Computer Engineering Department

Reinforcement Learning:
Model Based RL

Mohammad Hossein Rohban, Ph.D.

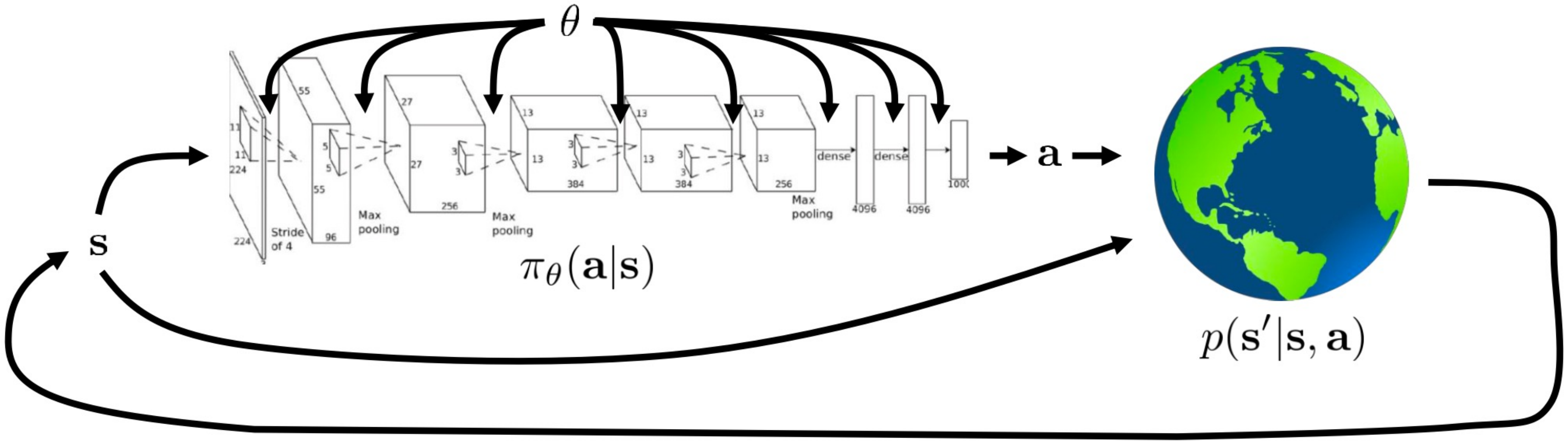
Spring 2025

Courtesy: Most of slides are adopted from CS 285 Berkeley.

Overview

- Introduction to model-based reinforcement learning
- What if we know the dynamics? How can we make decisions?
- Stochastic optimization methods
- Monte Carlo tree search (MCTS)
- Trajectory optimization
- Goal: Understand how we can perform planning with known dynamics models in discrete and continuous spaces

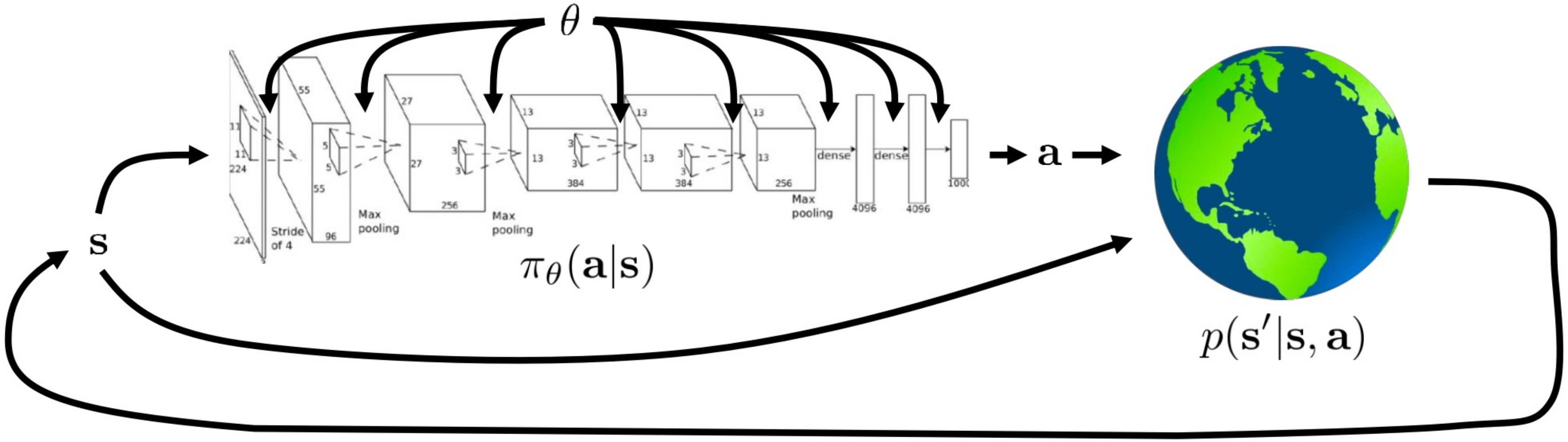
Recap: Model-Free RL



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

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

Recap: 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
  - 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 can learn the dynamics
  - System identification – fit unknown parameters of a known model
  - Learning – fit a general-purpose model to observed transition data

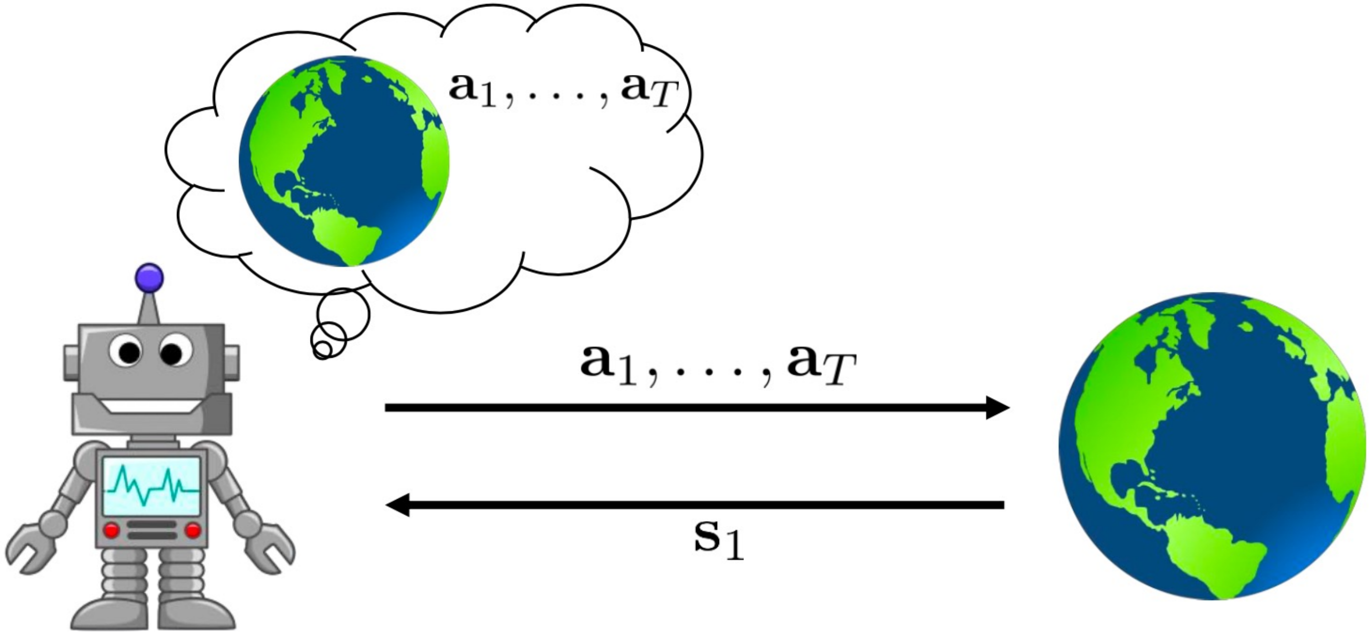
Does knowing the dynamics make things easier?

Often, yes!

Model-based RL

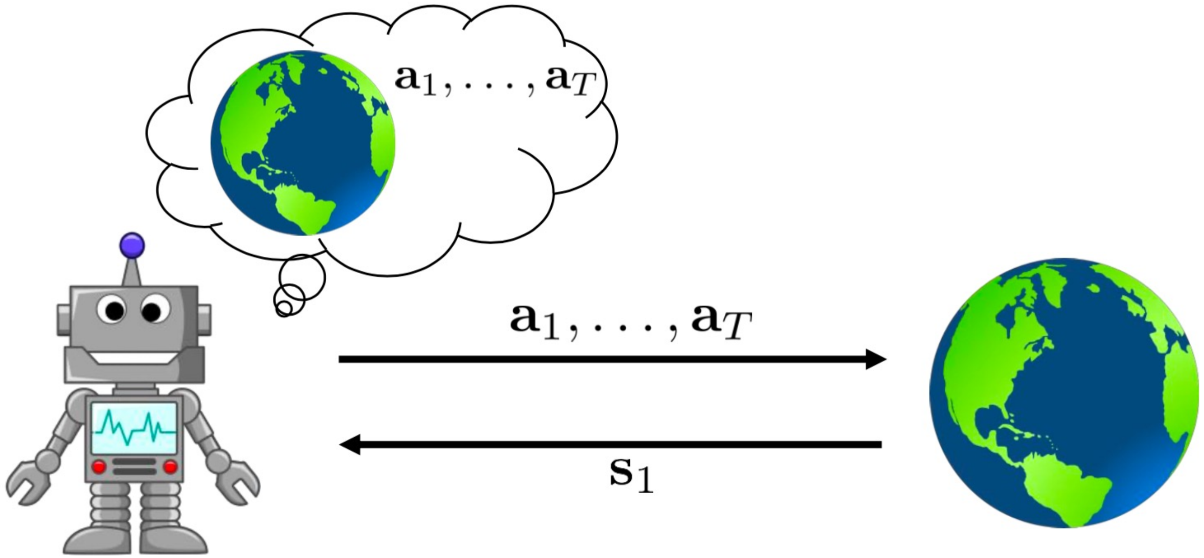
- Model-based reinforcement learning: learn the transition dynamics, then figure out how to choose actions.
- Today: how can we make decisions if we know the dynamics?
  - a. How can we choose actions under perfect knowledge of the system dynamics?
  - b. Optimal control, trajectory optimization, planning

The deterministic case



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

The stochastic open-loop case



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

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

why is this suboptimal?

The stochastic open-loop case

کری می خواست به عیادت بیماری برود.اندیشید که هنگام احوال پرسی ممکن است صدای اورانشنوم وپاسخی ناشایسته بدهم.ازاین رودرپی چاره برآمدوبالاخره باخود گفت:بهتراست پرسشهارا پیش ازرفتن بسنجم وپاسخ رانیزبرآورد کنم تادچاراشتباه نشوم.

بنابراین پرسشهای خودراچنین پیش بینی کرد:

-ابتداازاومی پرسم حالت بهتراست؟ اوخواهد گفت "آری" من درجواب می گویم:خدا را شکر

-بعدازاومی پرسم چه خورده ای؟ لابد نام غذایی راخواهد آورد.من می گویم گوارا باد.

-درپایان می پرسم پزشکت کیست؟ نام پزشکی رامی گویدومن پاسخ می دهم:مقدمش مبارک باد.

....

چون به خانه ی بیماررسید همان گونه که ازپیش آماده شده بودبه احوال پرسی پرداخت:

-کر گفت:"چگونه ای؟"

بیمارگفت: مُردم

کر گفت: خدارا شکر

بیمارازاین سخن بیجا برآشفت.

-بعدازآن پرسید:"چه خورده ای؟"

بیمارگفت: زهر

کر گفت: گواراباد.داروی خوبی است.

بیمار ازاین پاسخ نیزبیشتربه خود پیچید.

-بعد ازآن کر گفت:" ازطبیبان کیست او کاوهمی آید به چاره پیش تو؟"

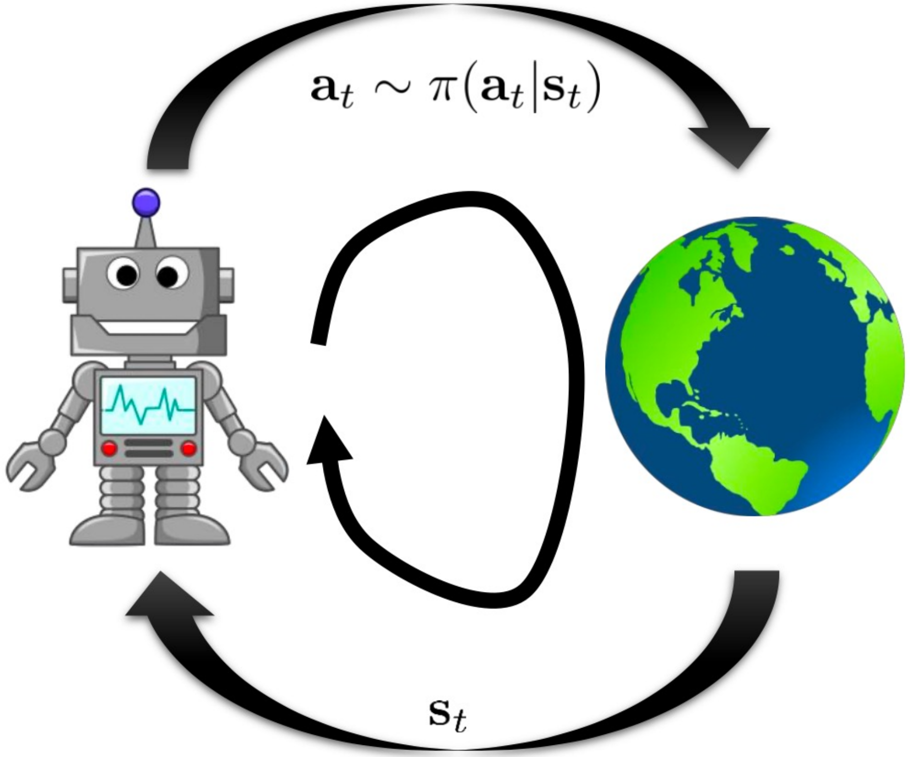
بیمار که آشفتگی وناراحتی اش به نهایت رسیده بود در پاسخ گفت:

عزرائیل می آید, برو.

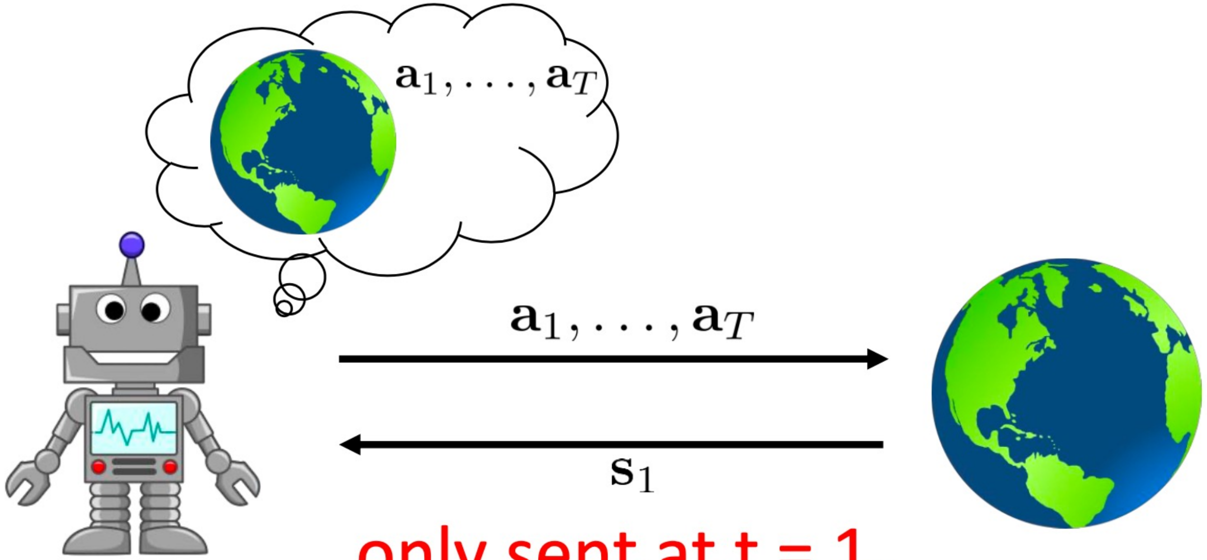
کر گفت: پایش بس مبارک.شاد شو!

open-loop vs. closed-loop case

closed-loop

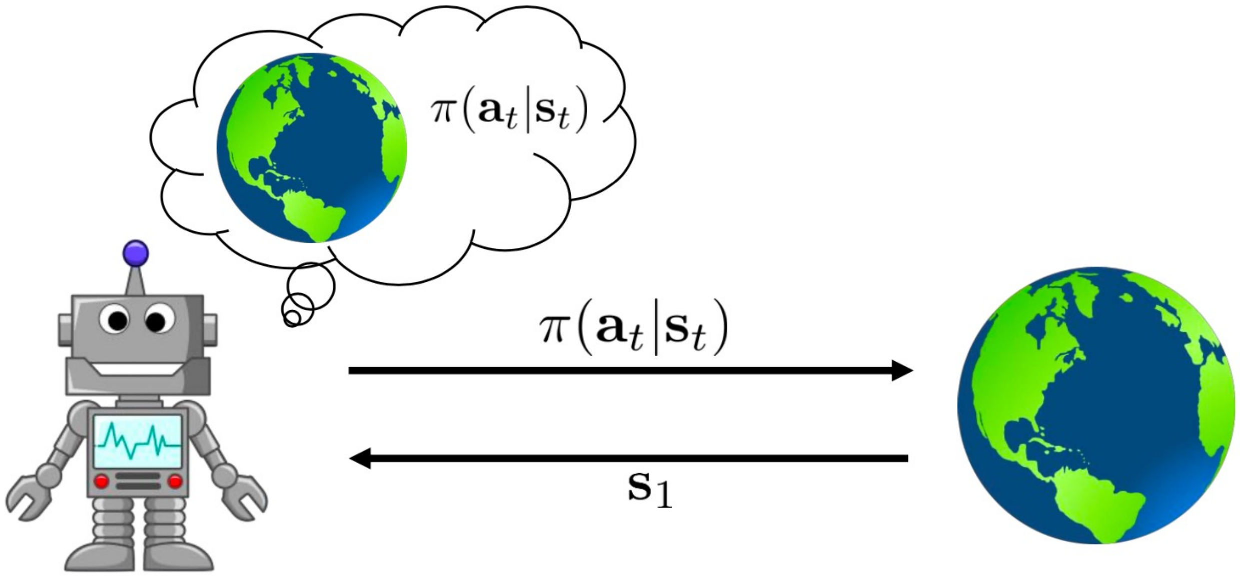


open-loop



only sent at t = 1, then it's one-way!

The stochastic open-loop case



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

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

form of π?

neural net global

time-varying linear

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

Stochastic optimization

abstract away optimal control/planning:

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

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

simplest method: guess & check "random shooting method"

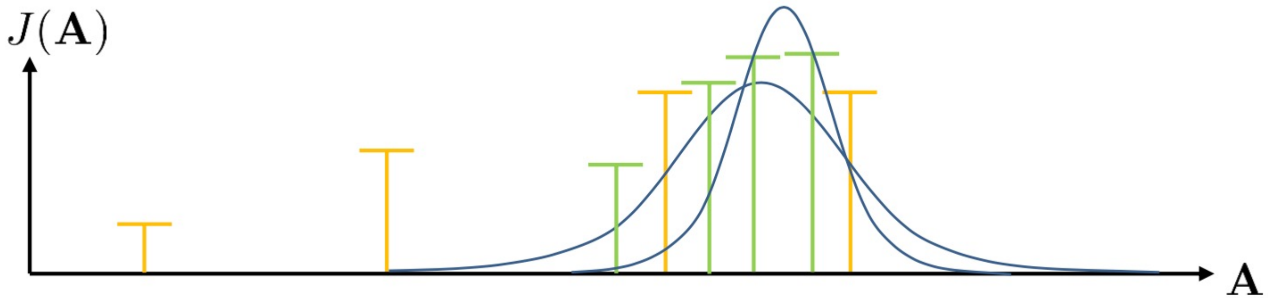
1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ from some distribution (e.g., uniform)
2. choose $\mathbf{A}_i$ based on $\arg\max_i J(\mathbf{A}_i)$

Cross-entropy Method (CEM)

1. pick $\mathbf{A}_1, \ldots, \mathbf{A}_N$ 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 $\mathbf{A}_1, \ldots, \mathbf{A}_N$ 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 $\mathbf{A}_{i_1}, \ldots, \mathbf{A}_{i_M}$

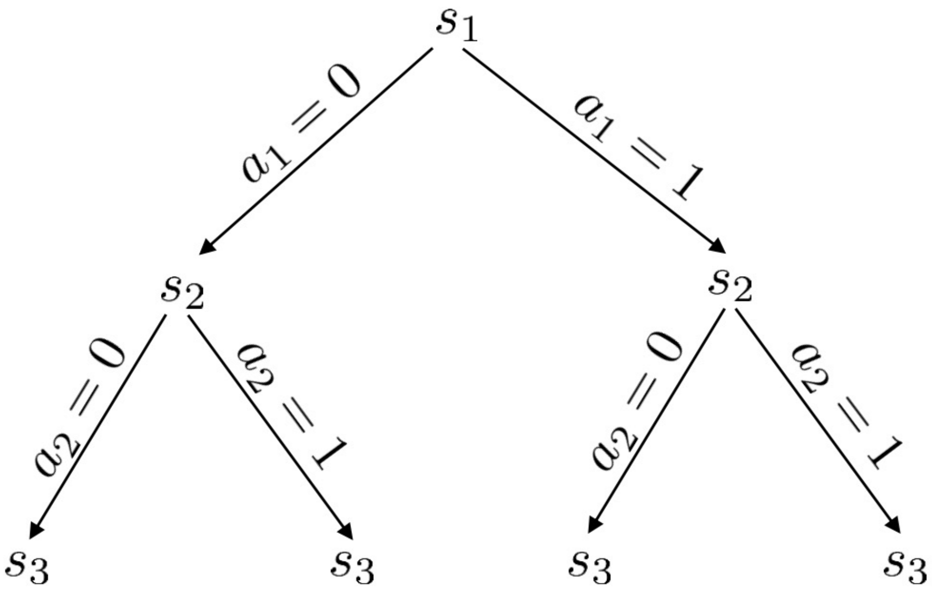
Pros and Cons

- Pros
  - Could be very fast (Parallelizable)
  - Extremely simple
- Cons
  - Very harsh dimensionality limit
  - Only open-loop planning

Discrete Case: Monte Carlo Tree Search

s_t a_t

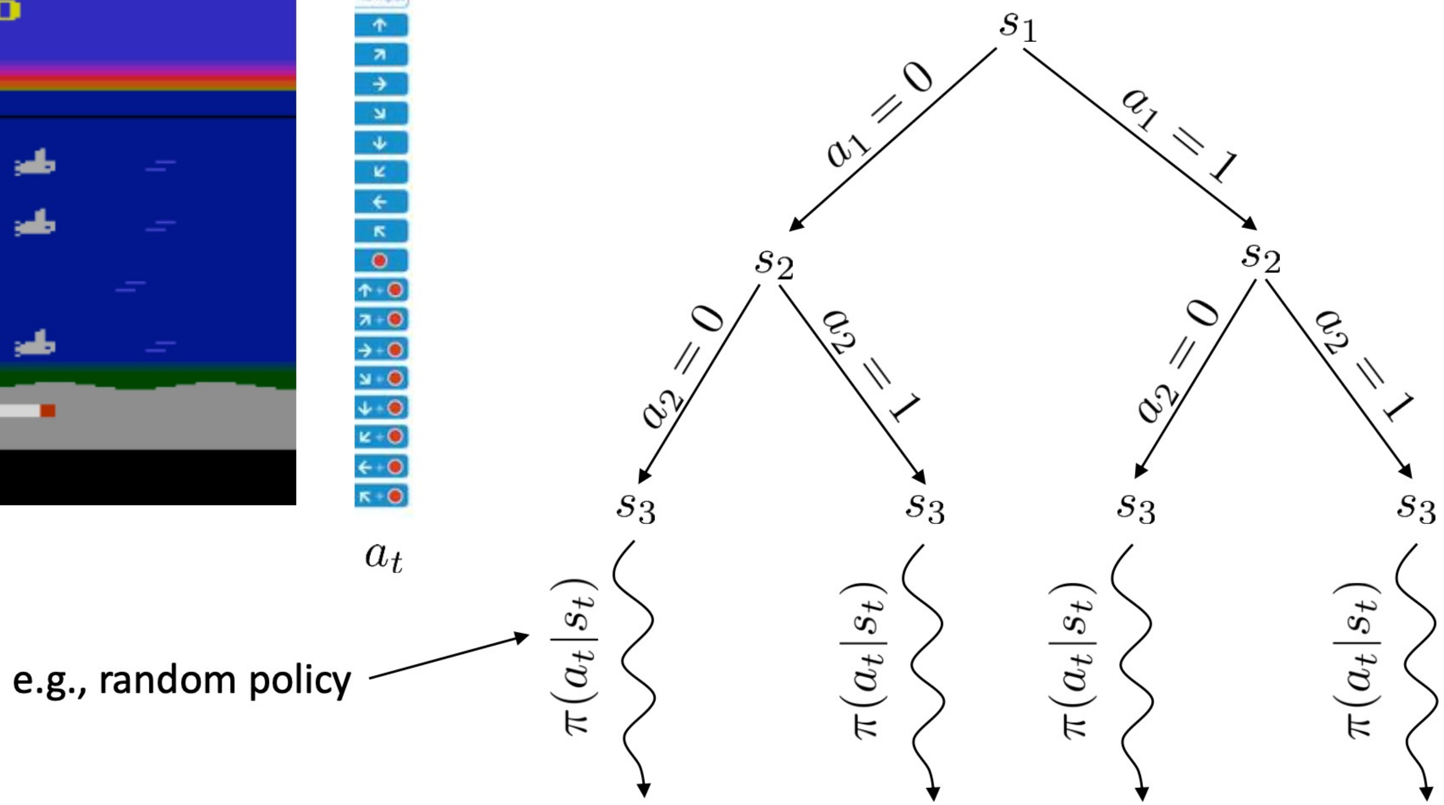
discrete planning as a search problem



Discrete Case: Monte Carlo Tree Search

s_t a_t

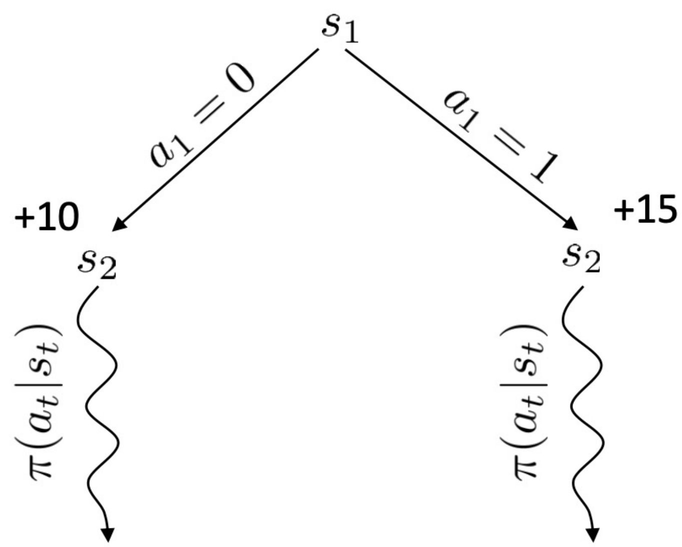
how to approximate value without full tree?



Discrete Case: Monte Carlo Tree Search

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

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

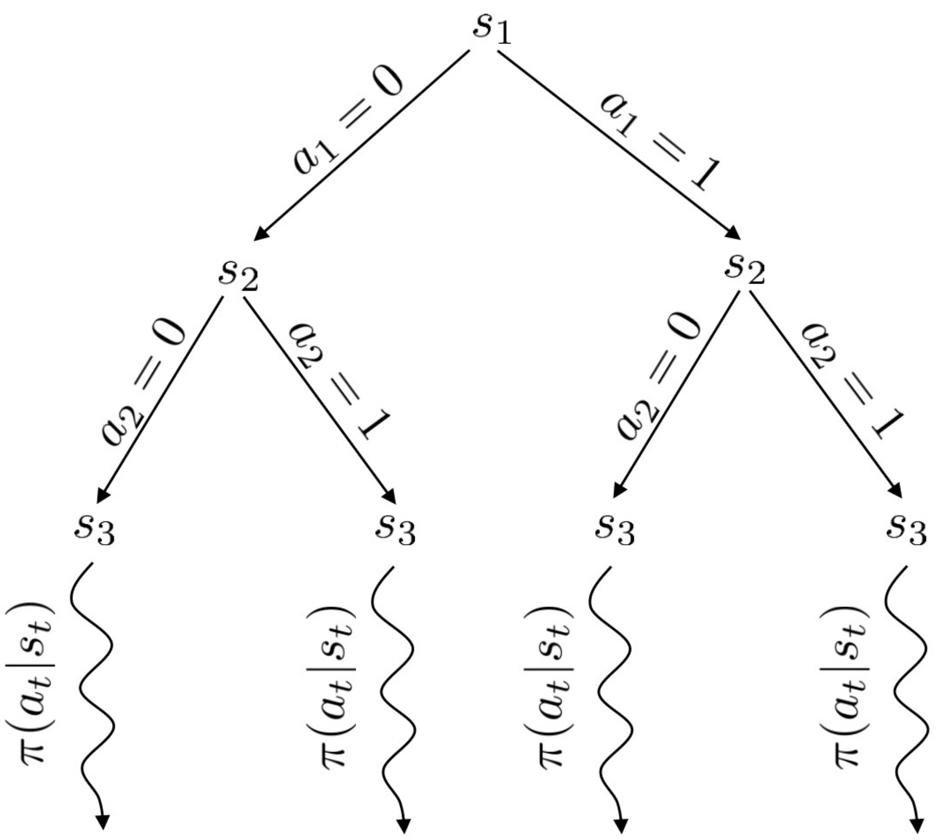
take best action from s_1

UCT TreePolicy(s_t)

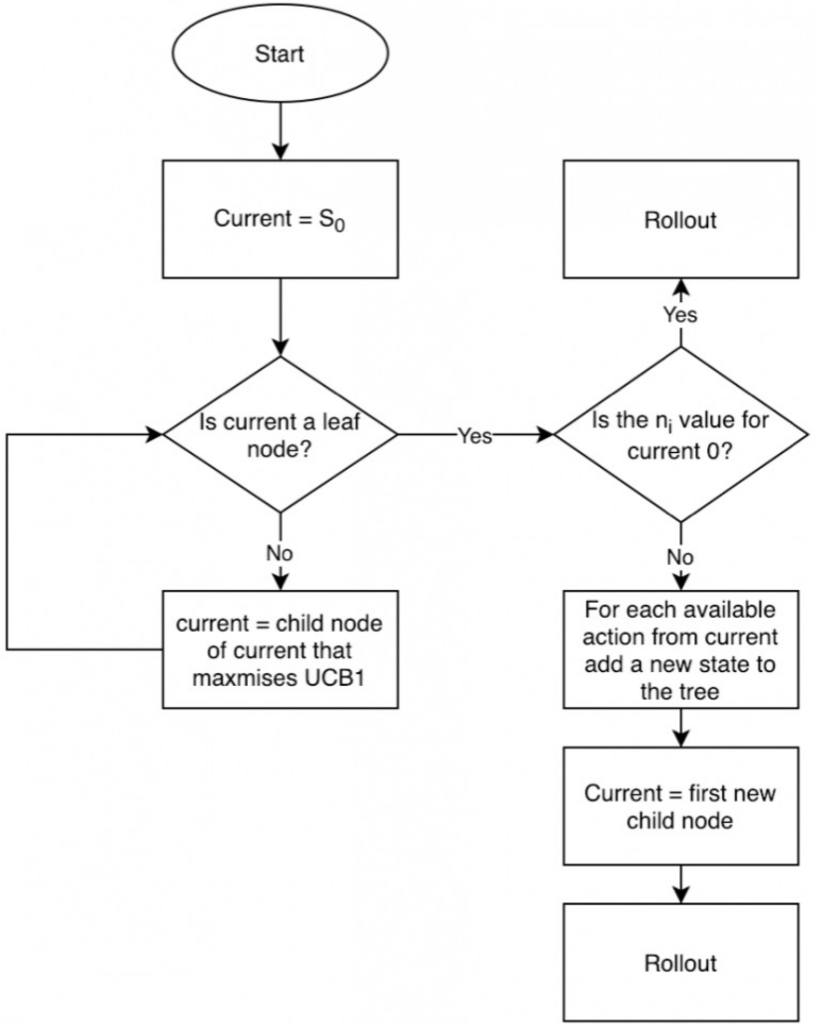
if s_t not fully expanded, choose new a_t

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

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



Discrete Case: Monte Carlo Tree Search



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.

Today’s Lecture

1. Basics of model-based RL: learn a model, use model for control
    - Why does naïve approach not work?
    - The effect of distributional shift in model-based RL
2. Uncertainty in model-based RL
3. Model-based Policy Learning

- Goals:
    - Understand how to build model-based RL algorithms
    - Understand the important considerations for model-based RL
    - Understand the tradeoffs between different model class choices

Why learn the model?

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

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

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

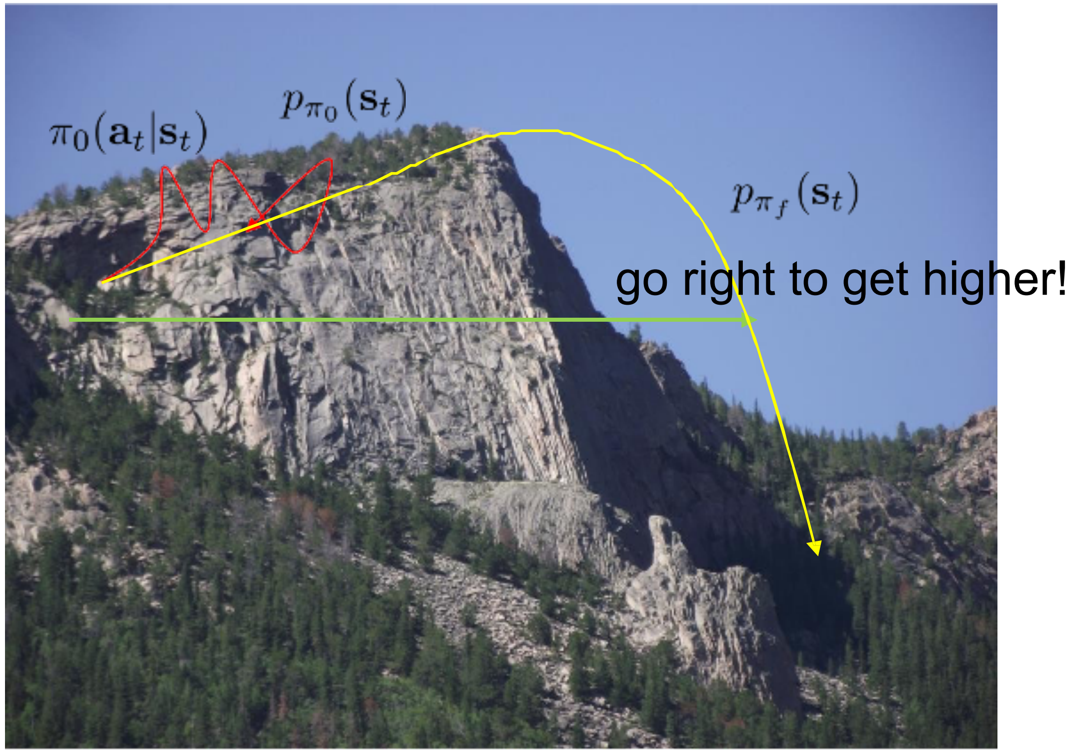
model-based reinforcement learning version 0.5:

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

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

Can we do better?

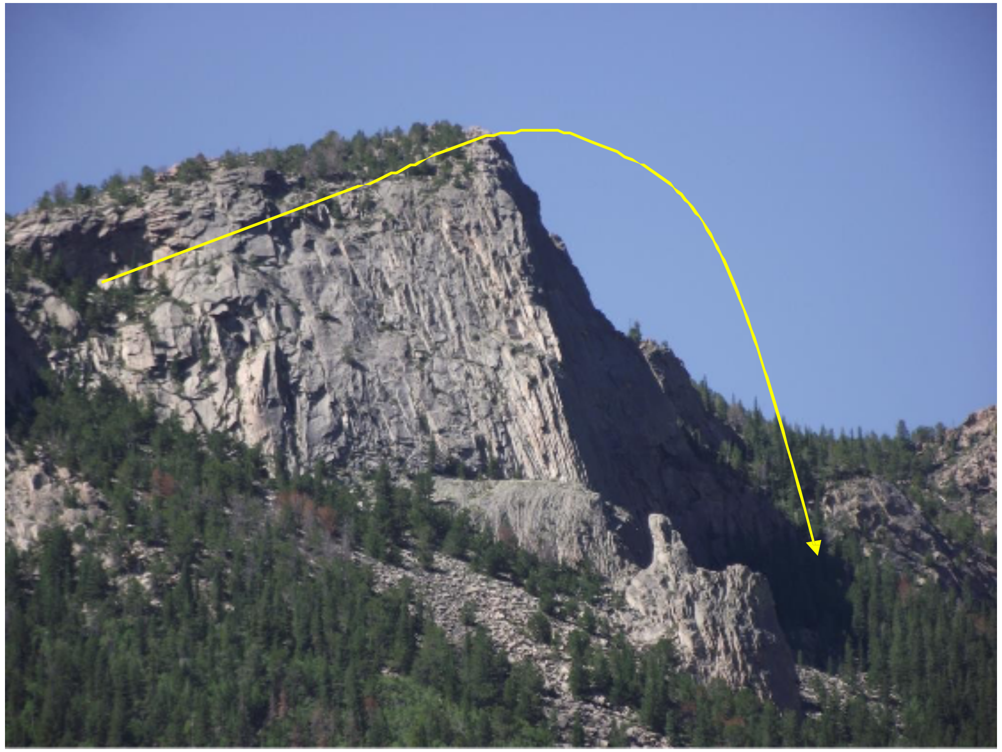
can we make $p_{\pi_0}(\mathbf{s}_t) = p_{\pi_f}(\mathbf{s}_t)$?

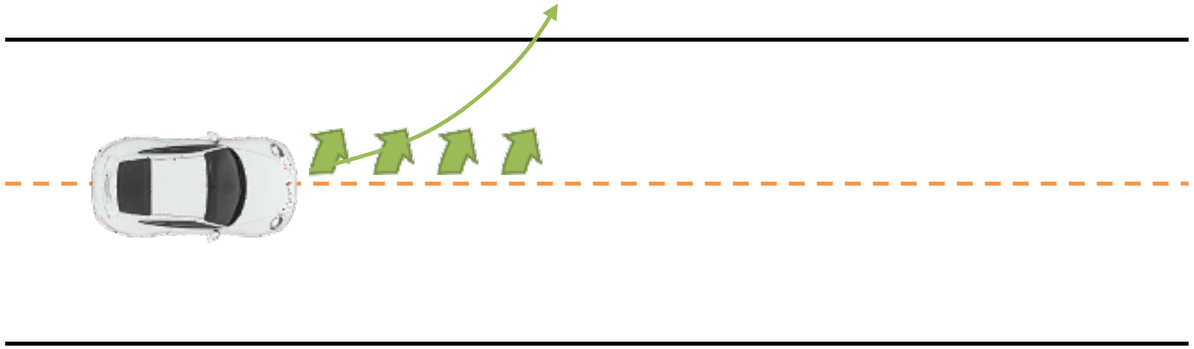
where have we seen that before? need to collect data from $p_{\pi_f}(\mathbf{s}_t)$

model-based reinforcement learning version 1.0:

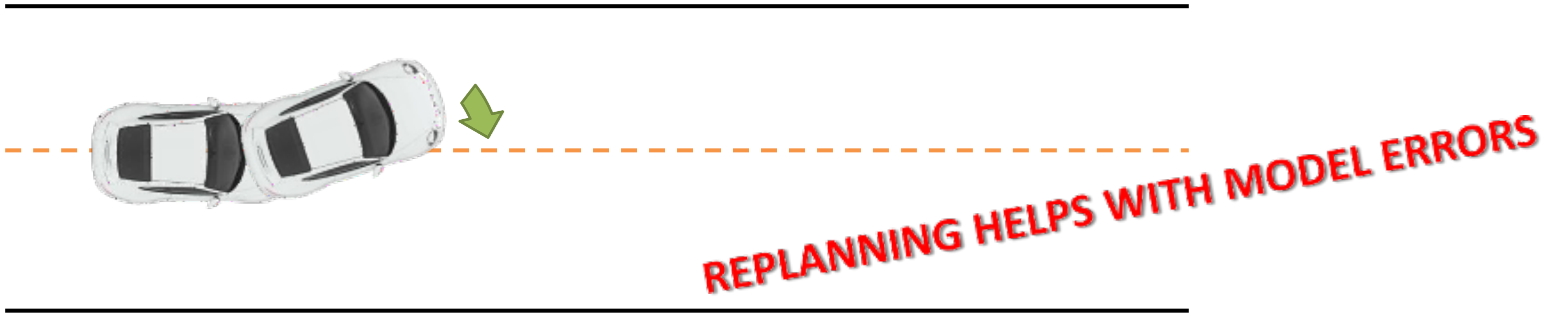
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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





Can we do better?



model-based reinforcement learning version 1.5:

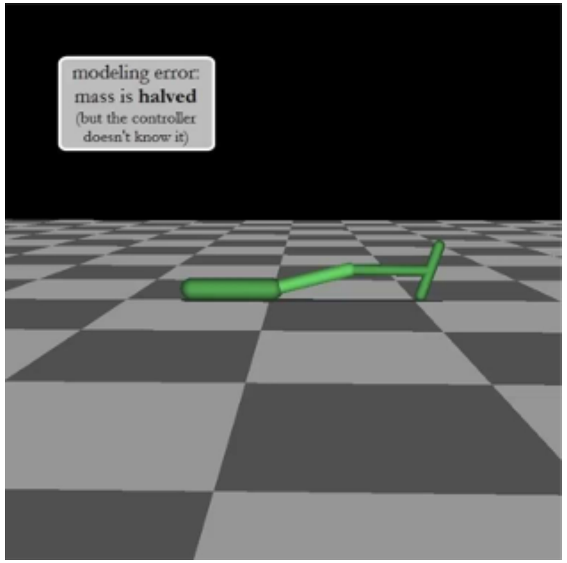
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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

How to replan?

model-based reinforcement learning version 1.5:

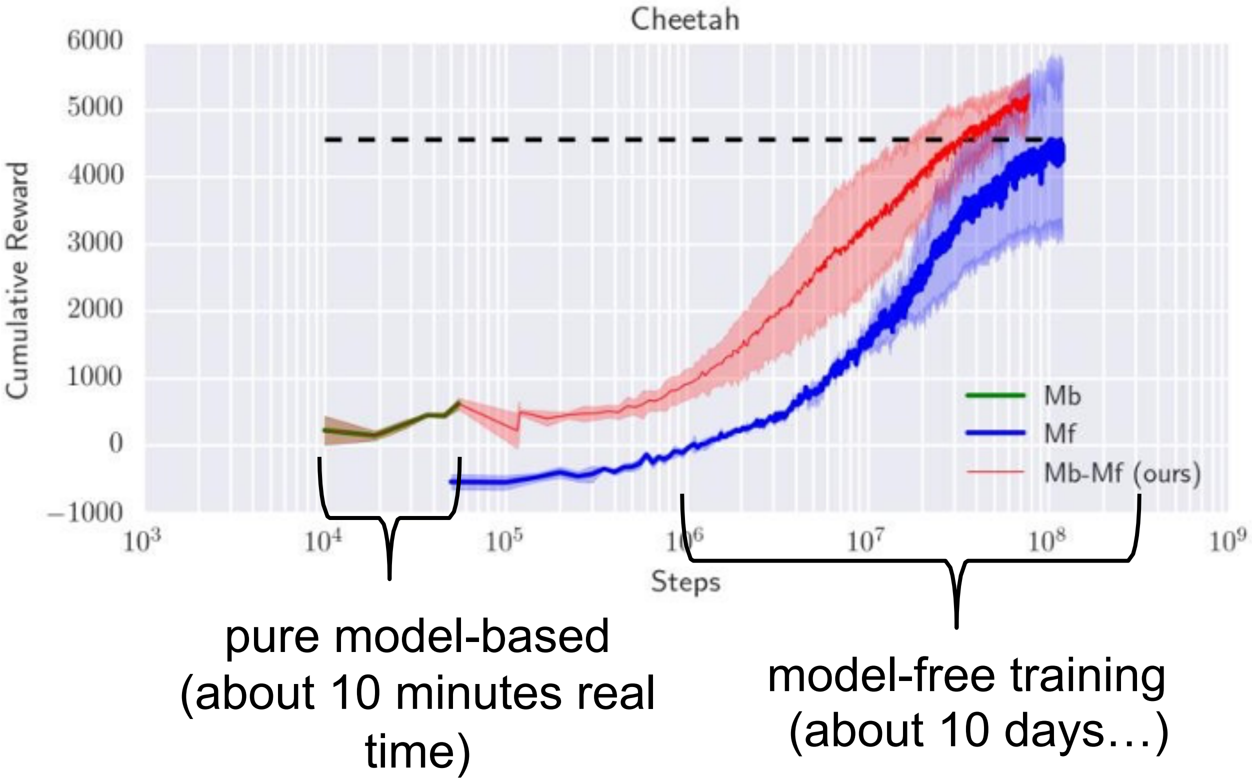
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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

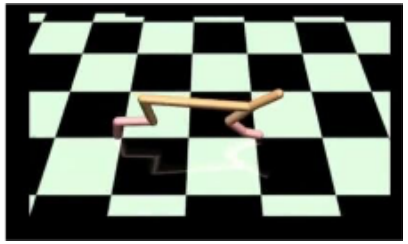
- The more you replan, the less perfect each individual plan needs to be
- Can use shorter horizons
- Even random sampling can often work well here!

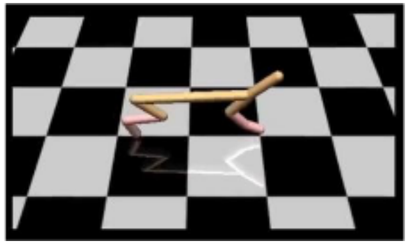


Uncertainty in Model-Based RL

A performance gap in model-based RL

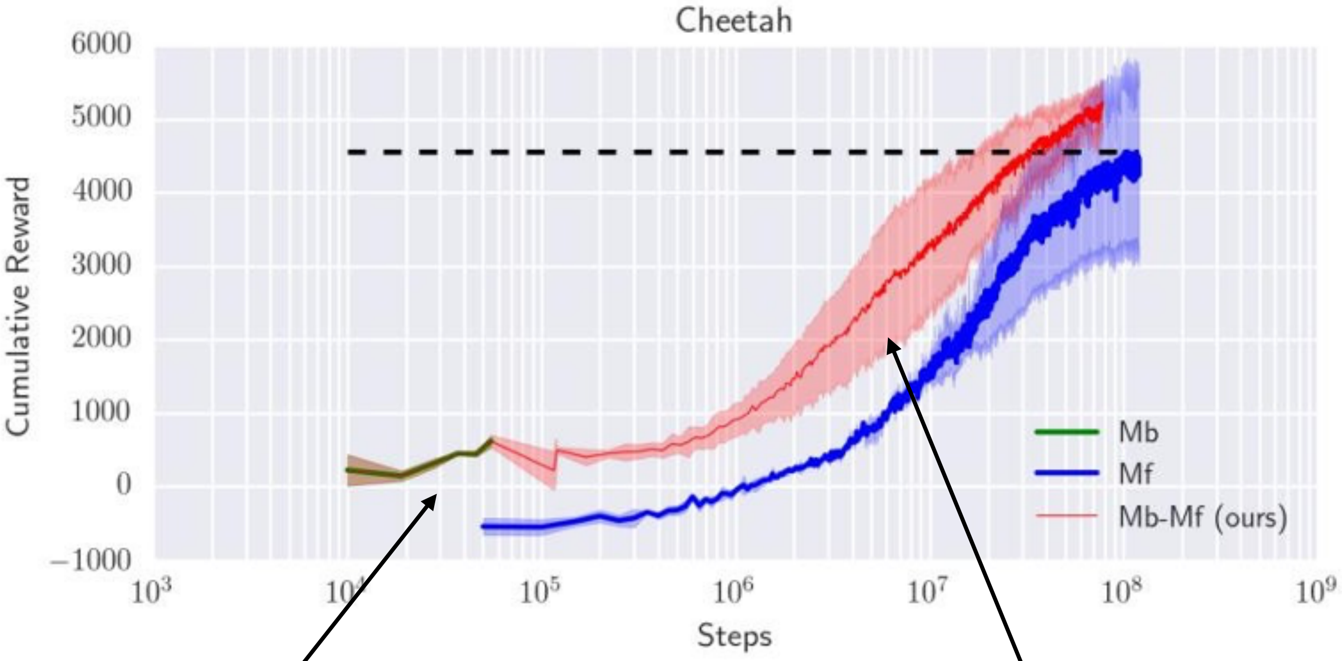






Nagabandi, Kahn, Fearing, L. ICRA 2018

Why the performance gap?



need to not overfit here…

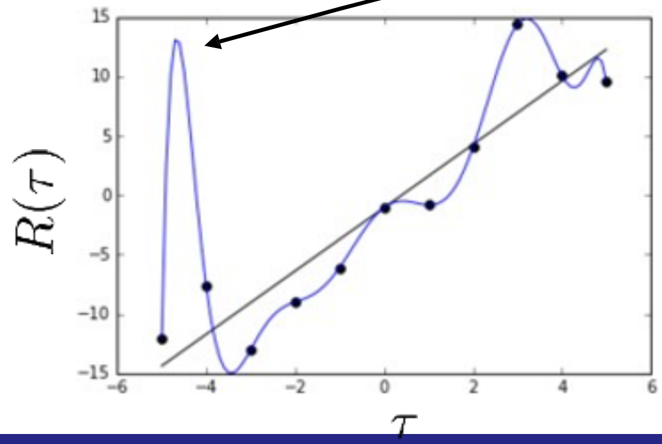
…but still have high capacity over here

Why the performance gap?

model-based reinforcement learning version 1.5:

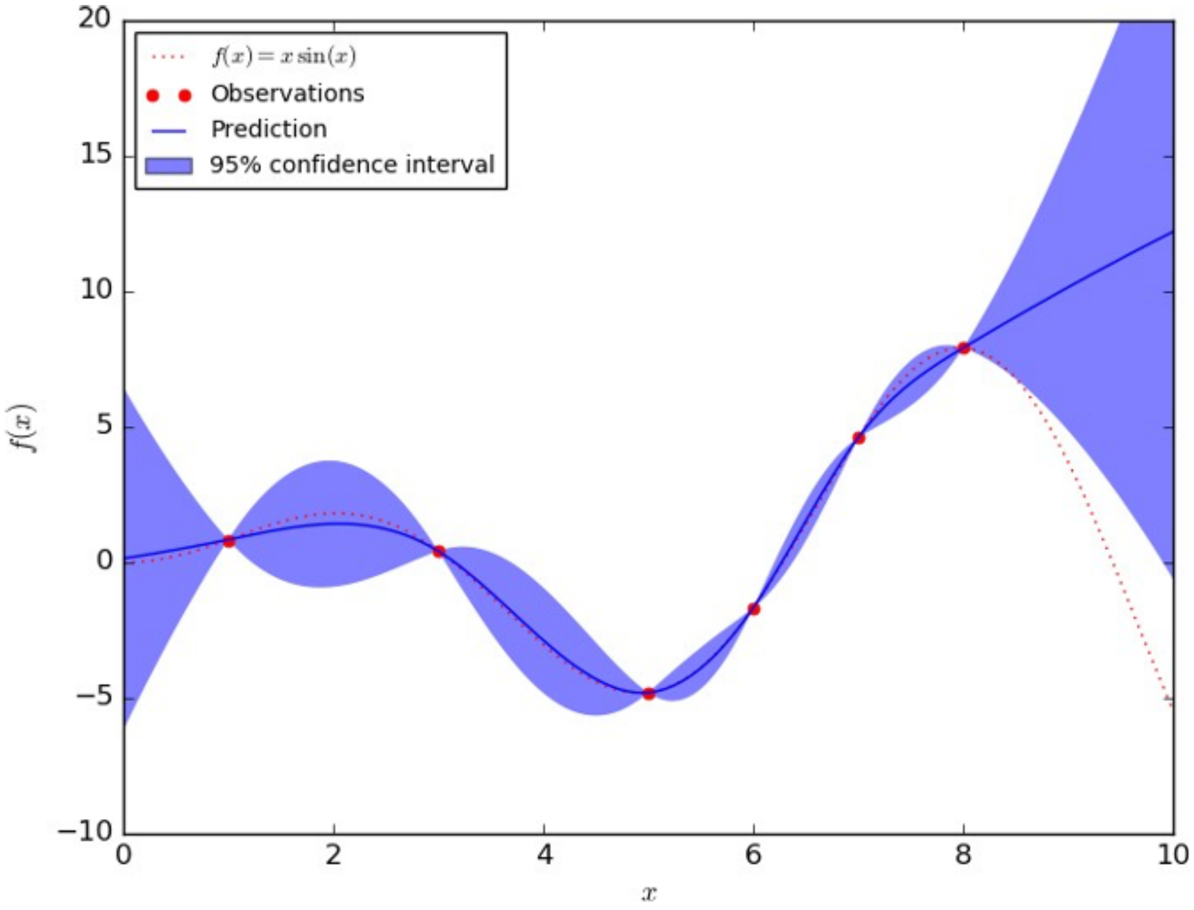
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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

How can uncertainty estimation help?



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

expected reward under high-variance prediction is **very** low, even though mean is the same!

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

There are a few caveats…

Need to explore to get better

Expected value is not the same as pessimistic value

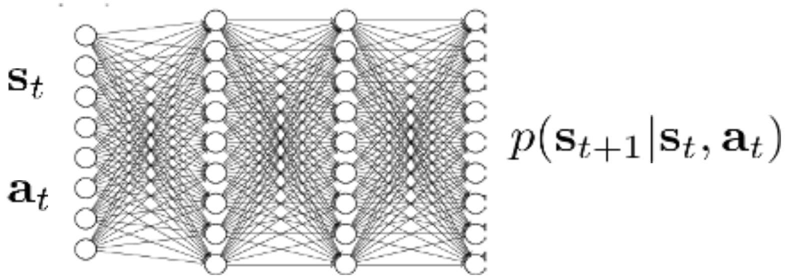
Expected value is not the same as optimistic value

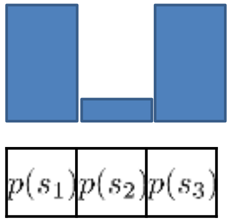
…but expected value is often a good start

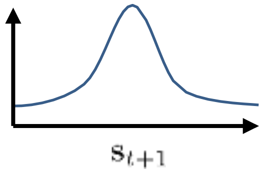
Uncertainty-Aware Neural Net Models

How can we have uncertainty-aware models?

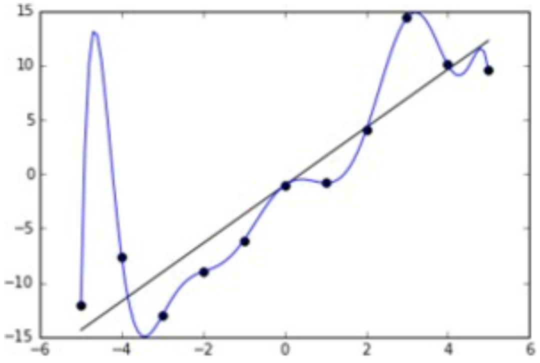
Idea 1: use output entropy







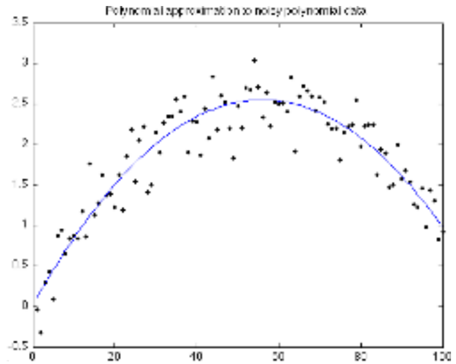
why is this not enough?



Two types of uncertainty:

aleatoric or *statistical* uncertainty

epistemic or *model* uncertainty



what is the variance here?

“the model is certain about the data, but we are not certain about the model”

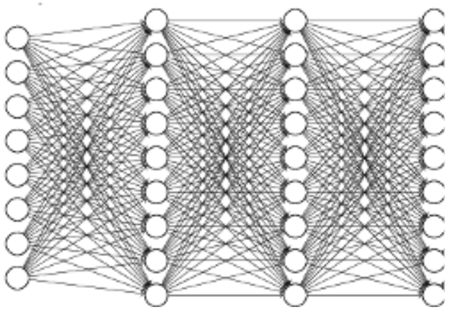
How can we have uncertainty-aware models?

Idea 2: estimate model uncertainty

“the model is certain about the data, but we are not certain about the model”

$\mathbf{s}_t$

$\mathbf{a}_t$



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

parameters θ

usually, we estimate

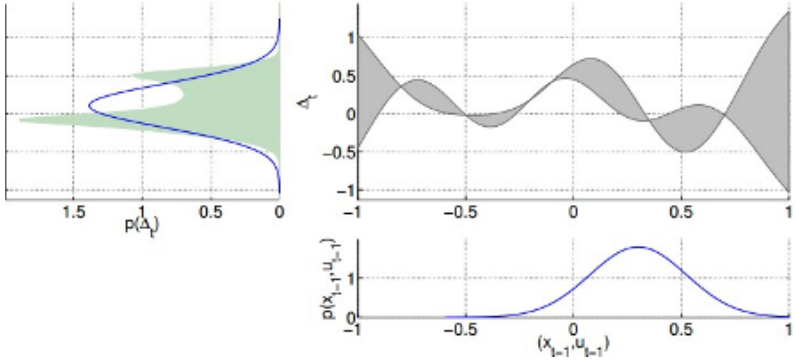
$$\arg\max_\theta \log p(\theta|\mathcal{D}) = \arg\max_\theta \log p(\mathcal{D}|\theta)$$

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

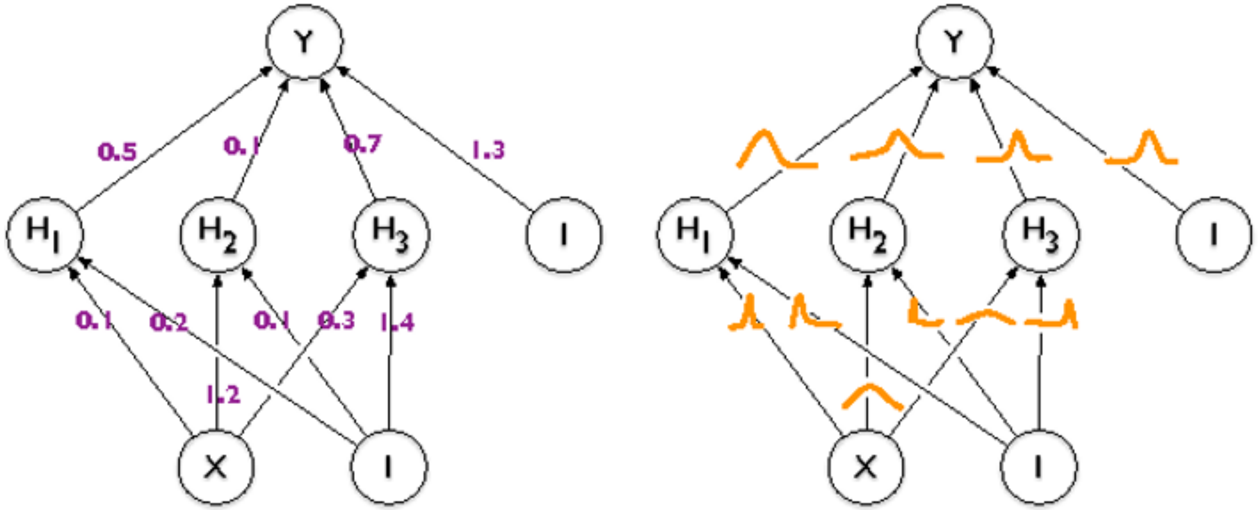
the entropy of this tells us the model uncertainty!

predict according to:

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



Quick overview of Bayesian neural networks



common approximation:

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

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

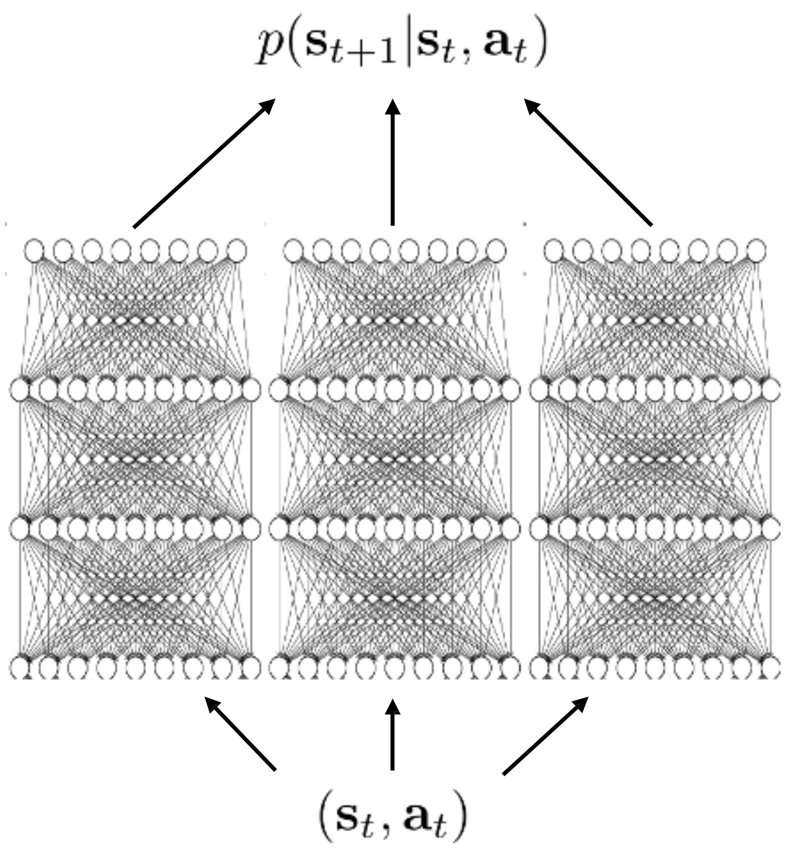
expected weight

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

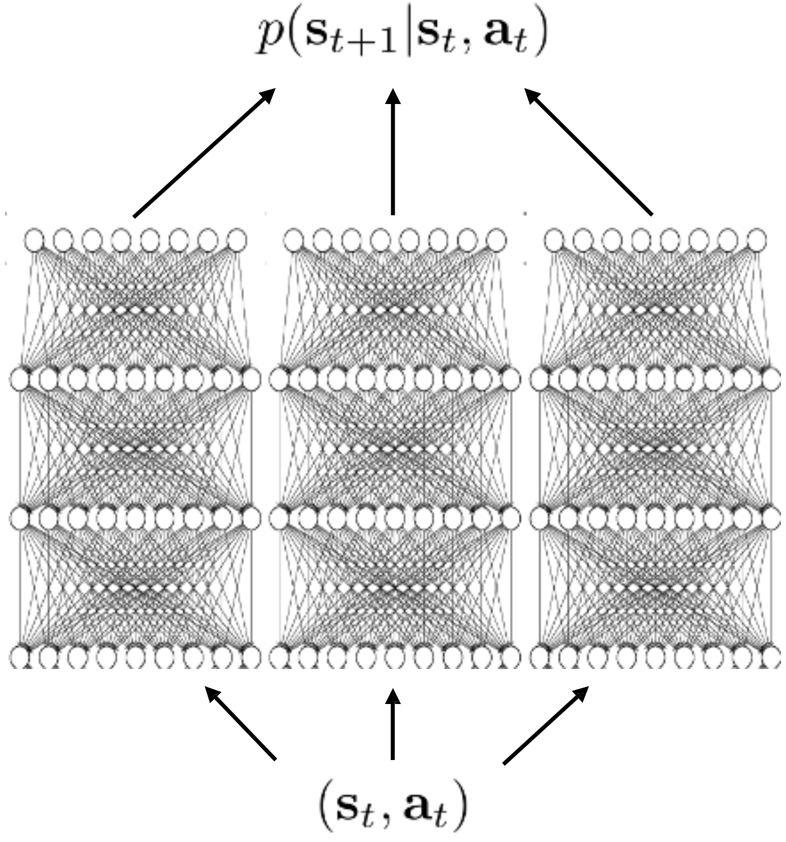
$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta) p(\theta|\mathcal{D}) d\theta \approx \frac{1}{N}\sum_i p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta_i)$$

How to train?

Main idea: need to generate "independent" datasets to get "independent" models

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

Planning with Uncertainty, Examples

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 $R = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$

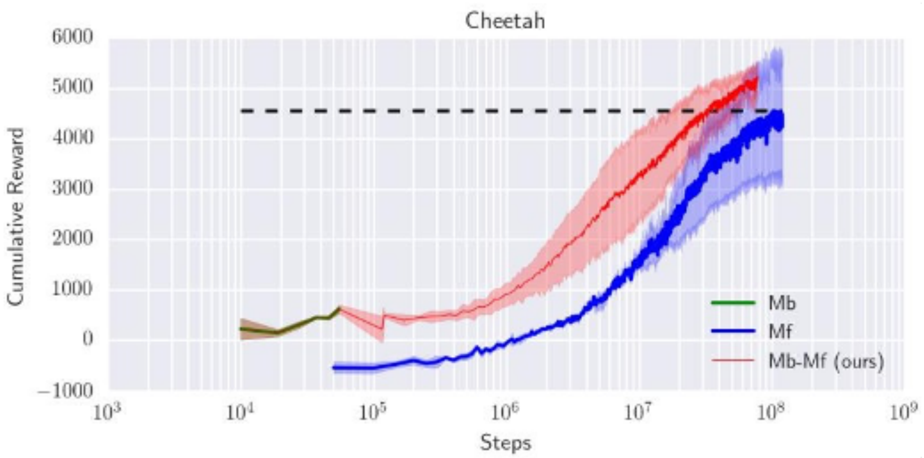
Step 4: repeat steps 1 to 3 and accumulate the average reward

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

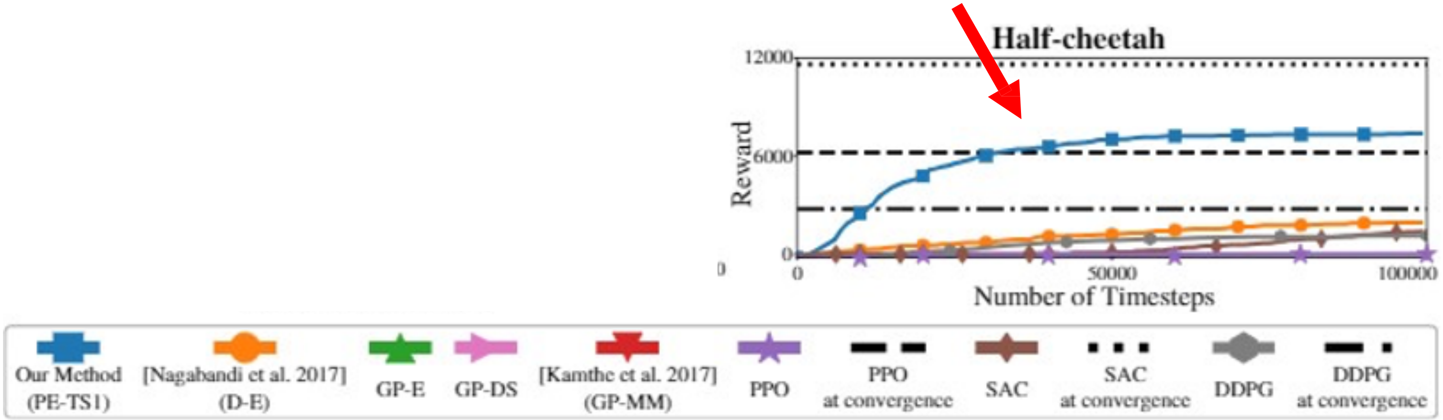
Example: model-based RL with ensembles

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

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



before



after