

# UC Berkeley·CS285 | Deep Reinforcement Learning (2020)

## CS285 (2020)· 课程资料包 @ShowMeAI



视频

中英双语字幕



课件

一键打包下载



笔记

官方笔记翻译



代码

作业项目解析



视频·B 站 [ 扫码或点击链接 ]

<https://www.bilibili.com/video/BV12341167kL>



课件 & 代码·博客 [ 扫码或点击链接 ]

<http://blog.showmeai.tech/cs285>

Berkeley

actor-critic

reinforcement learning

exploration

元学习

exploitation

梯度策略

探索与利用

DQN

迁移学习

逆强化学习

Q-learning / Q 学习

deep q network

meta learning

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列，覆盖最具知名度的 **TOP20+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets· 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2: 扫码点击**底部菜单栏**

称为 **AI 内容创作者**? 回复 [ 添砖加瓦 ]

# Model-Based Policy Learning

CS 285

Instructor: Sergey Levine  
UC Berkeley

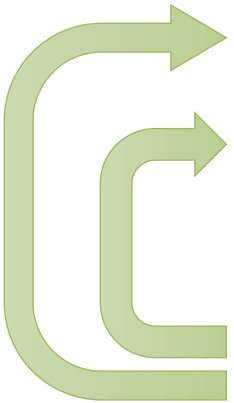


# Last time: model-based RL with MPC

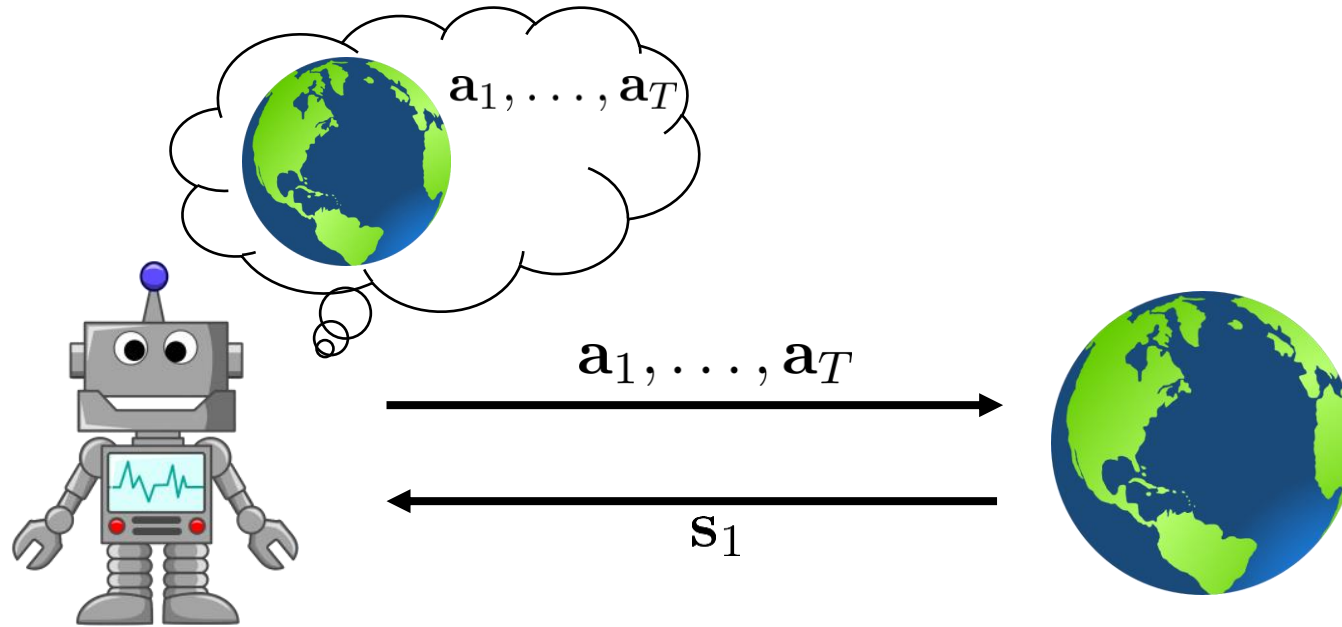
model-based reinforcement learning version 1.5:

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 stochastic open-loop case

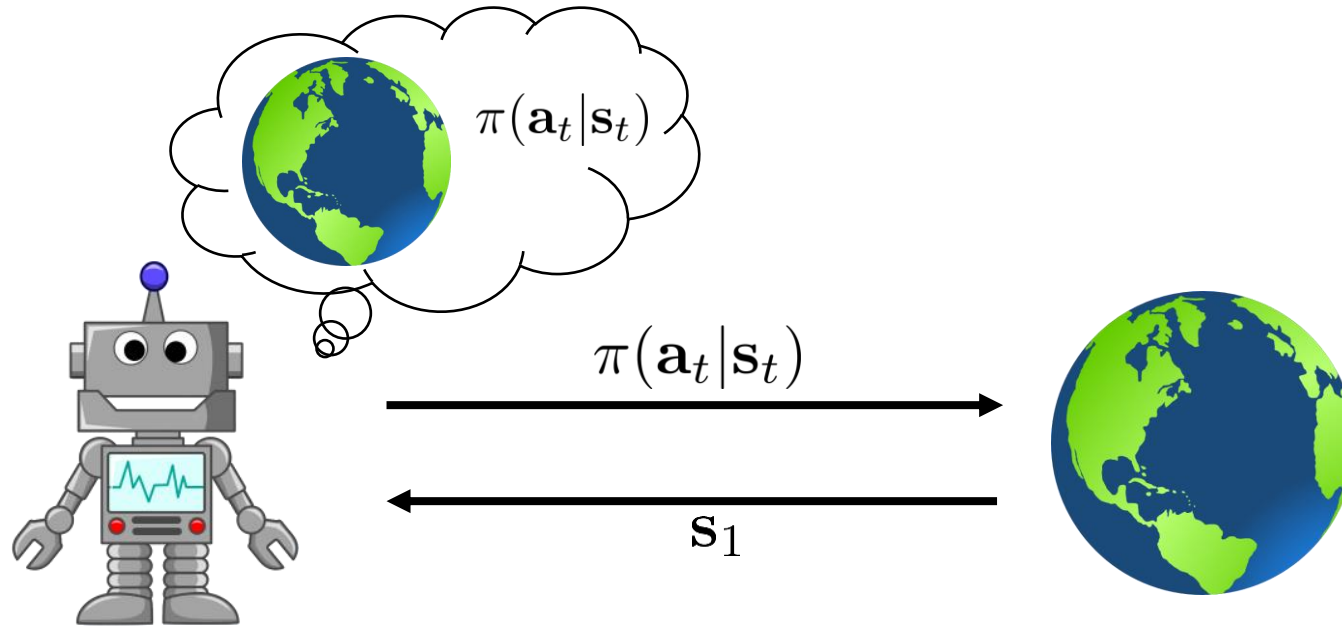


$$p_{\theta}(\mathbf{s}_1, \dots, \mathbf{s}_T | \mathbf{a}_1, \dots, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\mathbf{a}_1, \dots, \mathbf{a}_T = \arg \max_{\mathbf{a}_1, \dots, \mathbf{a}_T} E \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{a}_1, \dots, \mathbf{a}_T \right]$$

why is this suboptimal?

# The stochastic **closed**-loop case



$$p(\mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^T \pi(\mathbf{a}_t | \mathbf{s}_t) p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$$

$$\pi = \arg \max_{\pi} E_{\tau \sim p(\tau)} \left[ \sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

form of  $\pi$ ?

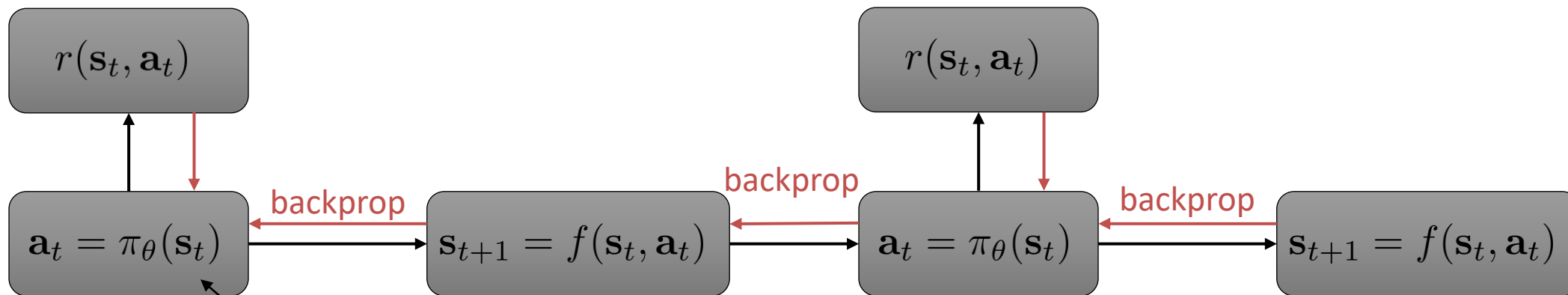
neural net

time-varying linear

$\mathbf{K}_t \mathbf{s}_t + \mathbf{k}_t$

global  
local

# Backpropagate directly into the policy?

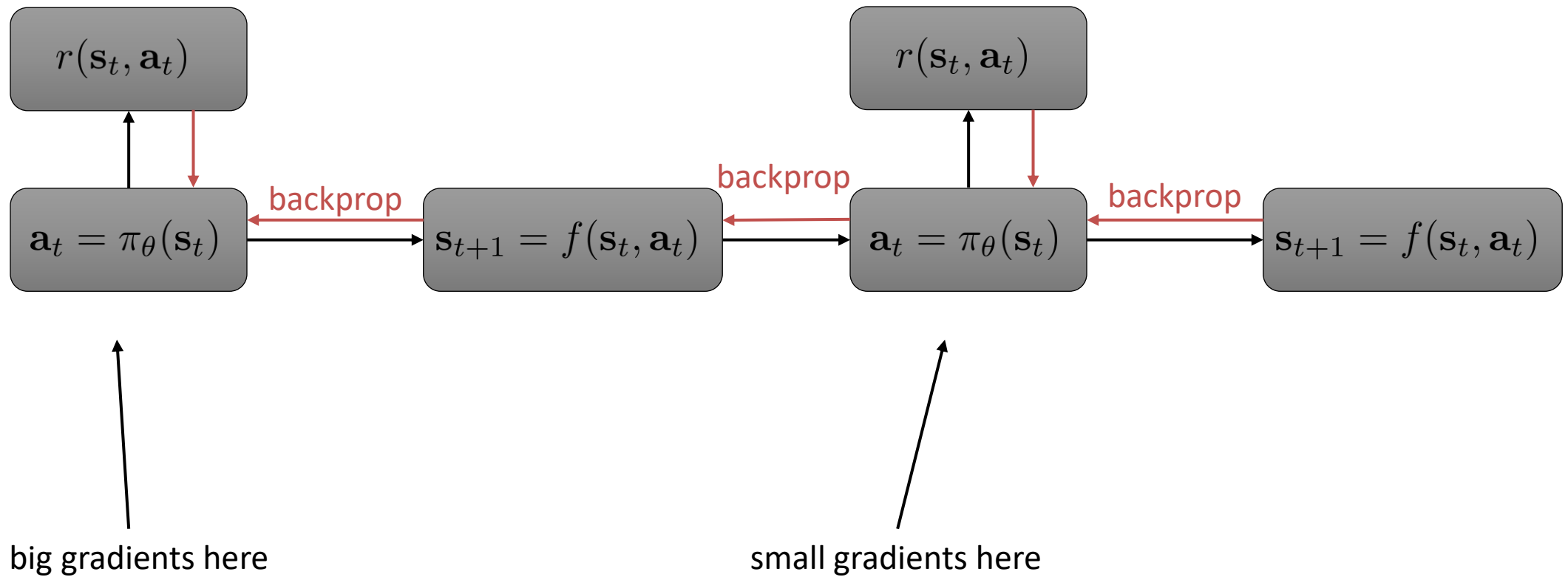


easy for deterministic policies, but also possible for stochastic policy

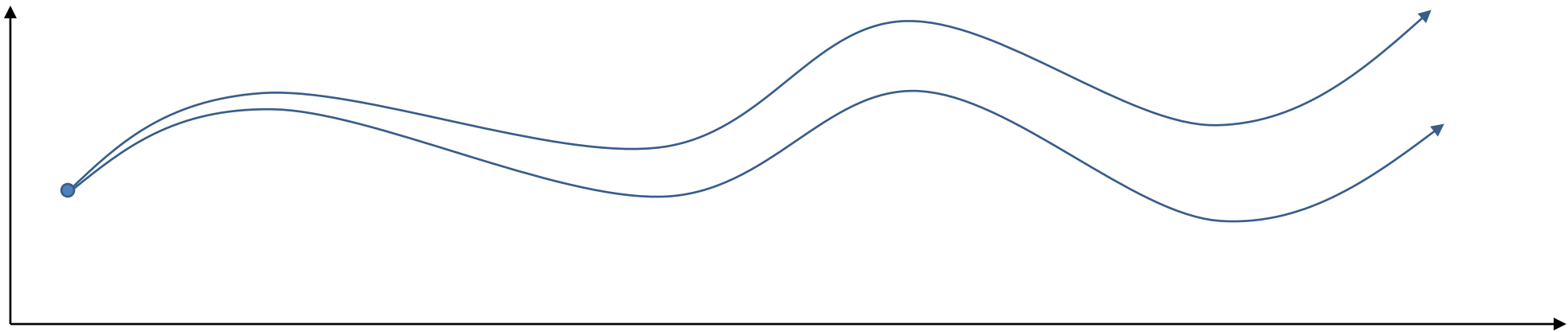
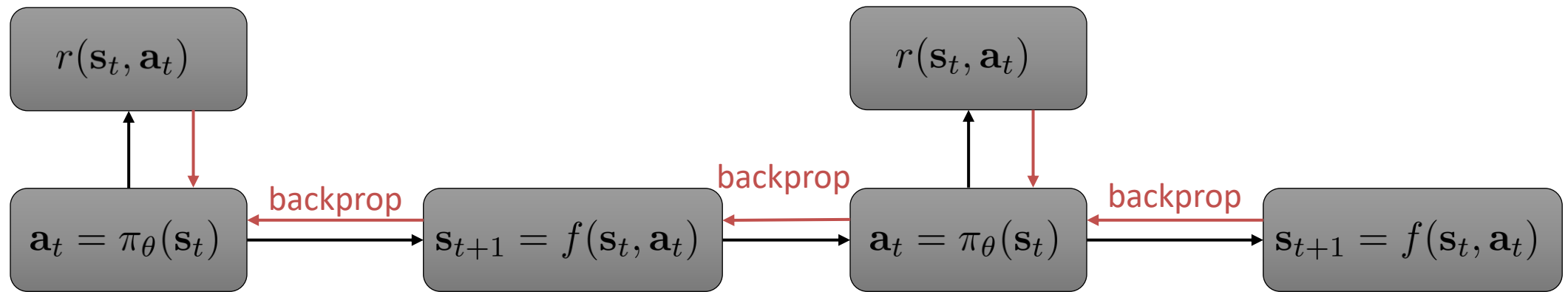
model-based reinforcement learning version 2.0:

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's the problem with backprop into policy?

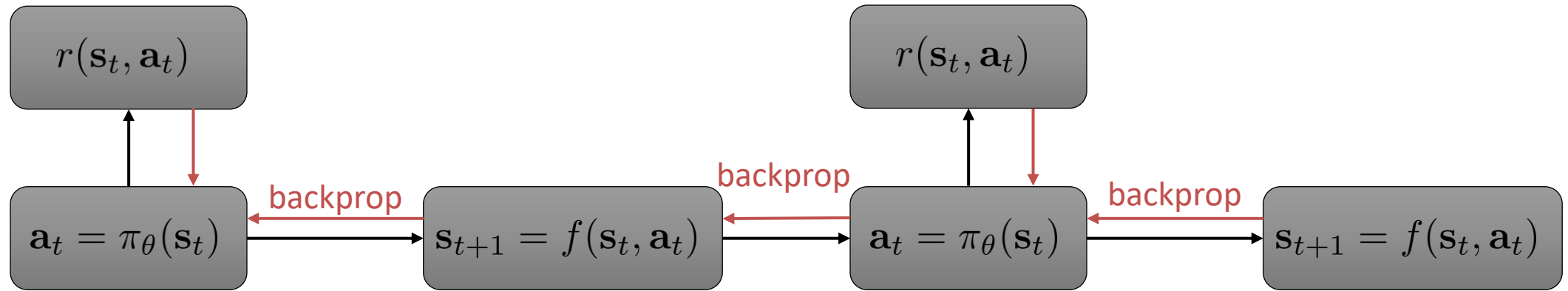


# What's the problem with backprop into policy?





# What's the problem with backprop into policy?



- Similar parameter sensitivity problems as shooting methods
  - But no longer have convenient second order LQR-like method, because policy parameters couple all the time steps, so no dynamic programming
- Similar problems to training long RNNs with BPTT
  - Vanishing and exploding gradients
  - Unlike LSTM, we can't just "choose" a simple dynamics, dynamics are chosen by nature

# What's the solution?

- Use derivative-free (“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 nets
  - LQR with learned models (LQR-FLM – **F**itted **L**ocal **M**odels)
  - Train **local** policies to solve simple tasks
  - Combine them into **global** policies via supervised learning

# Model-Free Learning With a Model

# What's the solution?

- Use derivative-free (“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 nets
  - LQR with learned models (LQR-FLM – Fitted Local Models)
  - Train **local** policies to solve simple tasks
  - Combine them into **global** policies via supervised learning

# Model-free optimization with a model

Policy gradient: 
$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) \hat{Q}_{i,t}^{\pi}$$

Backprop (pathwise) gradient: 
$$\nabla_{\theta} J(\theta) = \sum_{t=1}^T \frac{dr_t}{d\mathbf{s}_t} \prod_{t'=2}^t \frac{d\mathbf{s}_{t'}}{d\mathbf{a}_{t'-1}} \frac{d\mathbf{a}_{t'-1}}{d\mathbf{s}_{t'-1}}$$

- Policy gradient might be more *stable* (if enough samples are used) because it does not require multiplying many Jacobians
- See a recent analysis here:
  - Parmas et al. '18: PIPP: Flexible Model-Based Policy Search Robust to the Curse of Chaos

# Model-free optimization with a model

## Dyna

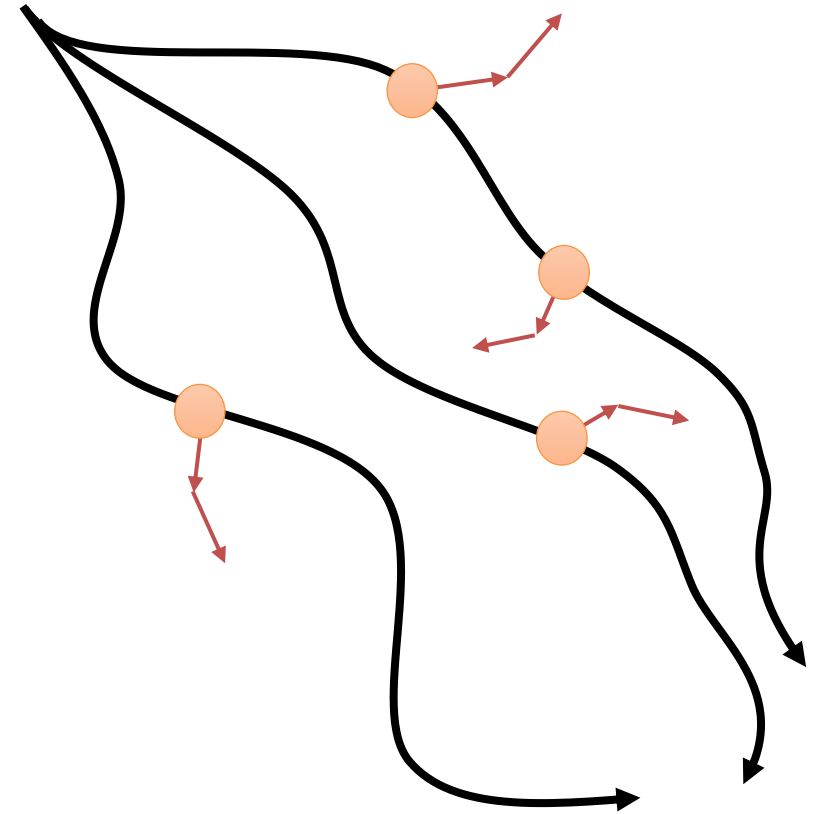
online Q-learning algorithm that performs model-free RL with a model

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:  $Q(s, a) \leftarrow Q(s, a) + \alpha E_{s', r}[r + \max_{a'} Q(s', a') - Q(s, a)]$

# 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



# Model-Based Acceleration (MBA)

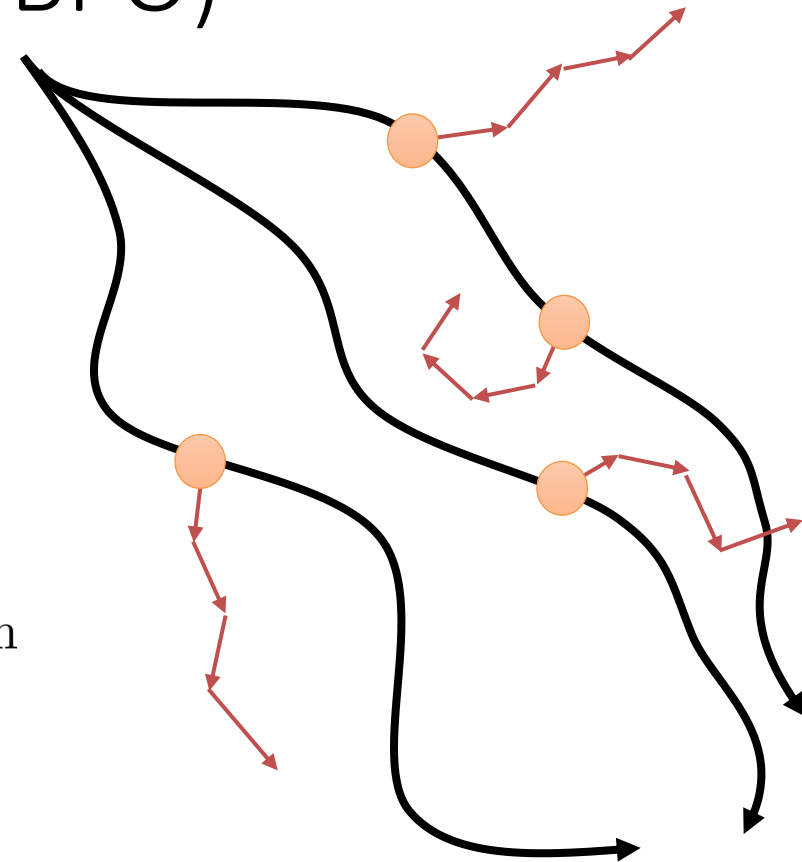
## Model-Based Value Expansion (MVE)

## Model-Based Policy Optimization (MBPO)

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

+ why is this a *good* idea?

- why is this a *bad* idea?



Gu et al. Continuous deep Q-learning with model-based acceleration. '16

Feinberg et al. Model-based value expansion. '18

Janner et al. When to trust your model: model-based policy optimization. '19



# Local Models

# What's the solution?

- Use derivative-free (“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 nets
  - LQR with learned models (LQR-FLM – **F**itted **L**ocal **M**odels)
  - Train **local** policies to solve simple tasks
  - Combine them into **global** policies via supervised learning

# What's the solution?

- Use derivative-free (“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 nets
  - LQR with learned models (LQR-FLM – **Fitted Local Models**)
  - Train **local** policies to solve simple tasks
  - Combine them into **global** policies via supervised learning

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

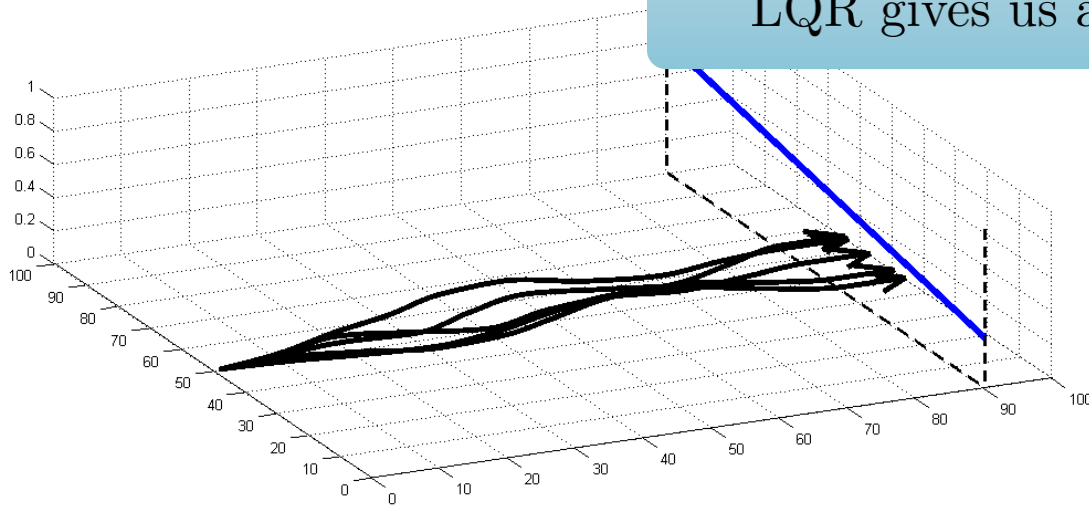
# Local models

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!

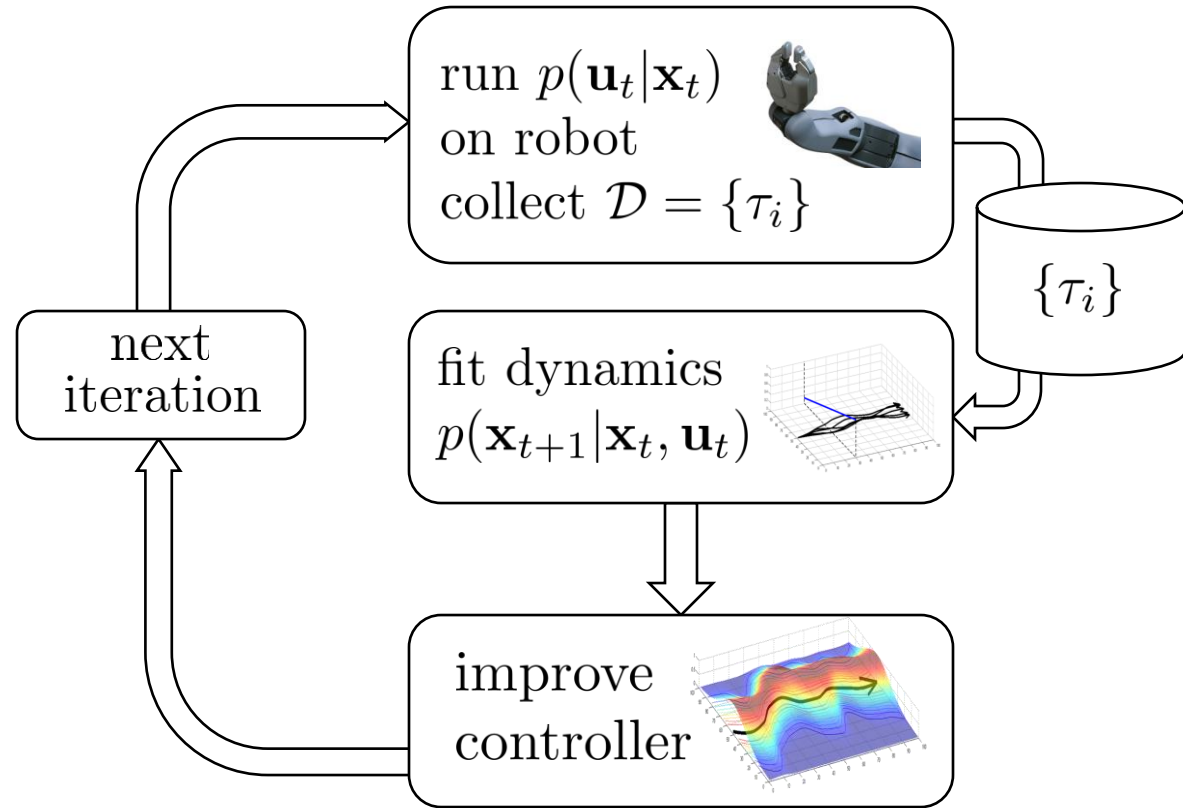


# Local models

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

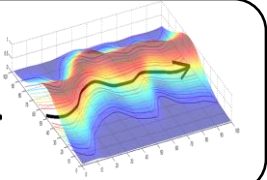
$$f(\mathbf{x}_t, \mathbf{u}_t) \approx \mathbf{A}_t \mathbf{x}_t + \mathbf{B}_t \mathbf{u}_t$$

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



# What controller to execute?

improve  
controller



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

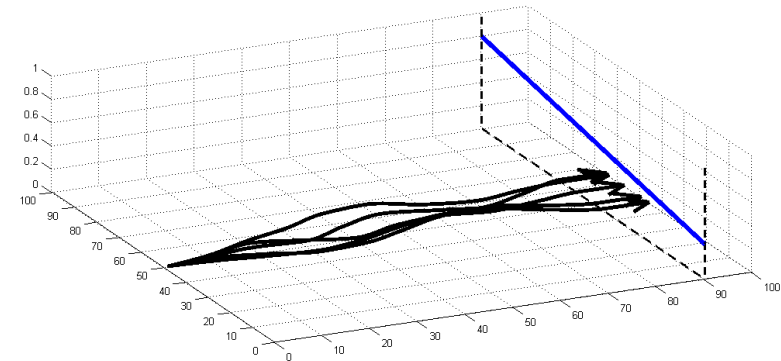
Version 1.0:  $p(\mathbf{u}_t|\mathbf{x}_t) = \delta(\mathbf{u}_t = \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t) + \mathbf{k}_t + \hat{\mathbf{u}}_t)$

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!

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

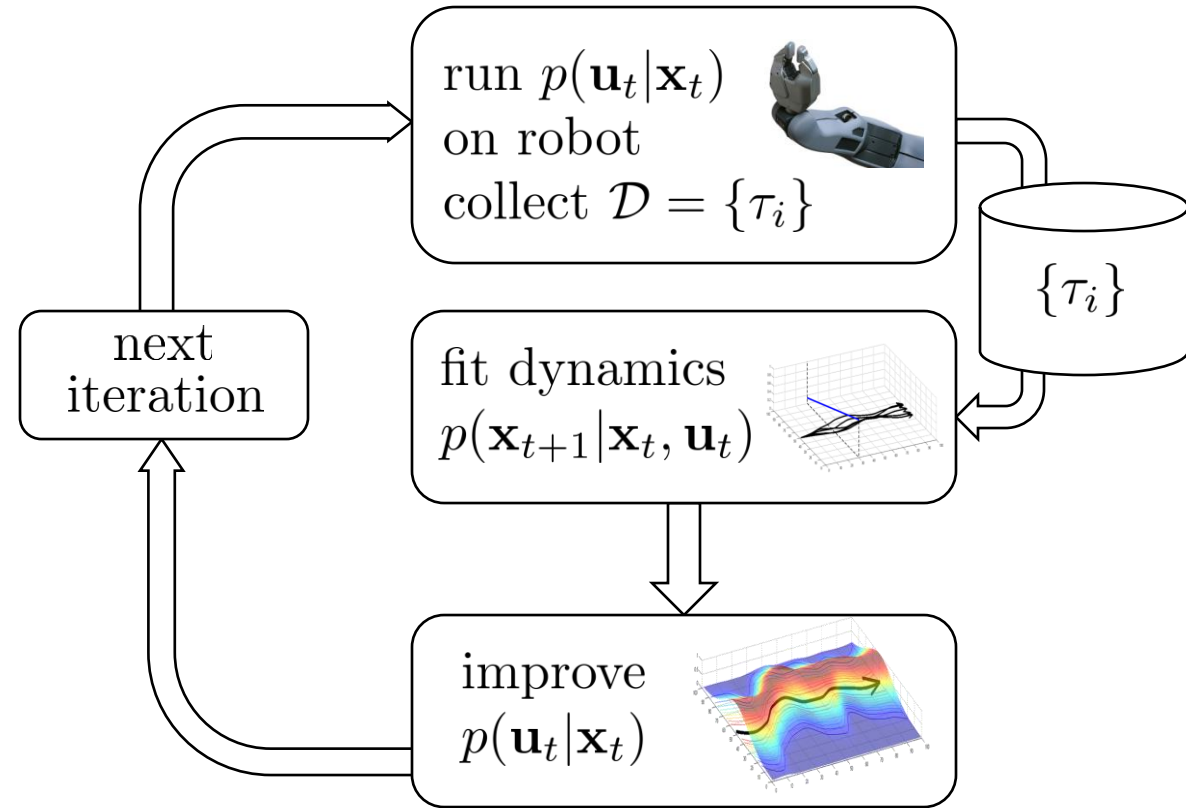


# Local models

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

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

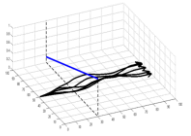
$$\mathbf{A}_t = \frac{df}{d\mathbf{x}_t} \quad \mathbf{B}_t = \frac{df}{d\mathbf{u}_t}$$





# How to fit the dynamics?

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

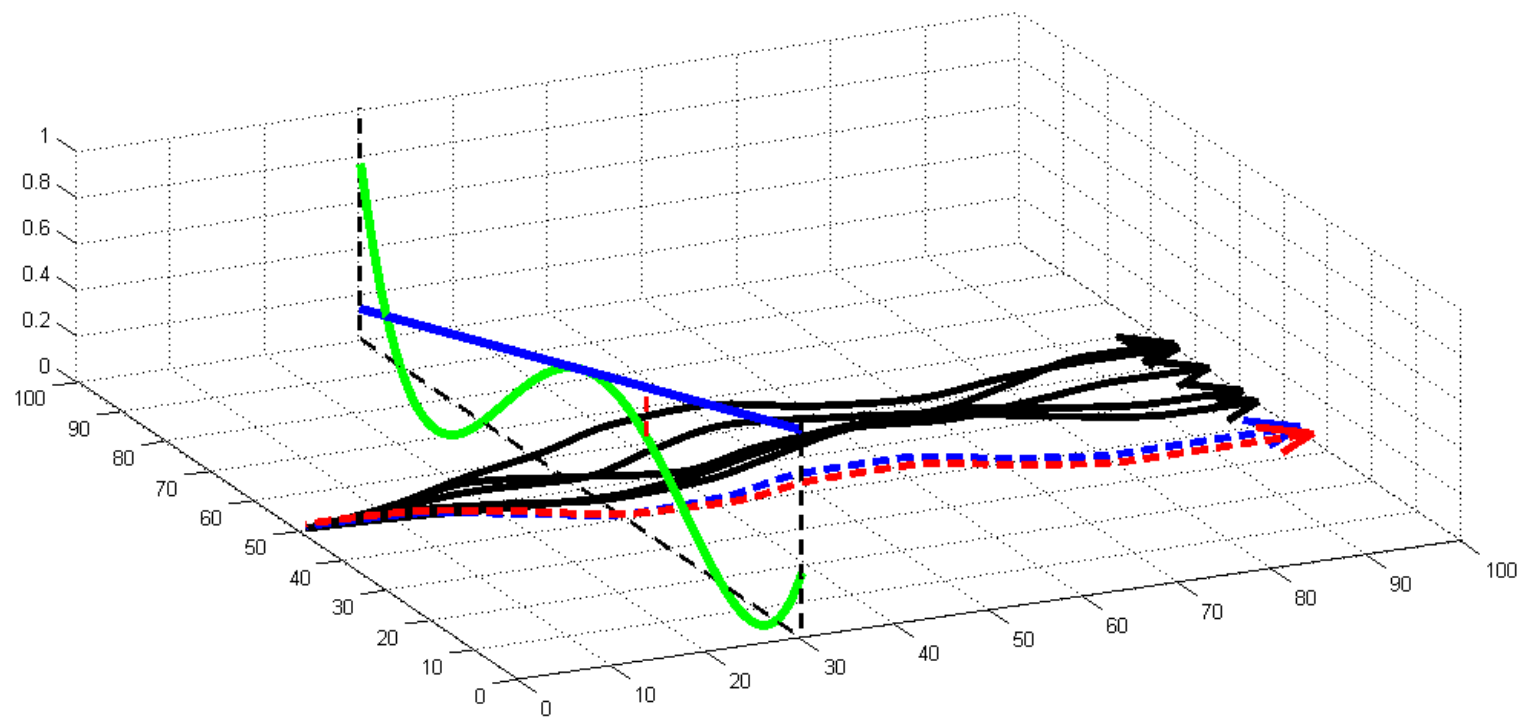


$$\{(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})_i\}$$

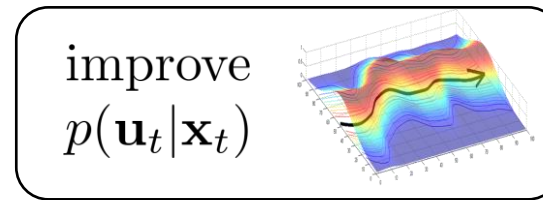
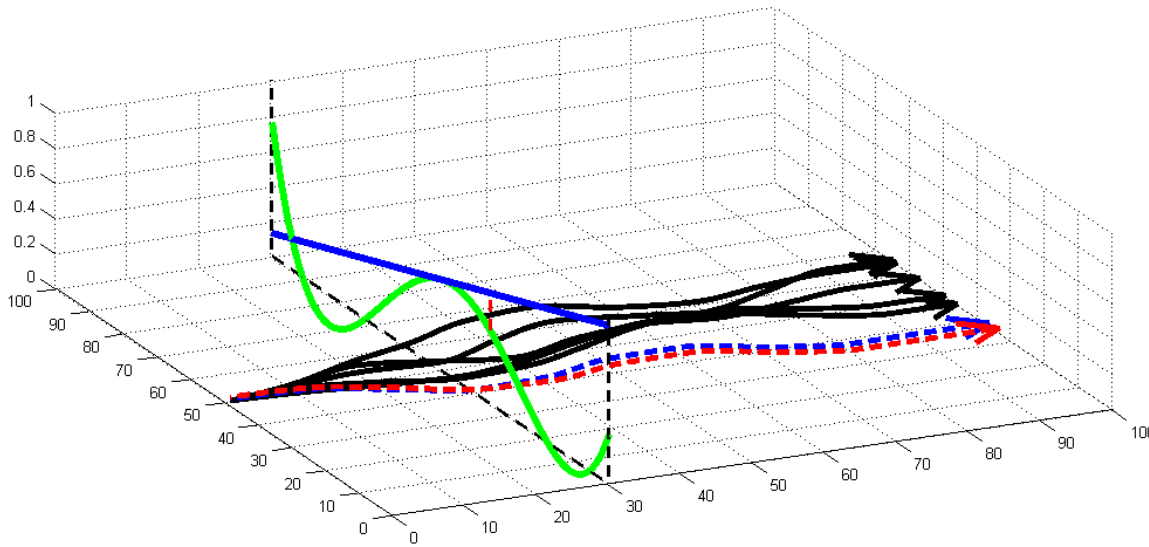
fit  $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$  at each time step using linear regression

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

# What if we go too far?



# How to stay close to old controller?



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

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

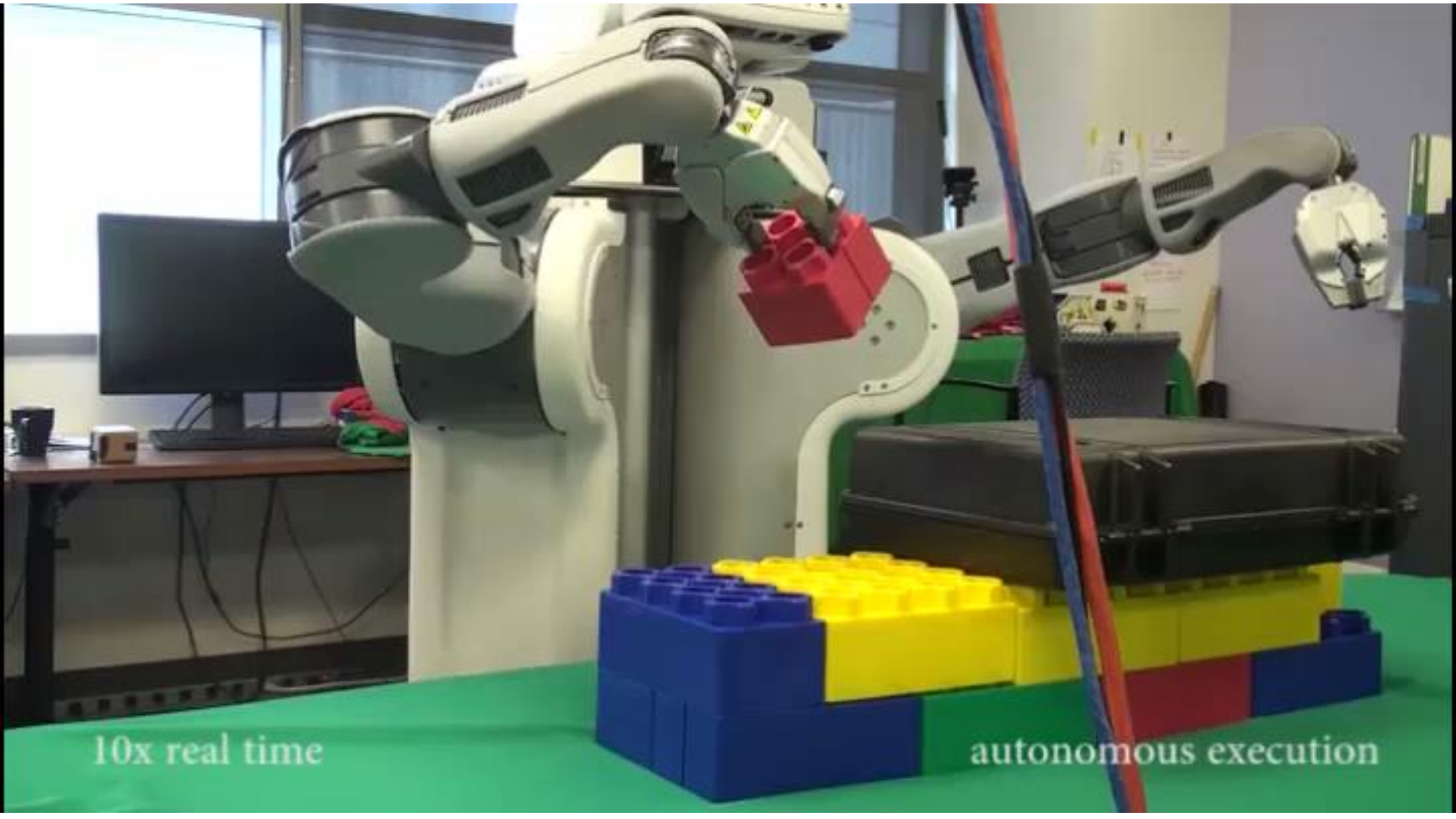
What if the new  $p(\tau)$  is “close” to the old one  $\bar{p}(\tau)$ ?

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

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

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

For details, see: “**Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics**”



10x real time

autonomous execution

# Global Policies from Local Models

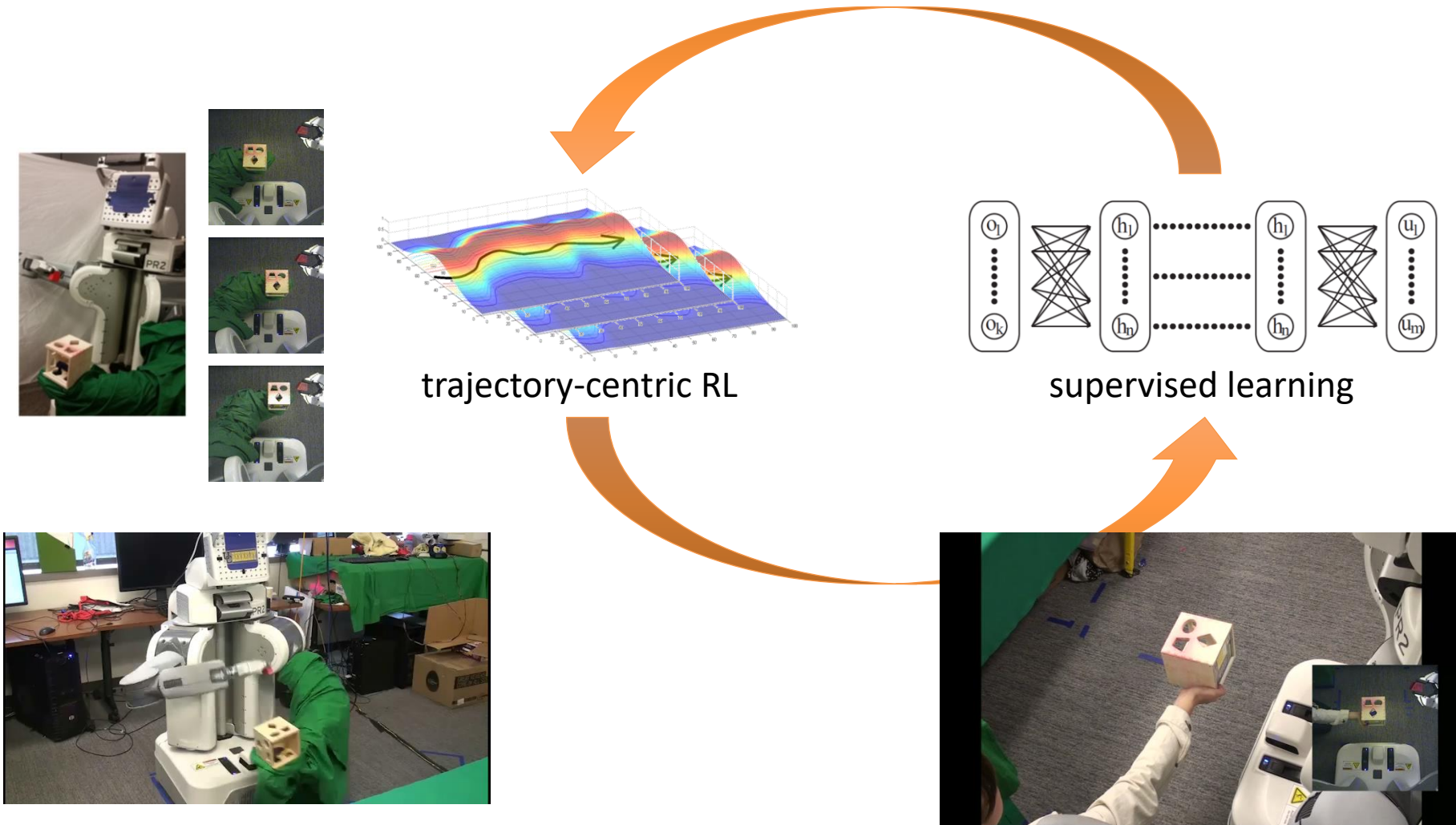
# What's the solution?

- Use derivative-free (“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 nets
  - LQR with learned models (LQR-FLM – **Fitted Local Models**)
  - Train **local** policies to solve simple tasks
  - Combine them into **global** policies via supervised learning

# What's the solution?

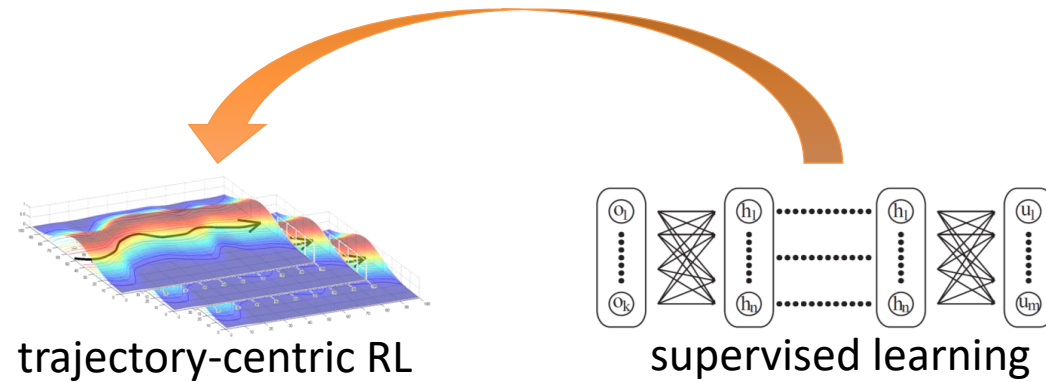
- Use derivative-free (“model-free”) RL algorithms, with the model used to generate synthetic samples
  - Seems weirdly backwards
  - Actually works very well
  - Essentially “model-based acceleration” for model-free RL
- Use simpler policies than neural nets
  - LQR with learned models (LQR-FLM – Fitted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

# Guided policy search: high-level idea





# Guided policy search: algorithm sketch



modified cost to keep  $\pi_{\text{LQR},i}$  close to  $\pi_\theta$

1. optimize each local policy  $\pi_{\text{LQR},i}(\mathbf{u}_t|\mathbf{x}_t)$  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  $\pi_{\text{LQR},i}(\mathbf{u}_t|\mathbf{x}_t)$
3. update cost function  $\tilde{c}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = c(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

For details, see: “End-to-End Training of Deep Visuomotor Policies” Lagrange multiplier

# 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?**

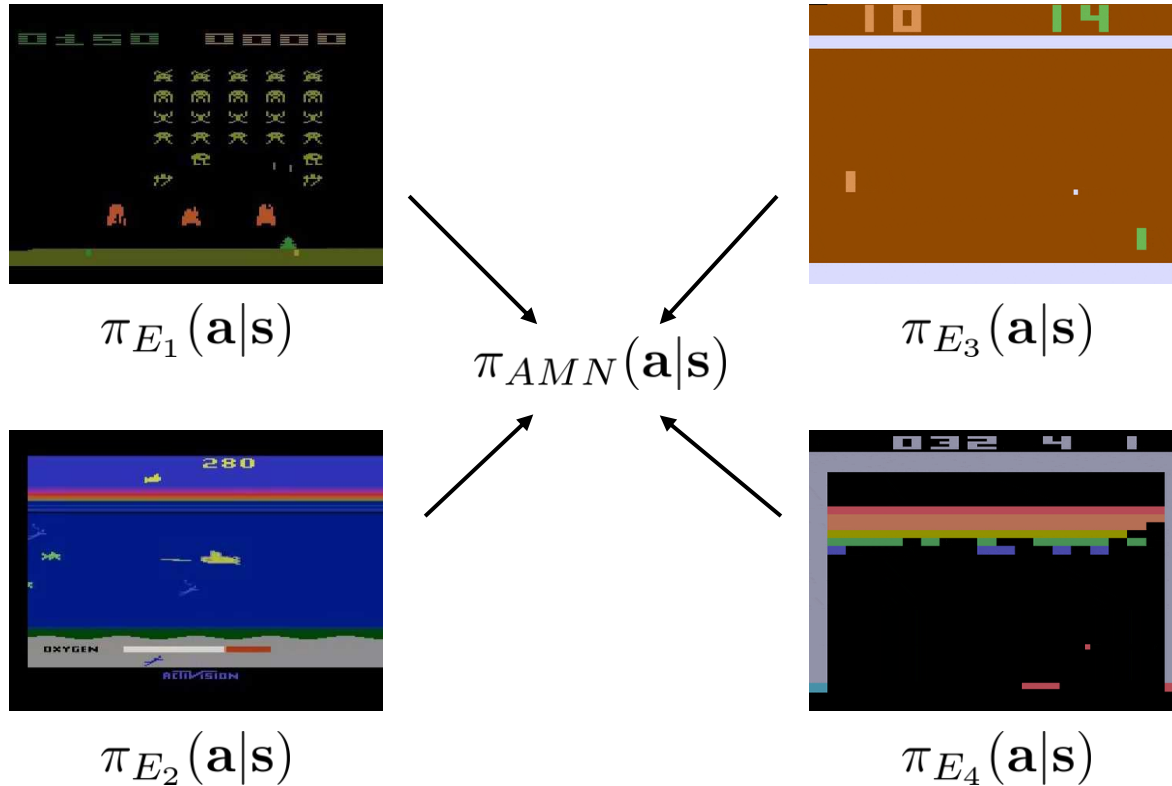
**Distillation:** train on the ensemble's predictions as “soft” targets

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

logit  $\rightarrow$   $\exp(z_i/T)$   $\leftarrow$  temperature

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

# Distillation for Multi-Task Transfer



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

(just supervised learning/distillation)

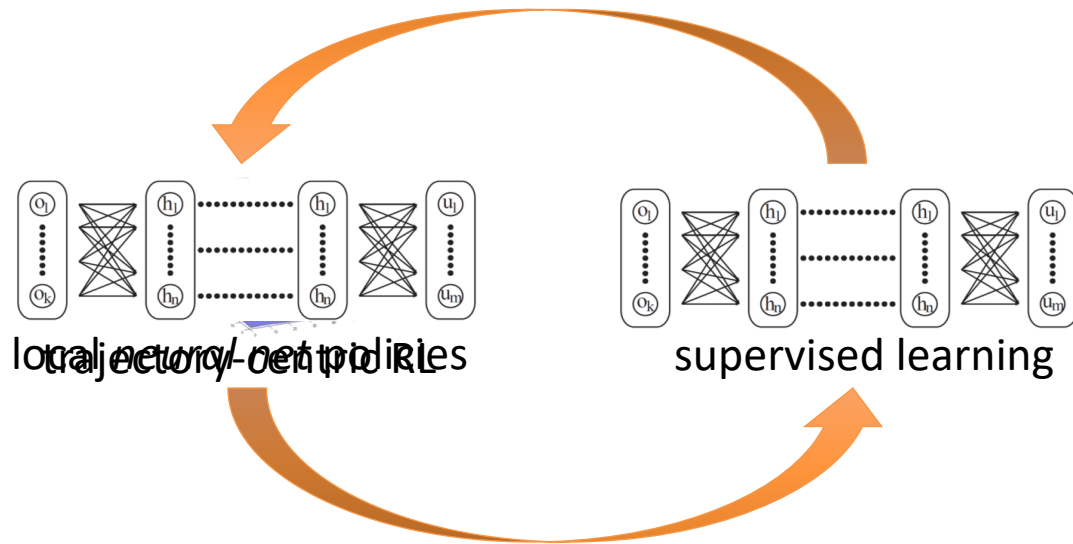
analogous to guided policy search, but  
for multi-task learning

some other details

(e.g., feature regression objective)

– see paper

# 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  $\tilde{r}_{k+1,i}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \lambda_{k+1,i} \log \pi_{\theta}(\mathbf{u}_t|\mathbf{x}_t)$

For details, see: “Divide and Conquer Reinforcement Learning”

# Readings: guided policy search & distillation

- L.\*, Finn\*, et al. End-to-End Training of Deep Visuomotor Policies. 2015.
- Rusu et al. Policy Distillation. 2015.
- Parisotto et al. Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning. 2015.
- Ghosh et al. Divide-and-Conquer Reinforcement Learning. 2017.
- Teh et al. Distral: Robust Multitask Reinforcement Learning. 2017.

# UC Berkeley·CS285 | Deep Reinforcement Learning (2020)

## CS285 (2020)· 课程资料包 @ShowMeAI



视频

中英双语字幕



课件

一键打包下载



笔记

官方笔记翻译



代码

作业项目解析



视频·B 站 [ 扫码或点击链接 ]

<https://www.bilibili.com/video/BV12341167kL>



课件 & 代码·博客 [ 扫码或点击链接 ]

<http://blog.showmeai.tech/cs285>

Berkeley

actor-critic

reinforcement learning

exploration

元学习

exploitation

梯度策略

探索与利用

DQN

迁移学习

逆强化学习

Q-learning / Q 学习

deep q network

meta learning

Awesome AI Courses Notes Cheatsheets 是 [ShowMeAI](#) 资料库的分支系列，覆盖最具知名度的 **TOP20+** 门 AI 课程，旨在为读者和学习者提供一整套高品质中文学习笔记和速查表。

点击课程名称，跳转至课程**资料包**页面，**一键下载**课程全部资料！

机器学习	深度学习	自然语言处理	计算机视觉
Stanford · CS229	Stanford · CS230	Stanford · CS224n	Stanford · CS231n
# Awesome AI Courses Notes Cheatsheets· 持续更新中			
知识图谱	图机器学习	深度强化学习	自动驾驶
Stanford · CS520	Stanford · CS224W	UCBerkeley · CS285	MIT · 6.S094



微信公众号

资料下载方式 2: 扫码点击**底部菜单栏**

称为 **AI 内容创作者**? 回复 [ 添砖加瓦 ]