Model-Based Reinforcement Learning

Spring, 2024

Institute of Media, Information, and Network

Outline

Table of Contents

Model-Based Reinforcement Learning

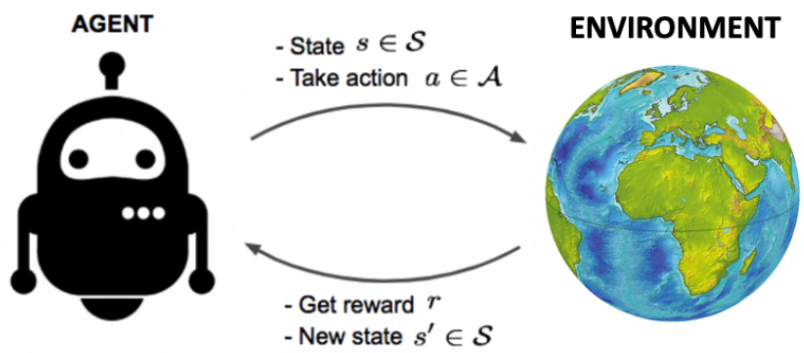
- *Previous lectures*: learn policy or value function directly from experience
- This lecture: learn model directly from experience
  - and use planning to construct a value function or policy
  - Integrate learning and planning into a single architecture

Model-Based and Model-Free RL

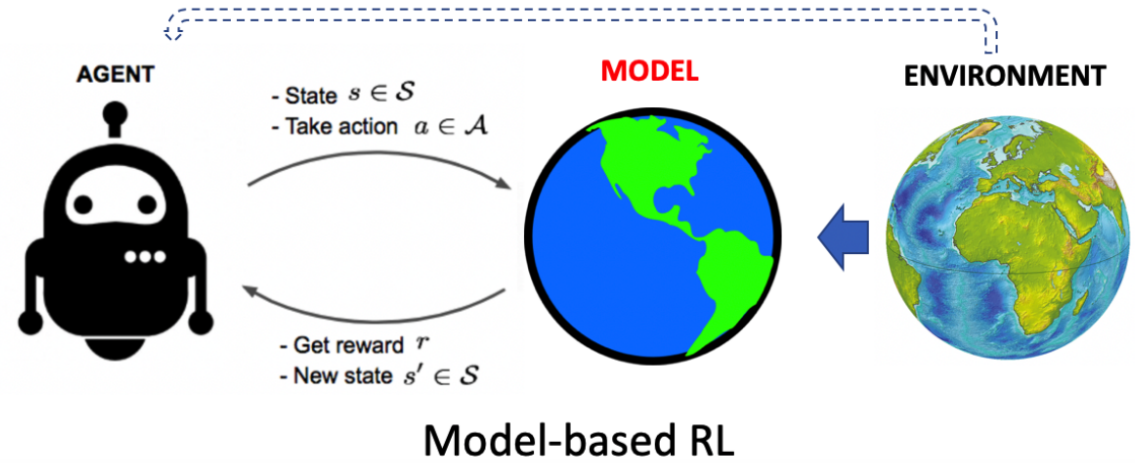
- Model-Free RL
  - No model
  - Learn value function (and/or policy) from experience
  - High sample complexity
- Model-Based RL
  - Learn a model from experience
  - Plan value function (and/or policy) from model
  - Better sample efficiency

Model-Based and Model-Free RL

1. Diagram of model-free reinforcement learning



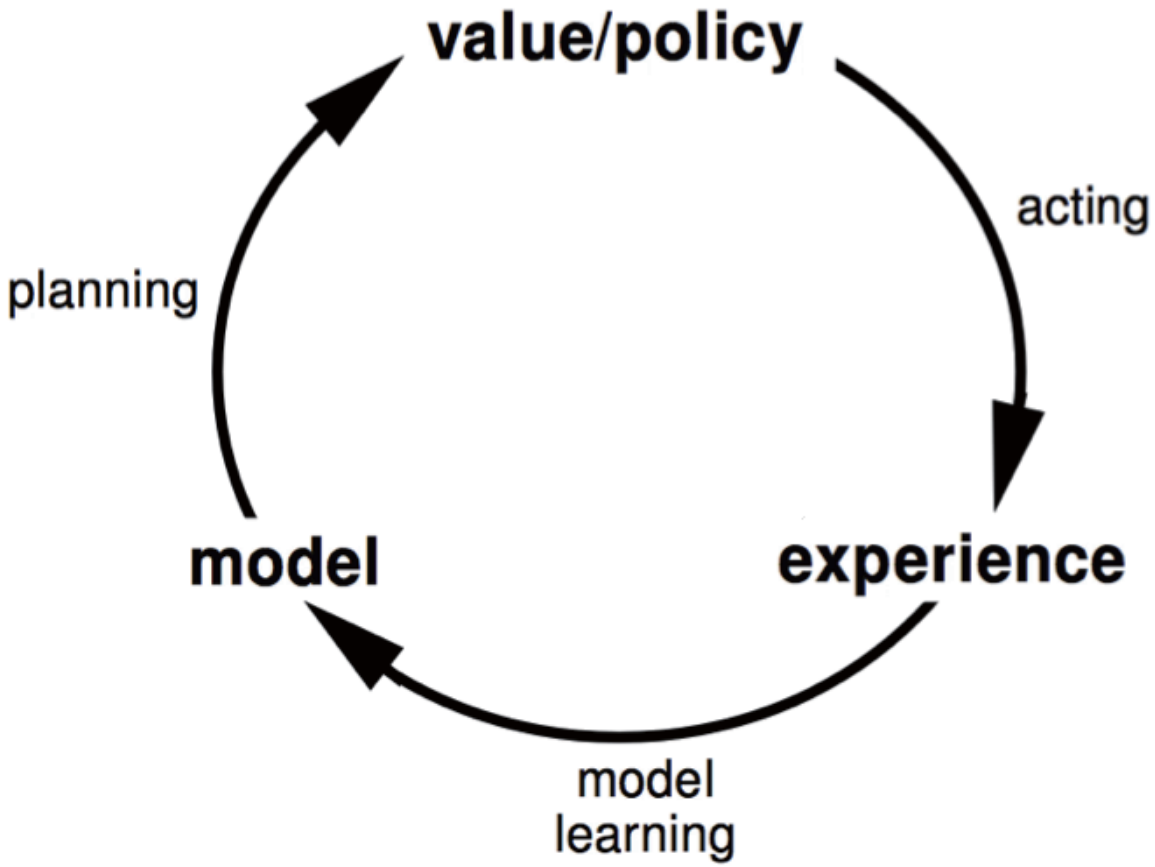
2. Diagram of model-based reinforcement learning



Iedia,
and Network

Model-Based RL

- Planning is "trying things in your head", using an internal model of the world.



Institute of Media, Information, and Network

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 methods share a common structure

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

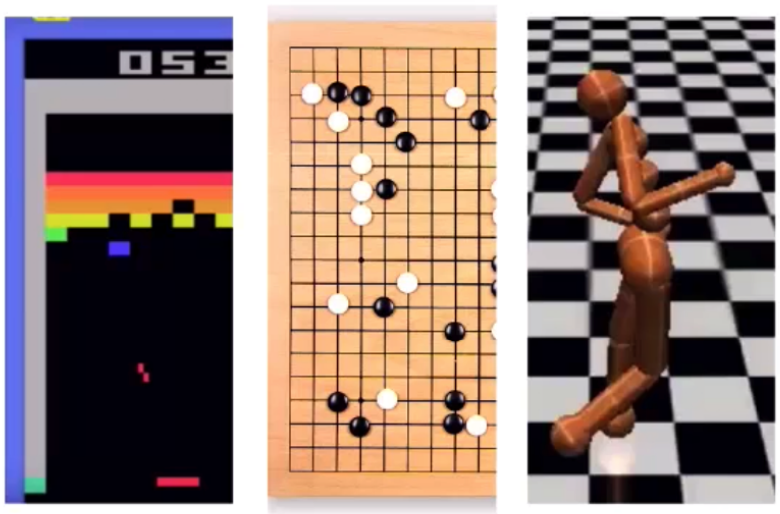
4. Model-based policy optimization methods have a simpler structure as

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

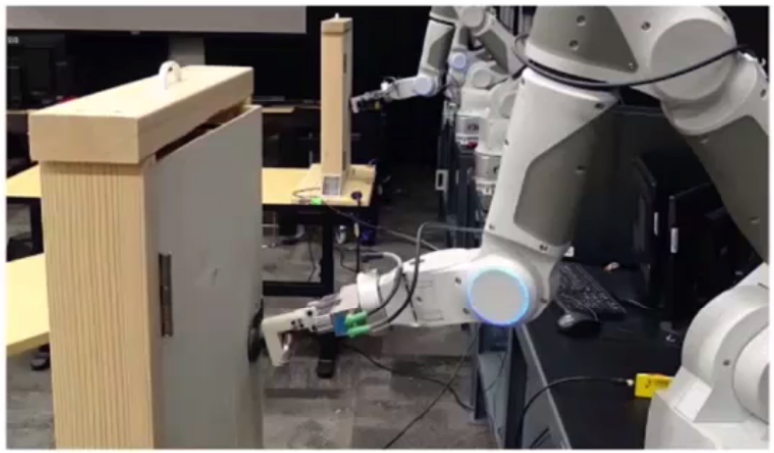
Advantages of Model-Based RL

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

work

Advantages of Model-Based RL

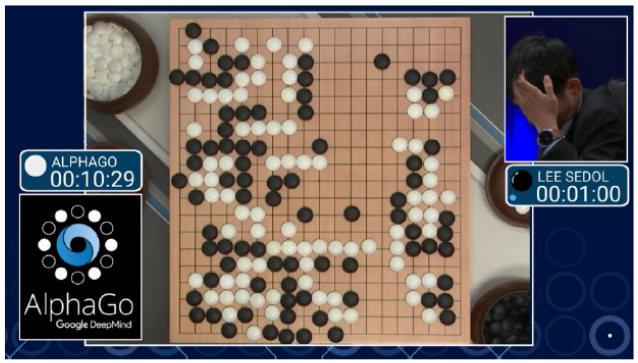
1. Better sample efficiency

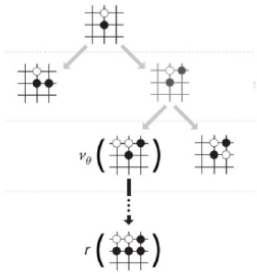
2. Cons:
   1. First learning a model then constructing a value function or policy function leads to two sources of approximation error
   2. Difficult to come up with guarantee of convergence

Institute of Media, Information, and Network

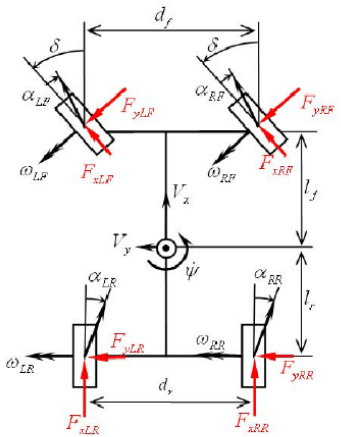
Sometimes it is easy to access the model

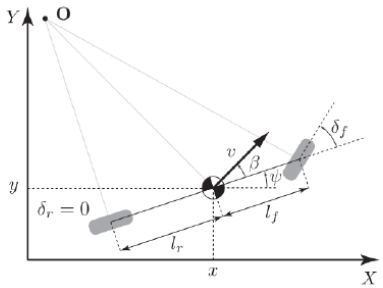
1. Known models: Game of Go: the rule of the game is the model





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





Network

Table of Contents

What is a Model?

- A *model* $\mathcal{M}$ is a representation of an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, parametrized by η
- We will assume state space $\mathcal{S}$ and action space $\mathcal{A}$ are known
- So a model $\mathcal{M} = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$ represents state transitions $\mathcal{P}_\eta \approx \mathcal{P}$ and rewards $\mathcal{R}_\eta \approx \mathcal{R}$

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

- Typically assume conditional independence between state transitions and rewards

$$\mathbb{P}[S_{t+1}, R_{t+1}|S_t, A_t] = \mathbb{P}[S_{t+1}|S_t, A_t]\mathbb{P}[R_{t+1}|S_t, A_t]$$

Model Learning

- Goal: estimate model $\mathcal{M}_\eta$ from experience $\{S_1, A_1, R_2, \cdots, S_T\}$
- This is a supervised learning problem

$$S_1, A_1 \to R_2, S_2$$
$$S_2, A_2 \to R_3, S_3$$
$$\vdots$$
$$S_{T-1}, A_{T-1} \to R_T, S_T$$

- Learning $s, a \to r$ is a *regression* problem
- Learning $s, a \to s'$ is a *density estimation* problem
- Pick loss function, e.g. mean-squared error, KL divergence
- Find parameters η that minimize empirical loss

Example of Models

- Table Lookup Model
- Linear Expectation Model
- Linear Gaussian Model
- Gaussian Process Model
- Deep Belief Network Model
- Deep Neural Network Model
- ...

Table Lookup Model

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

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

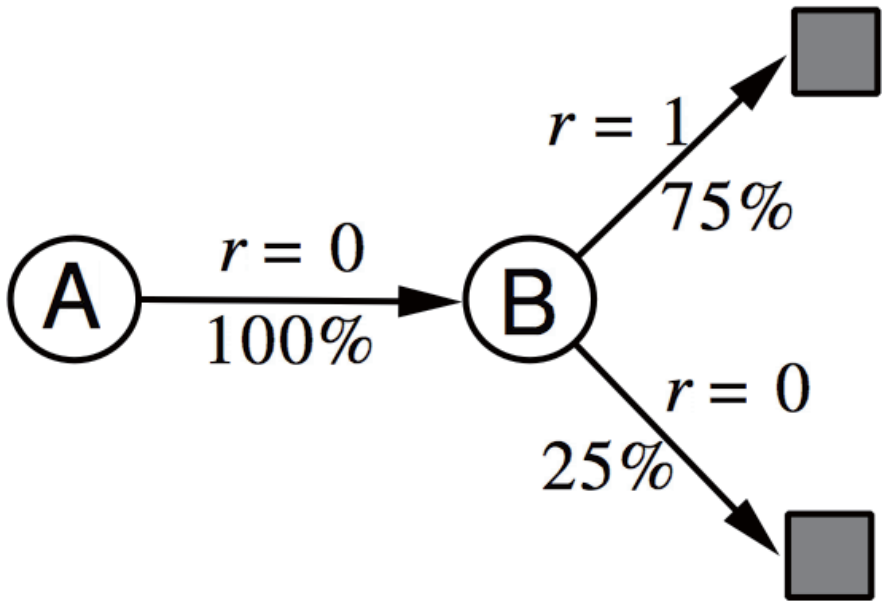
$$\hat{\mathcal{R}}^a_s = \frac{1}{N(s,a)} \sum_{t=1}^{T} \mathbf{1}(S_t, A_t, = s, a) R_t$$

- Alternatively
  - At each time-step t, record experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$
  - To sample model, randomly pick tuple matching $\langle s, a, \cdot, \cdot \rangle$

AB Example

Two states A, B; no discounting; 8 episodes of experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



We have constructed a table lookup model from the experience

Planning with a Model

- Given a model $\mathcal{M}_\eta = \langle \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Solve the MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- Using favourite planning algorithm
  - Value iteration
  - Policy iteration
  - Tree search
  - ...

Sample-Based Planning

- A simple but powerful approach to planning
- Use the model only to generate samples
- Sample experience from model

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

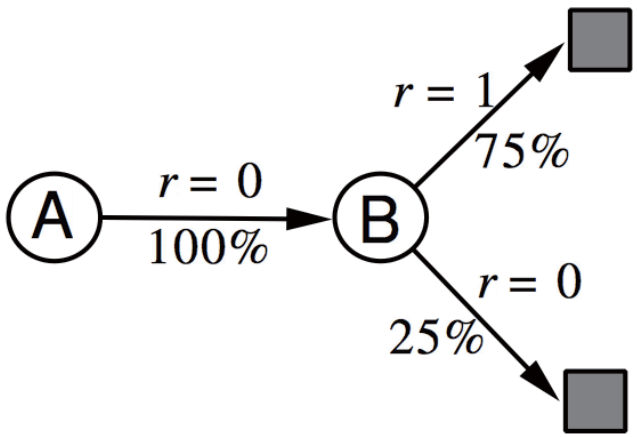
- Apply model-free RL to samples, e.g.:
  - Monte-Carlo control
  - Sarsa
  - Q-learning
- Sample-based planning methods are often more efficient

Back to the AB Example

- Construct a table-lookup model from real experience
- Apply model-free RL to sample experience

Real experience

A, 0, B, 0
B, 1
B, 1
B, 1
B, 1
B, 1
B, 1
B, 0



Sampled experience

B, 1
B, 0
B, 1
A, 0, B, 1
B, 1
A, 0, B, 1
B, 1
B, 0

e.g. Monte-Carlo learning: $V(A) = 1, V(B) = 0.75$

Sometimes it is easy to access the 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

We consider two sources of experience

Real experience Sampled from environment (true MDP)

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

Simulated experience Sampled from model (approximate MDP)

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

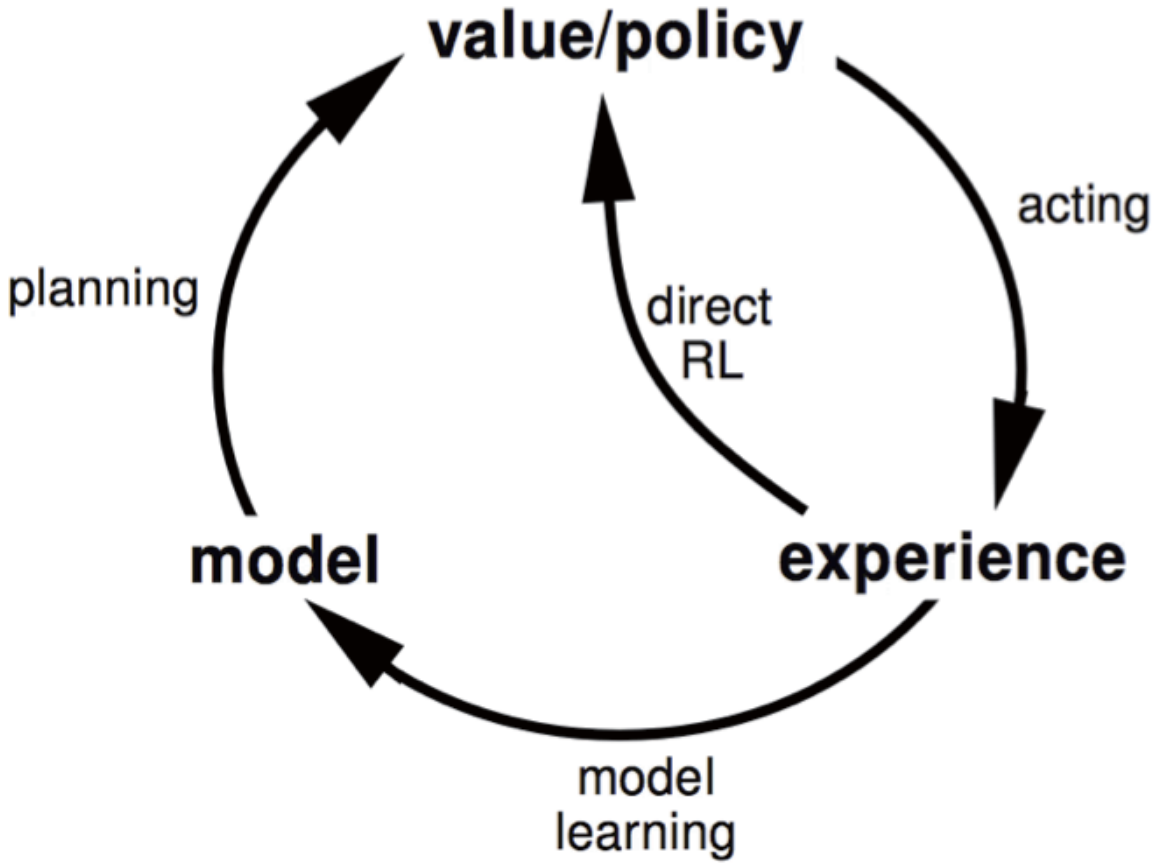
Institute of Media, Information, and Network

Integrating Learning and Planning

- Model-Free RL
  - No model
  - Learn value function (and/or policy) from real experience
- Model-Based RL (using Sample-Based Planning)
  - Learn a model from real experience
  - Plan value function (and/or policy) from simulated experience
- Dyna
  - Learn a model from real experience
  - Learn and plan value function (and / or policy) from real and simulated experience

Dyna Architecture

R. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, AAAI 1991.



Institute of Media, Information, and Network

Dyna-Q Algorithm

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) $A \leftarrow \varepsilon$-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:

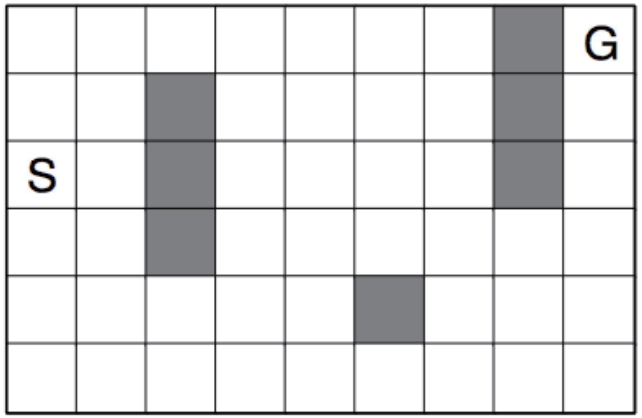
$S \leftarrow$ random previously observed state

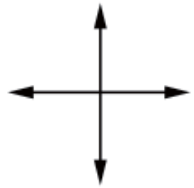
$A \leftarrow$ random action previously taken in S

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

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

Dyna-Q on a Simple Maze

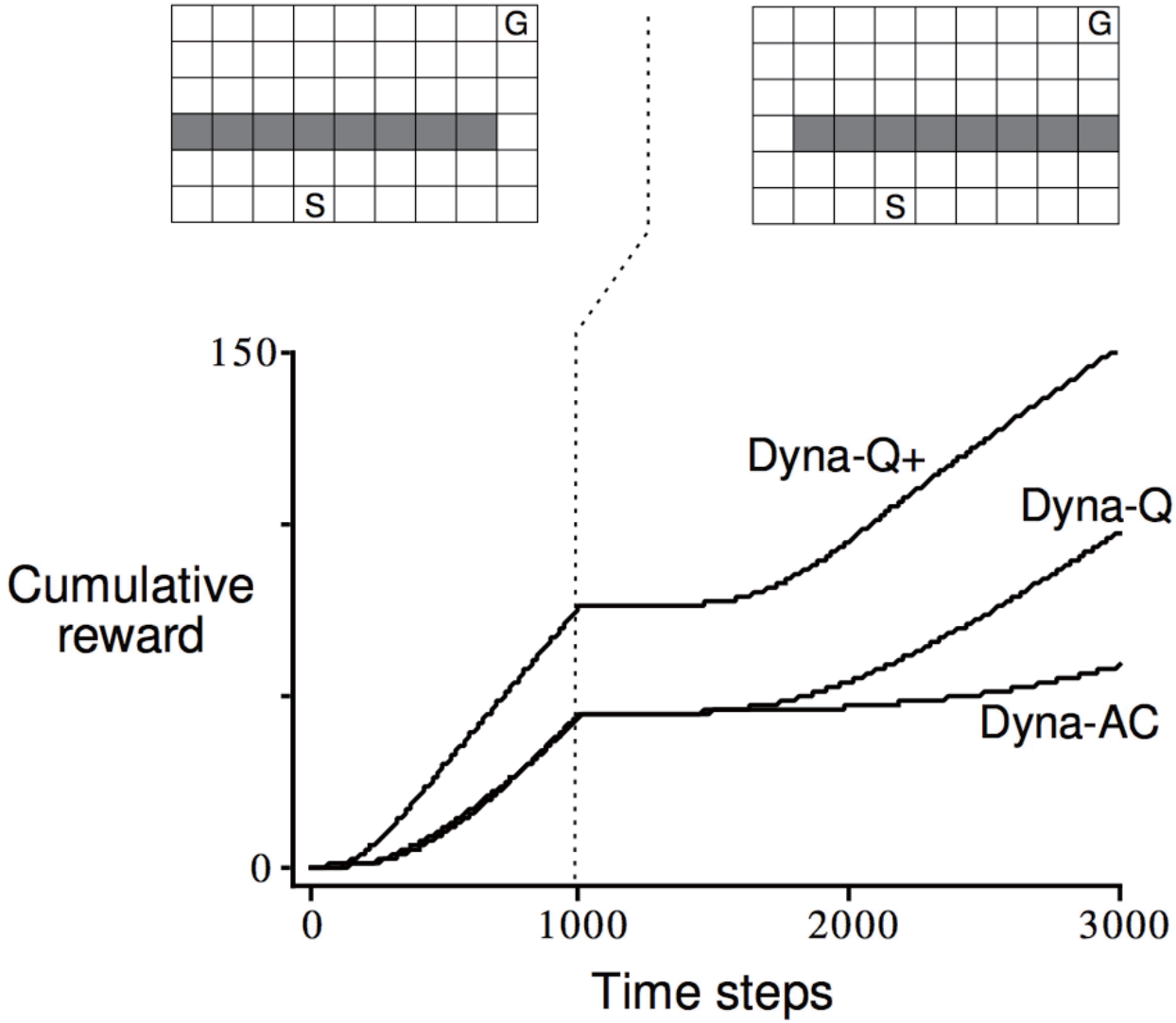




actions

Dyna-Q with an Inaccurate Model

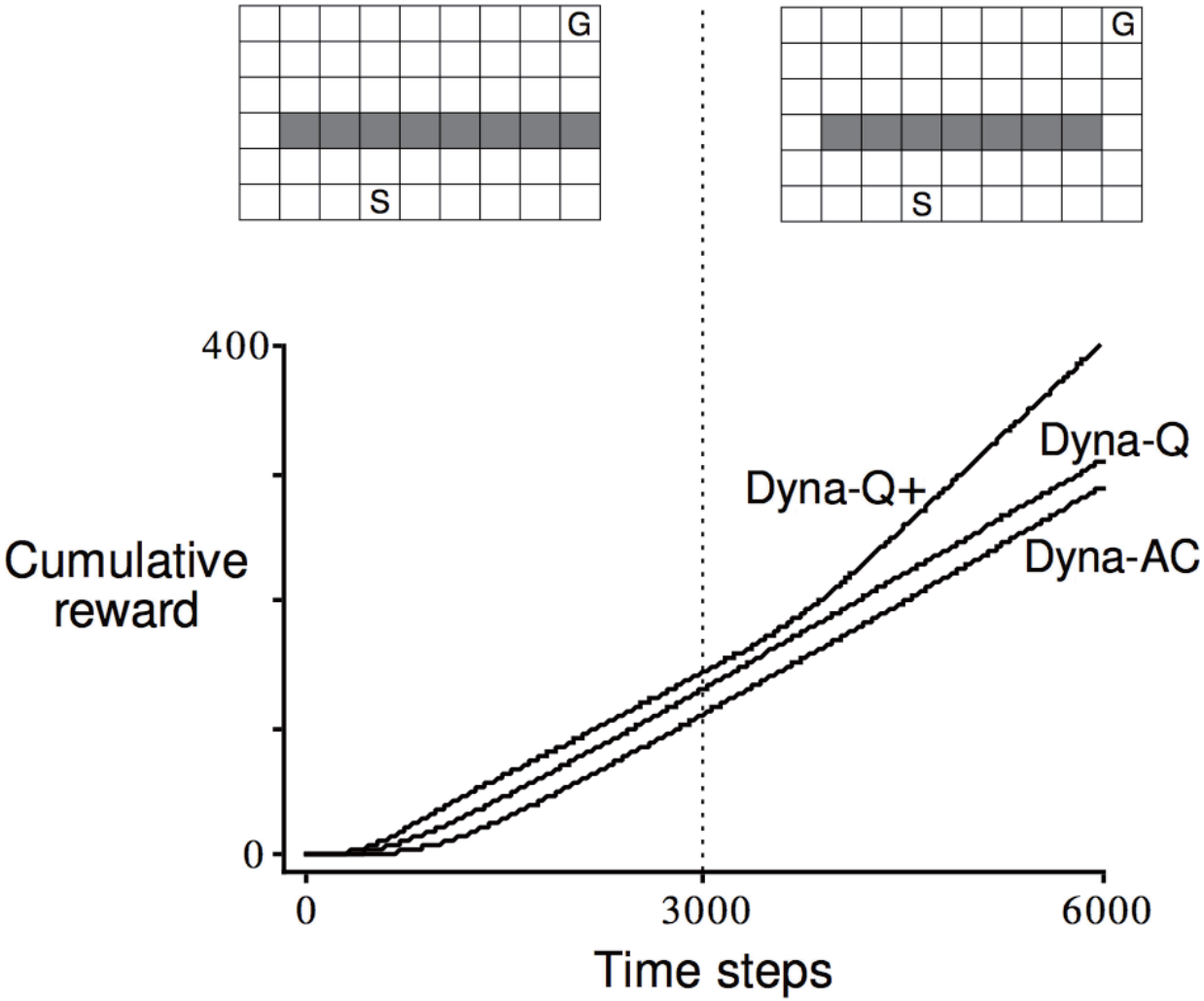
- The changed environment is harder



stitute of Media,
formation, and Network

Dyna-Q with an Inaccurate Model (2)

- The changed environment is easier



nstitute of Media,
nformation, and Network

Table of Contents

Policy Optimization with Model-based RL

1. Previous model-based value-based RL:

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

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

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

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

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

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

a,
Network

Model-based Policy Optimization in RL

Why learn the model?

If we knew $f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$, we could use the tools from last week.

(or $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ in the stochastic case)

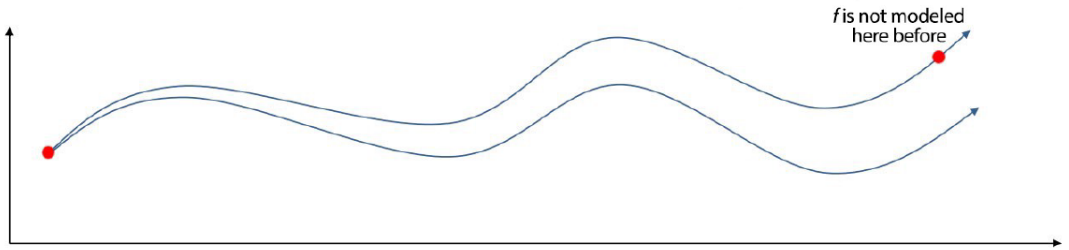
So let's learn $f(\mathbf{s}_t, \mathbf{a}_t)$ from data, and *then* plan through it!

model-based reinforcement learning version 0.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

Model-Based Policy Optimization

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*

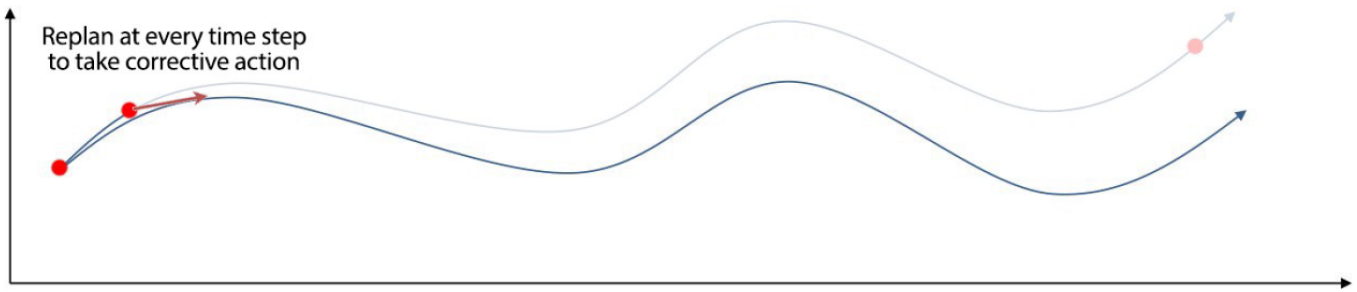
model-based reinforcement learning version 1.0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute those actions and add the resulting data $\{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_j\}$ to $\mathcal{D}$

Iedia,
Information, and Network

Model-Based Policy Optimization

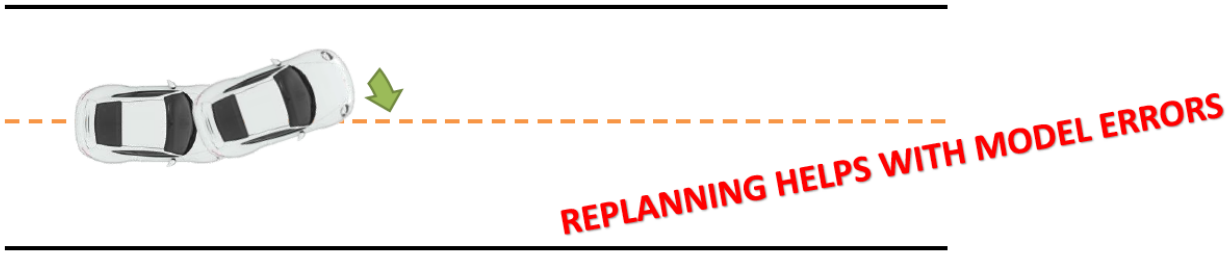
1. Nevertheless, the previous method executes all planned actions before fitting the model again. We may be off-grid too far already
2. So we can use Model Predictive Control (MPC) that we optimize the whole trajectory but we take the first action only, then we observe and replan again
3. In MPC, we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



Institute of Media, Information, and Network

Model-Based Policy Optimization

Can we do better?



model-based reinforcement learning version 1.5:

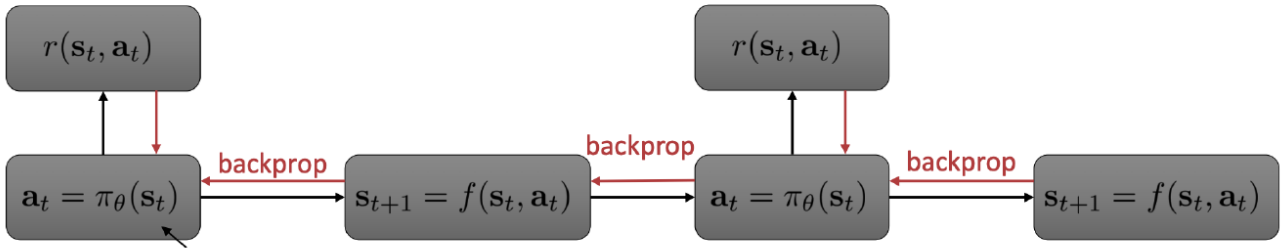
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

every N steps

Institute of Media, Information, and Network

Model-Based Policy Optimization

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 $\mathcal{D}$

Institute of Media, Information, and Network

Vanilla model-based deep reinforcement learning

- It includes model learning and policy learning.
- The model and the policy are represented by neural networks.
- In model learning, samples are collected from interaction with the environment, supervised learning is used to fit a dynamics model.
- In policy learning, the learned model is used to search an improved policy.
- Dynamics modeling with neural networks does not perform well and limited to relatively simple low-dimensional tasks.
- Model-based methods are more sample efficient and their performance is worse due to model bias.
- Model-free methods achieve better performance at the expense of higher sample complexity.

Model-Ensemble Trust-Region Policy Optimization, ICLR 2018.

Model learning

- Assume that state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function $\mathcal{R}$ are known
- Transition dynamics is modeled with a neural network, to predict the next state given a state and an action as inputs, i.e.,

$$s_{t+1} = \hat{f}_\phi(s_t, a_t)$$

- The objective is to find a parameter ϕ to minimize the loss function, i.e.,

$$\min_\phi \frac{1}{|\mathcal{D}|} \sum_{(s_t, a_t, s_{t+1}) \in \mathcal{D}} \left\| s_{t+1} - \hat{f}_\phi(s_t, a_t) \right\|_2^2$$

Policy learning

- During training, model-based methods maintain an approximate MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_\eta, \mathcal{R}_\eta \rangle$
- The policy is then updated w.r.t. the approximate MDP, to maximize the expected sum of rewards, i.e.,

$$\hat{\eta}(\theta; \phi) := \mathbb{E}_{\hat{\tau}}[\sum_{t=0}^{T} r(s_t, a_t)]$$

- The gradient can be estimated by BPTT method, i.e.,

$$\nabla_\theta \hat{\eta} = \mathbb{E}_{s_0 \sim \rho_0(s_0), \zeta_i \sim \mathcal{N}(0, I_m)}[\nabla_\theta \sum_{t=0}^{T} r(s_t, a_t)]$$

Vanilla model-based deep RL algorithm

Algorithm 1 Vanilla Model-Based Deep Reinforcement Learning

Initialize a policy π_θ and a model $\hat{f}_\phi$.
Initialize an empty dataset D.
repeat
    Collect samples from the real environment f using π_θ and add them to D.
    Train the model $\hat{f}_\phi$ using D.
    repeat
        Collect fictitious samples from $\hat{f}_\phi$ using π_θ.
        Update the policy using BPTT on the fictitious samples.
        Estimate the performance $\hat{\eta}(\theta; \phi)$.
    until the performance stop improving.
until the policy performs well in real environment f.

Institute of Media, Information, and Network