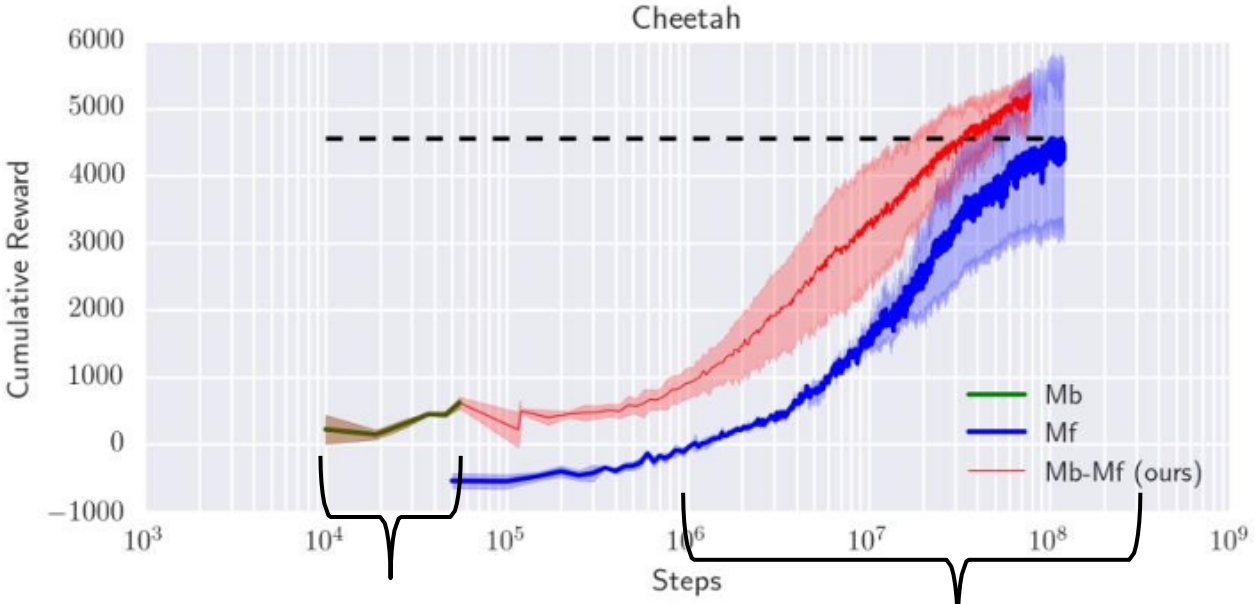
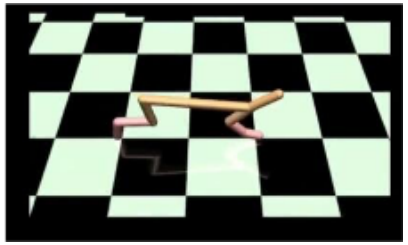
Uncertainty in Model-Based RL

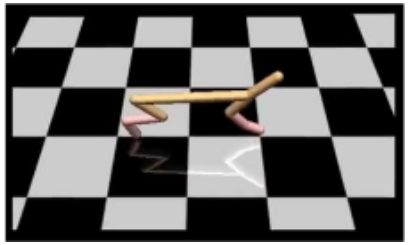
A performance gap in model-based RL



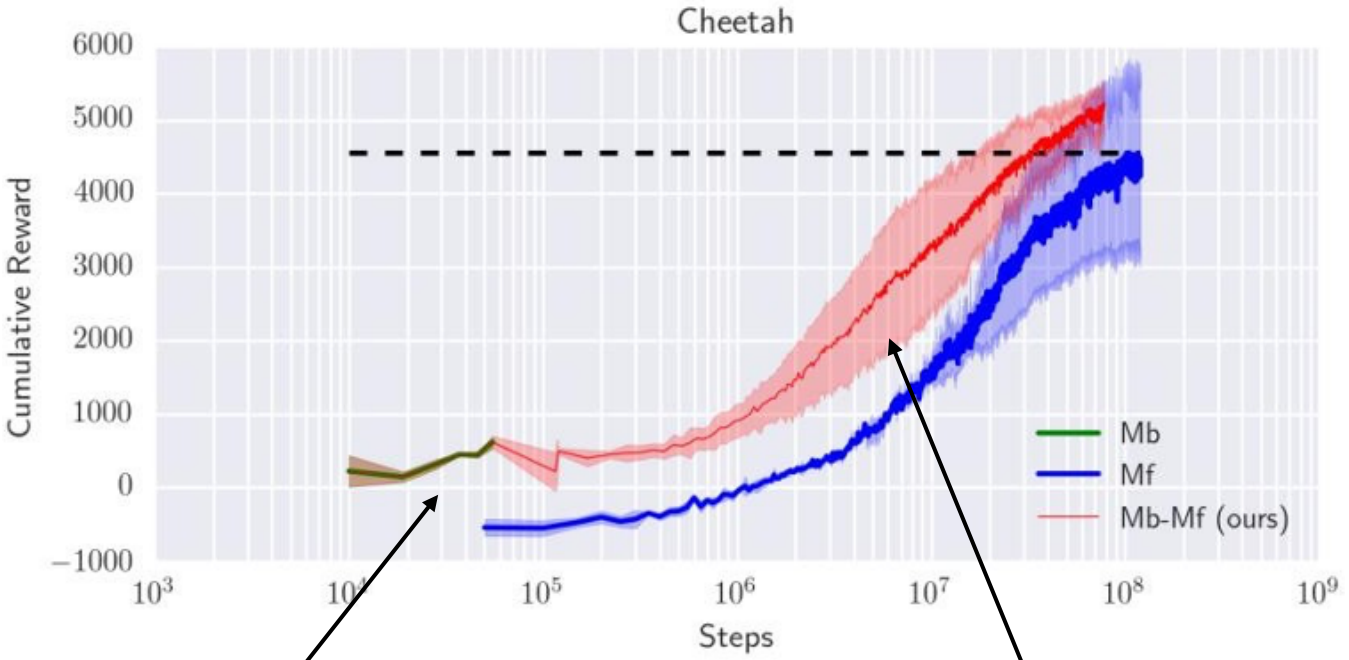
pure model-based (about 10 minutes real time)

model-free training (about 10 days…)





Why the performance gap?



need to not overfit here…

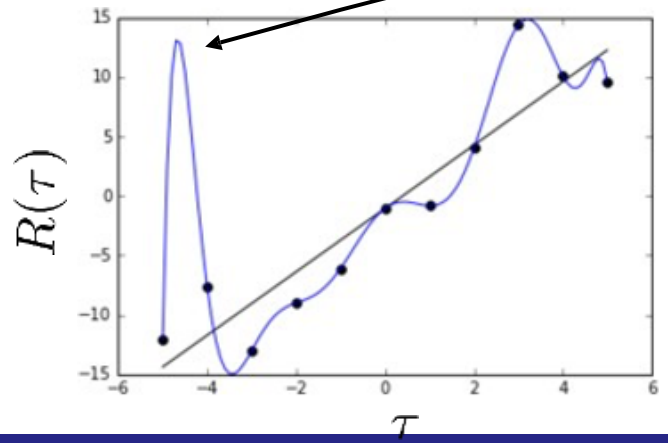
…but still have high capacity over here

Why the performance gap?

model-based reinforcement learning version 1.5:

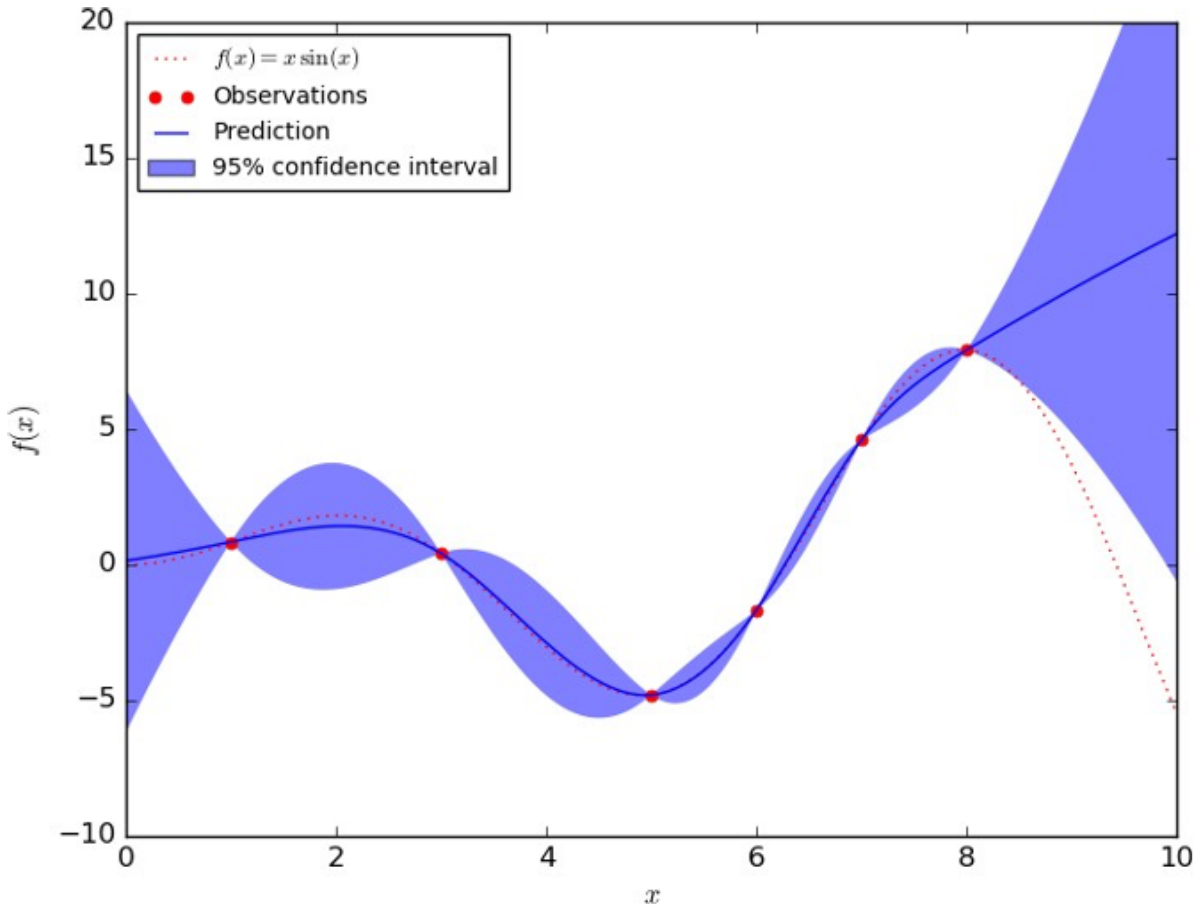
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s},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

every N steps

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

This avoids "exploiting" the model

The model will then adapt and get better

There are a few caveats…

Need to explore to get better

Expected value is not the same as pessimistic value

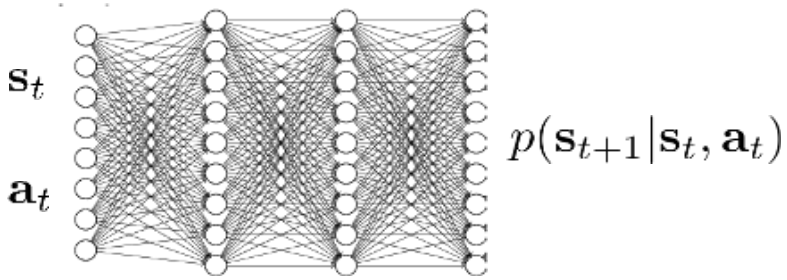
Expected value is not the same as optimistic value

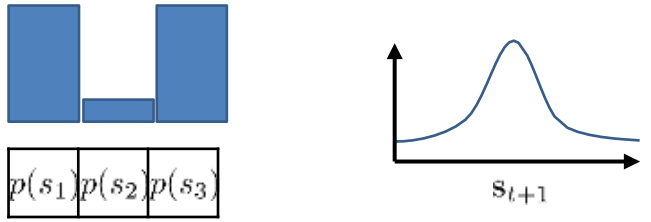
…but expected value is often a good start

Uncertainty-Aware Neural Net Models

How can we have uncertainty-aware models?

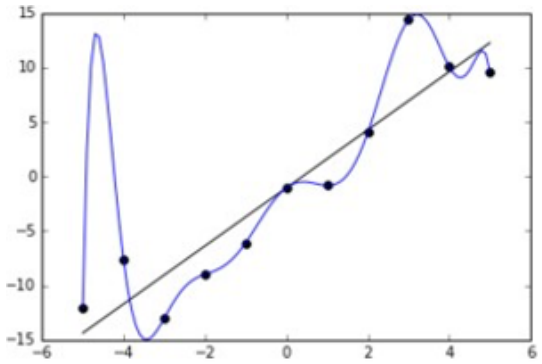
Idea 1: use output entropy





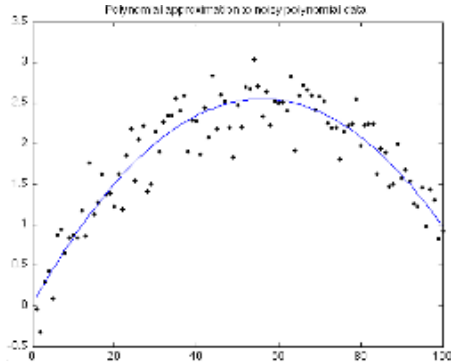
why is this not enough?

Two types of uncertainty:



aleatoric or *statistical* uncertainty

epistemic or *model* uncertainty



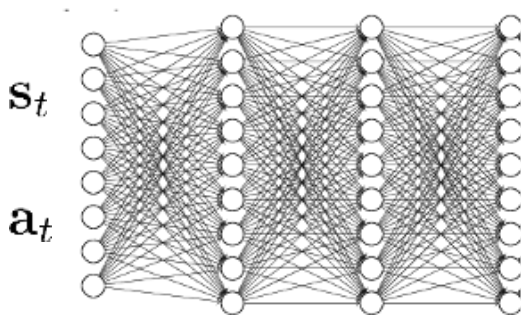
what is the variance here?

"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"



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

parameters θ

usually, we estimate

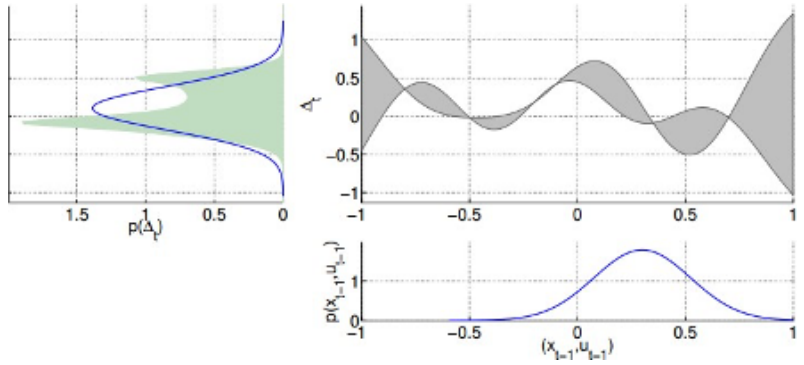
$$\arg\max_\theta \log p(\theta|\mathcal{D}) = \arg\max_\theta \log p(\mathcal{D}|\theta)$$

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

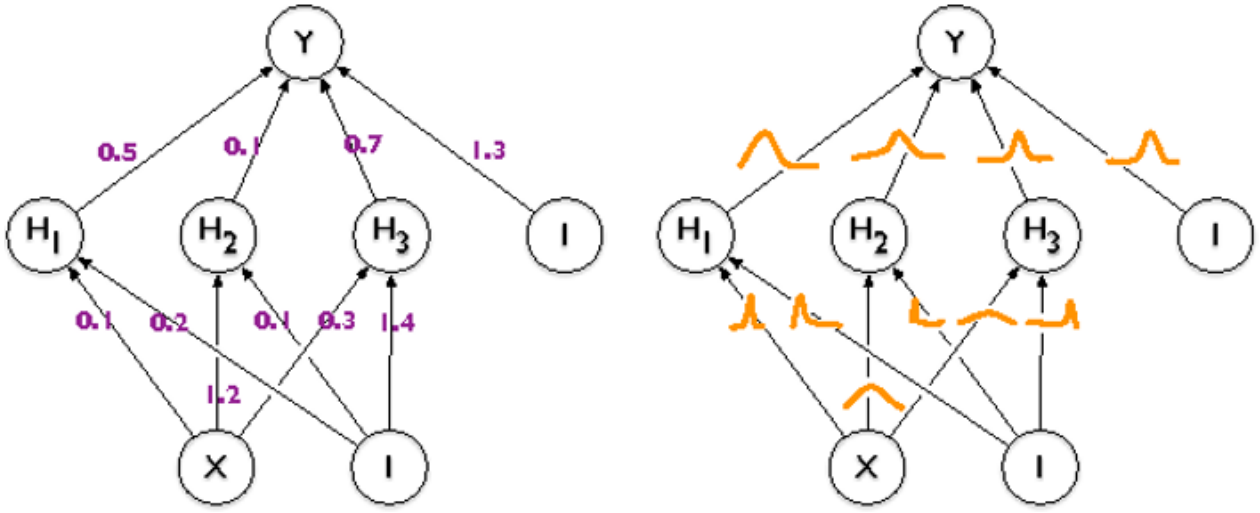
the entropy of this tells us the model uncertainty!

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:

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

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

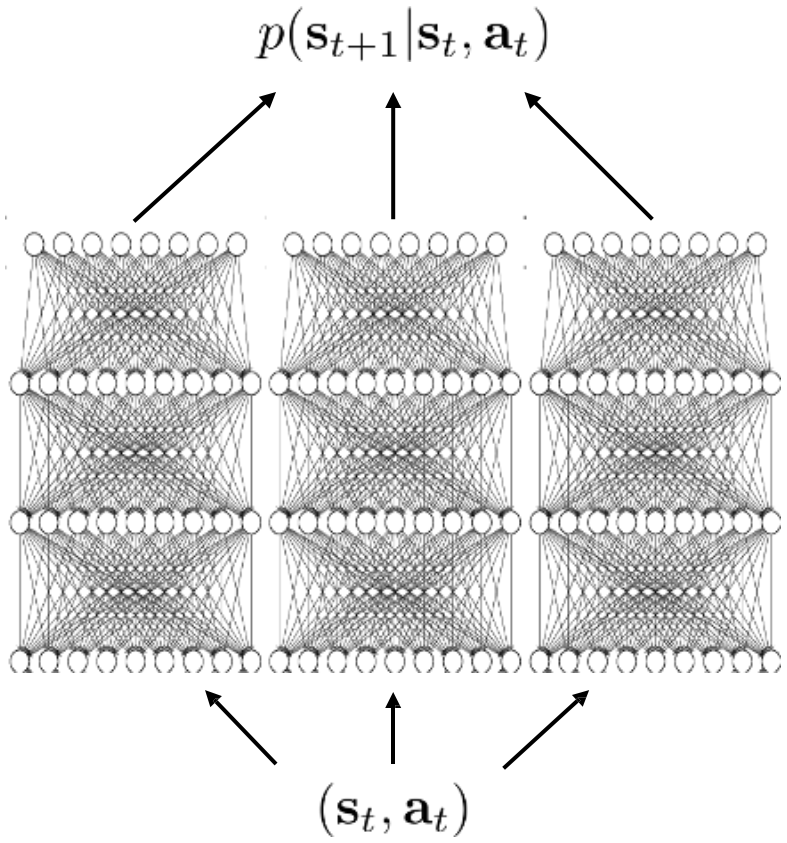
expected weight

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!

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

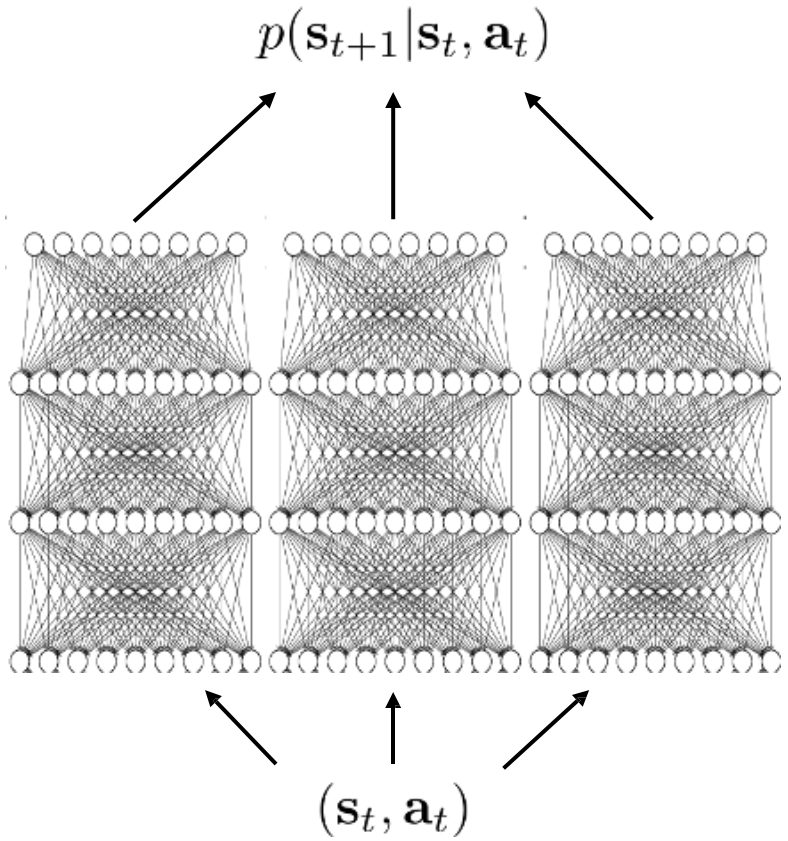
$$\int p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{N}\sum_i p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \theta_i)$$

How to train?

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

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

Bootstrap ensembles in deep learning



This basically works

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

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

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

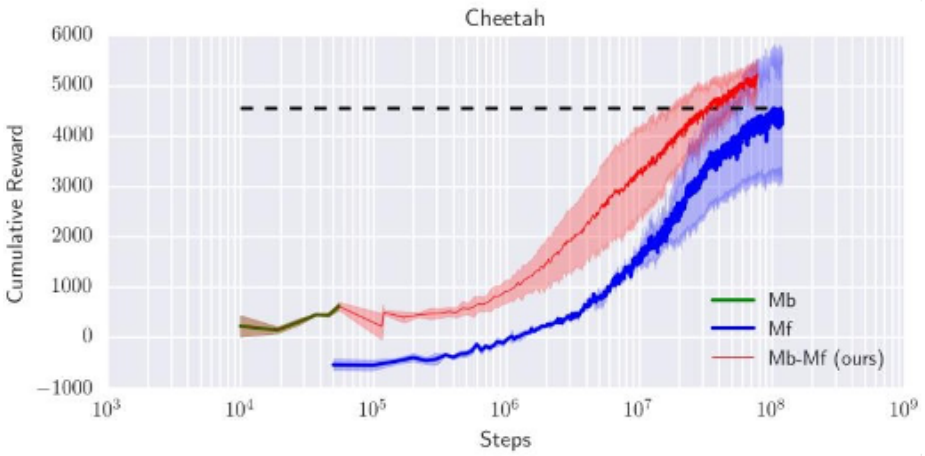
Step 4: repeat steps 1 to 3 and accumulate the average reward

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

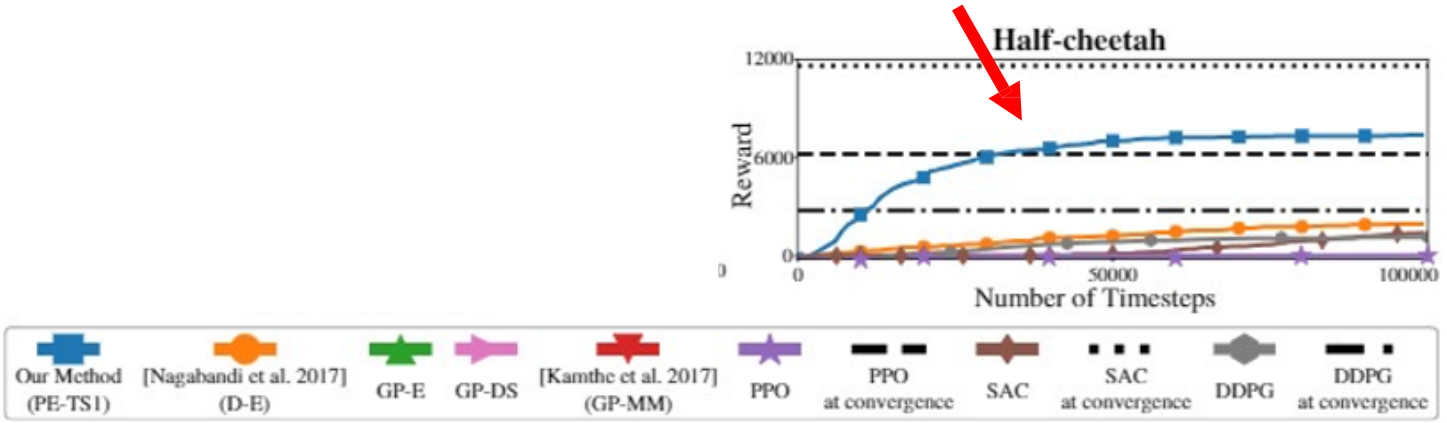
Example: model-based RL with ensembles

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)

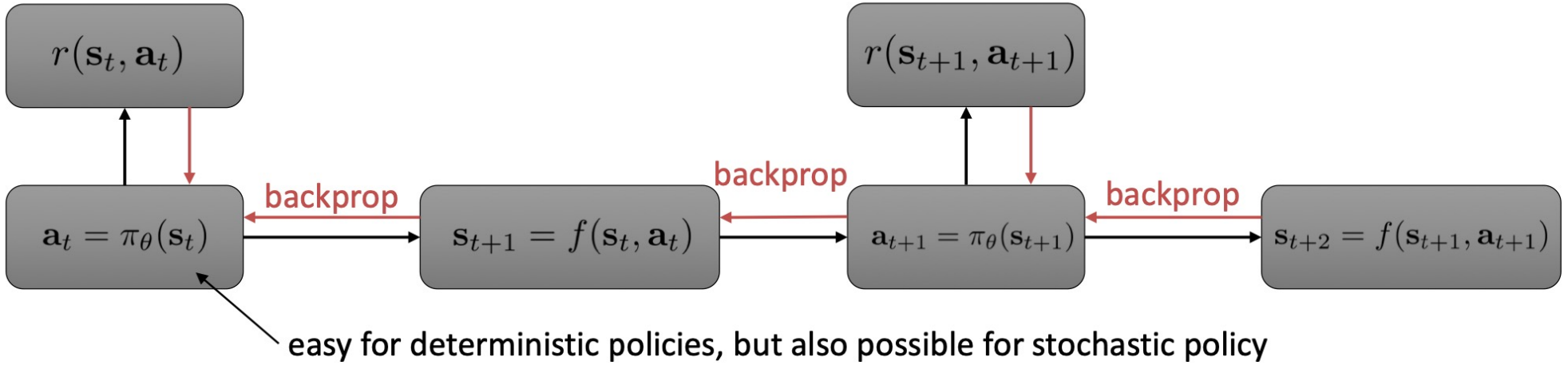


before



after

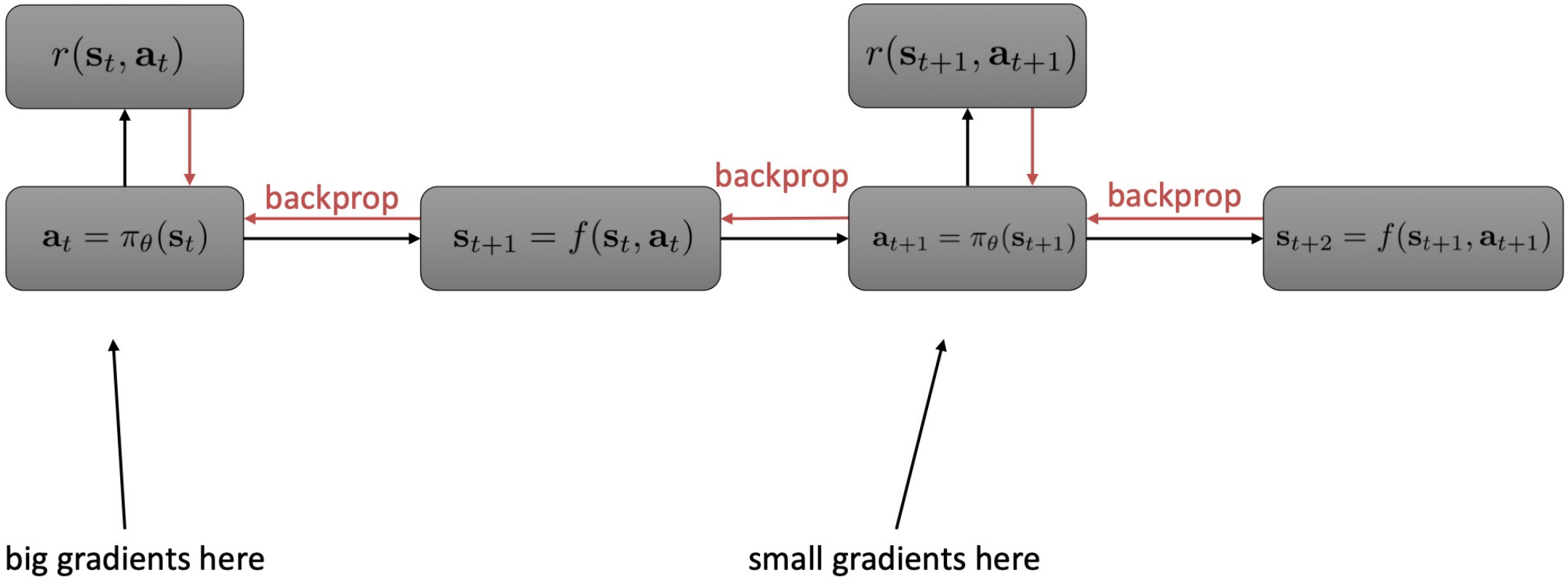
Backpropagate directly into the policy?



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?

- 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 problem with backprop into policy?

- 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

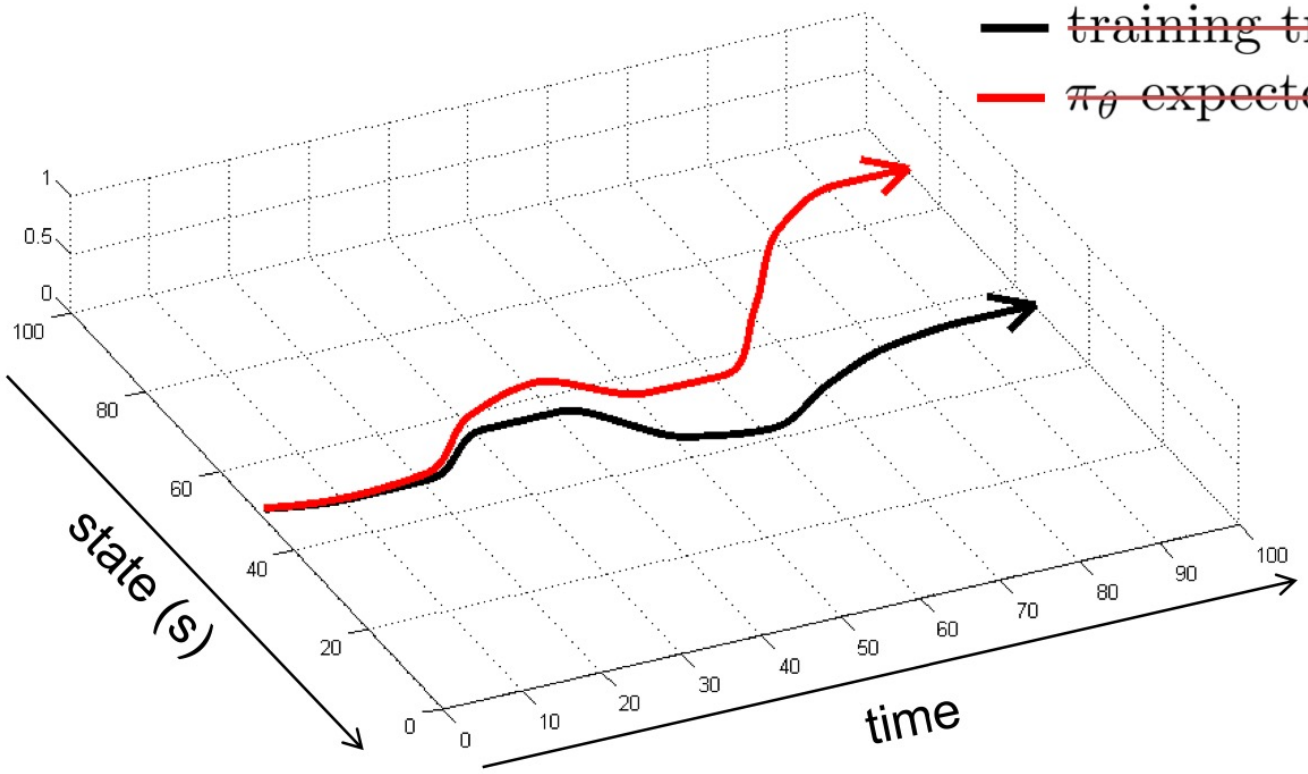
Model-based RL via policy gradient

model-based reinforcement learning version 2.5:

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. use $f(\mathbf{s}, \mathbf{a})$ to generate trajectories $\{\tau_i\}$ with policy $\pi_\theta(\mathbf{a}|\mathbf{s})$
4. use $\{\tau_i\}$ to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ via policy gradient
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

What's a potential **problem** with this approach?

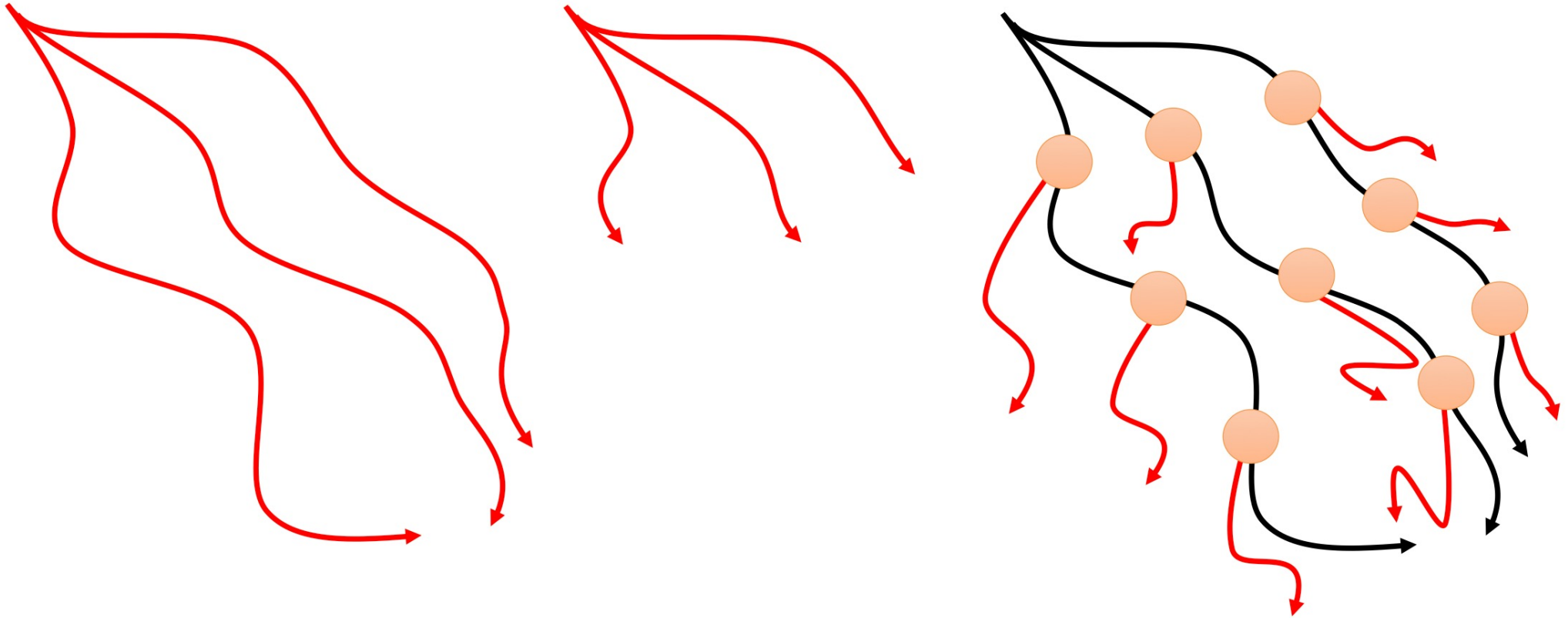
The curse of long model-based rollouts



How quickly does error accumulate?

$$\mathcal{O}(\epsilon T^2)$$

How to get away with accumulated errors?



- huge accumulating error

+ much lower error
- never see later time steps

+ much lower error
+ see all time steps
- wrong state distribution

Model-based RL with short rollouts

model-based reinforcement learning version 3.0:

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. pick states $\mathbf{s}_i$ from $\mathcal{D}$, use $f(\mathbf{s}, \mathbf{a})$ to make *short* rollouts from them
4. use *both* real and model data to improve $\pi_\theta(\mathbf{a}|\mathbf{s})$ with *off-policy RL*
5. run $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, appending the visited tuples $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to $\mathcal{D}$

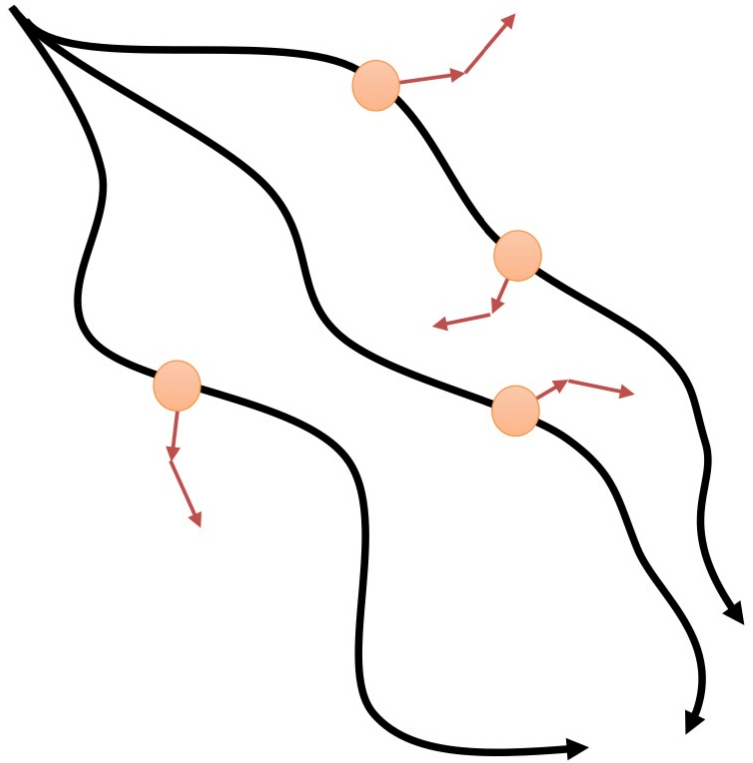
Dyna

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.

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 π, 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

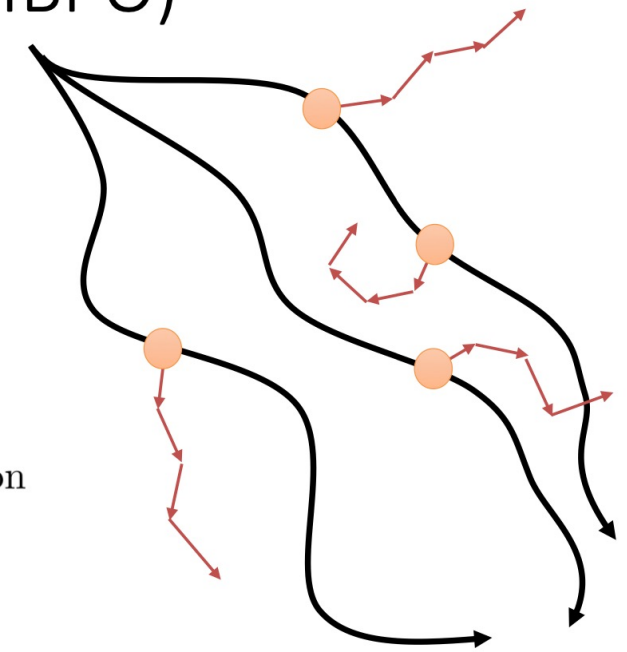
Instantiations

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