

# Lecture 10: Model-Based Reinforcement Learning

Zhi Wang & Chunlin Chen

Department of Control and Systems Engineering  
Nanjing University

Dec. 3rd, 2020

# Table of Contents

## 1 Optimal Control and Planning

- Make decisions using known dynamics
  - Stochastic optimization, cross-entropy method
  - Monte Carlo tree search
  - Linear-quadratic regulator

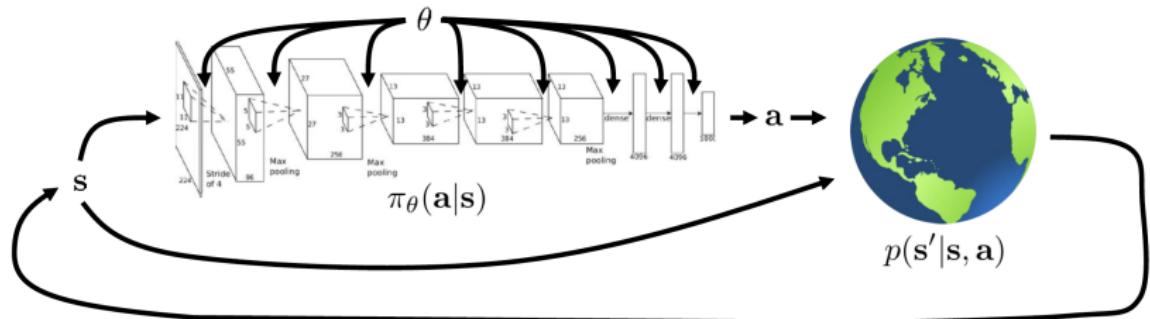
## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

# Recall: The RL objective



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

$$\theta^\star = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right]$$

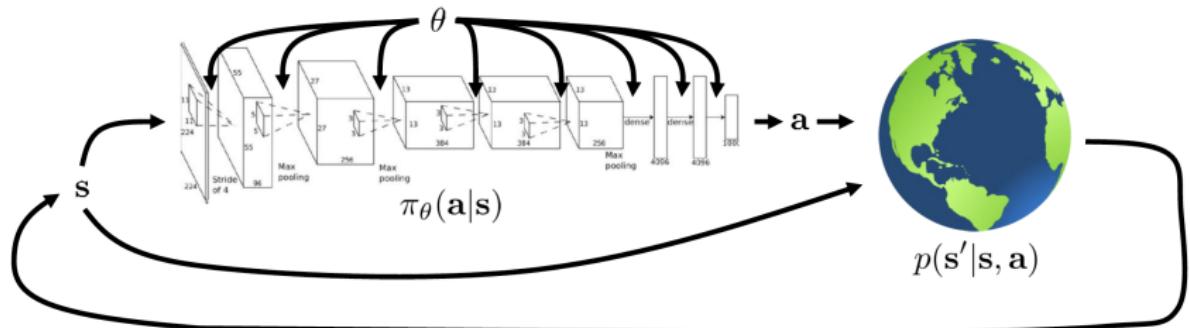
## Dynamics of the MDP $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$

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

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

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

# Recall: Model-free RL



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

assume this is unknown  
don't even attempt to learn it

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right]$$

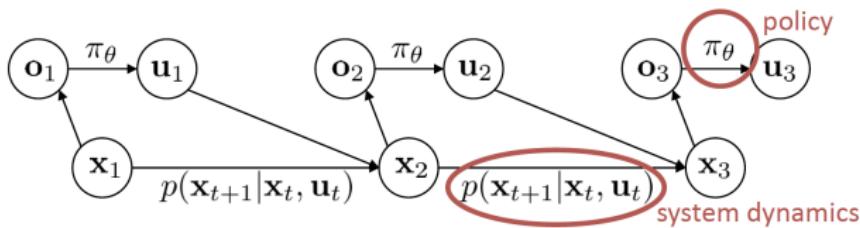
# What if we knew the transition dynamics?

- Often we do know the dynamics
  - Games (e.g., Atari games, chess, Go)
  - Easily modeled systems (e.g., navigating a car)
  - simulated environments (e.g., simulated robots, video games)
- Often we learn the dynamics
  - System identification - fit unknown parameters of a known model
  - 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

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

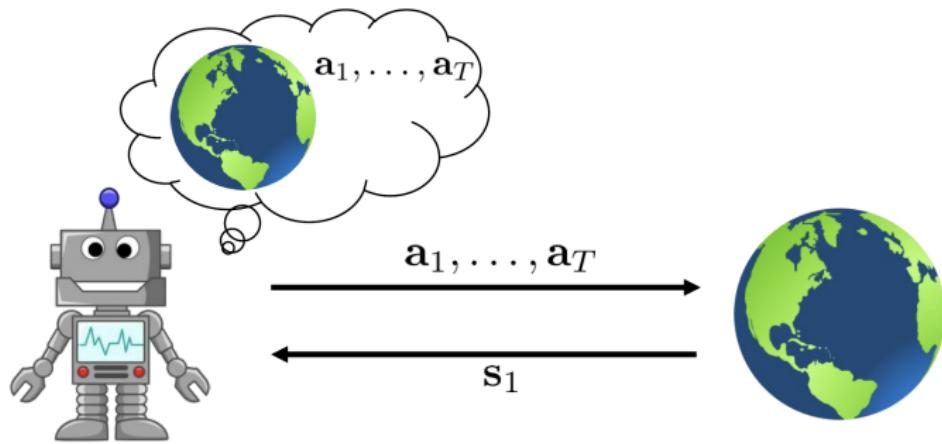
## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

# The deterministic case



$$a_1, \dots, a_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} \sum_{t=1}^T r(s_t, \mathbf{a}_t) \text{ s.t. } \mathbf{a}_{t+1} = f(s_t, \mathbf{a}_t)$$

# Method 1: Stochastic optimization

abstract away optimal control/planning:

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} J(\mathbf{a}_1, \dots, \mathbf{a}_T)$$

$$\mathbf{A} = \arg \max_{\mathbf{A}} J(\mathbf{A})$$

don't care what this is

simplest method: guess & check      “random shooting method”

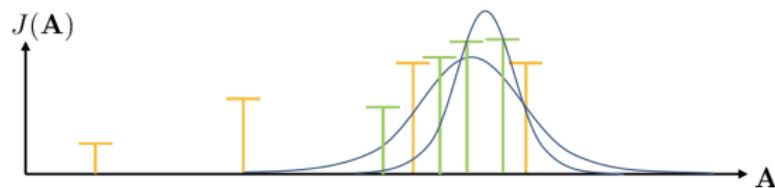
1. pick  $\mathbf{A}_1, \dots, \mathbf{A}_N$  from some distribution (e.g., uniform)
2. choose  $\mathbf{A}_i$  based on  $\arg \max_i J(\mathbf{A}_i)$

## Method 2: Cross-entropy method (CEM)

1. pick  $\mathbf{A}_1, \dots, \mathbf{A}_N$  from some distribution (e.g., uniform)

can we do better?

2. choose  $\mathbf{A}_i$  based on  $\arg \max_i J(\mathbf{A}_i)$



cross-entropy method with continuous-valued inputs:

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

typically use  
Gaussian  
distribution

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

# 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

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

# Discrete case: Monte Carlo tree search (MCTS)

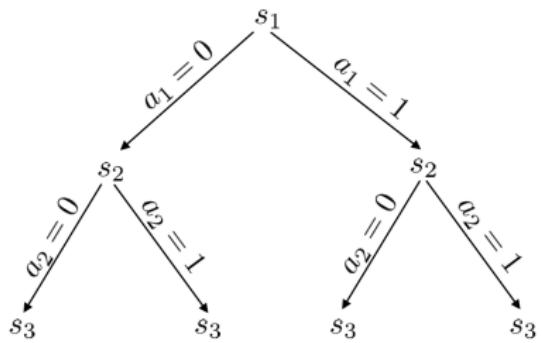


$s_t$



$a_t$

discrete planning as a search problem



# Discrete case: Monte Carlo tree search (MCTS)

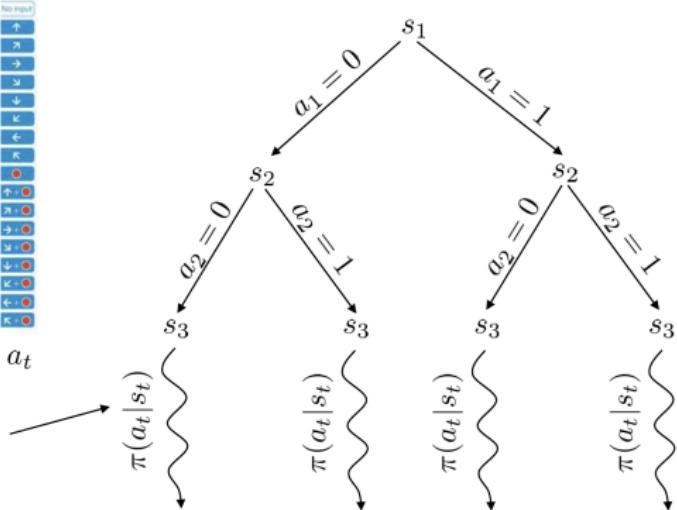
how to approximate value without full tree?



$s_t$



e.g., random policy



# Discrete case: Monte Carlo tree search (MCTS)

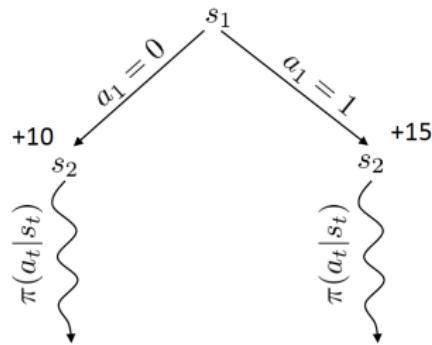


$s_t$



$a_t$

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

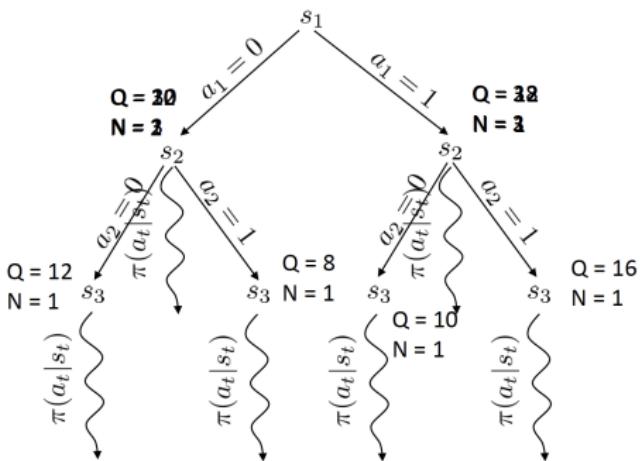
1. find a leaf  $s_l$  using TreePolicy( $s_1$ )
2. evaluate the leaf using DefaultPolicy( $s_l$ )
3. update all values in tree between  $s_1$  and  $s_l$

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

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

# Can we use derivatives?

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

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

usual story: differentiate via backpropagation and optimize!

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

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

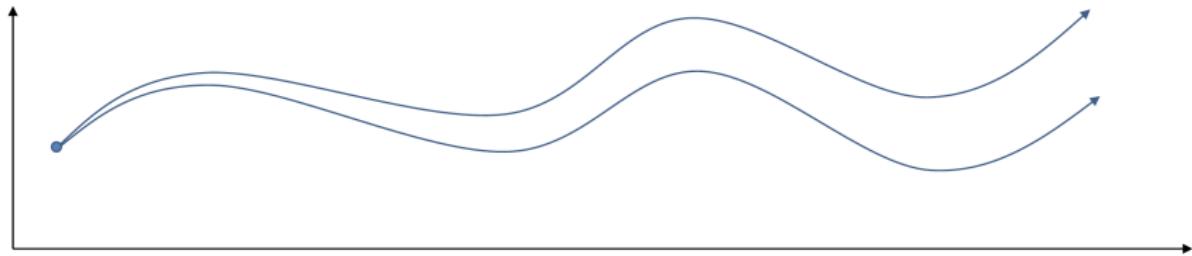
in practice, it really helps to use a 2<sup>nd</sup> order method!



# Shooting methods vs. collocation

shooting method: optimize over actions only

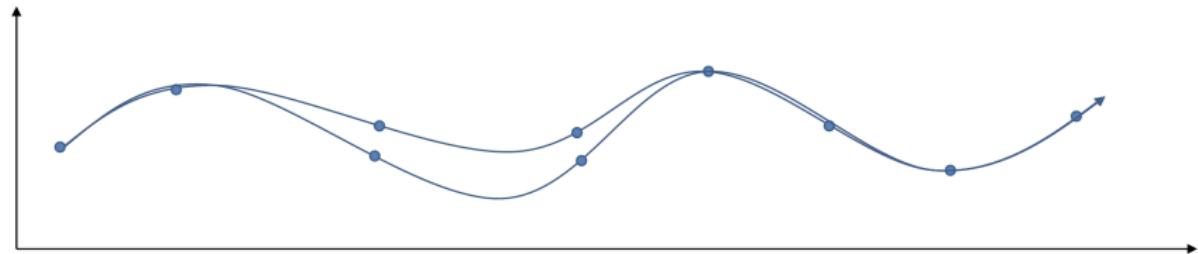
$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots) \dots), \mathbf{u}_T)$$



# Shooting methods vs. collocation

collocation method: optimize over actions and states, with constraints

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



# 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

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

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

---

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

# The LQR algorithm

Backward recursion

for  $t = T$  to 1:

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

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

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

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

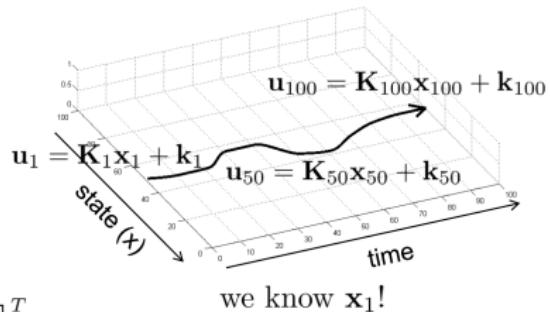
$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

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

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

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

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



Forward recursion

for  $t = 1$  to  $T$ :

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

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

# The LQR algorithm

Backward recursion

for  $t = T$  to 1:

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

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

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

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

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

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

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

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

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

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t \quad \xleftarrow{\text{total cost from now until end from state } \mathbf{x}_t} V(\mathbf{x}_t) = \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t)$$

# Table of Contents

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

- 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

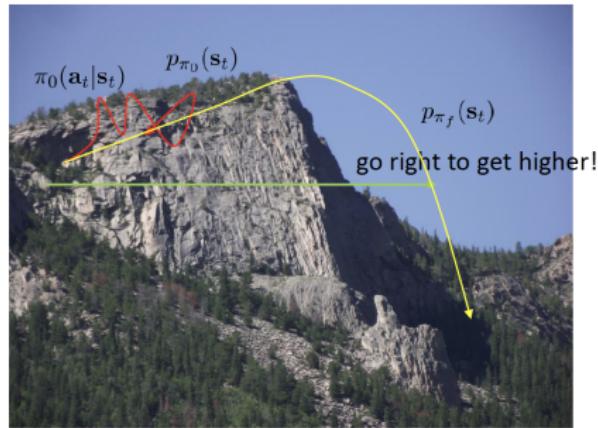
# Does it work? Yes!

- 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!

## 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

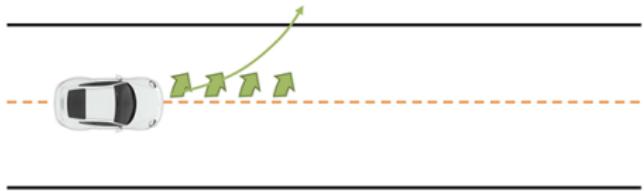
can we make  $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$ ?

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  $\mathcal{D}$



# 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  $\mathcal{D}$

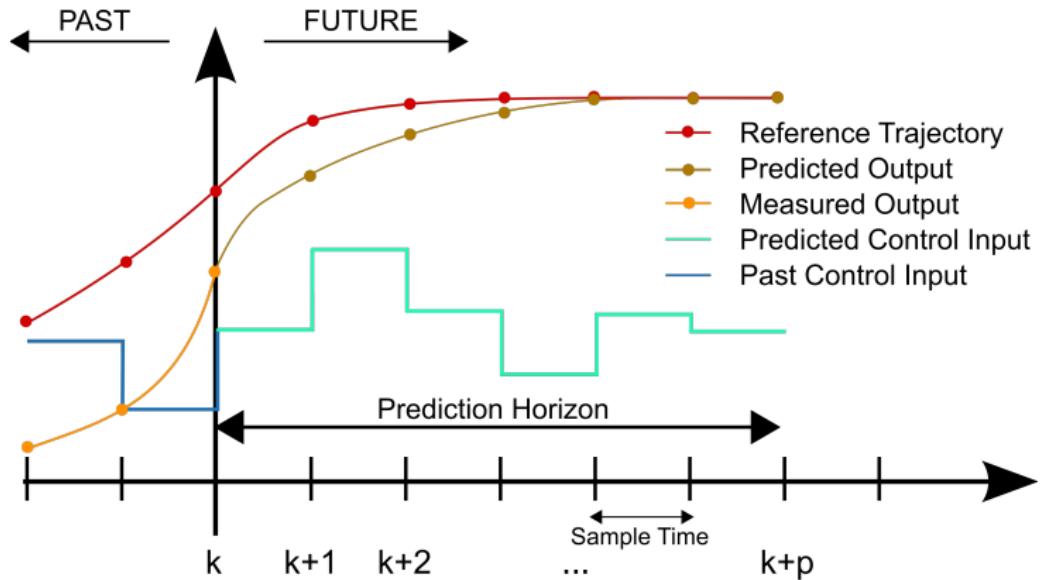


- The more you replan, the less perfect each individual plan needs to be
- Can use shorter horizons
- Even random sampling can often work well here!

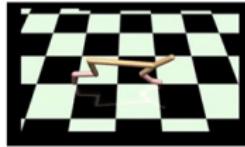
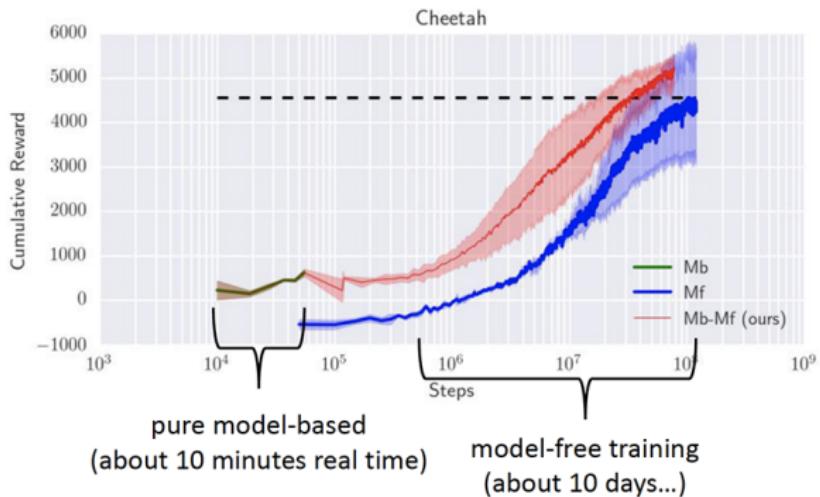
## Alternative: Model predictive control (MPC)

- 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)

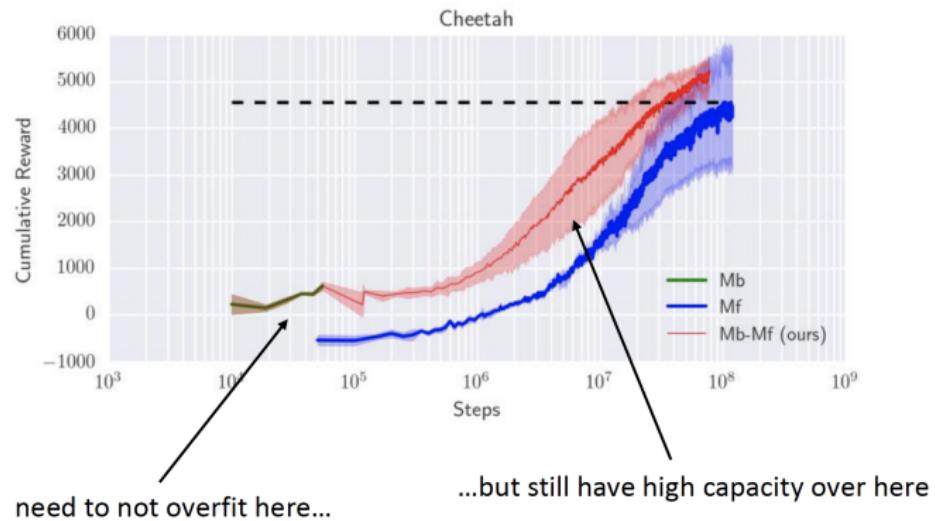


# A performance gap in model-based RL



- 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)*.

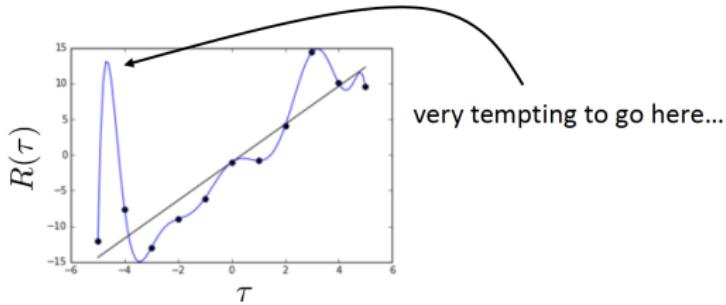
# Why the performance gap?



# 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  $\mathcal{D}$



# Table of Contents

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

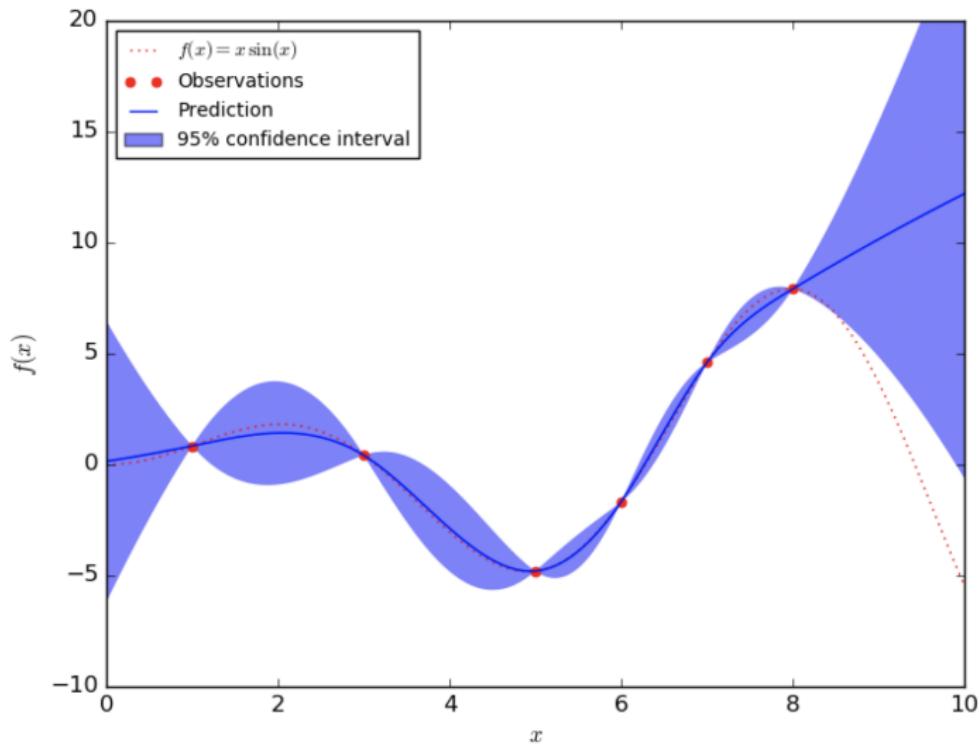
## 2 Model-Based RL

- Learn the “model”
- **Uncertainty Estimation**
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

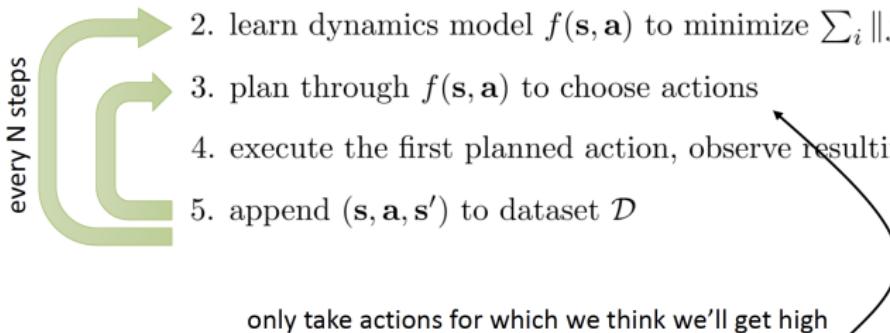
# Uncertainty - Bayesian models



# 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  $\mathcal{D}$



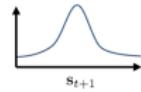
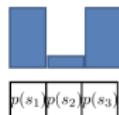
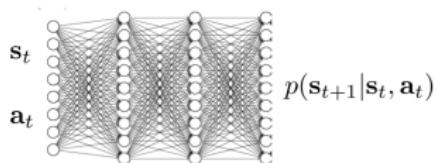
only take actions for which we think we'll get high reward in expectation (w.r.t. uncertain dynamics)

This avoids “exploiting” the model

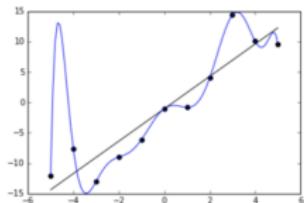
The model will then adapt and get better

# How can we have uncertainty-aware models?

Idea 1: use output entropy



why is this not enough?

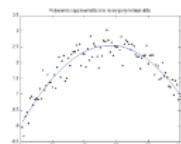


Two types of uncertainty:

*aleatoric or statistical uncertainty* →

← *epistemic or model uncertainty*

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

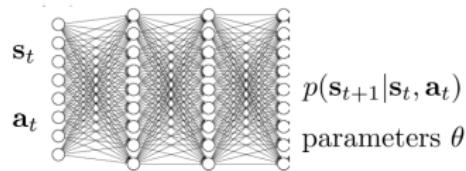


what is the variance here?

# 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"*



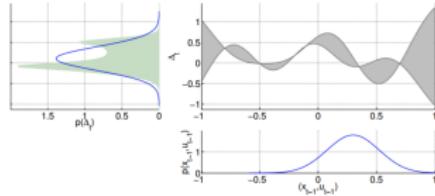
usually, we estimate

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

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

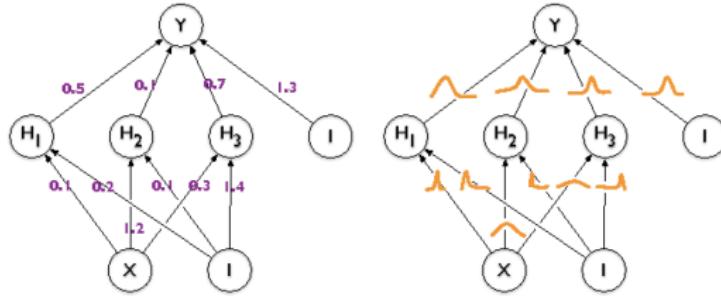
predict according to:

$$\int p(s_{t+1}|s_t, a_t, \theta) p(\theta|\mathcal{D}) d\theta$$



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

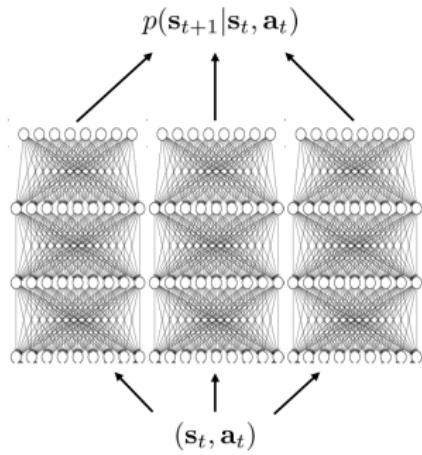
uncertainty  
about the weight

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!

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

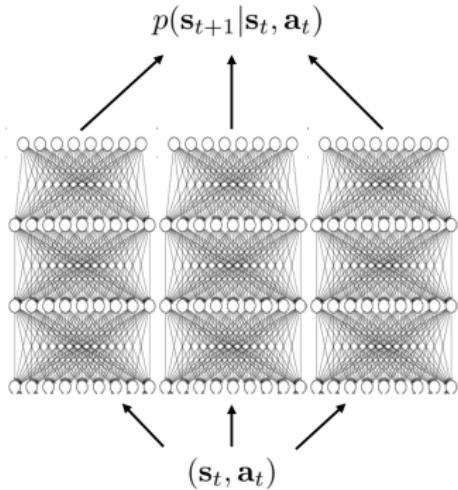
$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{N} \sum_i p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta_i)$$

How to train?

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

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

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

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

distribution over  
deterministic models

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

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

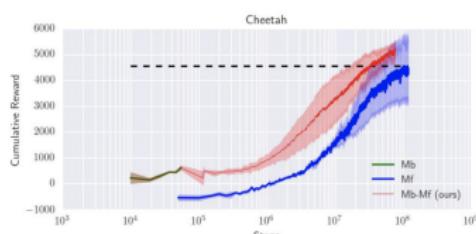
Step 3: calculate  $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

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

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

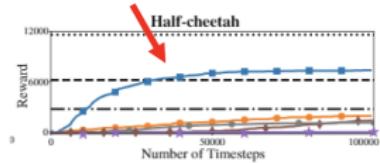
# Example: model-based RL with ensembles

## 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

## Further readings

- Nagabandi, et al., Deep dynamics models for learning dexterous manipulation, 2019.
- Nagabandi, et al., Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning
- Chua, et al., Deep reinforcement learning in a handful of trials using probabilistic dynamics models
- Feinberg, et al., Model-based value expansion for efficient model-free reinforcement learning
- Buckman, et al., Sample-efficient reinforcement learning with stochastic ensemble value expansion

# Table of Contents

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

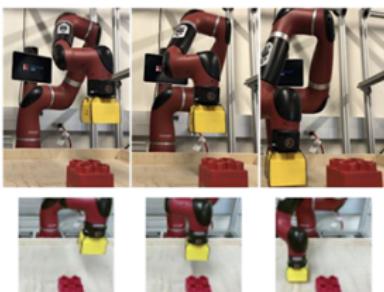
## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

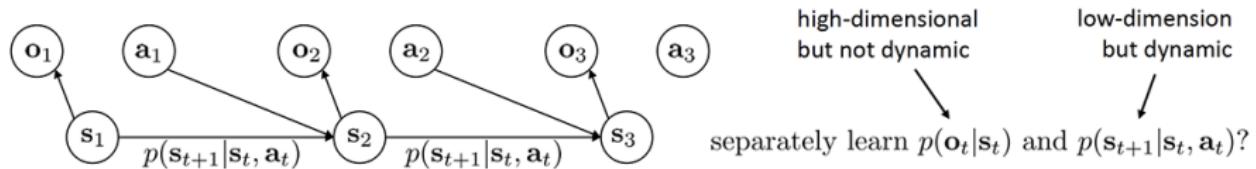
# What about complex observations?



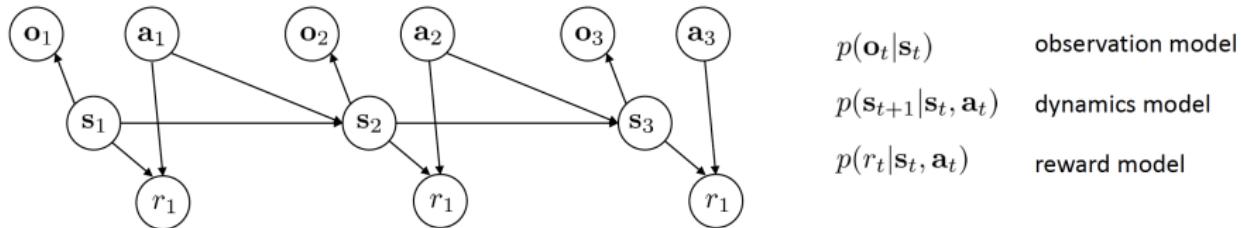
$$f(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$$

What is hard about this?

- High dimensionality
- Redundancy
- Partial observability



# State space (latent space) models



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

↑  
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

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

↑  
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

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

↑  
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*

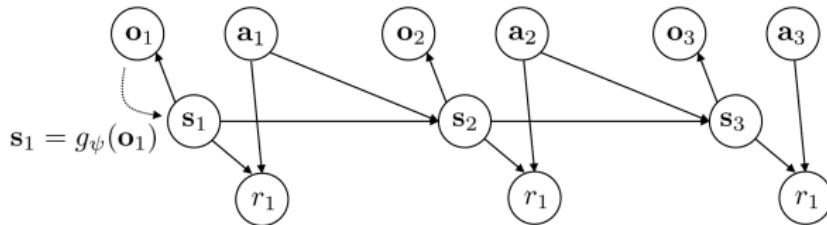
stochastic case requires variational inference

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

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

**Everything is differentiable, can train with backprop**

# Model-based RL with latent space models

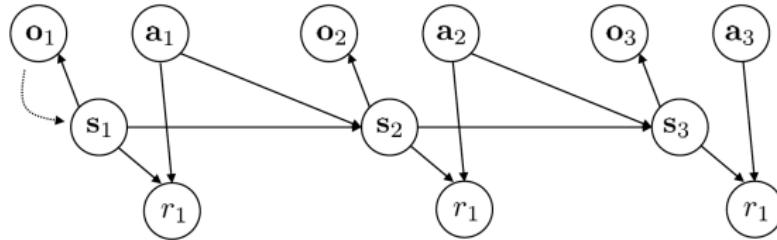


$$\max_{\phi, \psi} \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \log p_\phi(g_\psi(\mathbf{o}_{t+1,i}) | g_\psi(\mathbf{o}_{t,i}), \mathbf{a}_{t,i}) + \log p_\phi(\mathbf{o}_{t,i} | g_\psi(\mathbf{o}_{t,i})) + \log p_\phi(r_{t,i} | g_\psi(\mathbf{o}_{t,i}))$$

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  $p_\phi(s_{t+1}|s_t, a_t)$ ,  $p_\phi(r_t|s_t)$ ,  $p(\mathbf{o}_t|s_t)$ ,  $g_\psi(\mathbf{o}_t)$
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

## 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

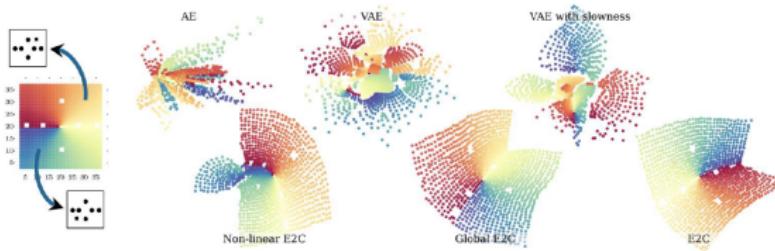
{watterm, springj, jboedeck}@cs.uni-freiburg.de

Martin Riedmiller

Google DeepMind

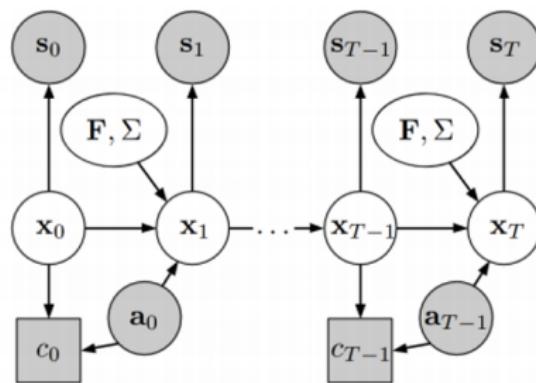
London, UK

riedmiller@google.com



# Example: Deep structured latent representations

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



# Table of Contents

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- Use simpler policies than neural networks

# Backpropagate into the policy with a model

- Policy gradient:

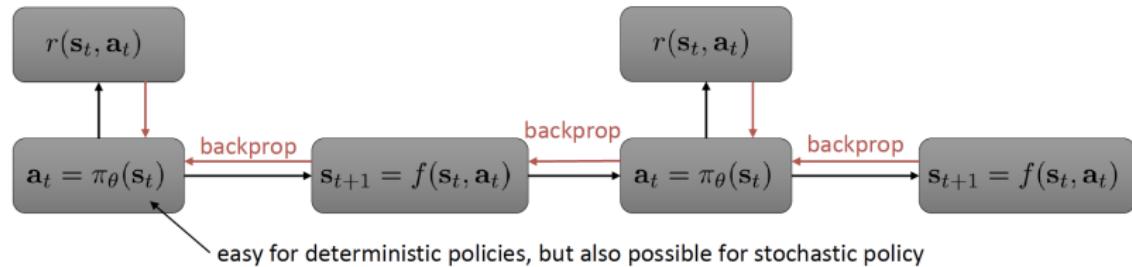
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t}) \hat{Q}_{i,t}^{\pi}$$

- Backpropagate with a model:

$$\nabla_{\theta} J(\theta) \approx \mathbb{E} \left[ \sum_{t=1}^T \frac{dr_t}{ds_t} \prod_{t'=2}^t \frac{ds_{t'}}{da_{t'-1}} \frac{da_{t'-1}}{ds_{t'-1}} \right]$$

- Policy gradient might be more stable
  - if enough samples are used
  - because it does not require multiplying many Jacobians

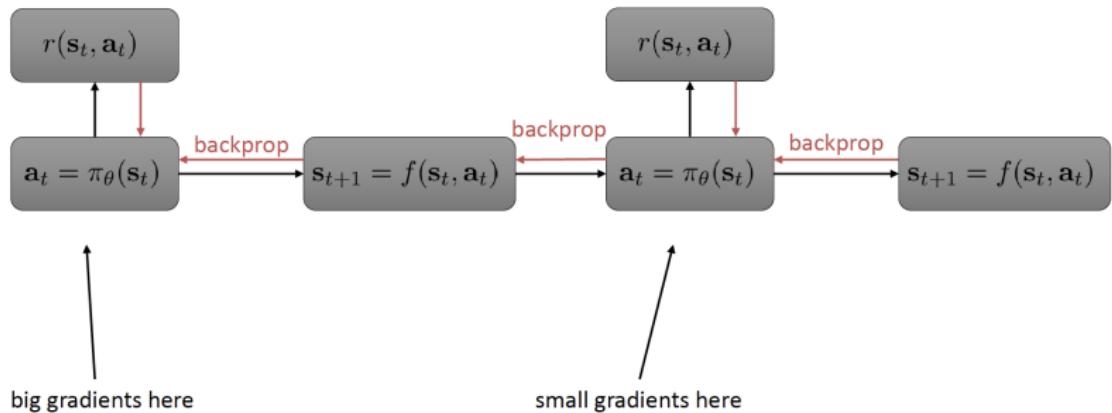
# Backpropagate into the policy with a model



model-based reinforcement learning version 2.0:

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. backpropagate through  $f(\mathbf{s}, \mathbf{a})$  into the policy to optimize  $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
4. run  $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ , appending the visited tuples  $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$  to  $\mathcal{D}$

# What is the problem?



- Parameter sensitivity problem: policy parameters couple all time steps
- Vanishing and exploding gradient

# What is the solution?

- Use model-free RL algorithms, with the model used to generate synthetic samples
  - Seems weirdly backwards
  - Actually works very well
  - Essentially “model-based acceleration” for model-free RL
- Use simpler policies than neural networks
  - LQR with learned models (LQR-FLM - **Fitted Local Models**)
  - Train **local** policies to solve simple tasks
  - Combine them into **global** policies via supervised learning

# Table of Contents

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- **Model-based acceleration for model-free learning**
- Use simpler policies than neural networks

# Model-free optimization with a model

- **Dyna:** online Q-learning that performs model-free RL with a model

1. given state  $s$ , pick action  $a$  using exploration policy
2. observe  $s'$  and  $r$ , to get transition  $(s, a, s', r)$
3. update model  $\hat{p}(s'|s, a)$  and  $\hat{r}(s, a)$  using  $(s, a, s')$
4. Q-update:  $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s', r}[r + \max_{a'} Q(s', a') - Q(s, a)]$
5. repeat  $K$  times:
  6. sample  $(s, a) \sim \mathcal{B}$  from buffer of past states and actions
  7. Q-update:  $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s', r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

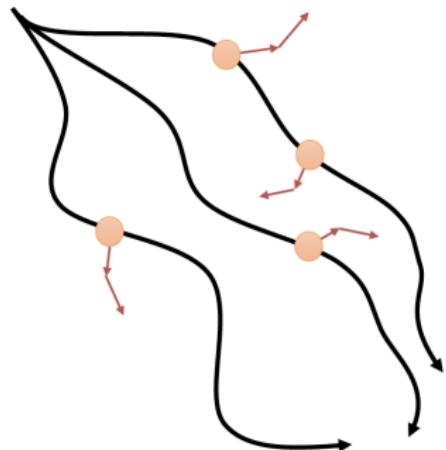
Richard S. Sutton, “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming,” *ICML*, 1990.

# General “Dyna-style” model-based RL recipe

1. collect some data, consisting of transitions  $(s, a, s', r)$
2. learn model  $\hat{p}(s'|s, a)$  (and optionally,  $\hat{r}(s, a)$ )
3. repeat K times:
  4. sample  $s \sim \mathcal{B}$  from buffer
  5. choose action  $a$  (from  $\mathcal{B}$ , from  $\pi$ , or random)
  6. simulate  $s' \sim \hat{p}(s'|s, a)$  (and  $r = \hat{r}(s, a)$ )
  7. train on  $(s, a, s', r)$  with model-free RL
  8. (optional) take  $N$  more model-based steps

+ only requires short (as few as one step) rollouts from model

+ still sees diverse states



# General “Dyna-style” model-based RL recipe

1. take some action  $\mathbf{a}_i$  and observe  $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ , add it to  $\mathcal{B}$
2. sample mini-batch  $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$  from  $\mathcal{B}$  uniformly
3. use  $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j\}$  to update model  $\hat{p}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$
4. sample  $\{\mathbf{s}_j\}$  from  $\mathcal{B}$
5. for each  $\mathbf{s}_j$ , perform model-based rollout with  $\mathbf{a} = \pi(\mathbf{s})$
6. use all transitions  $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$  along rollout to update Q-function

- Model-based acceleration (MBA)
  - Gu et al., “Continuous deep Q-learning with model-based acceleration,” 2016.
- Model-based value-expansion (MVE)
  - Feinberg et al., “Model-based value-expansion,” 2018.
- Model-based policy optimization (MBPO)
  - Janner et al., “When to trust your model: Model-based policy optimization,” 2019.

# Table of Contents

## 1 Optimal Control and Planning

- Make decisions using known dynamics
- Stochastic optimization, cross-entropy method
- Monte Carlo tree search
- Linear-quadratic regulator

## 2 Model-Based RL

- Learn the “model”
- Uncertainty Estimation
- Complex Observations

## 3 Model-based Policy Learning

- Backpropagate directly into the policy
- Model-based acceleration for model-free learning
- **Use simpler policies than neural networks**

# Local models

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

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

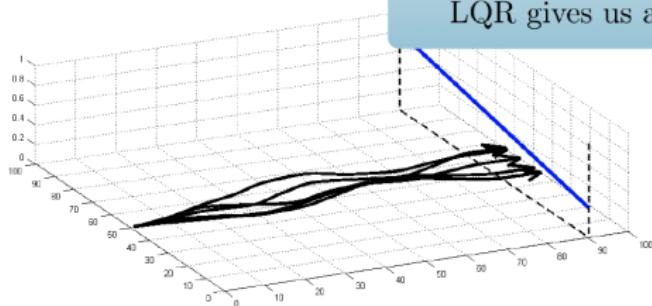
usual story: differentiate via backpropagation and optimize!

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

# Local models

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

idea: just fit  $\frac{df}{d\mathbf{x}_t}, \frac{df}{d\mathbf{u}_t}$  around current trajectory or policy!



LQR gives us a linear feedback controller

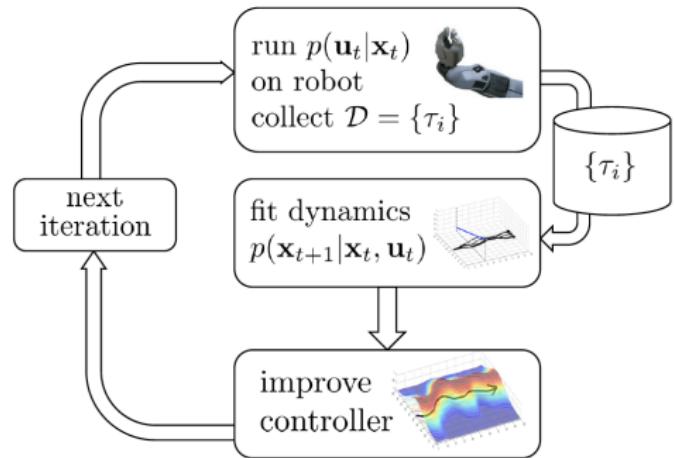
can **execute** in the real world!

# Local models

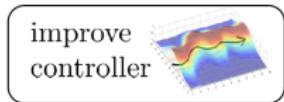
$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f(\mathbf{x}_t, \mathbf{u}_t), \Sigma)$$

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

$$\mathbf{A}_t = \frac{df}{d\mathbf{x}_t} \quad \mathbf{B}_t = \frac{df}{d\mathbf{u}_t}$$



# What controller to execute?



iLQR produces:  $\hat{\mathbf{x}}_t$ ,  $\hat{\mathbf{u}}_t$ ,  $\mathbf{K}_t$ ,  $\mathbf{k}_t$

$$\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t$$

Version 0.5:  $p(\mathbf{u}_t | \mathbf{x}_t) = \delta(\mathbf{u}_t = \hat{\mathbf{u}}_t)$

Doesn't correct deviations or drift

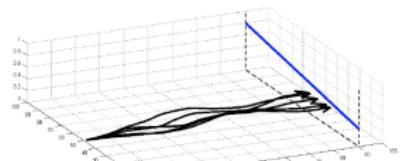
Version 1.0:  $p(\mathbf{u}_t | \mathbf{x}_t) = \delta(\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t)$

Better, but maybe a little too good?

Version 2.0:  $p(\mathbf{u}_t | \mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$

Add noise so that all samples don't look the same!

$$\text{Set } \Sigma_t = \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1}$$



# How to fit the dynamics?

fit dynamics  
 $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$

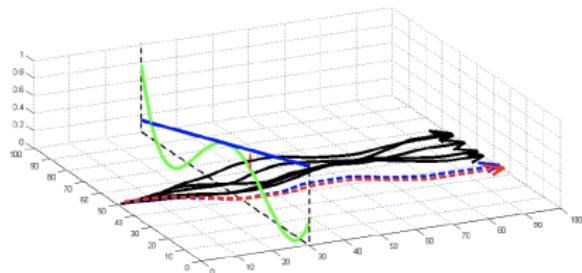


$$\{(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})_i\}$$

fit  $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$  at each time step using linear regression

$$p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(\mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t + \mathbf{c}, \mathbf{N}_t) \quad \mathbf{A}_t \approx \frac{df}{d\mathbf{x}_t} \quad \mathbf{B}_t \approx \frac{df}{d\mathbf{u}_t}$$

# How to stay close to the old controller?



improve  
 $p(\mathbf{u}_t | \mathbf{x}_t)$

$$p(\mathbf{u}_t | \mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t, \Sigma_t)$$

$$p(\tau) = p(\mathbf{x}_1) \prod_{t=1}^T p(\mathbf{u}_t | \mathbf{x}_t) p(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t)$$

What if the new  $p(\tau)$  is “close” to the old one  $\bar{p}(\tau)$ ?

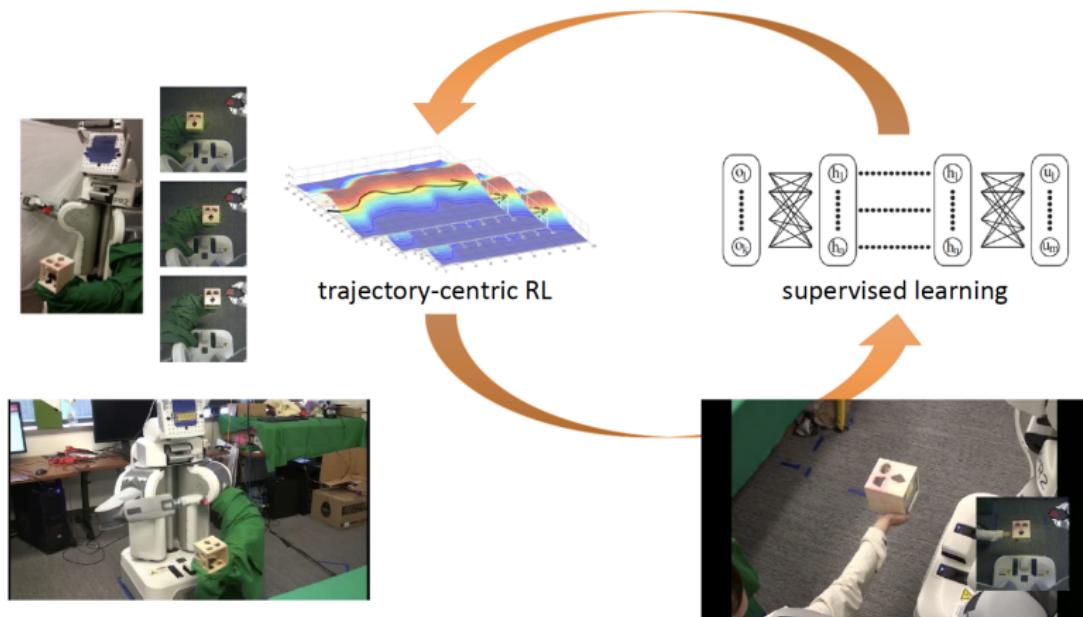
If trajectory distribution is close, then dynamics will be close too!

What does “close” mean?  $D_{\text{KL}}(p(\tau) \| \bar{p}(\tau)) \leq \epsilon$

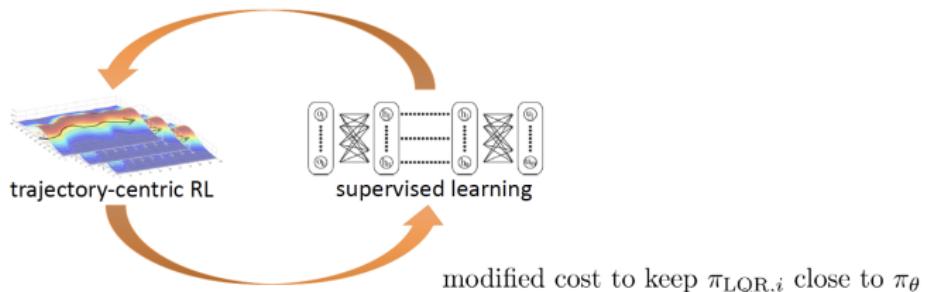
This is easy to do if  $\bar{p}(\tau)$  also came from linear controller!

# Global policies from local models

- Guided policy search: High-level idea



# Guided policy search: Algorithm sketch



1. optimize each local policy  $\pi_{LQR,i}(\mathbf{u}_t|\mathbf{x}_t)$  on initial state  $\mathbf{x}_{0,i}$  w.r.t.  $\tilde{c}_{k,i}(\mathbf{x}_t, \mathbf{u}_t)$
2. use samples from step (1) to train  $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$  to mimic each  $\pi_{LQR,i}(\mathbf{u}_t|\mathbf{x}_t)$
3. update cost function  $\tilde{c}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = c(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

Lagrange multiplier

- Levine et al., "End-to-end training of deep visuomotor policies," JMLR 2016.

# Underlying principle: Distillation

**Ensemble models:** single models are often not the most robust – instead train many models and average their predictions

this is how most ML competitions (e.g., Kaggle) are won

this is very expensive at test time

**Can we make a single model that is as good as an ensemble?**

**Distillation:** train on the ensemble's predictions as “soft” targets

$$p_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

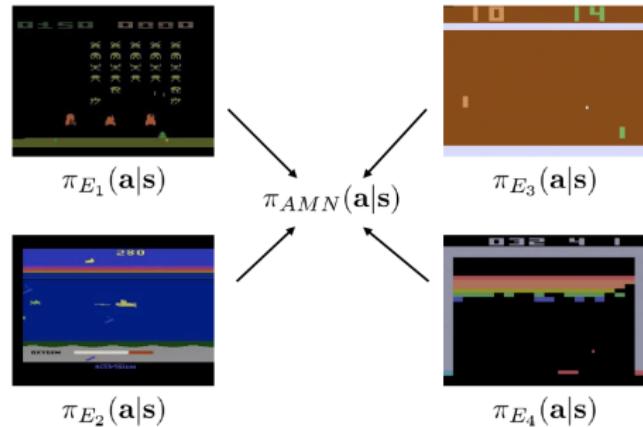
logit  $\xrightarrow{\exp(z_i/T)}$  temperature

**Intuition:** more knowledge in soft targets than hard labels!

- Hinton et al., “Distilling the knowledge in a neural network,” 2015.



# Distillation for multi-task transfer



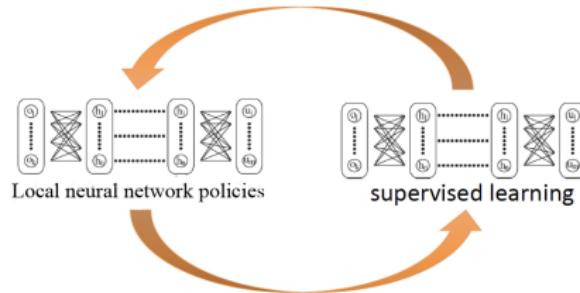
$$\mathcal{L} = \sum_a \pi_{E_i}(a|s) \log \pi_{AMN}(a|s)$$

(just supervised learning/distillation)

analogous to guided policy search, but  
for multi-task learning

- Parisotto et al., “Actor-mimic: Deep multitask and transfer reinforcement learning,” ICLR 2016.

# Combining weak policies into a strong policy



## Divide and Conquer Reinforcement Learning

Divide and conquer reinforcement learning algorithm sketch:

1. optimize each local policy  $\pi_{\theta_i}(\mathbf{a}_t|\mathbf{s}_t)$  on initial state  $\mathbf{s}_{0,i}$  w.r.t.  $\tilde{r}_{k,i}(\mathbf{s}_t, \mathbf{a}_t)$
2. use samples from step (1) to train  $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$  to mimic each  $\pi_{\theta_i}(\mathbf{u}_t|\mathbf{x}_t)$
3. update reward function  $\tilde{r}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

- Ghosh et al., "Divide and conquer reinforcement learning," ICLR 2018.

## Further readings: Guided policy search & Distillation

- Levine et al., “End-to-end training of deep visuomotor policies,” JMLR 2016.
- Rusu et al., “Policy distillation,” 2015.
- Parisotto et al., “Actor-mimic: Deep multitask and transfer reinforcement learning,” ICLR 2016.
- Ghosh et al., “Divide and conquer reinforcement learning,” ICLR 2018.
- Teh et al., “Distral: Robust multitask reinforcement learning,” 2017.

# 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
    - Uncertainty estimation, latent models...
  - Know how to backpropagate directly into the policy with a model
    - Model-based acceleration for model-free learning
    - Using local policies, combining them into a global one

# Model-based RL suggested readings

- Lecture 10 & 11 & 12 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>
  - <http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-12.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.

# THE END