experience} \xrightarrow{\text{learning}} \text{model} \xrightarrow{\text{search}} \text{policy}$$

Concepts in Model-based RL

- **Rollout:**predict a short trajectory $s_1, s_2, \ldots, s_T$if we start at s_0 and execute $a_0, a_1, \ldots, a_{N-1}$ sequentially using environment dynamics
- **Search:** takes a model as input and produces or improves a policy by rollouts with the modeled environment

$$\text{experience} \xrightarrow{\text{learning}} \text{model} \xrightarrow{\text{search}} \text{policy}$$

- **Model-based value optimization:**

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

Concepts in Model-based RL

- **Rollout:**predict a short trajectory $s_1, s_2, \ldots, s_T$if we start at s_0 and execute $a_0, a_1, \ldots, a_{N-1}$ sequentially using environment dynamics
- **Search:** takes a model as input and produces or improves a policy by rollouts with the modeled environment

$$\text{experience} \xrightarrow{\text{learning}} \text{model} \xrightarrow{\text{search}} \text{policy}$$

- **Model-based value optimization:**

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

- **Model-based policy optimization:**

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

Structure of Model-based RL

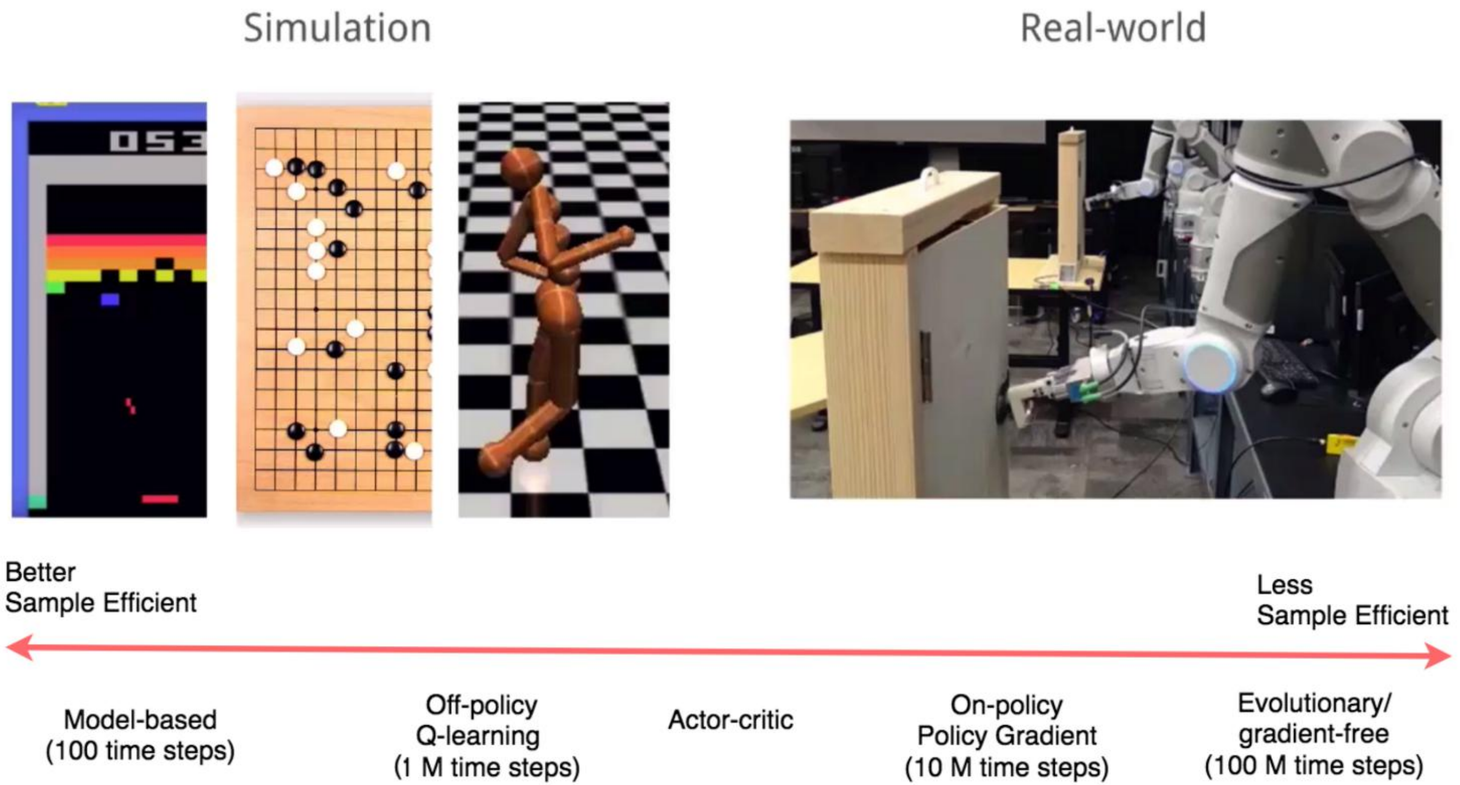
- Relationships of concepts

Structure of Model-based RL

- Relationships of concepts
- Two roles of real experience:
  - Improve the value and policy directly using off-policy methods
  - Improve the learned model to match the real environment more accurately : $p(s_{t+1}|s_t, a_t)$, $R(r_t|s_t, a_t)$

Advantage of Model-based RL

- Higher sample efficiency, which is crucial for real-world RL applications such as robotics



Models of the Environment

What is a Model

- A model M is a representation of an MDP parameterized by η
- Usually a model M represents state transition and reward function

$$s_{t+1} \sim P(s_{t+1}|s_t, a_t)$$
$$r_t \sim R(r_t|s_t, a_t)$$

- Typically we assume conditional independence between state transitions and rewards

$$P(s_{t+1}, r_t|s_t, a_t) = P(s_{t+1}|s_t, a_t)R(r_t|s_t, a_t)$$

Perfect Model Without Learning

- Games: Go, the rule of the game is known

- Simulated Environment: Inverted pendulum, kinematics and dynamics can be modeled using physics

Learning the Model

- Goal: learn model M_η from collected experience $\{s_0, a_0, r_0, \cdot, r_{T-1}\}$
  - The simplest way is to consider it s a supervised learning problem

$$
\begin{aligned}
s_0, a_0 &\longrightarrow r_1, s_2 \\
s_1, a_1 &\longrightarrow r_2, s_3 \\
&\vdots \\
s_{T-1}, a_{T-1} &\longrightarrow r_{T-1}, s_T
\end{aligned}
$$

- Learning $s, a \longrightarrow r$ is a regression problem
- Usually learning $s, a \longrightarrow s'$ is a density estimation problem. It can also be formulated as a regression problem
- Pick a loss function, e.g., mean-squares error, KL divergence, to optimize model parameters that minimize the empirical loss

Examples of Model Parameterization

The model of the environment can be parameterized by different functions, for example:

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Neural Network Model

Table Lookup Model

- Model is an explicit MDP for transition dynamics $P(s_{t+1}|s_t, a_t)$ and reward function $R(r_t|s_t, a_t)$
- Table lookup model: Count visits $N(s, a)$ to each state action pairs

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

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

Models-based Value Optimization

Simplest Way to Utilize Learned Model

- Use the model only to generate samples
- General procedure:
  - Sample experience from the model

$$S_{t+1} \sim P_\eta(S'|S_t, A_t)$$
$$R_t \sim R_\eta(R_t|S_t, A_t)$$

  - Apply model-free RL to sampled experiences: e.g., Q-Learning, policy gradient

Using an Inaccurate Model

- Given an imperfect model $\langle P_\eta, R_\eta \rangle \neq \langle P, R \rangle$
- Performance of model-based RL is limited to the optimal policy for approximate MDP
- When the model is inaccurate, learning from sampled trajectory will generate a suboptimal policy
- Possible solutions
  - When the accuracy of the model is low, use model-free RL
  - Reason explicitly about the model uncertainty (how confident we are for the estimated state): use probabilistic model such as Bayesian Network or Gaussian Process

Real and Simulated Experience

We now have two sources of experience:

Real and Simulated Experience

We now have two sources of experience:

- **Real Experience:** sampled from the environment (true MDP)

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

Real and Simulated Experience

We now have two sources of experience:

- **Real Experience:** sampled from the environment (true MDP)

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

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

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

Integrating Real and Simulated Experience

- Model-free RL:
  - No model
  - Learn value function (and/or policy) from real experience
- Model-based RL that only using samples from learned model
  - Learn a model from real experience
  - Update value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn and update value function (and/or policy) from both real and simulated experience

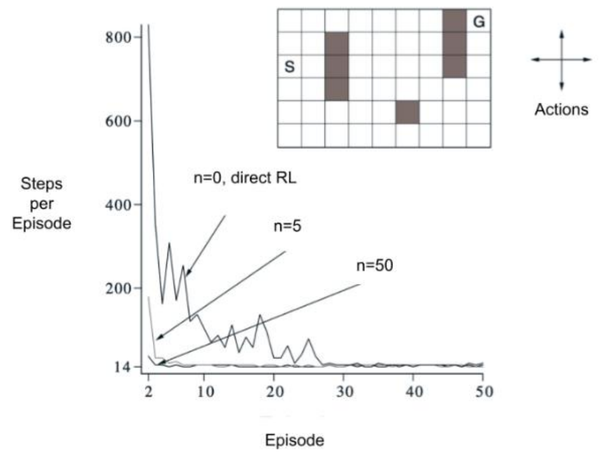
Tabular Dyna-Q

Dyna-Q: combines direct RL, model learning, and model-based sampling together

- Initialize $Q(s,a)$ and model $M_\eta(s,a)$
- Repeat:
  - $S \longleftarrow$ current(non-terminal) state
  - $A \longleftarrow \epsilon$-greedy (S,Q)
  - Execute action A: observe resultant reward R, and state S'
  - Update Q: $Q(s,a) \longleftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(S',a) - Q(s,a)]$
  - Update $M_\eta(s,a)$ via R and s'
  - Repeat for n times:
    - $s \longleftarrow$ random previously observed state
    - $a \longleftarrow$ random previously action taken at s
    - $r, s' \longleftarrow M_\eta(s,a)$
    - $Q(s,a) \longleftarrow Q(s,a) + \alpha[R + \gamma \max_a Q(s',a) - Q(s,a)]$

Result of Dyna

- A simple maze environment: travel from start state to goal as quickly as possible
- Learning curves vary with n in previous page: the number of simulated steps used per real step



- It learns faster with more number of simulated steps, but finally converges to the same performance

Models-based Policy Optimization

Policy Optimization with Model-based RL

- Previous model-based RL:

$$\text{model} \to \text{simulated trajectories} \xrightarrow{\text{backups}} \text{value} \to \text{policy}$$

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

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

Model-based Policy Optimization in RL

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

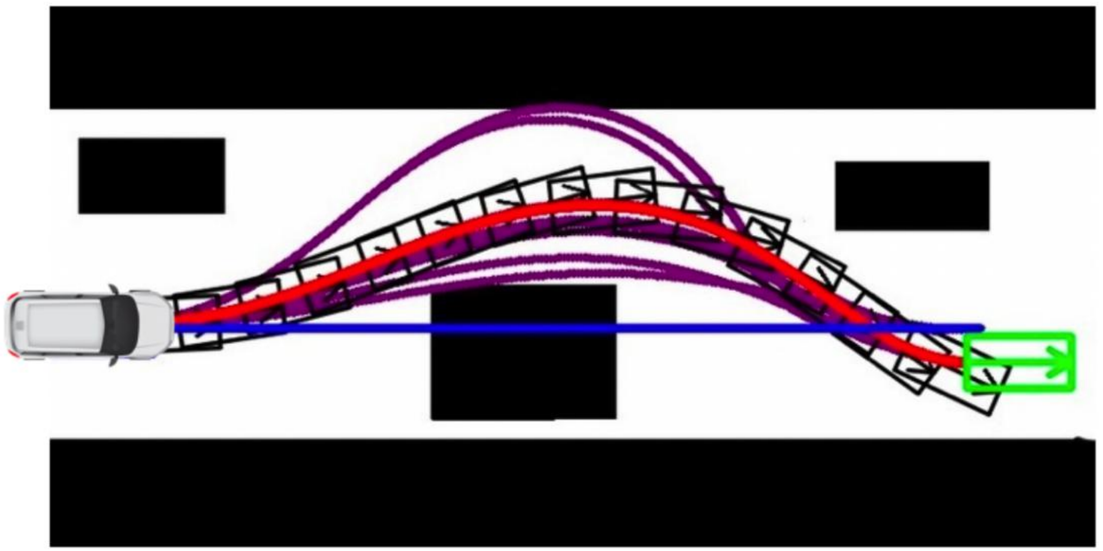
$$\tau = \{s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}\} \sim \pi_\theta(a|s)$$

$$\underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{\tau\sim\pi_\theta}[\sum_t \gamma^t r(s_t, a_t)]$$

- Can we do better if we know the model or are able to learn the model?

Model-based Policy Optimization in RL

- Model-based policy optimization in RL is strongly influenced by the optimal control theory
- In optimal control, the algorithm uses the model (transition function in L11) $\dot{x} = f(x, u)$ to compute the optimal control signal to minimize the cost, recall LQR in L11
- If the dynamics and reward(cost) is known, we can use optimal control to solve the problem



Model Learning for Trajectory Optimization

- If the dynamics model is unknown, we can combine model learning and trajectory optimization
- A Simple Algorithm for Trajectory Optimization with Model Learning:
  - Run base policy $\pi_0(a_t, s_t)$ (e.g., random policy) to collect $\{(s, a, s', r)\}$
  - Learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
  - Trajectory optimization through f(s,a) to choose actions

Model Learning for Trajectory Optimization

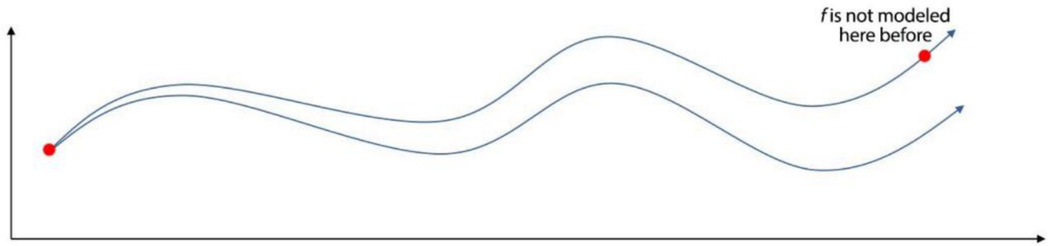
- If the dynamics model is unknown, we can combine model learning and trajectory optimization
- A Simple Algorithm for Trajectory Optimization with Model Learning:
  - Run base policy $\pi_0(a_t, s_t)$ (e.g., random policy) to collect $\{(s, a, s', r)\}$
  - Learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
  - Trajectory optimization through f(s,a) to choose actions
- Step 2 is just supervised learning to minimize the least square error from the sampled data

Model Learning for Trajectory Optimization

- If the dynamics model is unknown, we can combine model learning and trajectory optimization
- A Simple Algorithm for Trajectory Optimization with Model Learning:
  - Run base policy $\pi_0(a_t, s_t)$ (e.g., random policy) to collect $\{(s, a, s', r)\}$
  - Learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
  - Trajectory optimization through f(s,a) to choose actions
- Step 2 is just supervised learning to minimize the least square error from the sampled data
- Step 3 can be solved by any optimal control algorithm, e.g. LQR, to calculate the optimal trajectory using the model and a cost function

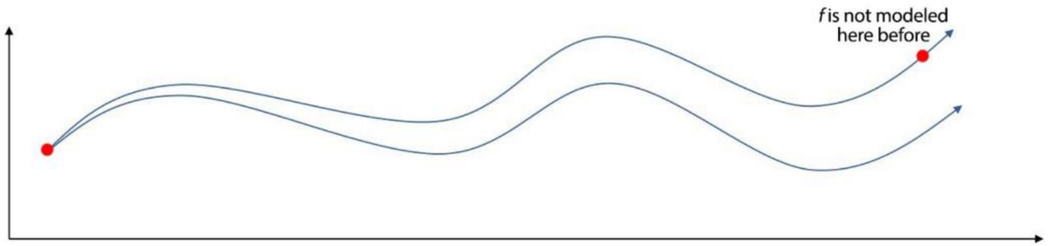
Model Learning for Trajectory Optimization

- The environment model is not easy to learn and a tiny error accumulates fast along the trajectory
- We may also land in areas where the model has not been learned yet



Model Learning for Trajectory Optimization

- The environment model is not easy to learn and a tiny error accumulates fast along the trajectory
- We may also land in areas where the model has not been learned yet



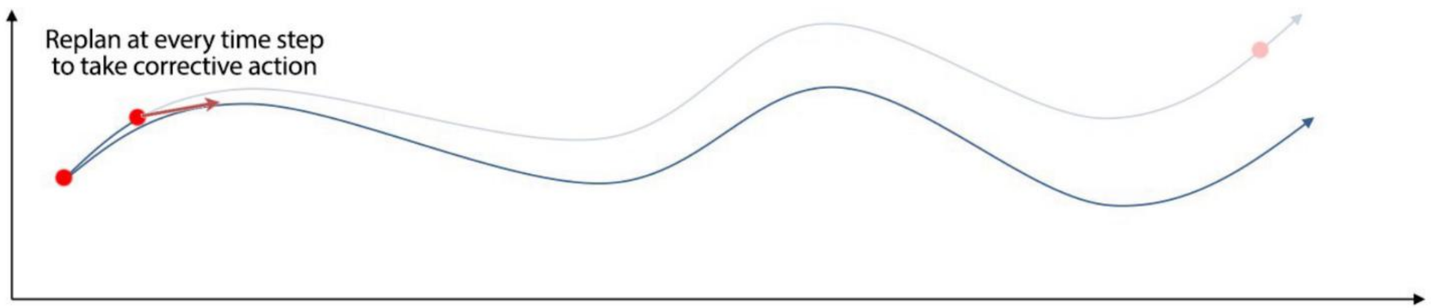
- An Improved Algorithm for Trajectory Optimization with Model Learning Iteratively
  - Run base policy $\pi_0(a_t, s_t)$ (e.g., random policy) to collect $\{(s, a, s', r)\}$
  - Repeat:
    - Learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
    - Trajectory optimization through f(s,a) to choose actions
    - Execute those actions and add the result data to the collect buffer $\{(s, a, s', r)\}$

Model Learning for Trajectory Optimization

- Nevertheless, the previous method executes all computed actions before fitting the model again. We may be off-grid too far already
- So we can use Model Predictive Control (MPC) to optimize the whole trajectory but we take the first action only, then we observe and replan again

Model Learning for Trajectory Optimization

- Nevertheless, the previous method executes all computed actions before fitting the model again. We may be off-grid too far already
- So we can use Model Predictive Control (MPC) to optimize the whole trajectory but we take the first action only, then we observe and replan again
- Model Predictive Control (MPC): optimize the whole trajectory but we take the first action only. It can be used with perfect model or learned model
- The replan (since it only take first action) in MPC gives us a chance to take corrective action after observed the current state again



Model Learning with MPC

- Similar as before, run base policy $\pi_0(a_t, s_t)$ (e.g., random policy) to collect $\{(s, a, s', r)\}$
- Repeat n steps:
  - Learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
  - Loop each step:
    - Search through $f(s, a)$ to choose action and rollout trajectories
    - Execute the first action in the trajectory and observe the resulting state s' (MPC)
    - Append (s, a, s', r) to collected dataset $\{(s, a, s', r)\}$

Search with Model: Random Shooting

- Given environment model, we can compute the action with LQR. However, LQR does not work well for model which is highly non-linear and running iLQR for each step is time consuming
- Random shooting is the simplest alternatives to search for an action with environment model

Search with Model: Random Shooting

- Given environment model, we can compute the action with LQR. However, LQR does not work well for model which is highly non-linear and running iLQR for each step is time consuming
- Random shooting is the simplest alternatives to search for an action with environment model
- It follows the routine of guess and check:
  - Sample m random action sequences from some distribution, e.g. uniform distribution:

$$a_0^0, a_1^0, \ldots, a_N^0$$

$$a_0^1, a_1^1, \ldots, a_N^1$$

$$\vdots$$

$$a_0^{m-1}, a_1^{m-1}, \ldots, a_N^{m-1}$$

  - Then evaluate the reward of each action sequence and choose the best one

Search with Model: Cross Entropy Method

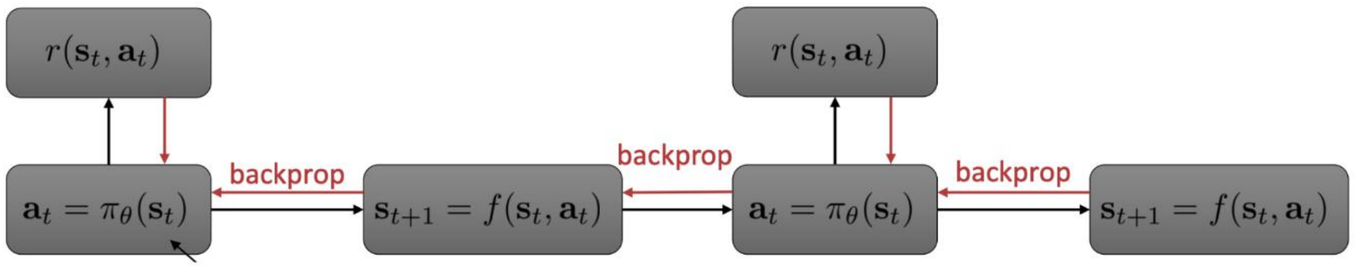
- In random shooting, a uniform distribution is used to sample action
- Can we have a more informative sampling?
- Intuition: using previous evaluation result to fit a distribution
- Cross Entropy Method (CEM) fit the distribution of solution with a Gaussian distribution

Search with Model: Cross Entropy Method

- Given: start state s_0 , population size M , planning horizon T , num of elites e, num of iters I , A Gaussian distribution $\mathcal{N}(\mu, \sigma)$
- Repeat for I steps:
  - Sample M action sequences from $\mathcal{N}(\mu, \sigma)$, each sequence has a horizon T
  - Repeat for M sequences:
    - For each action sequences, rollout T steps with environment model $p(s_{t+1}|s_t, a_t)$
    - Evaluate the rollouts using the reward function
  - Select the top-e elites from M action sequences with highest reward
  - Fit a new Gaussian distribution based on elites to update μ, σ
- Return the μ, σ after I steps, execute μ in the real environment

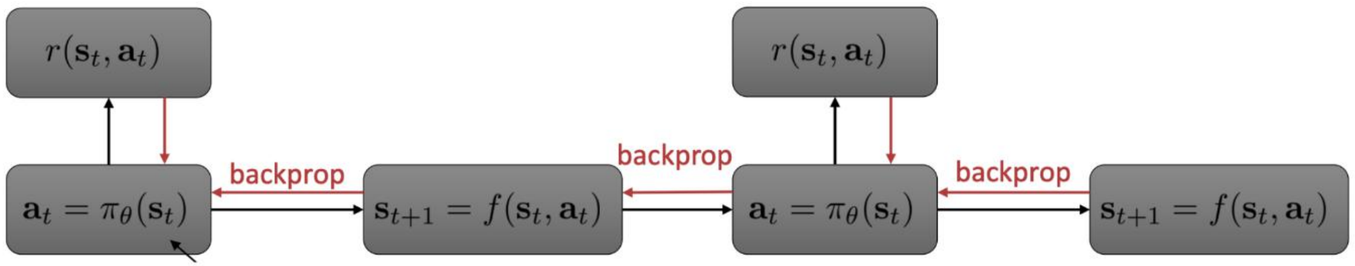
Learning Model and Policy Together

- If we represent the environment model using a neural network, then it becomes differentiable
- Thus we can plug the policy learning along with model learning via a differentiable pipeline



Learning Model and Policy Together

- If we represent the environment model using a neural network, then it becomes differentiable
- Thus we can plug the policy learning along with model learning via a differentiable pipeline



- **Learning Model and Policy Together:**
  - Similar as before, run base policy $\pi_0(a_t, s_t)$ (e.g., random policy) to collect $\{(s, a, s', r)\}$
  - Repeats:
    - Learn dynamics model $s' = f(s, a)$ to minimize $\sum_i ||f(s_i, a_i) - s'_i||^2$
    - Backpropagate through $f(s, a)$ into the policy to optimize $\pi_\theta(a_t|s_t)$
    - Run policy $\pi_\theta(a_t|s_t)$ and append (s, a, s', r) to collected dataset $\{(s, a, s', r)\}$

Parameterizing the Model

What function is used to parameterize the dynamics?

Parameterizing the Model

What function is used to parameterize the dynamics?

- Global model: $s_{t+1} = f(s_t, a_t)$ is represented by a single neural network
  - Pros: simple and straight-forward, can directly use lots of data to fit
  - Cons: not so great in low data regimes, and connot express model uncertainty

Parameterizing the Model

What function is used to parameterize the dynamics?

- Global model: $s_{t+1} = f(s_t, a_t)$ is represented by a single neural network
  - Pros: simple and straight-forward, can directly use lots of data to fit
  - Cons: not so great in low data regimes, and connot express model uncertainty
- Local model: model the transition as time-varying linear-Gaussian dynamics

$$p(x_{t+1}|x_t, u_t) = \mathcal{N}(f(x_t, u_t))$$
$$f(x_t, u_t) = A_t x_t + B_t u_t$$

  - Pros: very data-efficient and can express model uncertainty
  - What we need are only local gradients $A_t = \frac{df}{dx_t}$ and $B_t = \frac{df}{du_t}$
  - Cons: not great with non-smooth dynamics
  - Cons: very slow when dataset is large

Global Model versus Local Model

Local model as time-varying linear-gaussian

$$p(x_{t+1}|x_t, u_t) = \mathcal{N}(f(x_t, u_t))$$
$$f(x_t, u_t) = A_t x_t + B_t u_t$$

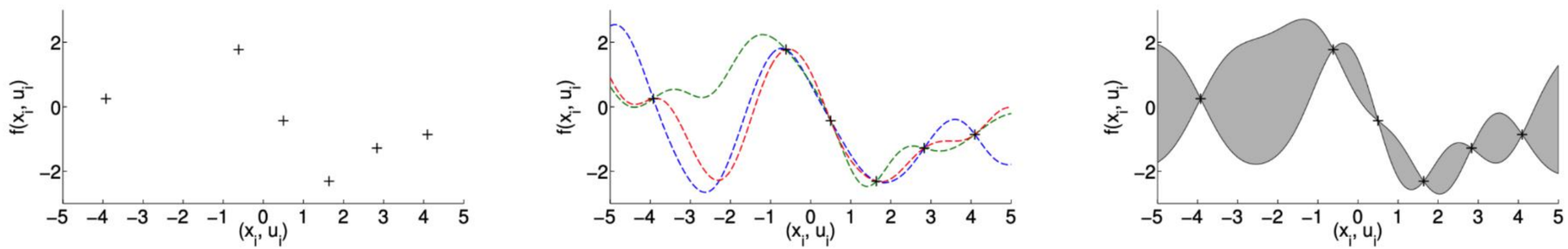


Figure 1. Small data set of observed transitions (left), multiple plausible deterministic function approximators (center), probabilistic function approximator (right). The probabilistic approximator models uncertainty about the latent function.

End