Lecture 8: Model-Based RL, Transfer RL, Hierarchical RL

Chunlin Chen & Zhi Wang

Department of Control and Systems Engineering
Nanjing University

Dec. 24th, 2019

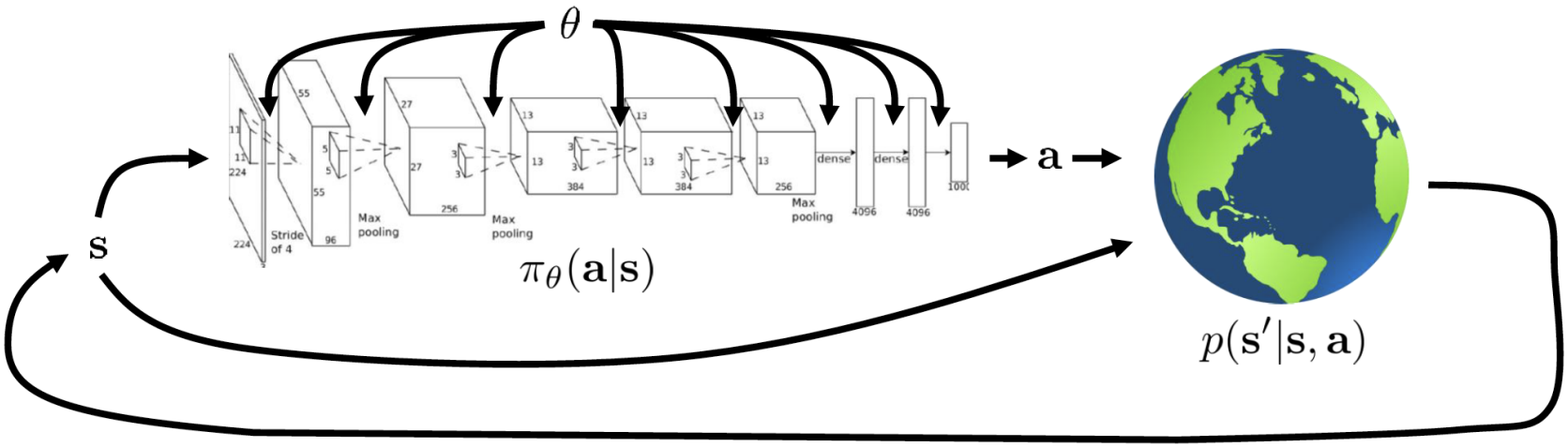
Table of Contents

Contents and Goals

- Optimal control and planning with known dynamics
- Basics of model-based RL: learn a model, use model for control
- Uncertainty in model-based RL
- Model-based RL with complex observations

- Goals
  - Understand how we can perform planning with known dynamics models in discrete and continuous spaces
  - Understand how to build model-based RL algorithms
  - Understand the important considerations for model-based RL
  - Understand the tradeoffs between different model class choices

Recall: The RL objective



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

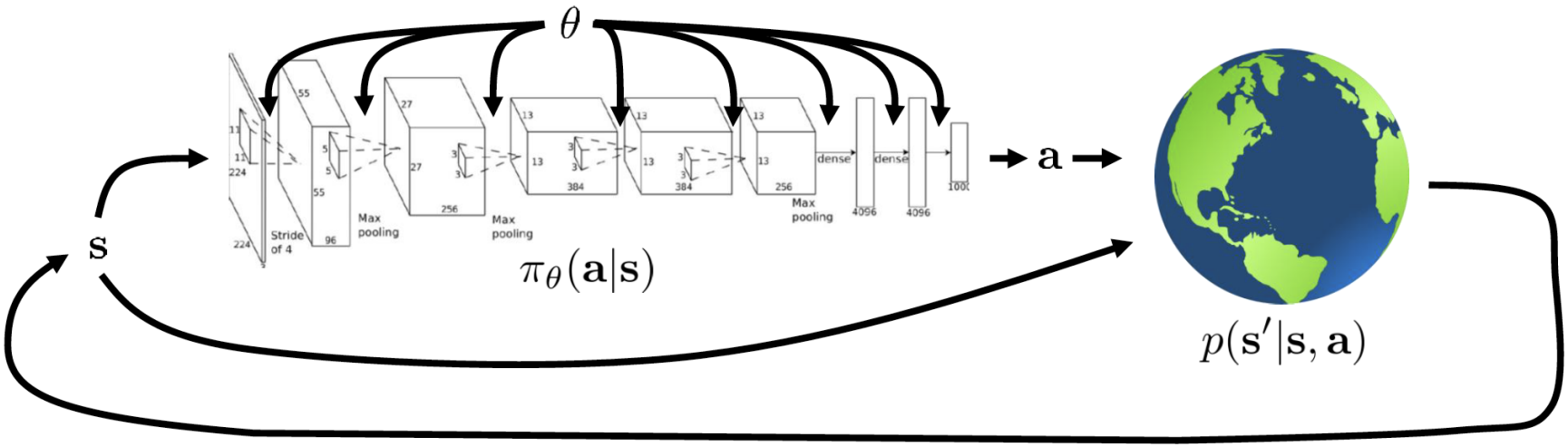
Dynamics of the MDP $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$

$$p(s', r|s, a) = Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}$$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

- The probabilities given by p completely characterize the environment's dynamics

Recall: Model-free RL



$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

assume this is unknown

don't even attempt to learn it

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

What if we knew the transition dynamics?

- Often we do know the dynamics
  1. Games (e.g., Atari games, chess, Go)
  2. Easily modeled systems (e.g., navigating a car)
  3. Simulated environments (e.g., simulated robots, video games)
- Often we can learn the dynamics
  1. System identification – fit unknown parameters of a known model
  2. Learning – fit a general-purpose model to observed transition data

Does knowing the dynamics make things easier?

Often, yes!

How can we make decisions if we know the dynamics?

- How can we choose actions under perfect knowledge of the system dynamics?
- Optimal control, trajectory optimization, planning

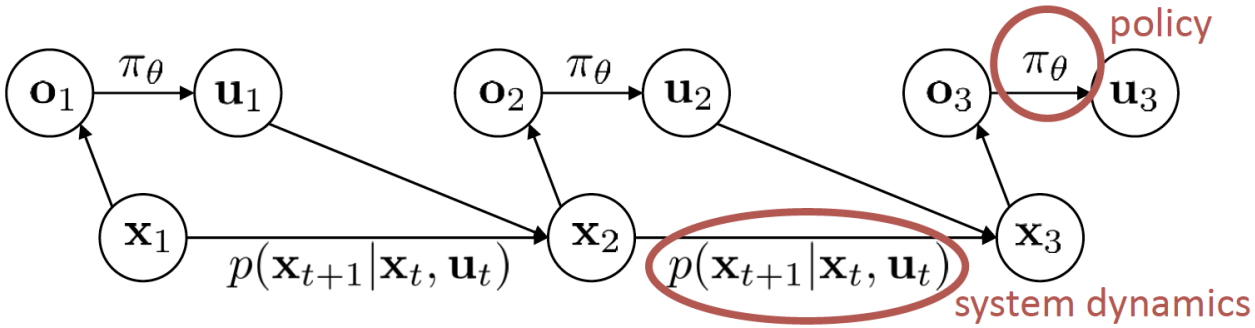
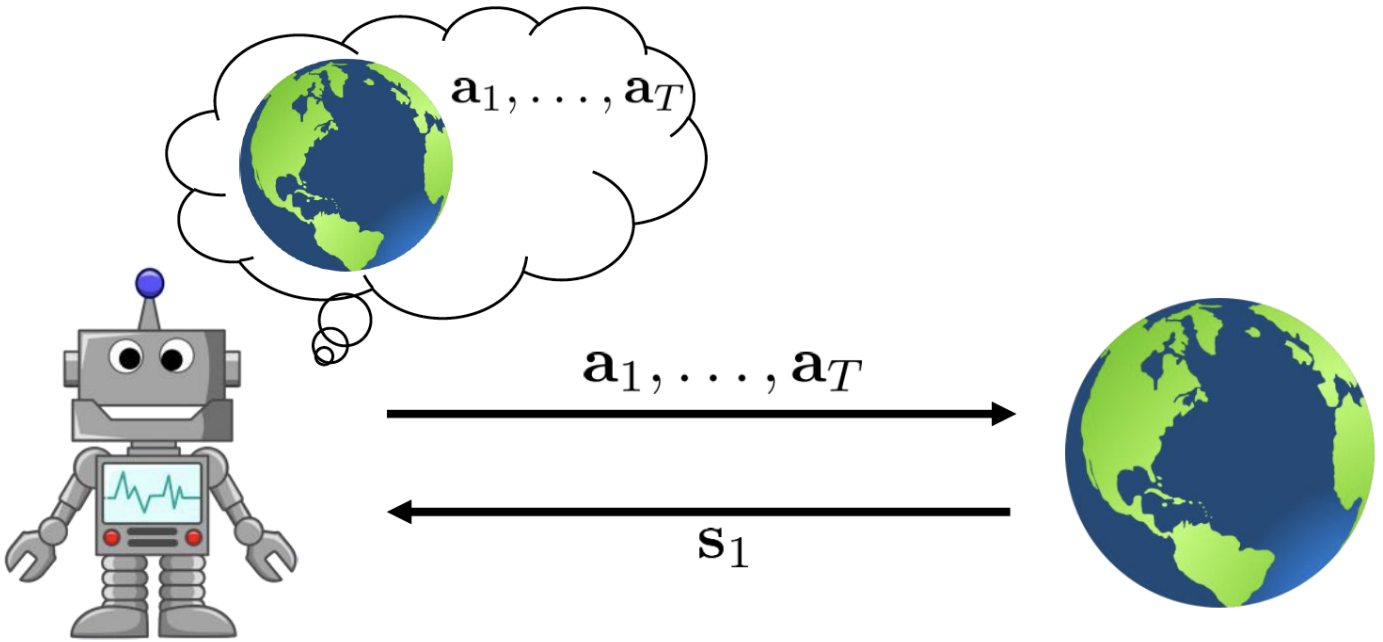


Table of Contents

The deterministic case



$$\mathbf{a}_1, \ldots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \ldots, \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$$

Method 1: Stochastic optimization

abstract away optimal control/planning:

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg\max_{\mathbf{a}_1,\dots,\mathbf{a}_T} \underbrace{J(\mathbf{a}_1, \dots, \mathbf{a}_T)}_{\text{don't care what this is}} \qquad \mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A})$$

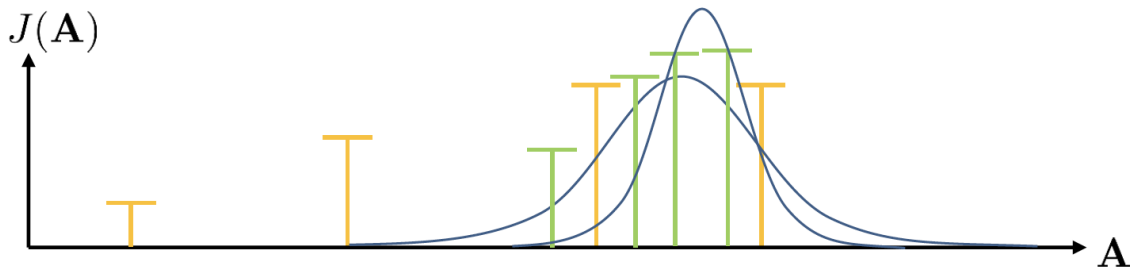
simplest method: guess & check "random shooting method"

1. pick $\mathbf{A}_1, \dots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

Method 2: Cross-entropy method (CEM)

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

can we do better?



typically use Gaussian distribution

see also: CMA-ES (sort of like CEM with momentum)

cross-entropy method with continuous-valued inputs:

1. sample $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from $p(\mathbf{A})$
2. evaluate $J(\mathbf{A}_1), \ldots, J(\mathbf{A}_N)$
3. pick the *elites* $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$ with the highest value, where $M < N$
4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$

Analysis of Methods 1 & 2

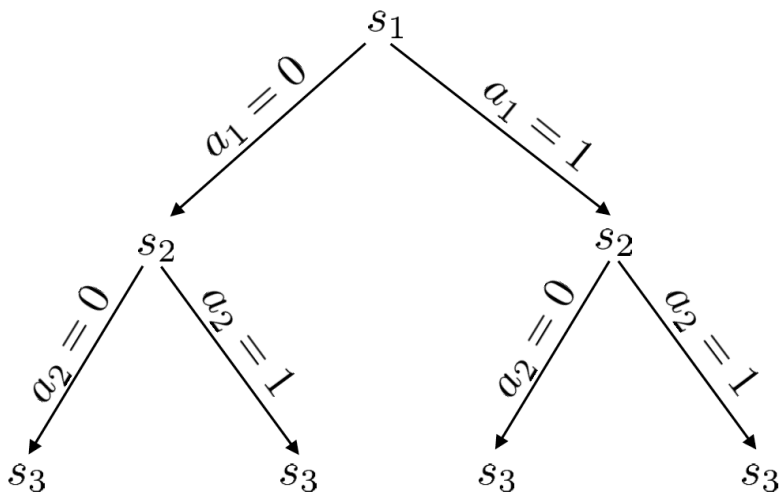
- What's the upside?
  - Very fast if parallelized
  - Extremely simple
- What's the problem?
  - Very harsh dimensionality limit
  - Only open-loop planning

Table of Contents

Discrete case: Monte Carlo tree search (MCTS)

s_t a_t

discrete planning as a search problem

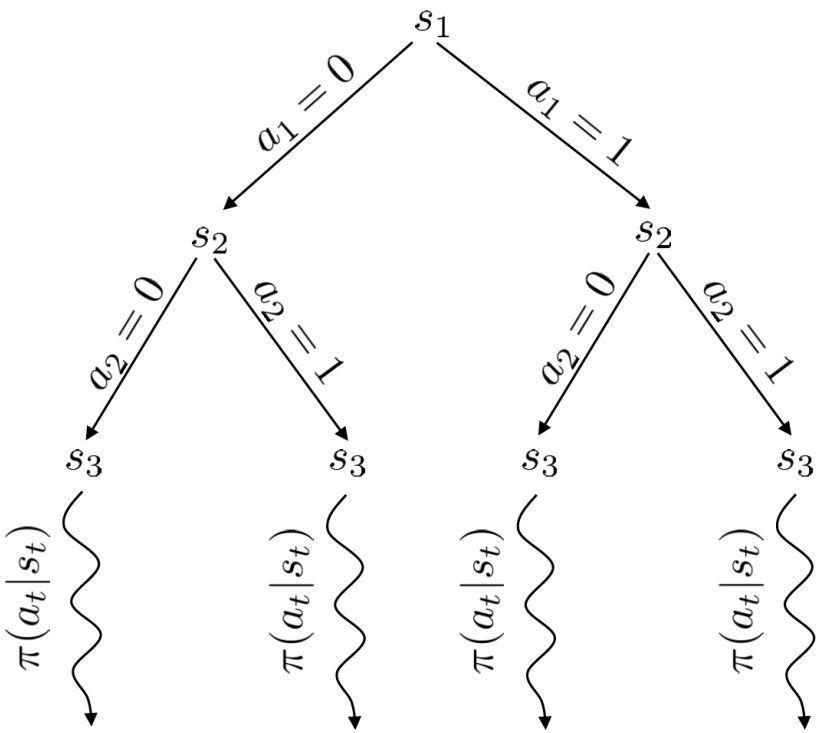


Discrete case: Monte Carlo tree search (MCTS)

s_t

a_t

how to approximate value without full tree?

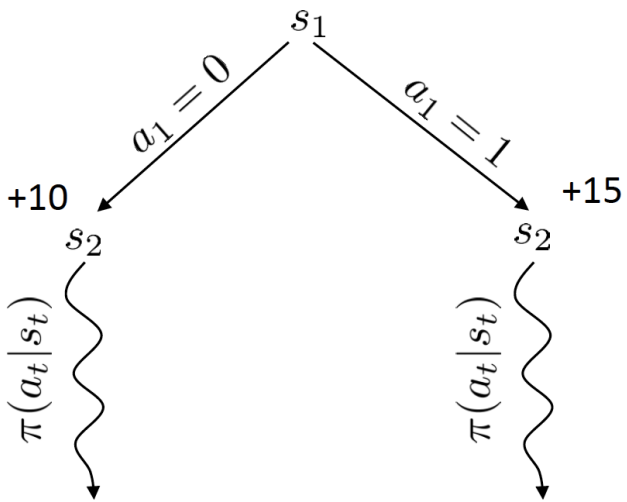


e.g., random policy

Discrete case: Monte Carlo tree search (MCTS)

s_t a_t

can't search all paths – where to search first?



intuition: choose nodes with best reward, but also prefer rarely visited nodes

Discrete case: Monte Carlo tree search (MCTS)

generic MCTS sketch

1. find a leaf s_l using TreePolicy(s_1)
2. evaluate the leaf using DefaultPolicy(s_l)
3. update all values in tree between s_1 and s_l

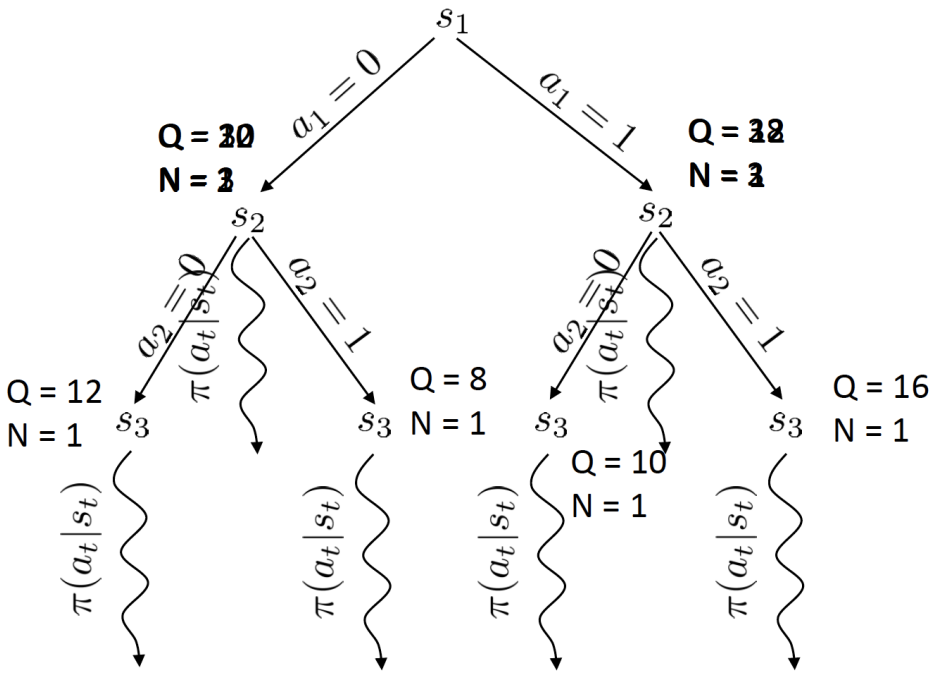
take best action from s_1

UCT TreePolicy(s_t)

if s_t not fully expanded, choose new a_t

else choose child with best Score(s_{t+1})

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$



Additional reading

- Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, Colton. (2012). **A Survey of Monte Carlo Tree Search Methods**.
  - Survey of MCTS methods and basic summary

Table of Contents

Can we use derivatives?

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

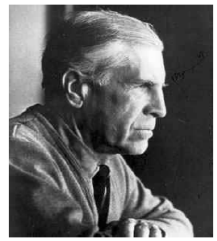
usual story: differentiate via backpropagation and optimize!

need $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}, \frac{dc}{d\mathbf{x}_t}, \frac{dc}{d\mathbf{u}_t}$

in practice, it really helps to use a 2^{nd} order method!

$\mathbf{s}_t$ – state
$\mathbf{a}_t$ – action

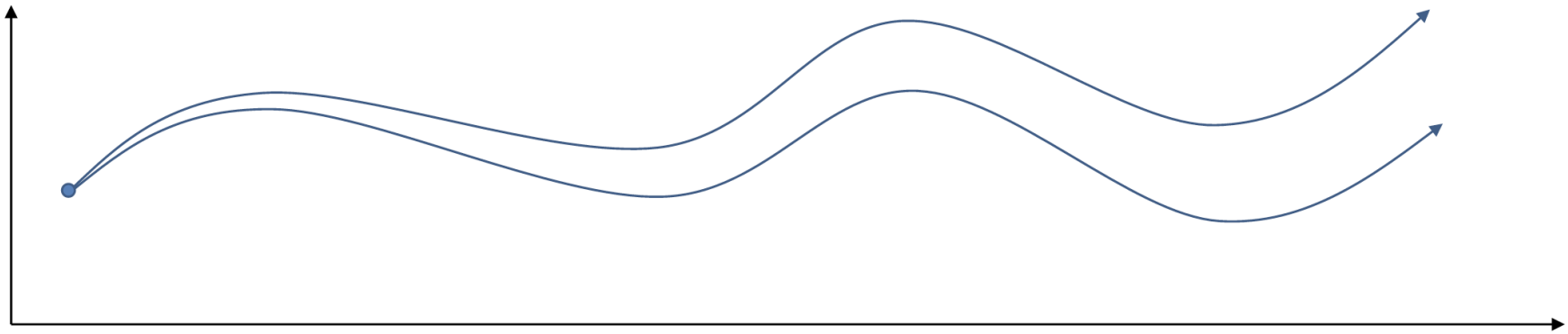
$\mathbf{x}_t$ – state
$\mathbf{u}_t$ – action



Shooting methods vs. collocation

shooting method: optimize over actions only

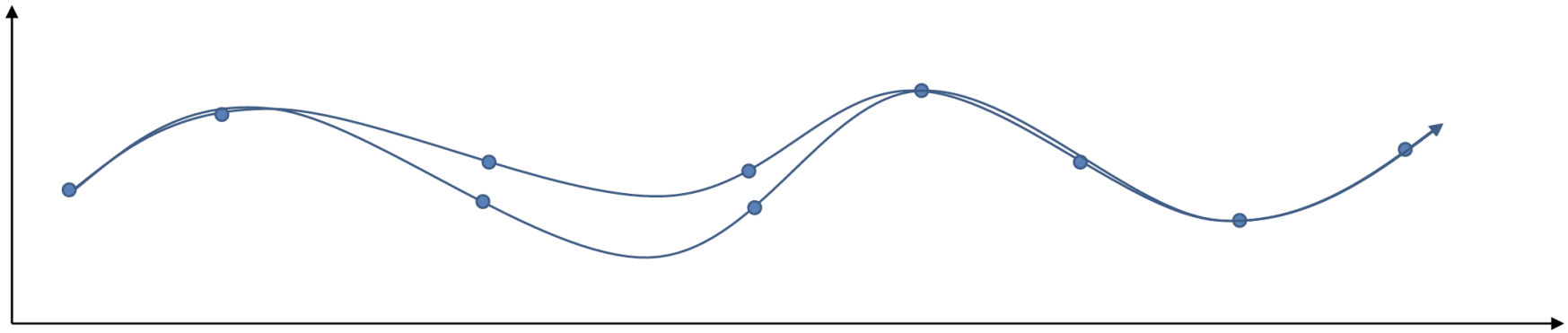
$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + c(f(f(\dots)\dots),\mathbf{u}_T)$$



Shooting methods vs. collocation

collocation method: optimize over actions and states, with constraints

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T} \sum_{t=1}^{T} c(\mathbf{x}_t,\mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1},\mathbf{u}_{t-1})$$



Linear case: Linear-quadratic regulator (LQR)

- The theory of optimal control is concerned with operating a dynamic system at minimum cost
- The LQ problem: the system dynamics are described by a set of linear differential equations, the cost is described by a quadratic function
- LQR: a feedback controller, one of the most fundamental problems in control theory

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + c(f(f(\ldots)\ldots),\mathbf{u}_T)$$

$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

linear

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

quadratic

Linear case: LQR

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1,\mathbf{u}_1) + c(f(\mathbf{x}_1,\mathbf{u}_1),\mathbf{u}_2) + \cdots + \underbrace{c(f(f(\ldots)\ldots),\mathbf{u}_T)}_{\mathbf{x}_T \text{ (unknown)}}$$

$$c(\mathbf{x}_t,\mathbf{u}_t) = \frac{1}{2}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{C}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix}^T \mathbf{c}_t$$

only term that depends on $\mathbf{u}_T$

$$f(\mathbf{x}_t,\mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix}\mathbf{x}_t\\ \mathbf{u}_t\end{bmatrix} + \mathbf{f}_t$$

Base case: solve for $\mathbf{u}_T$ *only*

$$\mathbf{C}_T = \begin{bmatrix}\mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T}\\ \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\end{bmatrix}$$

$$\mathbf{c}_T = \begin{bmatrix}\mathbf{c}_{\mathbf{x}_T}\\ \mathbf{c}_{\mathbf{u}_T}\end{bmatrix}$$

$$Q(\mathbf{x}_T,\mathbf{u}_T) = \text{const} + \frac{1}{2}\begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix}^T \mathbf{C}_T \begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix} + \begin{bmatrix}\mathbf{x}_T\\ \mathbf{u}_T\end{bmatrix}^T \mathbf{c}_T$$

$$\nabla_{\mathbf{u}_T} Q(\mathbf{x}_T,\mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}\mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0$$

$$\mathbf{u}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\left(\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T}\mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T}\right) \qquad \mathbf{u}_T = \mathbf{K}_T\mathbf{x}_T + \mathbf{k}_T$$

$$\mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \qquad \mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1}\mathbf{c}_{\mathbf{u}_T}$$

Linear case: LQR

$$\mathbf{u}_T = \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \qquad \mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1} \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \qquad \mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T}^{-1} \mathbf{c}_{\mathbf{u}_T}$$

$$Q(\mathbf{x}_T, \mathbf{u}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{c}_T$$

Since $\mathbf{u}_T$ is fully determined by $\mathbf{x}_T$, we can eliminate it via substitution!

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{c}_T$$

$$\begin{aligned} V(\mathbf{x}_T) = & \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T + \\ & \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{k}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{k}_T + \mathbf{x}_T^T \mathbf{c}_{\mathbf{x}_T} + \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{c}_{\mathbf{u}_T} + \text{const} \end{aligned}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T$$

$$\mathbf{V}_T = \mathbf{C}_{\mathbf{x}_T,\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{K}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{x}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{K}_T$$

$$\mathbf{v}_T = \mathbf{c}_{\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T,\mathbf{u}_T} \mathbf{k}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T,\mathbf{u}_T} \mathbf{k}_T$$

Linear case: LQR

Solve for $\mathbf{u}_{T-1}$ in terms of $\mathbf{x}_{T-1}$

$\mathbf{u}_{T-1}$ affects $\mathbf{x}_T$!

$$f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{x}_T = \mathbf{F}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \mathbf{f}_{T-1}$$

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + \overbrace{V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))}^{V(\mathbf{x}_T) = \text{const} + \frac{1}{2}\mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{v}_T}_{\text{linear}}$$

Linear case: LQR

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{v}_T}_{\text{linear}}$$

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{Q}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{q}_{T-1}$$

$$\mathbf{Q}_{T-1} = \mathbf{C}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}$$

$$\mathbf{q}_{T-1} = \mathbf{c}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{v}_T$$

$$\nabla_{\mathbf{u}_{T-1}} Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}} \mathbf{x}_{T-1} + \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}} \mathbf{u}_{T-1} + \mathbf{q}_{\mathbf{u}_{T-1}}^T = 0$$

$$\mathbf{u}_{T-1} = \mathbf{K}_{T-1} \mathbf{x}_{T-1} + \mathbf{k}_{T-1} \qquad \mathbf{K}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}} \qquad \mathbf{k}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{q}_{\mathbf{u}_{T-1}}$$

Linear case: LQR

Backward recursion

for $t = T$ to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

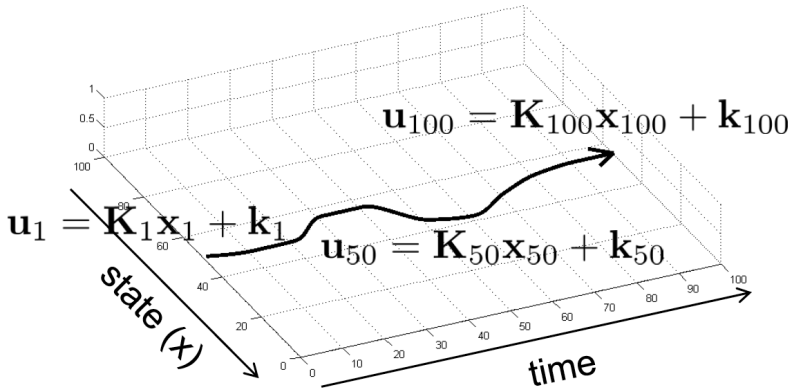
$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$



we know $\mathbf{x}_1$!

Forward recursion

for $t = 1$ to T:

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

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

Linear case: LQR

Backward recursion

for $t = T$ to 1:

total cost from now until end if we take $\mathbf{u}_t$ from state $\mathbf{x}_t$

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg\min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$

total cost from now until end from state $\mathbf{x}_t$

$$V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t)$$

Table of Contents

Model-based RL

- Often the system's dynamics are **unknown**
- Model-based RL: learn the transition dynamics, then figure out how to choose actions

Why learn the model?

- If we knew $f(s_t, a_t) = s_{t+1}$, we could use tools from "Optimal Control and Planning"
  - Or $p(s_{t+1}|s_t, a_t)$ in the stochastic case
- So let's learn $f(s_t, a_t)$ from data, and then **plan** through it

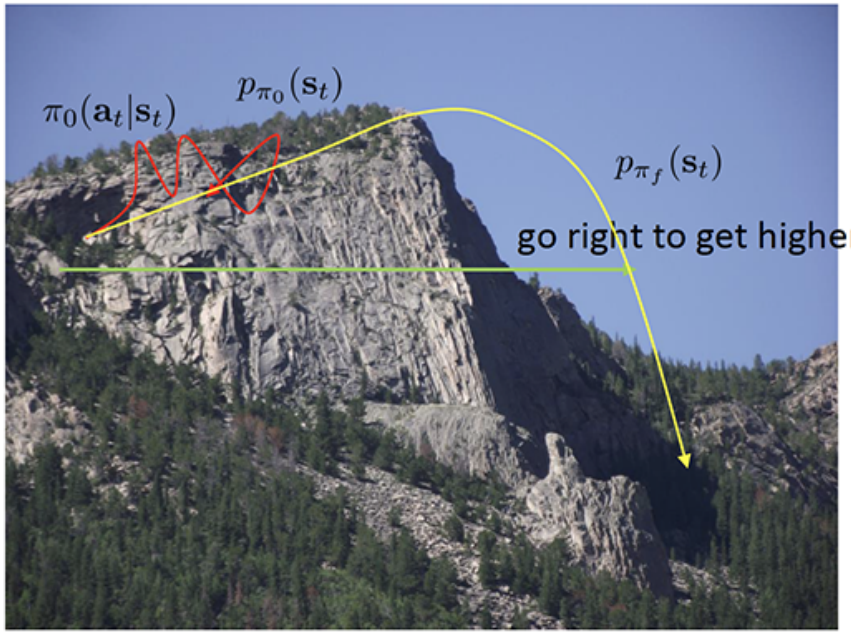
Model-based RL version 0.5

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

Does it work? Yes!

- Essentially how system identification works in classical robotics
- Some care should be taken to design a good base policy
- Particularly effective if we can hand engineer a dynamics representation using our knowledge of physics, and fit just a few parameters

Does it work? No!



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

$$p_{\pi_f}(\mathbf{s}_t) \neq p_{\pi_0}(\mathbf{s}_t)$$

- Distribution mismatch problem becomes exacerbated as we use more expressive model classes

Model-based RL version 1.0

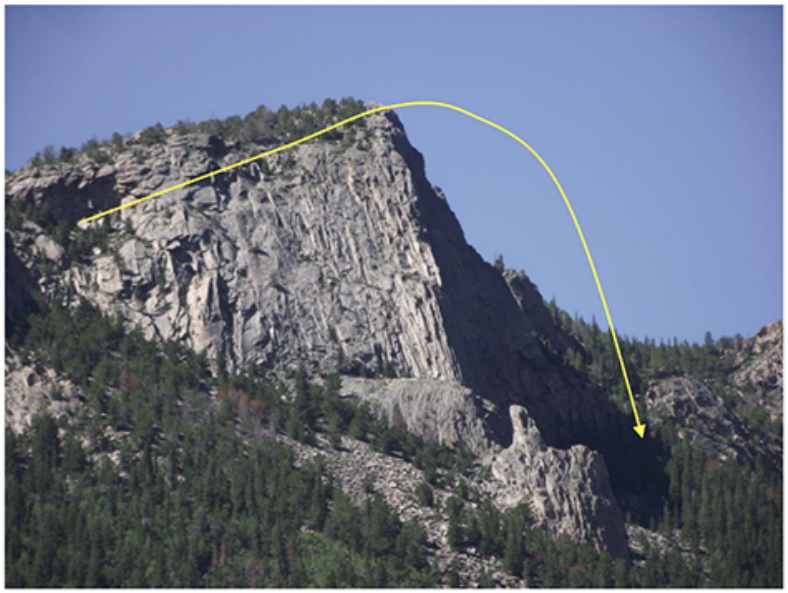
can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?

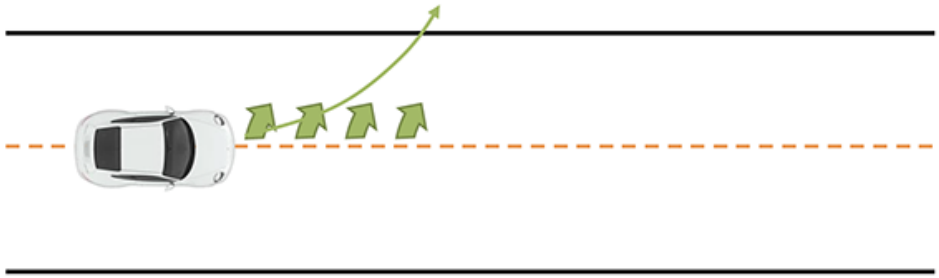
where have we seen that before? need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

model-based reinforcement learning version 1.0:

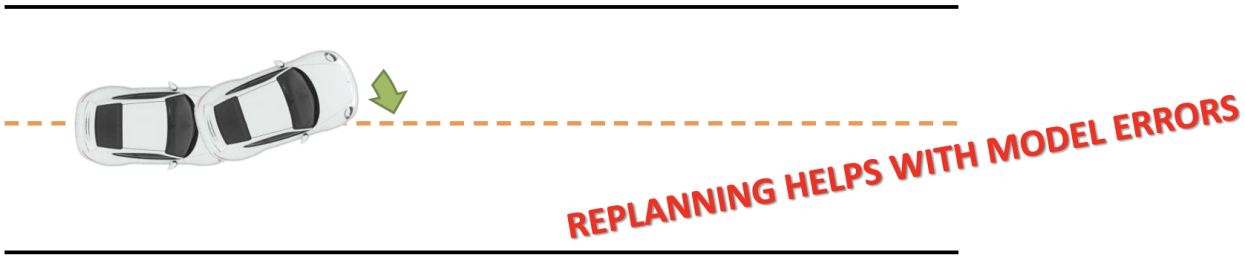
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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

What if we make a mistake? - Error cumulation





Model-based RL version 1.5



model-based reinforcement learning version 1.5:

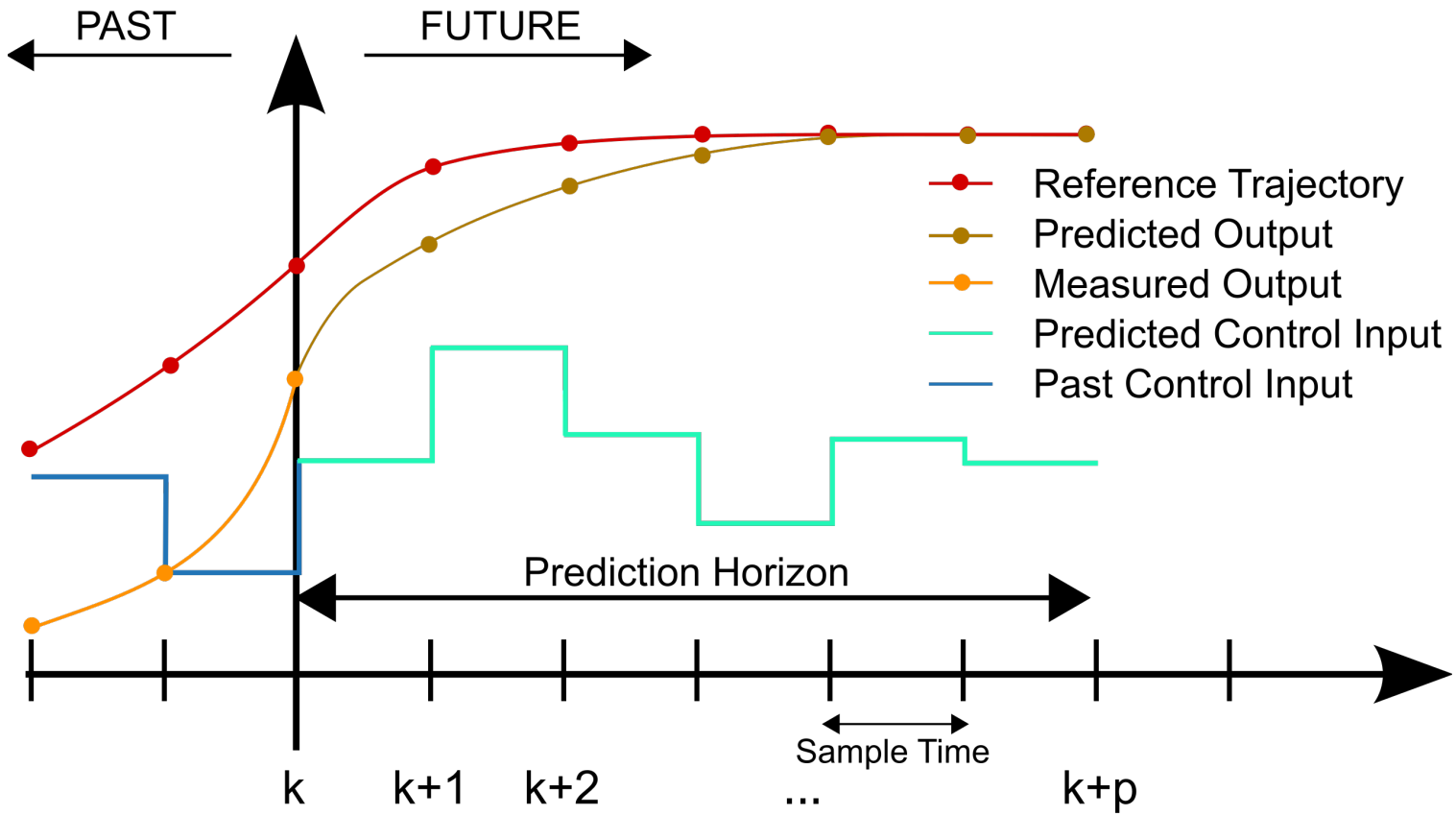
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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

Model predictive control (MPC)

- An advanced method of process control that is used to control a process while satisfying a set of constraints
- Allows the current timeslot to be optimized, while keeping future timeslots in account
- Optimizes a finite time-horizon, but only implementing the current timeslot and then optimizing again, repeatedly

Model predictive control (MPC)



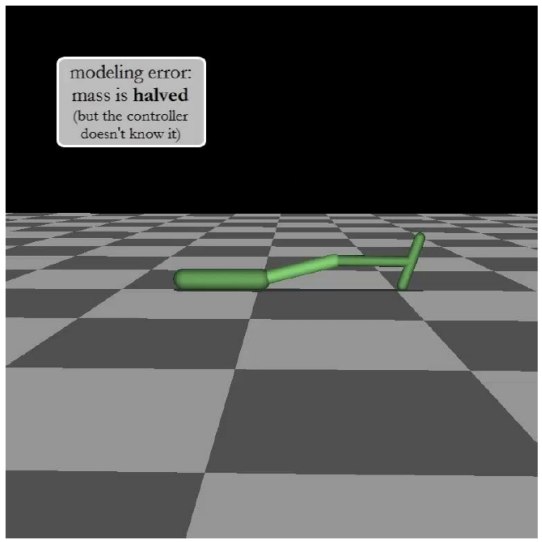
How to replan? - Refer to "Optimal Control and Planning"

model-based reinforcement learning version 1.5:

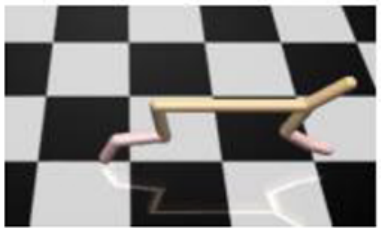
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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

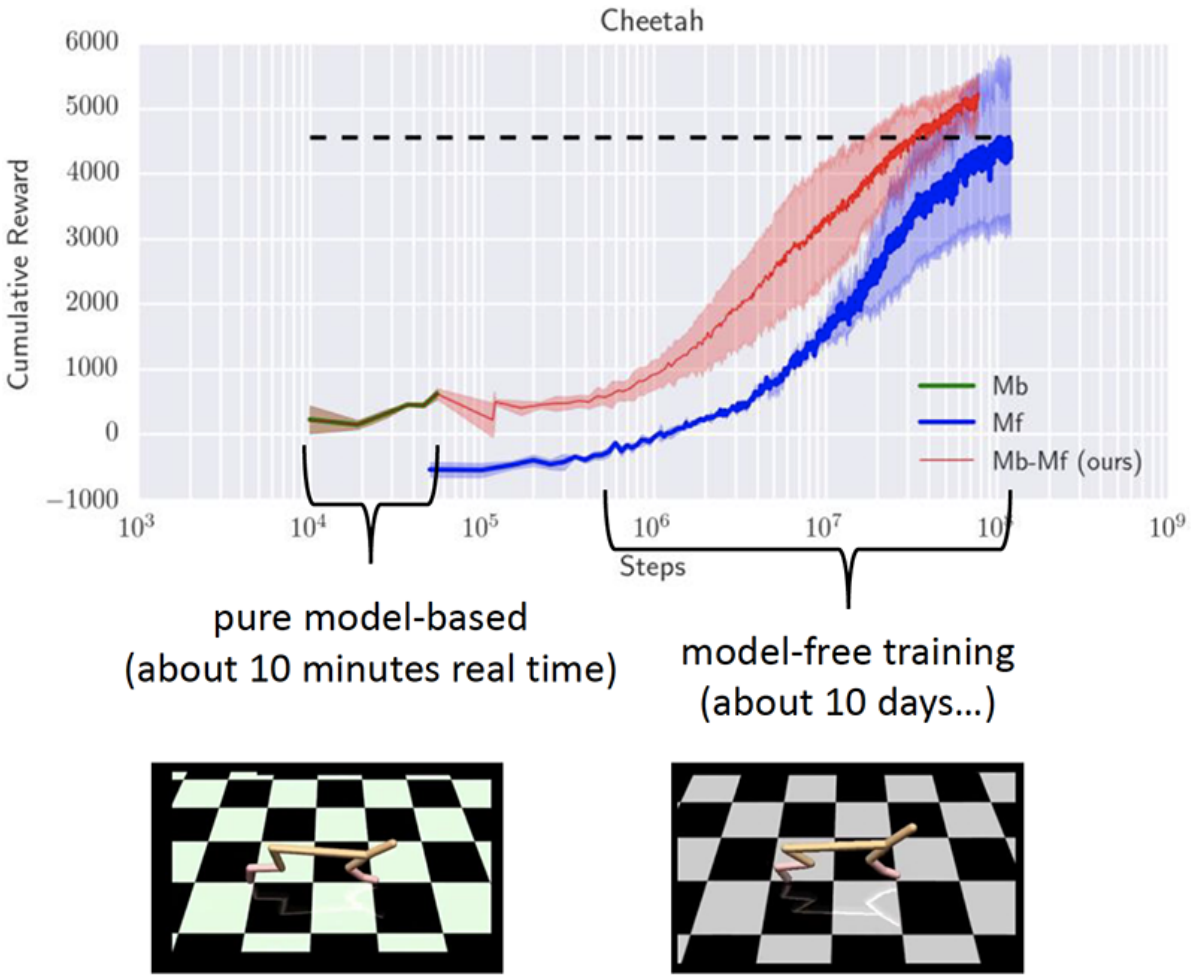
every N steps

- The more you replan, the less perfect each individual plan needs to be
- Can use shorter horizons
- Even random sampling can often work well here!



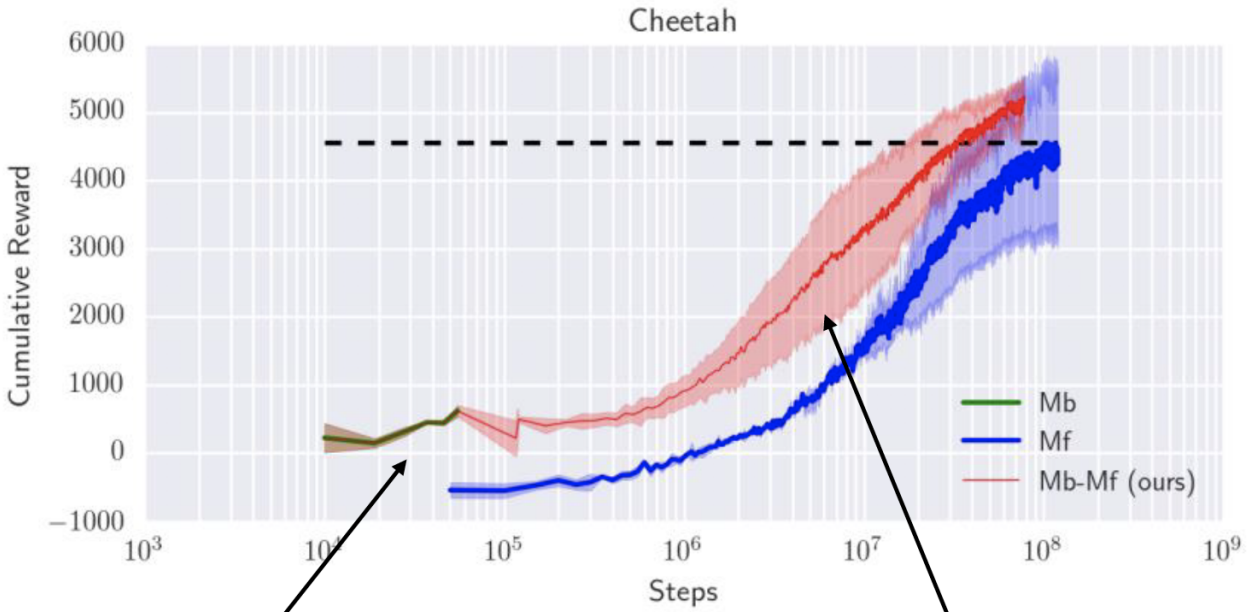
A performance gap in model-based RL





- Anusha Nagabandi, et al., "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning," *IEEE International Conference on Robotics and Automation (ICRA)*.

Why the performance gap?

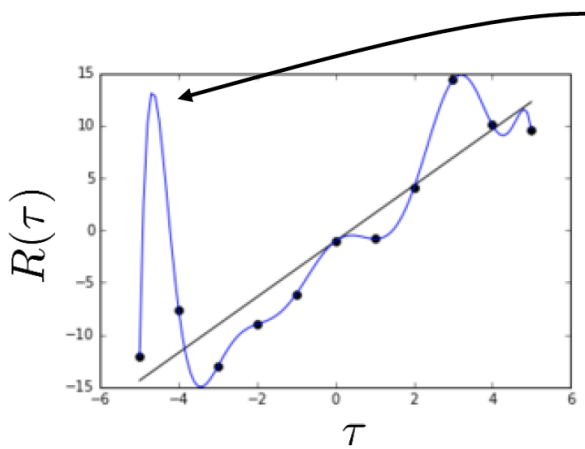


Why the performance gap?

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 $\mathcal{D}$

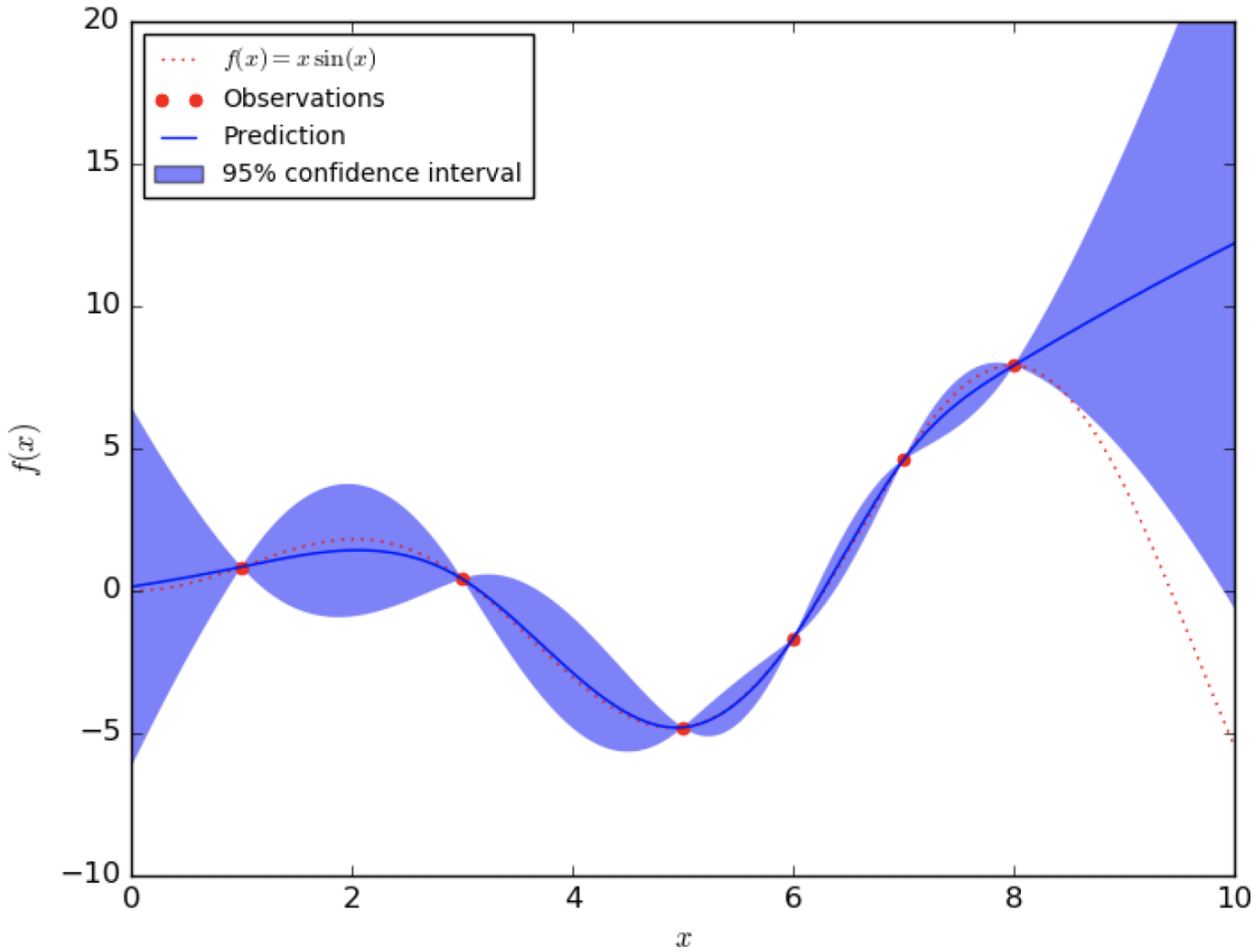
every N steps



very tempting to go here...

Table of Contents

Uncertainty - Bayesian models



Intuition behind uncertainty-aware RL

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

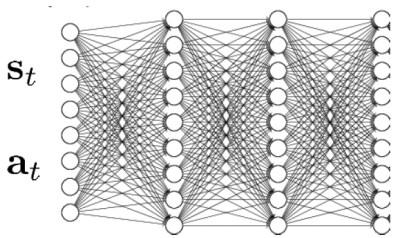
only take actions for which we think we'll get high reward in expectation (w.r.t. uncertain dynamics)

This avoids "exploiting" the model

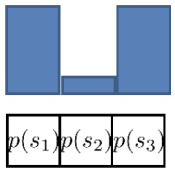
The model will then adapt and get better

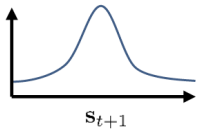
How can we have uncertainty-aware models?

Idea 1: use output entropy

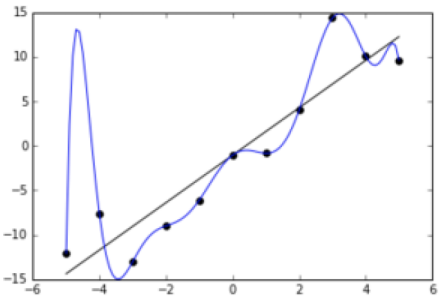


$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$





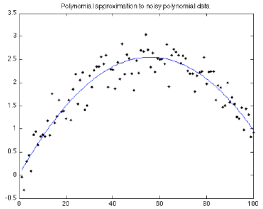
why is this not enough?



what is the variance here?

Two types of uncertainty:

aleatoric or *statistical* uncertainty →



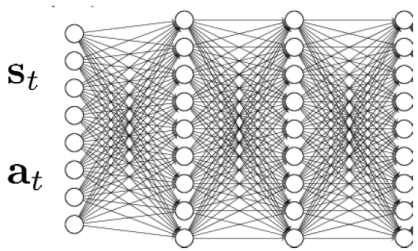
← *epistemic* or *model* uncertainty

"the model is certain about the data, but we are not certain about the model"

How can we have uncertainty-aware models?

Idea 2: estimate model uncertainty

"the model is certain about the data, but we are not certain about the model"



$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

parameters θ

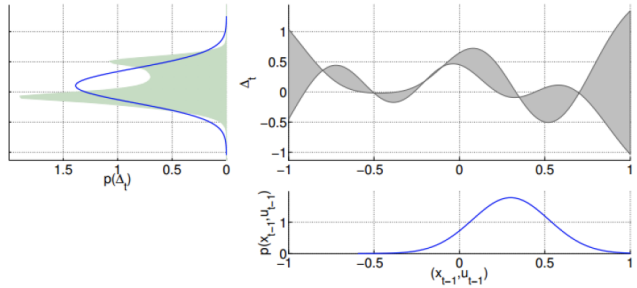
usually, we estimate

$$\arg\max_\theta \log p(\theta|\mathcal{D}) = \arg\max_\theta \log p(\mathcal{D}|\theta)$$

can we instead estimate $p(\theta|\mathcal{D})$?

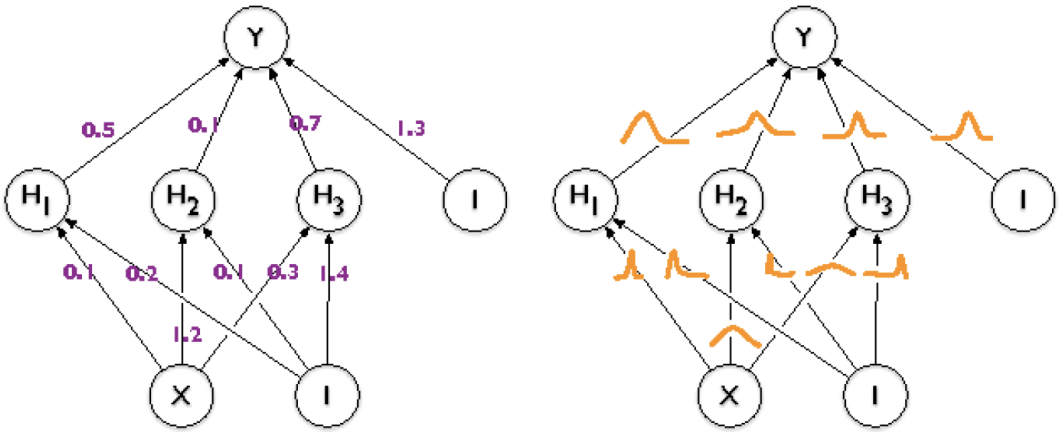
predict according to:

$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta) p(\theta|\mathcal{D}) d\theta$$



the entropy of this tells us the model uncertainty!

Quick overview of Bayesian neural networks



common approximation:

$$p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$$

$$p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$$

expected weight

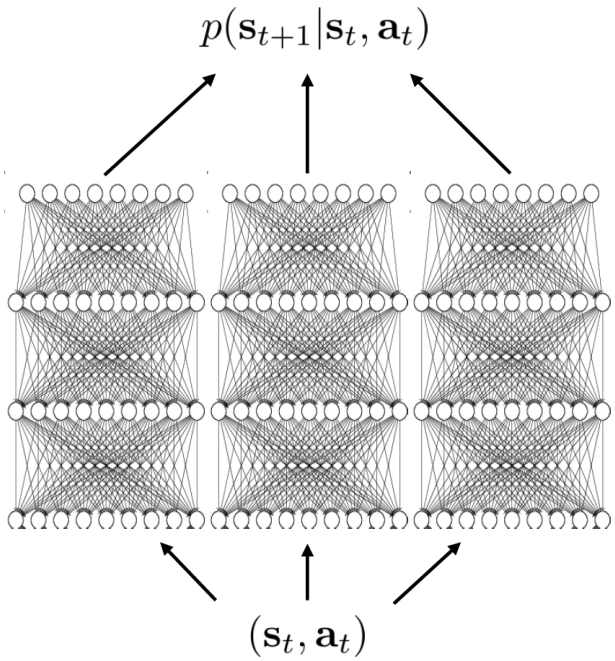
uncertainty about the weight

For more, see:

Blundell et al., Weight Uncertainty in Neural Networks

Gal et al., Concrete Dropout

Bootstrap ensembles



Train multiple models and see if they agree!

$$\text{formally:} \quad p(\theta|\mathcal{D}) \approx \frac{1}{N}\sum_i \delta(\theta_i)$$

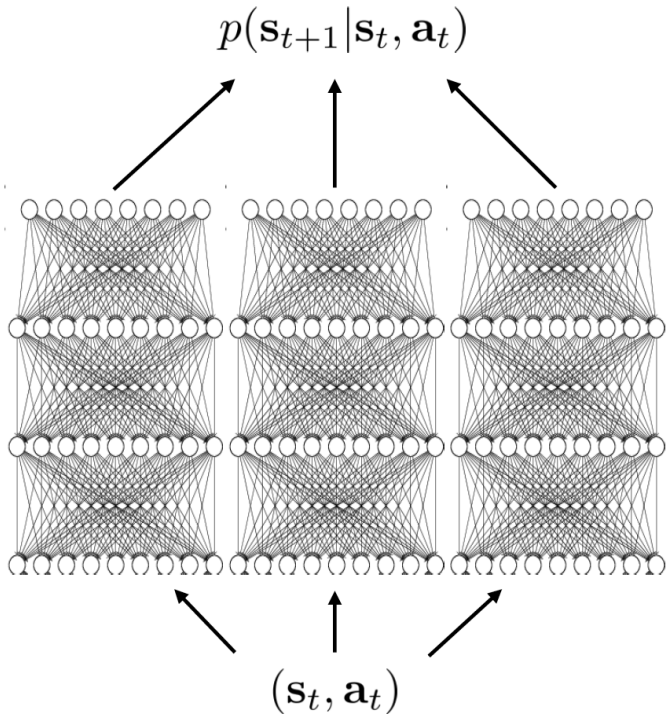
$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{N}\sum_i p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta_i)$$

How to train?

Main idea: need to generate “independent” datasets to get “independent” models

θ_i is trained on $\mathcal{D}_i$, sampled *with replacement* from $\mathcal{D}$

Bootstrap ensembles in deep learning



This basically works

Very crude approximation, because the number of models is usually small (< 10)

Resampling with replacement is usually unnecessary, because SGD and random initialization usually makes the models sufficiently independent

How to plan with uncertainty

Before: $J(\mathbf{a}_1, \ldots, \mathbf{a}_H) = \sum_{t=1}^{H} r(\mathbf{s}_t, \mathbf{a}_t)$, where $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t)$

Now: $J(\mathbf{a}_1, \ldots, \mathbf{a}_H) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} r(\mathbf{s}_{t,i}, \mathbf{a}_t)$, where $\mathbf{s}_{t+1,i} = f_i(\mathbf{s}_{t,i}, \mathbf{a}_t)$

distribution over deterministic models

In general, for candidate action sequence $\mathbf{a}_1, \ldots, \mathbf{a}_H$:

Step 1: sample $\theta \sim p(\theta|\mathcal{D})$

Step 2: at each time step t, sample $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)$

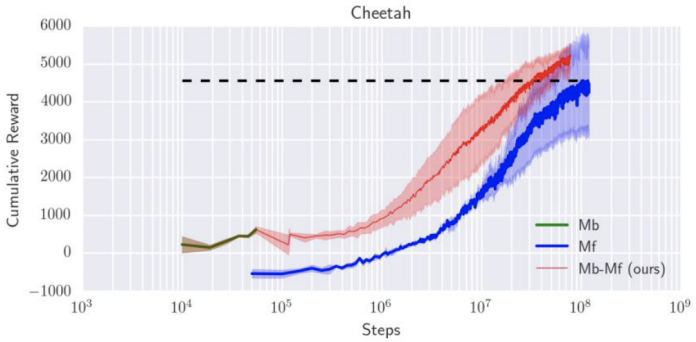
Step 3: calculate $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

Step 4: repeat steps 1 to 3 and accumulate the average reward

Other options: moment matching, more complex posterior estimation with BNNs, etc.

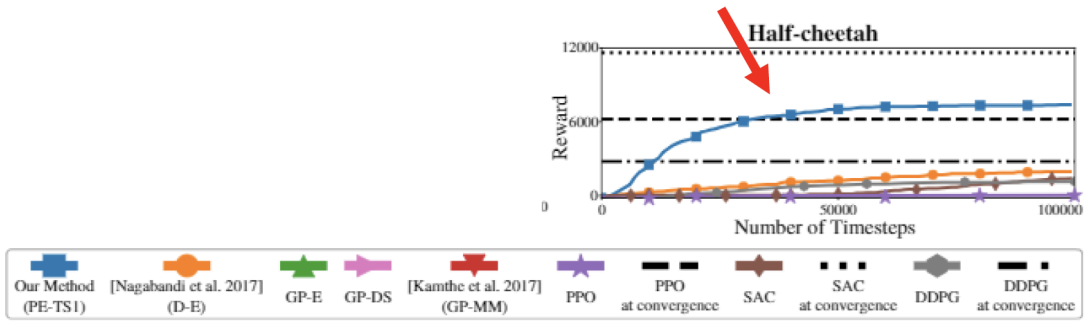
Example: model-based RL with ensembles

Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models



before

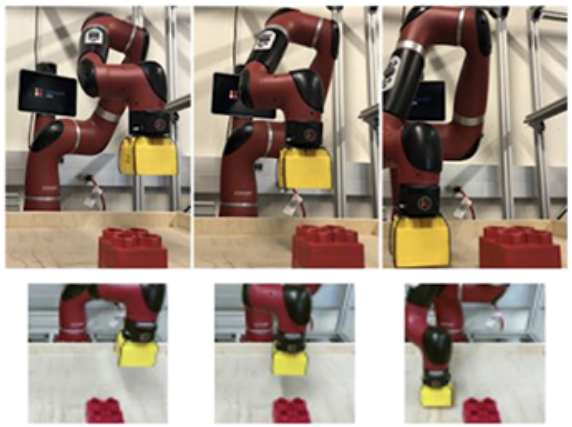
exceeds performance of model-free after 40k steps (about 10 minutes of real time)



after

Table of Contents

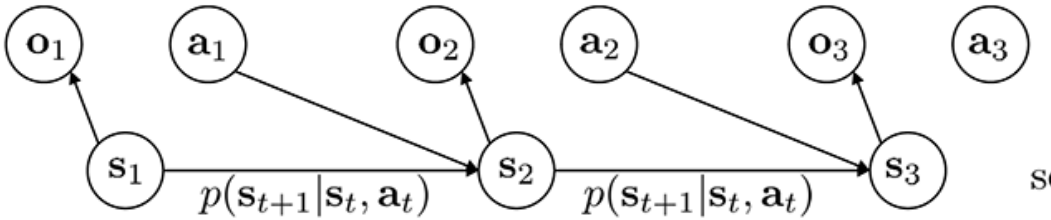
What about complex observations?



$$f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$$

What is hard about this?

- High dimensionality
- Redundancy
- Partial observability

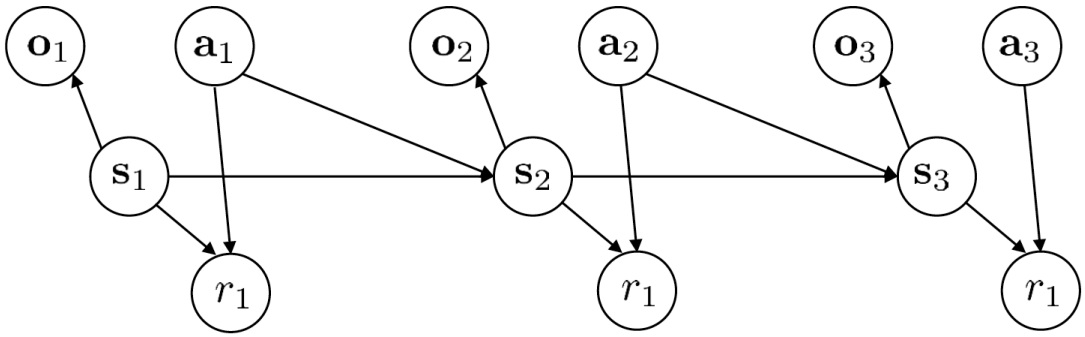


high-dimensional but not dynamic

low-dimension but dynamic

separately learn $p(\mathbf{o}_t|\mathbf{s}_t)$ and $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$?

State space (latent space) models



$p(\mathbf{o}_t|\mathbf{s}_t)$ observation model

$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ dynamics model

$p(r_t|\mathbf{s}_t, \mathbf{a}_t)$ reward model

How to train?

standard (fully observed) model: $\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i})$

latent space model: $\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} E\left[\log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i}|\mathbf{s}_{t,i})\right]$

expectation w.r.t. $(\mathbf{s}_t, \mathbf{s}_{t+1}) \sim p(\mathbf{s}_t, \mathbf{s}_{t+1}|\mathbf{o}_{1:T}, \mathbf{a}_{1:T})$

Model-based RL with latent space models

$$\max_{\phi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} E\left[\log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i}|\mathbf{s}_{t,i})\right]$$

expectation w.r.t. $(\mathbf{s}_t, \mathbf{s}_{t+1}) \sim p(\mathbf{s}_t, \mathbf{s}_{t+1}|\mathbf{o}_{1:T}, \mathbf{a}_{1:T})$

learn *approximate* posterior $q_\psi(\mathbf{s}_t|\mathbf{o}_{1:t}, \mathbf{a}_{1:t})$ "encoder"

many other choices for approximate posterior:

$q_\psi(\mathbf{s}_t, \mathbf{s}_{t+1}|\mathbf{o}_{1:T}, \mathbf{a}_{1:T})$ full smoothing posterior

+ most accurate
- most complicated

$q_\psi(\mathbf{s}_t|\mathbf{o}_t)$ single-step encoder

+ simplest
- least accurate

we'll talk about this one for now

Model-based RL with latent space models

$$\max_{\phi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} E\left[\log p_\phi(\mathbf{s}_{t+1,i}|\mathbf{s}_{t,i}, \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i}|\mathbf{s}_{t,i})\right]$$

expectation w.r.t. $\mathbf{s}_t \sim q_\psi(\mathbf{s}_t|\mathbf{o}_t), \mathbf{s}_{t+1} \sim q_\psi(\mathbf{s}_{t+1}|\mathbf{o}_{t+1})$

$q_\psi(\mathbf{s}_t|\mathbf{o}_t)$

simple special case: $q(\mathbf{s}_t|\mathbf{o}_t)$ is *deterministic*

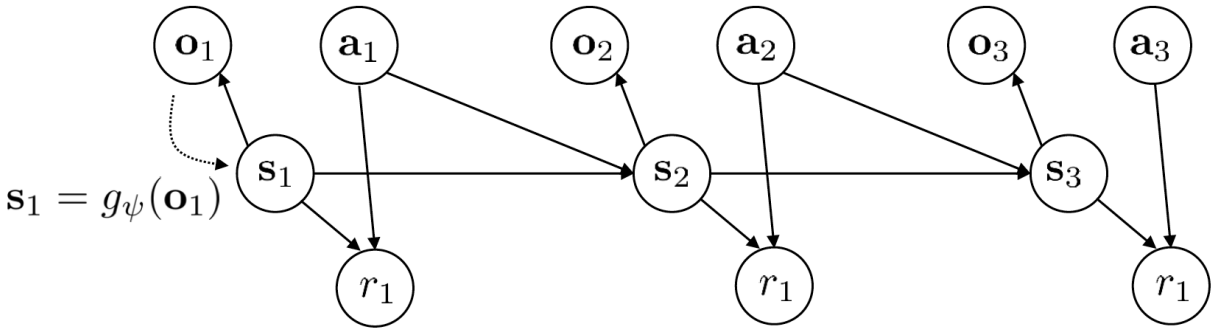
stochastic case requires variational inference

$q_\psi(\mathbf{s}_t|\mathbf{o}_t) = \delta(\mathbf{s}_t = g_\psi(\mathbf{o}_t)) \Rightarrow \mathbf{s}_t = g_\psi(\mathbf{o}_t)$ deterministic encoder

$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(g_\psi(\mathbf{o}_{t+1,i})|g_\psi(\mathbf{o}_{t,i}), \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i}|g_\psi(\mathbf{o}_{t,i}))$$

Everything is differentiable, can train with backprop

Model-based RL with latent space models

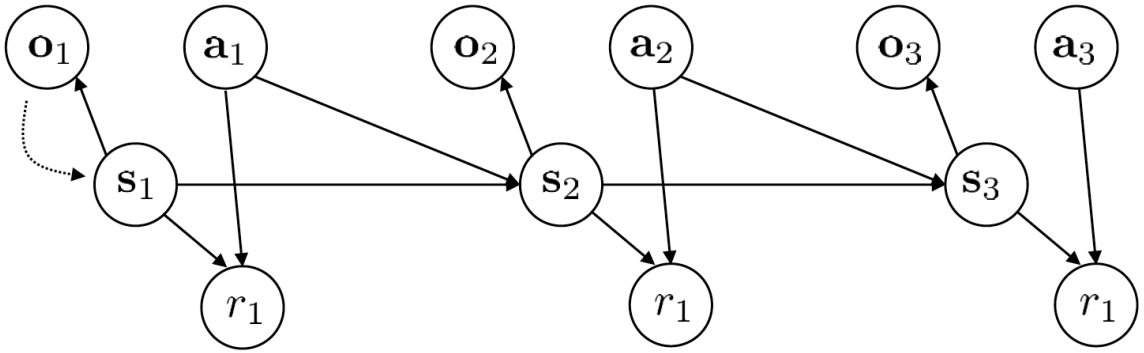


$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(g_\psi(\mathbf{o}_{t+1,i}) | g_\psi(\mathbf{o}_{t,i}), \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i} | g_\psi(\mathbf{o}_{t,i})) + \log p_\phi(r_{t,i} | g_\psi(\mathbf{o}_{t,i}))$$

latent space dynamics | image reconstruction | reward model

Many practical methods use a stochastic encoder to model uncertainty

Model-based RL with latent space models



model-based reinforcement learning with latent state:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{o}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{o}, \mathbf{a}, \mathbf{o}')_i\}$
2. learn $p_\phi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, $p_\phi(r_t|\mathbf{s}_t)$, $p(\mathbf{o}_t|\mathbf{s}_t)$, $g_\psi(\mathbf{o}_t)$
3. plan through the model to choose actions
4. execute the first planned action, observe resulting $\mathbf{o}'$ (MPC)
5. append $(\mathbf{o}, \mathbf{a}, \mathbf{o}')$ to dataset $\mathcal{D}$

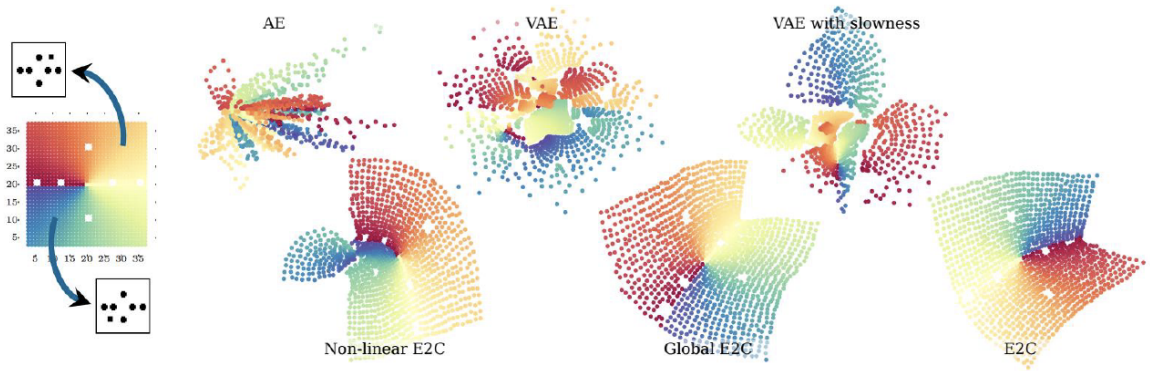
every N steps

Example: Latent dynamics model

Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images

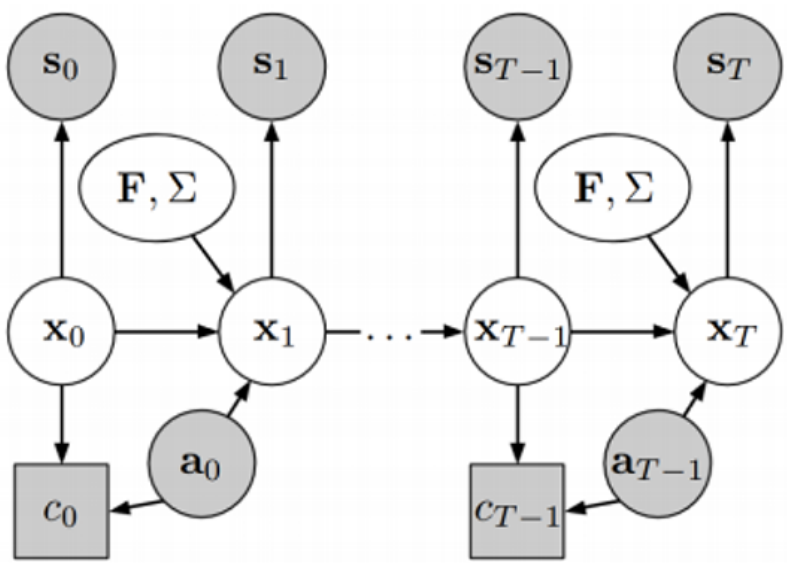
Manuel Watter* **Jost Tobias Springenberg***
Joschka Boedecker
University of Freiburg, Germany
{watterm, springj, jboedeck}@cs.uni-freiburg.de

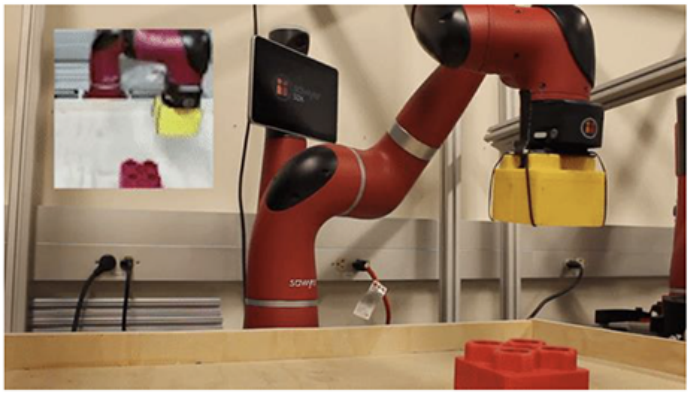
Martin Riedmiller
Google DeepMind
London, UK
riedmiller@google.com



Example: Deep structured latent representations

SOLAR: Deep Structured Latent Representations for Model-Based Reinforcement Learning





Learning objectives of this lecture

- You should be able to...
  - Know the common methods for optimal control and planning when the system's dynamics are known
    - Stochastic optimization, cross-entropy method, MCTS, LQR
  - Know how to learn the model, the different versions, practical considerations
  - Be aware of several advanced methods, uncertainty estimation, latent models

Model-based RL suggested readings

- Lecture 10 & 11 of CS285 at UC Berkeley, **Deep Reinforcement Learning, Decision Making, and Control**
  - `http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-10.pdf`
  - `http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-11.pdf`
- Recent papers
  - Nagabandi et al., **Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning**, IEEE ICRA 2017.
  - Grady Williams et al., **Aggressive Driving with Model Predictive Path Integral Control**, IEEE ICRA 2016.
  - Buckman et al., **Sample Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion**, NIPS 2018.

Table of Contents

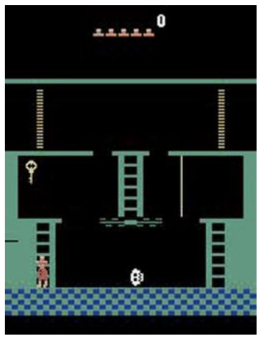
Transfer learning terminology

transfer learning: using experience from one set of tasks for faster learning and better performance on a new task

in RL, task = MDP!

source domain

target domain



"shot": number of attempts in the target domain

0-shot: just run a policy trained in the source domain

1-shot: try the task once

few shot: try the task a few times

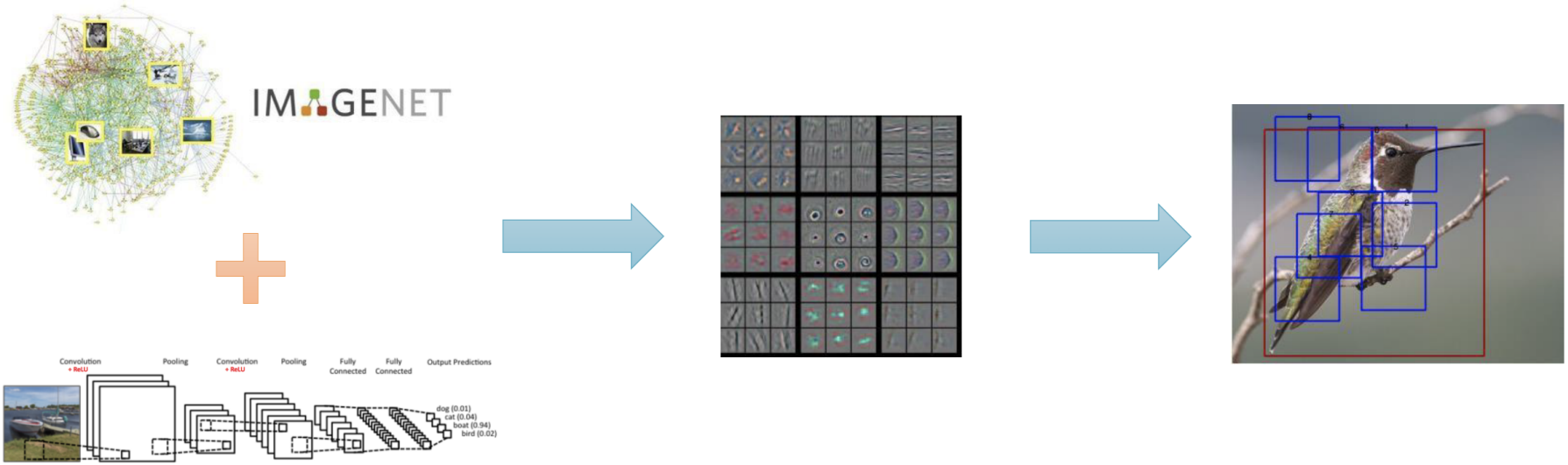
How can we frame transfer learning problems?

No single solution! Survey of various recent research papers

1. "Forward" transfer: train on one task, transfer to a new task
   a) Just try it and hope for the best
   b) Finetune on the new task
   c) Randomize source domain
2. Multi-task transfer: train on many tasks, transfer to a new task
   a) Generate highly randomized source domains
   b) Model-based reinforcement learning
   c) Model distillation
   d) Contextual policies
   e) Modular policy networks
3. Multi-task meta-learning: learn to learn from many tasks
   a) RNN-based meta-learning
   b) Gradient-based meta-learning

Finetuning

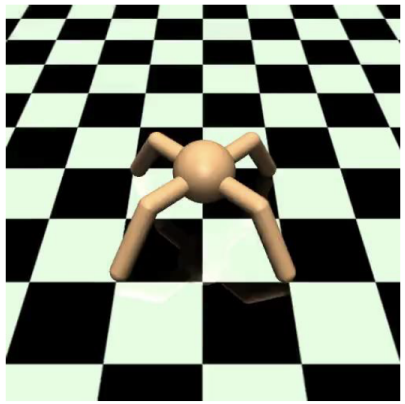
The most popular transfer learning method in (supervised) deep learning!



Where are the “ImageNet” features of RL?

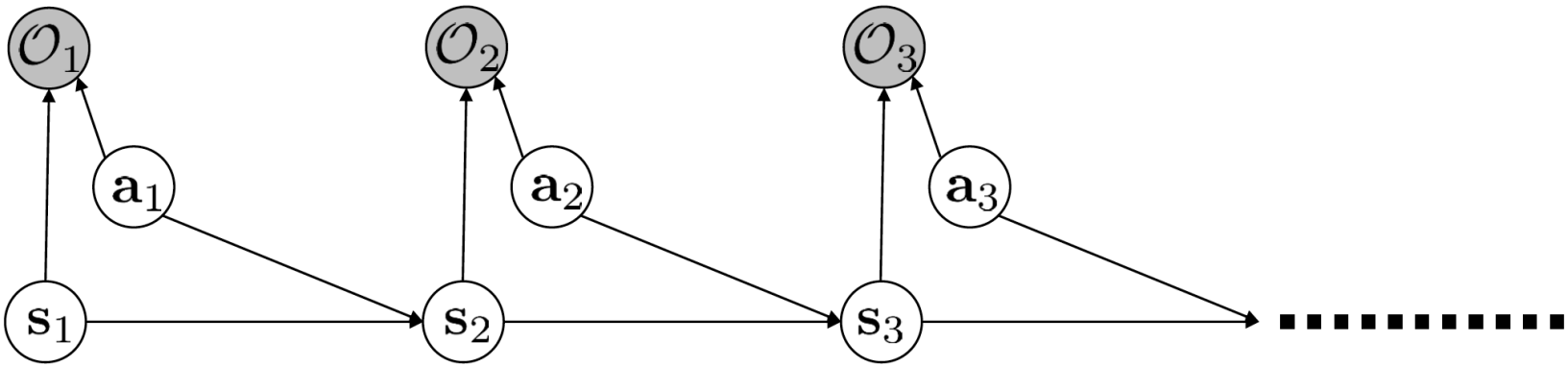
Challenges with finetuning in RL

1. RL tasks are generally much less diverse
   - Features are less general
   - Policies & value functions become overly specialized
2. Optimal policies in fully observed MDPs are deterministic
   - Loss of exploration at convergence
   - Low-entropy policies adapt very slowly to new settings



Finetuning with maximum-entropy policies

How can we increase diversity and entropy?

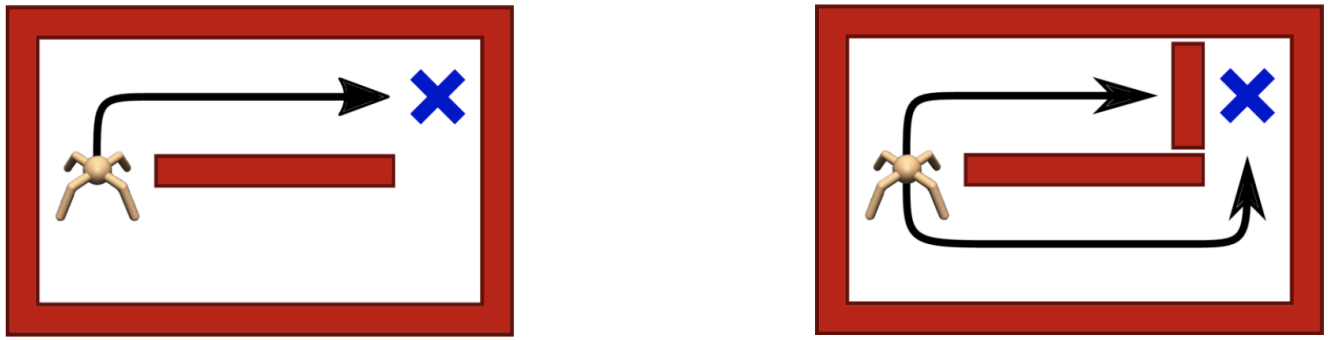


$$\pi(\mathbf{a}|\mathbf{s}) = \exp(Q_\phi(\mathbf{s},\mathbf{a}) - V(\mathbf{s})) \text{ optimizes } \sum_t E_{\pi(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)] + E_{\pi(\mathbf{s}_t)}[\mathcal{H}(\pi(\mathbf{a}_t|\mathbf{s}_t))]$$

policy entropy

Act **as randomly as possible** while collecting high rewards!

Example: pre-training for robustness



Learning to solve a task **in all possible ways** provides for more robust transfer!

Finetuning in RL: suggested readings

Finetuning via MaxEnt RL: Haarnoja*, Tang*, et al. (2017). **Reinforcement Learning with Deep Energy-Based Policies.**

Finetuning from transferred visual features (via VAE): Higgins et al. **DARLA: improving zero-shot transfer in reinforcement learning.** 2017.

Pretraining with hierarchical RL methods:

Andreas et al. **Modular multitask reinforcement learning with policy sketches.** 2017.

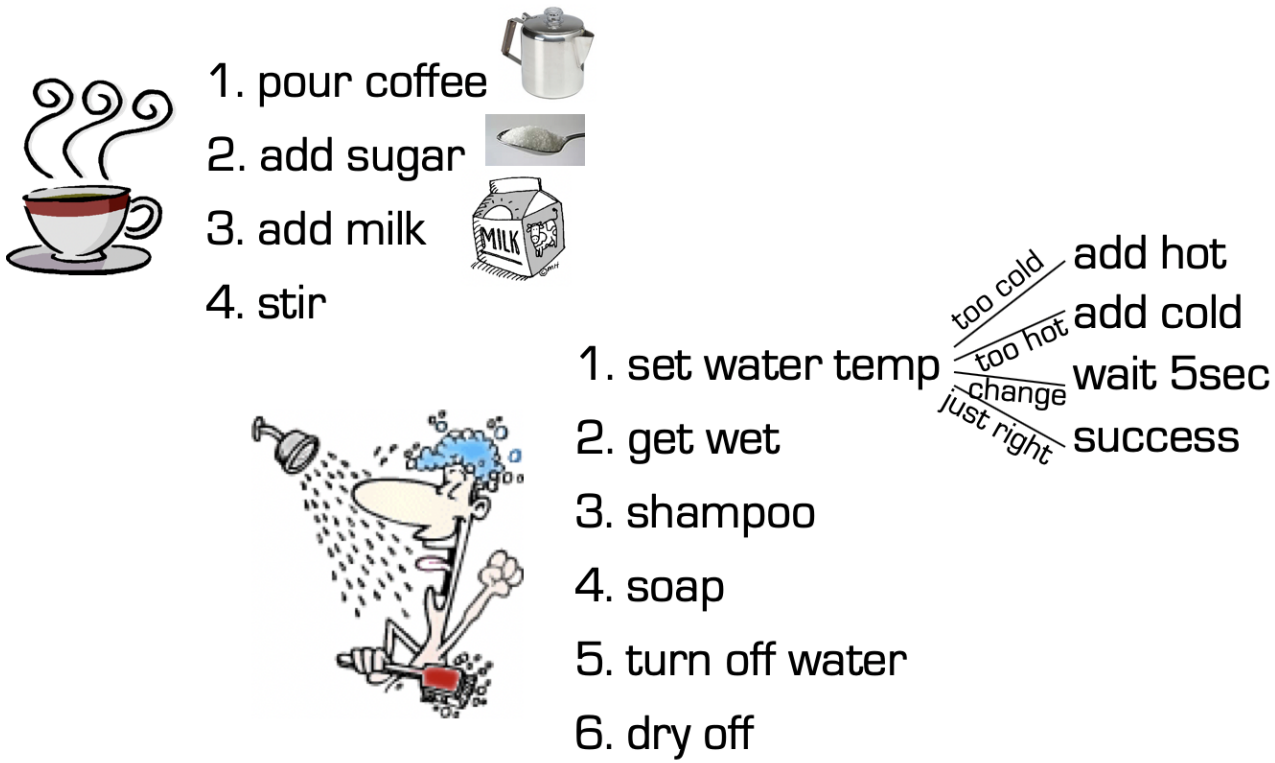
Florensa et al. **Stochastic neural networks for hierarchical reinforcement learning.** 2017.

...and many many others!

Table of Contents

Real-world behavior is hierarchical

- Humans plan modularly at different granularities of understanding
- Going out of one room is similar to going out of another room



Real-world behavior is hierarchical

- In the field of cognitive science, research has long suggested that human and animal behavior is based on a hierarchical structure

- Botvinick, Niv and Barto. **Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective**. *Cognition*, 2008.
- Badre, Kayser, and D'Esposito. **Frontal cortex and the discovery of abstract action rules**, *Neuron*, 2010.

Hierarchical RL (HRL)

- RL problems suffer from serious scaling issues. HRL is a computational approach intended to address these issues by learning to operate on different levels of temporal abstraction

- **Long-term credit assignment**: faster learning and better generalization
- **Structured exploration**: explore with sub-policies rather than primitive actions
- **Transfer learning**: different levels of hierarchy can encompass different knowledge and allow for better transfer

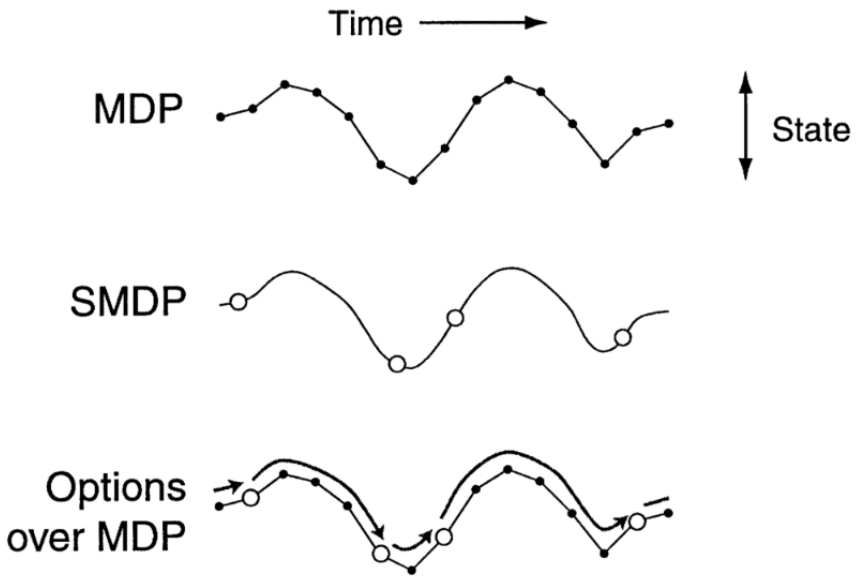
The "Option" framework

Bottom level is a sub-policy

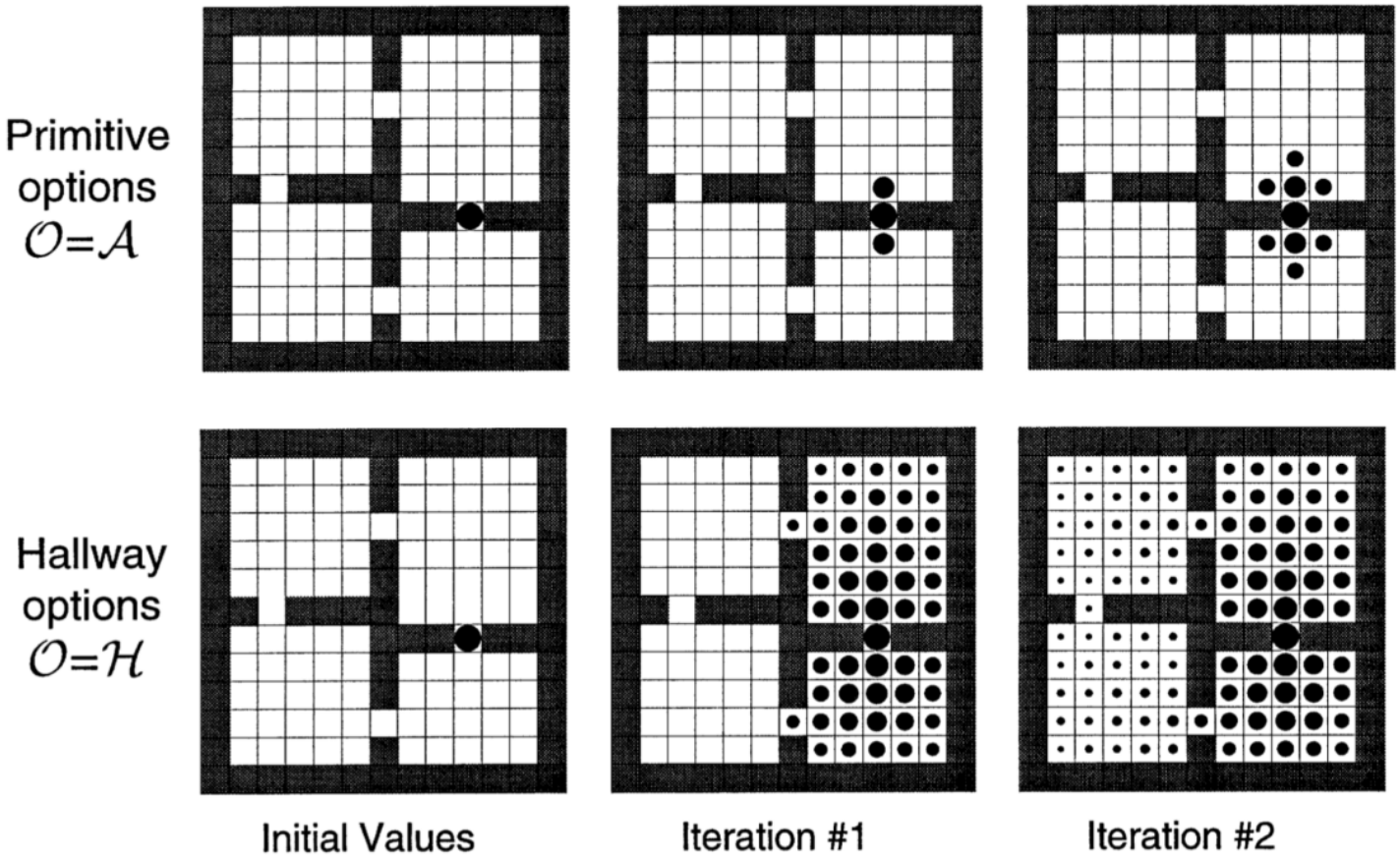
- takes environment observations
- outputs actions
- runs until termination

Top level is a policy-over-options:

- takes environment observations
- outputs sub-policies
- runs until termination



Option framework



HRL frameworks

	Temporal abstraction	State abstraction	Sub-tasks *fixed policy provided by the programmer*	Sub-tasks *non-deterministic finite-state controller*	Sub-tasks *termination predicate and a local reward function*	Algorithm with proven convergence	Optimal policy	Domain knowledge requirements
Feudal	✓	✓			✓		hierarchical	++++
Options	✓		✓			✓	hierarchical	+++
HAM	✓			✓		✓	hierarchical	++
MAXQ	✓	✓			✓	✓	recursive	+

Hierarchical RL: suggested readings

- M. Ghavamzadeh and S. Mahadevan, **Hierarchical Average Reward Reinforcement Learning**, JMLR 2007.
- TG Dietterich, **Hierarchical reinforcement learning with the MAXQ value function decomposition**, JAIR 2010.
- T. D. Kulkarni, et al., **Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation**, NIPS 2016.
- A. S. Vezhnevets, et al., **FeUdal networks for hierarchical reinforcement learning**, ICML 2017.
- O. Nachum, et al., **Data-Efficient Hierarchical Reinforcement Learning**, NIPS 2018.
- O. Nachum, et al., **Why does hierarchy (sometimes) work so well in reinforcement learning?** 2019.

Review of syllabus

Lecture Slides

Nov. 5th, 2019	Lecture 1: Introduction to RL
Nov. 12th, 2019	Lecture 2: Tabular RL Algorithms
Nov. 19th, 2019	Lecture 3: Introduction to Deep Reinforcement Learning
Nov. 26th, 2019	Lecture 4: Deep Reinforcement Learning - Policy Gradients
Dec. 3rd, 2019	Lecture 5: Deep Reinforcement Learning - Actor-Critic Algorithms
Dec. 10th, 2019	Lecture 6: Deep Reinforcement Learning - Value Function Methods - Part I
Dec. 17th, 2019	Lecture 7: Deep Reinforcement Learning - Value Function Methods - Part II
Dec. 24th, 2019	Lecture 8: Model-Based RL, Transfer RL, Hierarchical RL

THE END