Carnegie Mellon

School of Computer Science

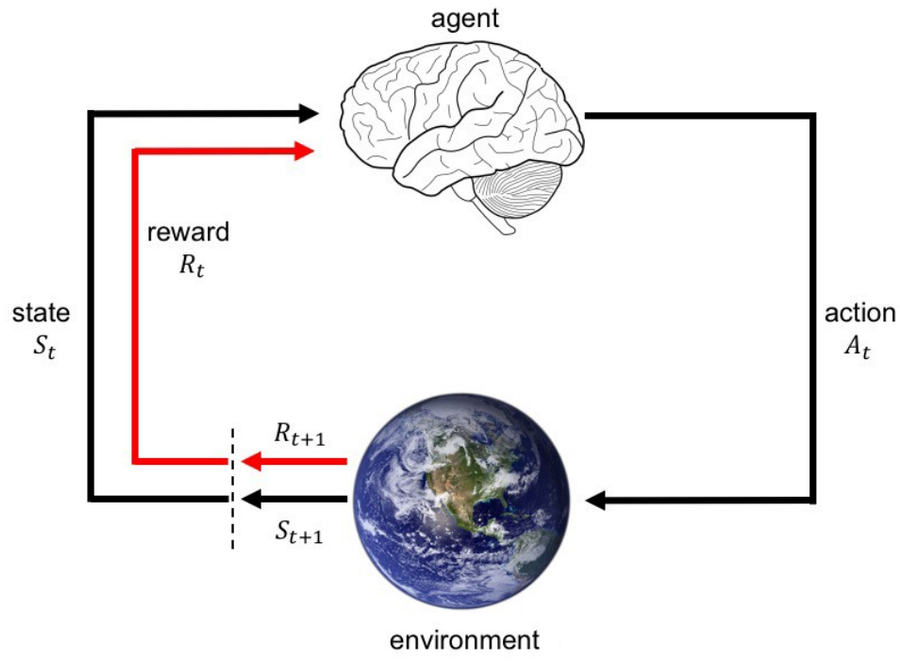
Deep Reinforcement Learning and Control

Model Based Reinforcement Learning

Spring 2021, CMU 10-403

Katerina Fragkiadaki

Model-free Reinforcement learning

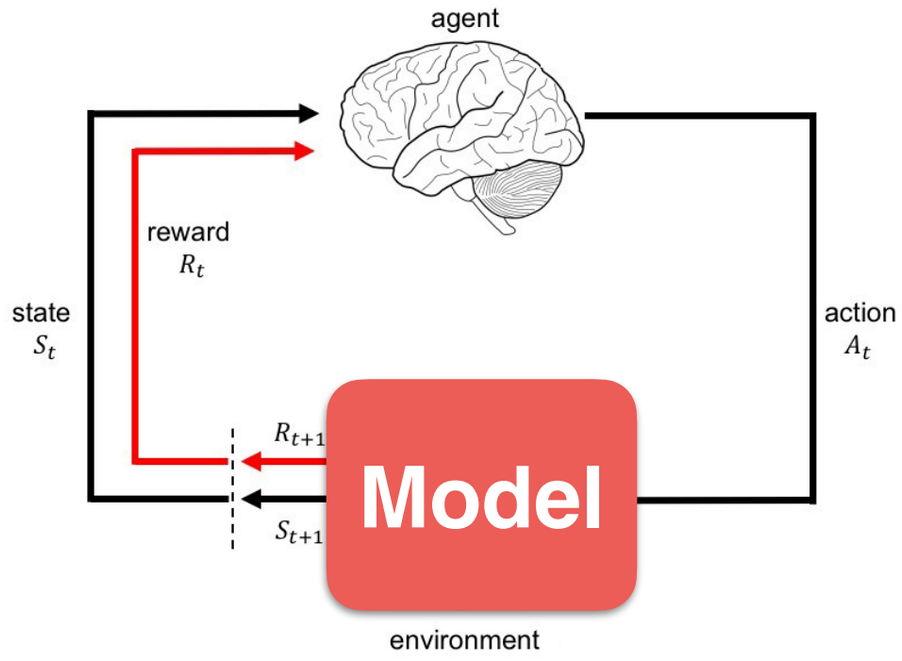


$$a_t \sim \pi_\theta(a_t \mid s_t)$$

$$p(\tau;\theta) = p(s_0)\left[\prod_{t=1}^{T} \underbrace{\begin{matrix} p(s_{t+1} \mid s_t, a_t) \\ p(r_{t+1} \mid s_t, a_t) \end{matrix}}_{\text{dynamics}} \cdot \underbrace{\pi_\theta(a_t \mid s_t)}_{\text{policy}}\right]$$

$$\max_\theta . \; U(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right]$$

The model: transition+reward function



$$a_t \sim \pi_\theta(a_t \mid s_t)$$

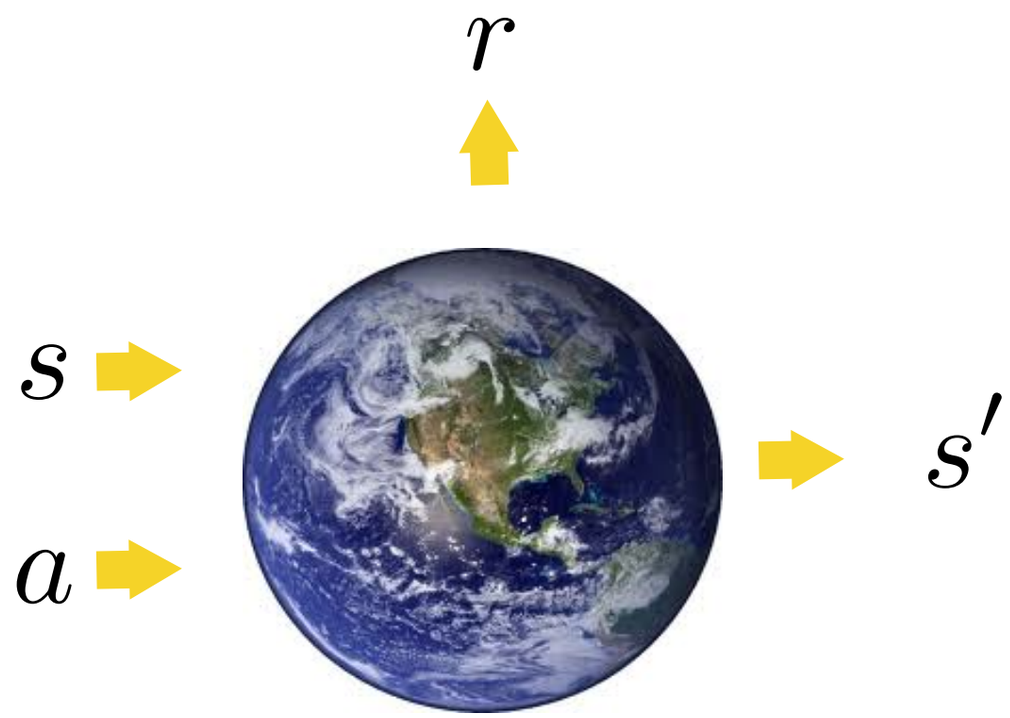
$$p(\tau;\theta) = p(s_0)\left[\prod_{t=1}^{T} \underbrace{p(s_{t+1} \mid s_t, a_t)}_{\text{dynamics}} \cdot \underbrace{\pi_\theta(a_t \mid s_t)}_{\text{policy}}\right]$$

$$p(s_{t+1} \mid s_t, a_t)$$
$$p(r_{t+1} \mid s_t, a_t)$$

$$\max_\theta . \; U(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right]$$

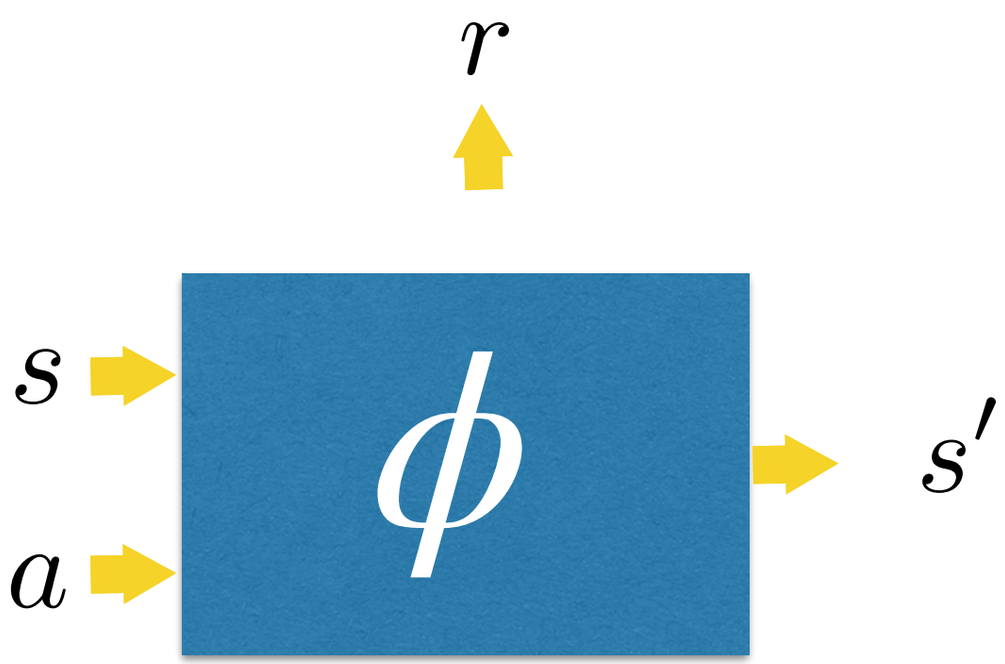
The model: transition+reward function

Anything the agent can use to predict how the environment will respond to its actions, concretely, the state transition function $T(s' \mid s, a)$ and reward function $R(s, a)$.



Model learning

Model-based reinforcement learning methods learn a model using supervised learning from experience tuples. Then use the model to select actions and learn policies.

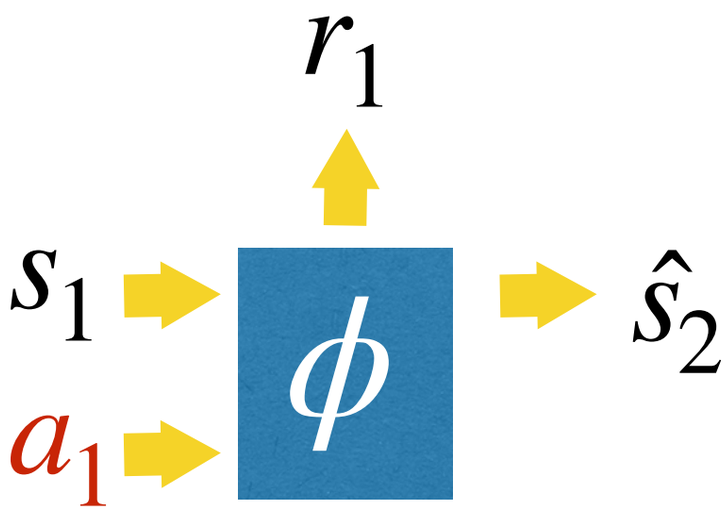


gaussian process,
random forest, deep
neural network,
linear function

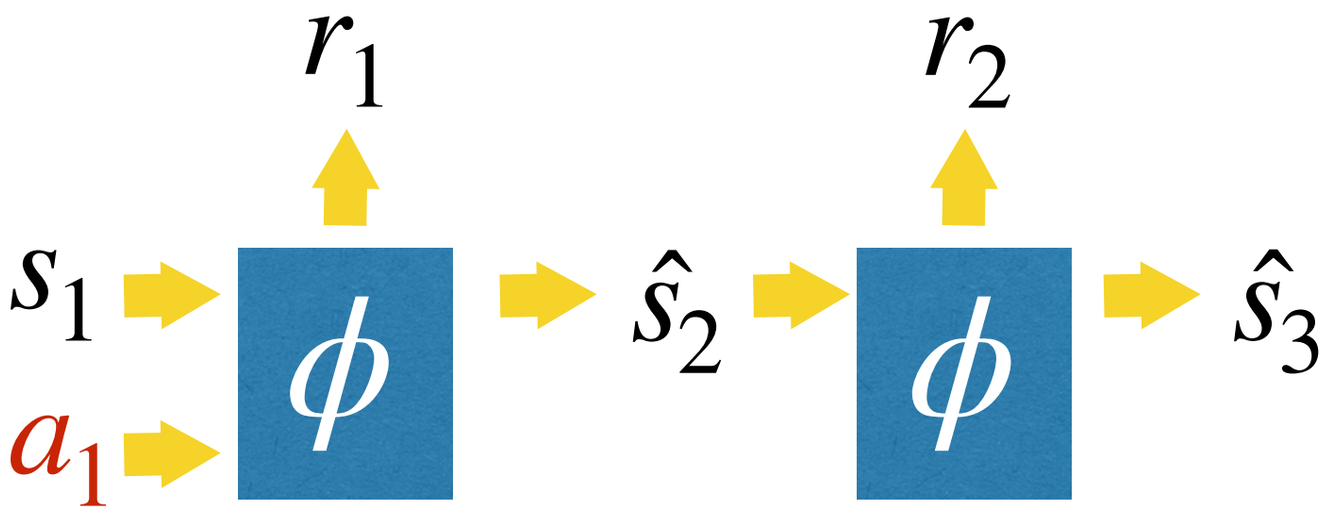
Model-based Reinforcement learning

- Assume we have a model. How can we use it for action selection and policy learning?
- Assume we do not have a model. How can we learn it and use it for action selection and policy learning?

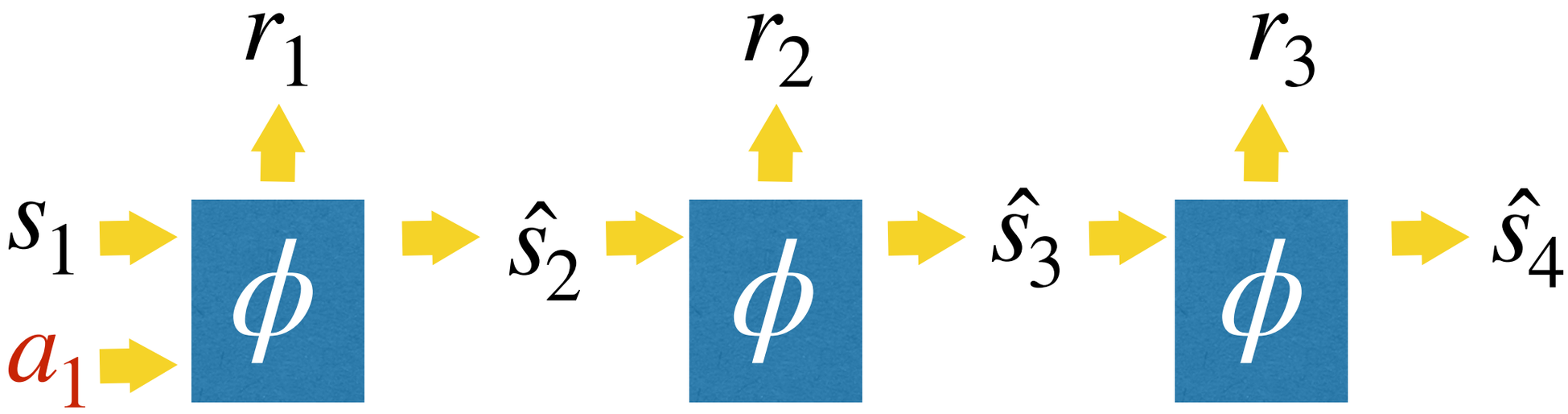
Model unrolling



Model unrolling



Model unrolling



Model-based control

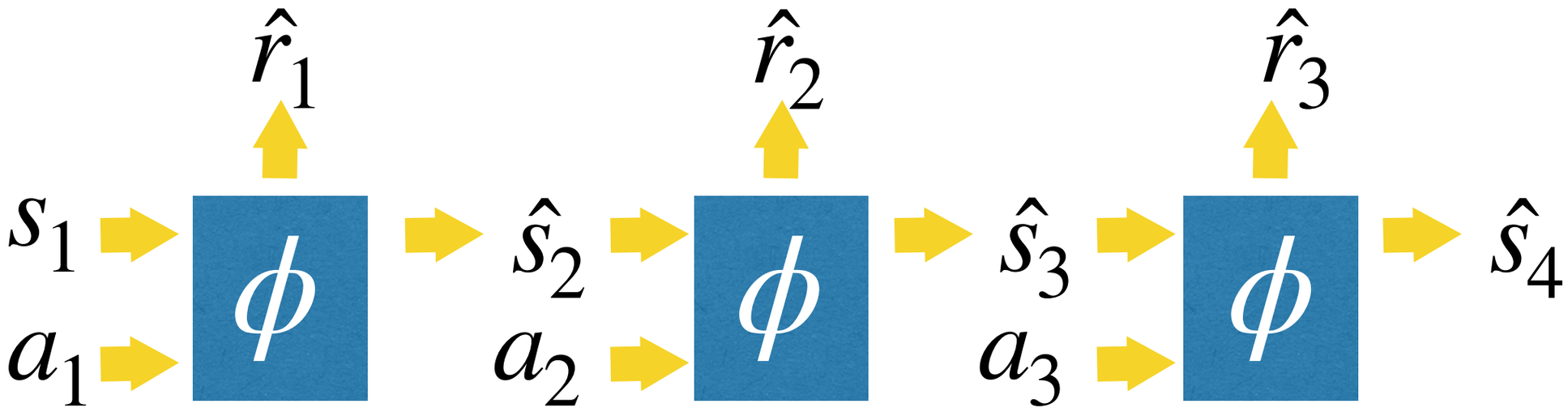
s_0 Given an initial state, estimate a sequence of actions to reach a desired goal or maximize sum of rewards by unrolling the model forward in time.

$$\min_{a_1 \cdots a_T} . \ \|s_T - s_*\|$$

$$\text{s.t. } . \forall t, \ s_{t+1} = f(s_t, a_t; \phi)$$

$$\max_{a_1 \cdots a_T} . \ \sum_{t=1}^{T} r_t$$

$$\text{s.t. } . \forall t, \ (s_{t+1}, r_{t+1}) = f(s_t, a_t; \phi)$$



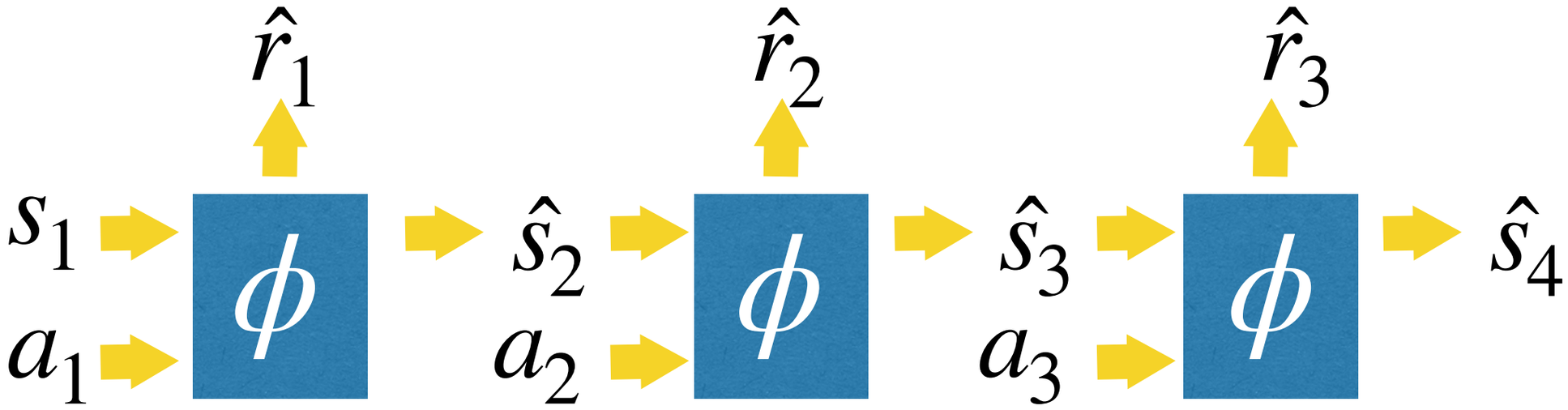
Model-based control

Given an initial state, estimate a sequence of actions to reach a desired goal or maximize sum of rewards by unrolling the model forward in time.

$$\min_{a_1 \cdots a_T} . \ \|s_T - s_*\|$$
$$\text{s.t. } . \forall t, \ s_{t+1} = f(s_t, a_t; \phi)$$

$$\max_{a_1 \cdots a_T} . \ \sum_{t=1}^{T} r_t$$
$$\text{s.t. } . \forall t, \ (s_{t+1}, r_{t+1}) = f(s_t, a_t; \phi)$$

If the dynamics are non-linear and the loss is not a quadratic, this optimization is difficult. We can use gradient descent optimization, evolutionary search, MCTS, etc..



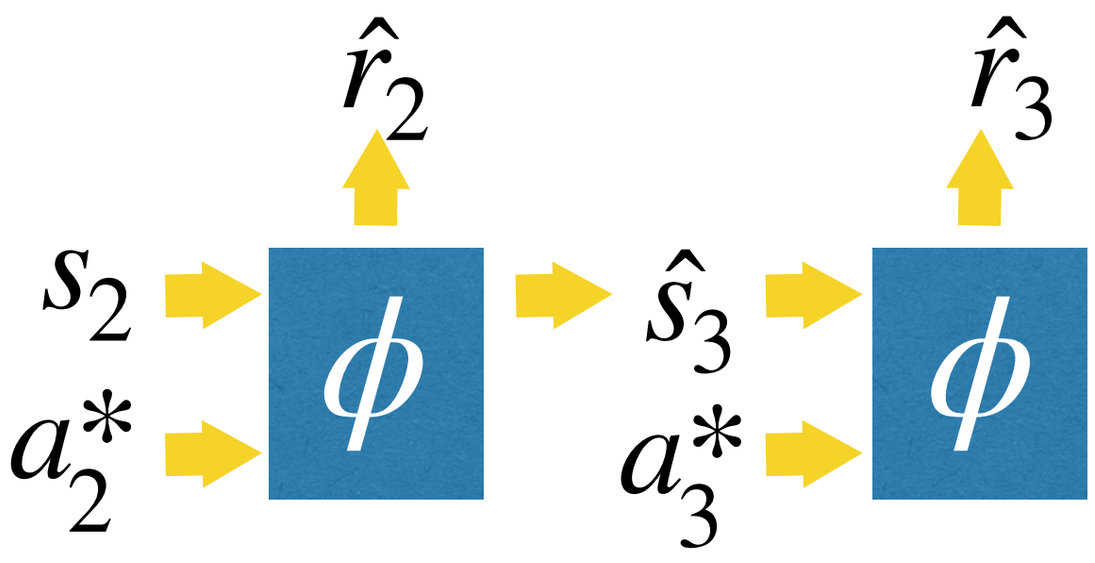
Model-predictive control

Execute the first action a_1^*, observe resulting state s_2, and re-optimize for 2…T.

Repeat.

Re-planning at each tilmestep help fight model error accumulation through unrolling.

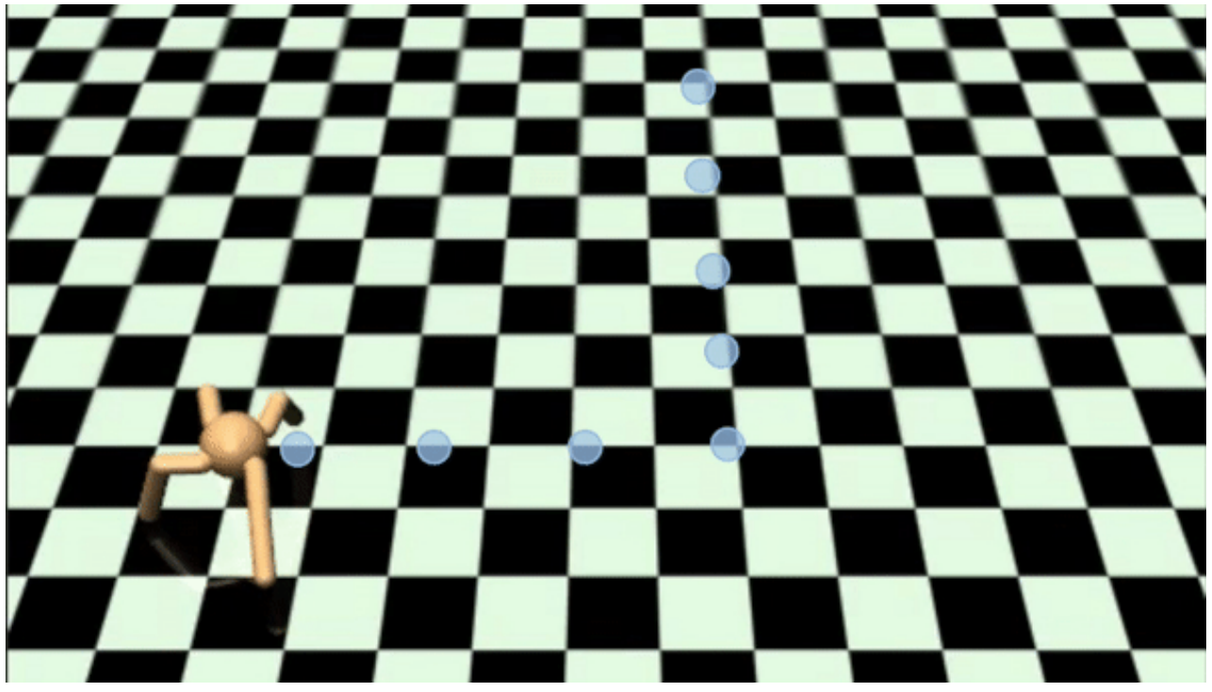
$$\max_{a_2 \cdots a_T} \sum_{t=2}^{T} \hat{r}_t$$

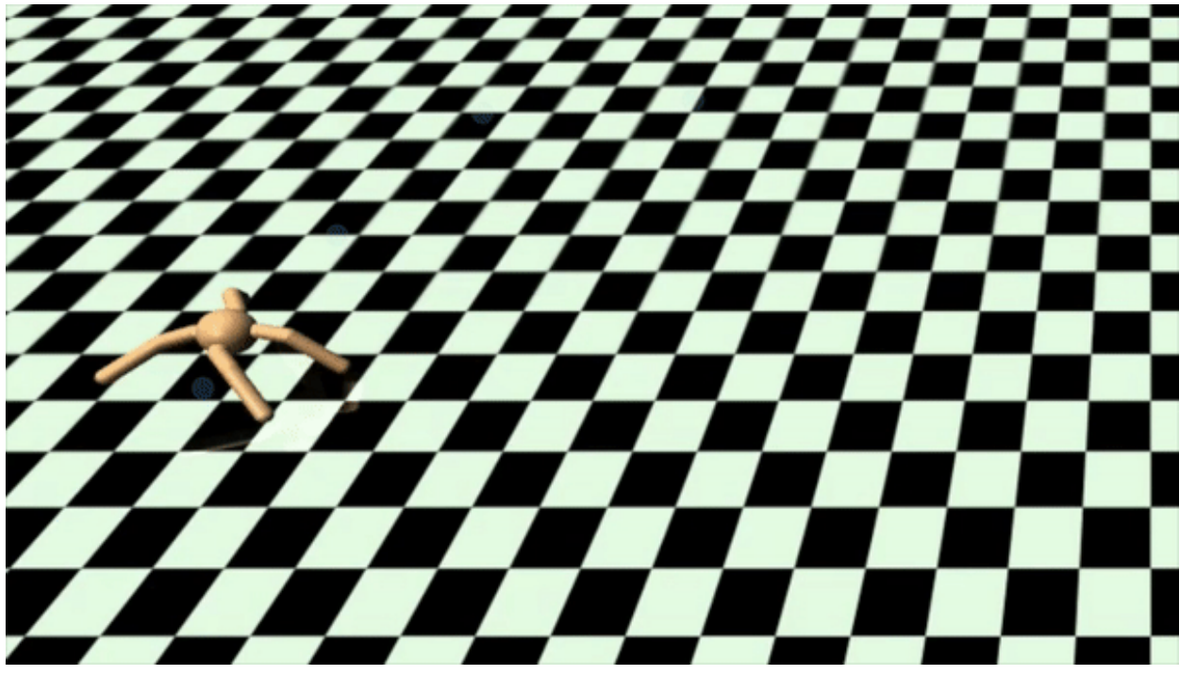


Benefits of Model-based Reinforcement learning

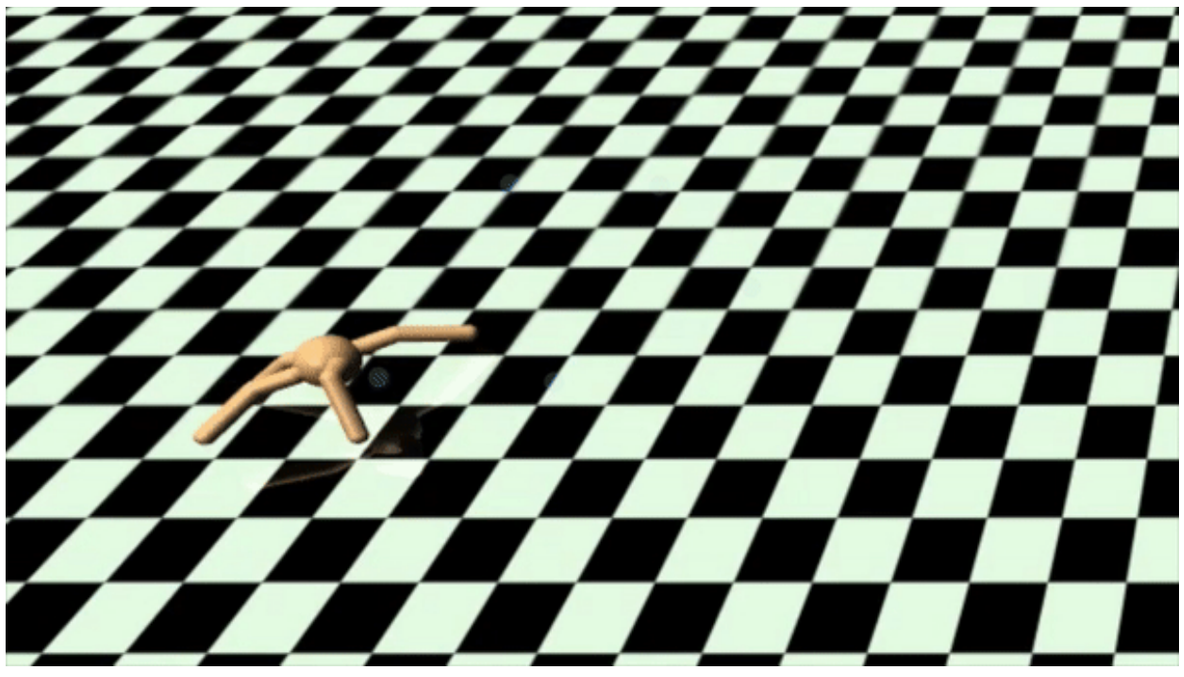
- Experience is not wasted. In model-free RL experience that does not yield any reward is not used.
- Models can support learning of multiple different tasks

Model-based RL





Training a model based controller allows to follow arbitrary trajectories at test time: the model allows you to optimize different reward function for different tasks, without any retraining.



Neural network dynamics for model-based Deep RI with model-free finetuning, *Nagabandi et al.*

Benefits of Model-based Reinforcement learning

- Experience is not wasted. In model-free RL experience that does not yield any reward is not used.
- Models can support learning of multiple different tasks
- The hope is that they can support much much more sample efficient behavior learning.

When models are learnt

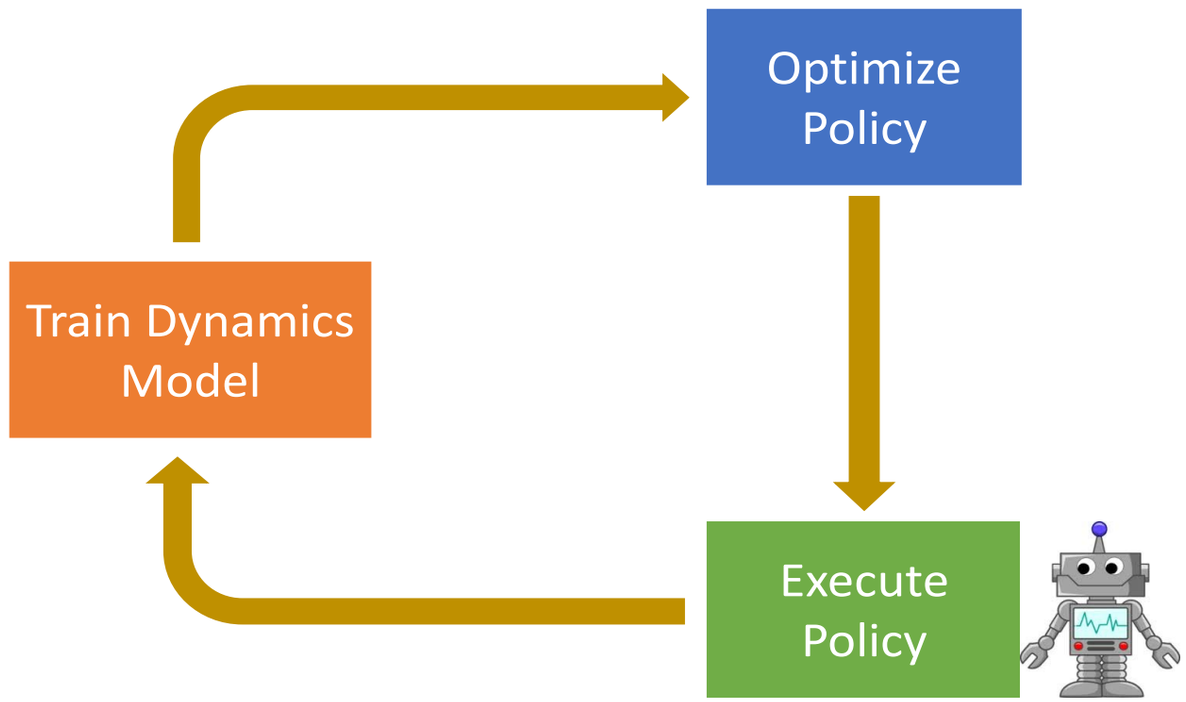
Alternating between model and policy learning

Initialize policy $\pi(s;\theta)$ and $D_{env} = \{\}$.

1. Run the policy and update experience tuples dataset D_{env}.
2. Train a dynamic model using D_{env}: $(s', r') = f(s, a; \phi)$
3. Update the policy by
    1. a model-free RL method on simulated experience D_{model} sampled from the model.
    2. Imitating a model-based controller (planner)
4. GOTO 1.

Why alternate?

Our model dynamics are only accurate close to the data collected by the policy. If we plan through the model or if we sample from it, we may land far from the data distribution of the policy. Deploying the suggested actions in the environment tries to bring the policy π_θ and the policy found by planning or sampling our model close to one another.



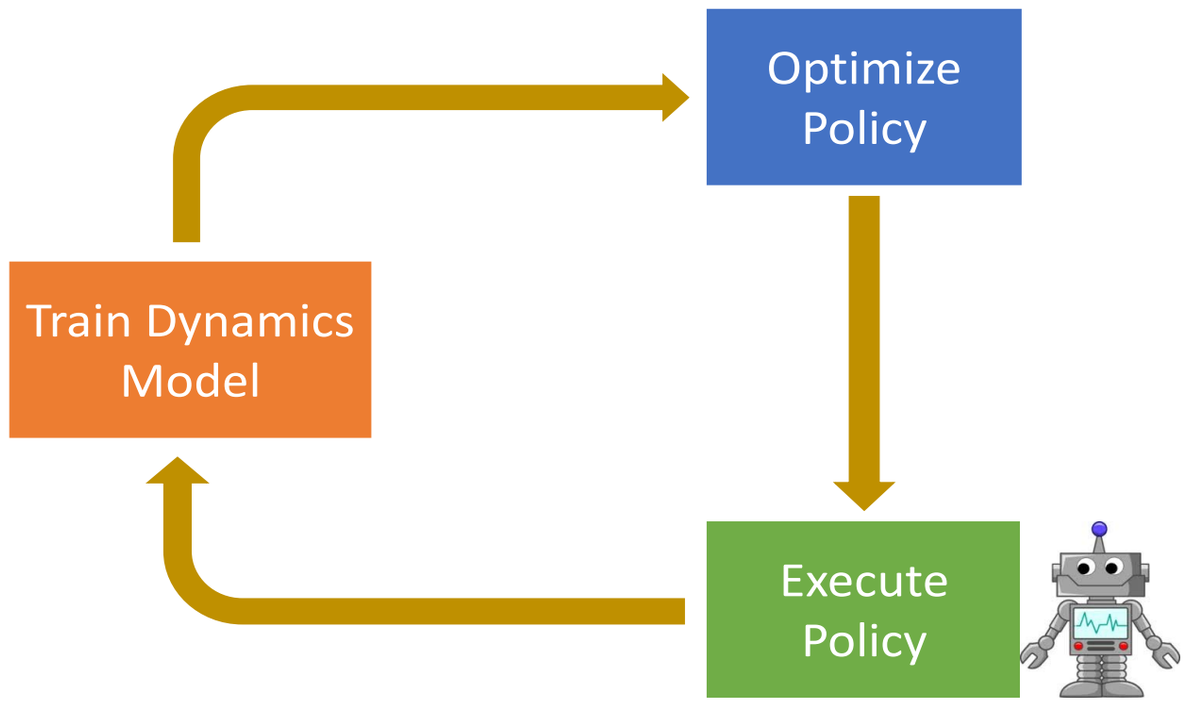
Alternating between model and policy learning

Initialize policy $\pi(s;\theta)$ and $D_{env} = \{\}$.

1. Run the policy and update experience tuples dataset D_{env}.
2. Train a dynamic model using D_{env}: $(s', r') = f(s, a; \phi)$
3. Update the policy by
    1. a model-free RL method on simulated experience D_{model} sampled from the model.
    2. Imitating a model-based controller (planner)
4. GOTO 1.

Why alternate?

Our model dynamics are only accurate close to the data collected by the policy. If we plan through the model or if we sample from it, we may land far from the data distribution of the policy. Deploying the suggested actions in the environment tries to bring the policy π_θ and the policy found by planning or sampling our model close to one another.



Challenges in model learning

- Under-modelling: If the model class is restricted (e.g., linear function or gaussian process) we have under-modeling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth. As a result, though we learn faster than model free in the beginning, MBRL ends up having worse asymptotic performance than model-free methods, that do not suffer from model bias.

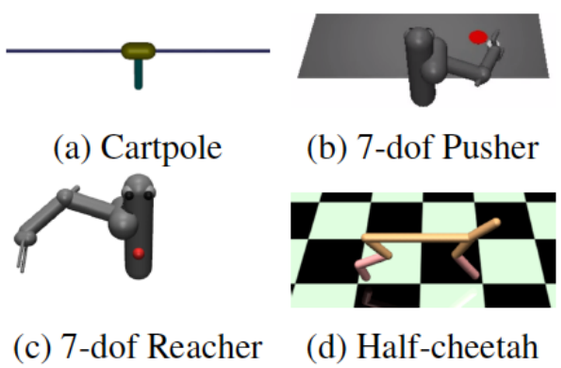
- Over-fitting: If the model class is very expressive (e.g., neural networks) the model will overfit, especially in the beginning of training, where we have very few samples. Action selection on top of model unrolling will surely exploit mistakes of the model.

- Errors compound through unrolling

- Need to capture different futures (stochasticity of the environment).

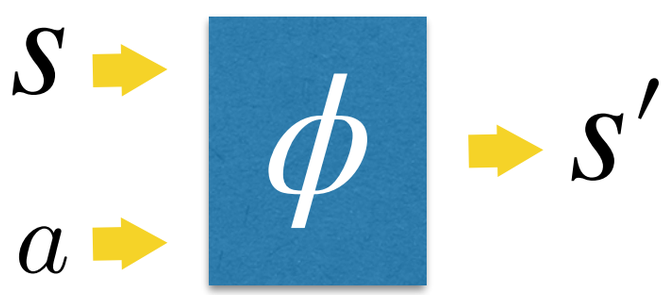
Model Learning

Where a low dimensional state is observed and given:



(a) Cartpole (b) 7-dof Pusher

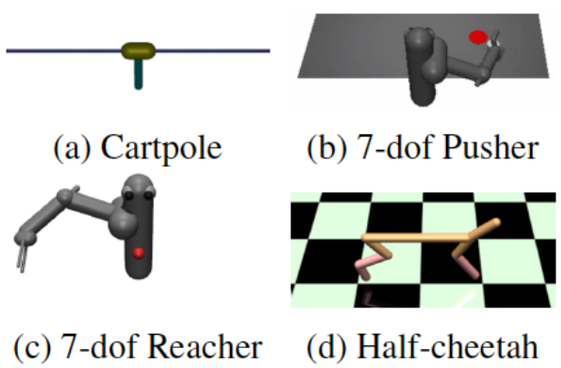
(c) 7-dof Reacher (d) Half-cheetah



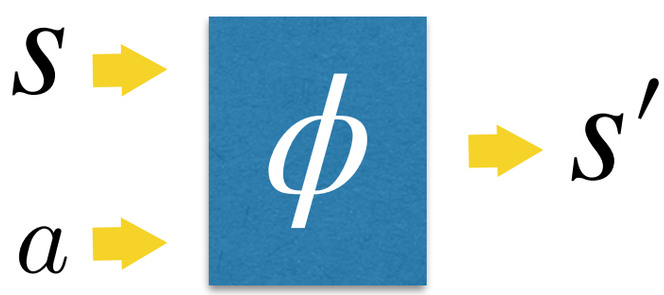
state can be 3D locations and 3D velocities of agent joints, actions can be torques

Model Learning

Where a low dimensional state is observed and given:

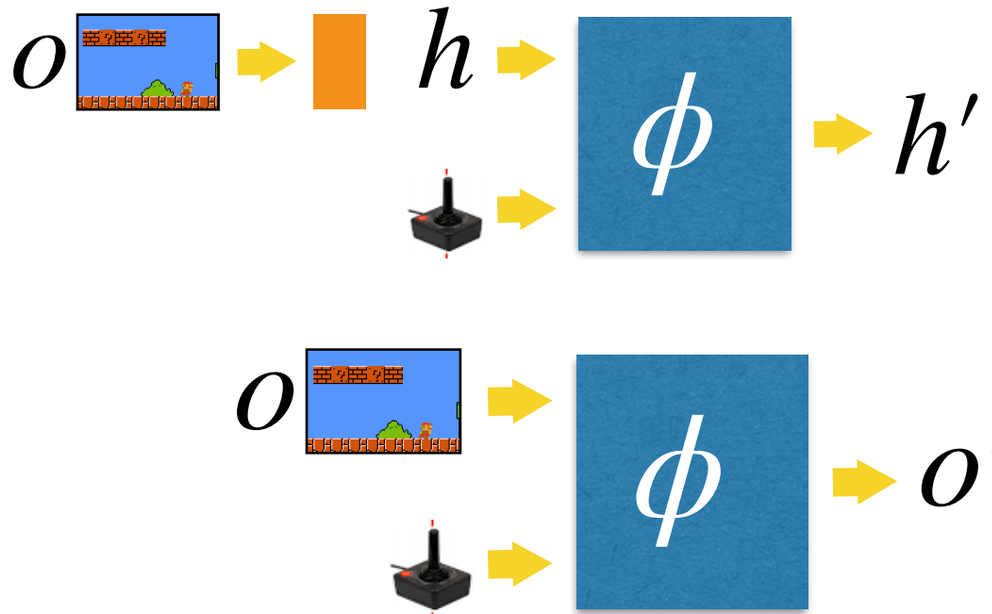


state can be 3D locations and 3D velocities of agent joints, actions can be torques



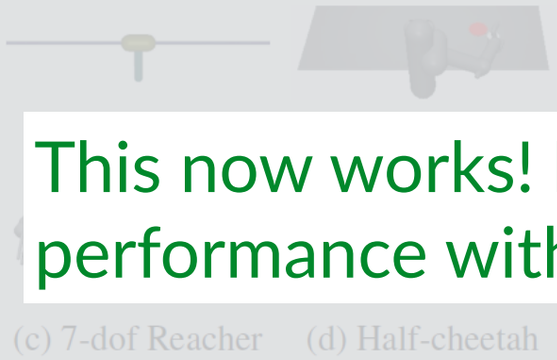
Where we only have access to (high dim) sensory input, e.g., images:

e.g., Atari game playing



Model Learning

- Where a low dimensional state is observed and given:



(c) 7-dof Reacher (d) Half-cheetah

state can be 3D locations and 3D velocities of agent joints, actions can be torques

This now works! It outperforms model-free RL methods: reaches same final performance with much fewer samples :-)

- Where we only have access to (high dim) sensory input, e.g., image or touch:

Still an open problem, overall models do not generalize under environment variations :-(

Model learning from sensory inputs is currently a central research problem.

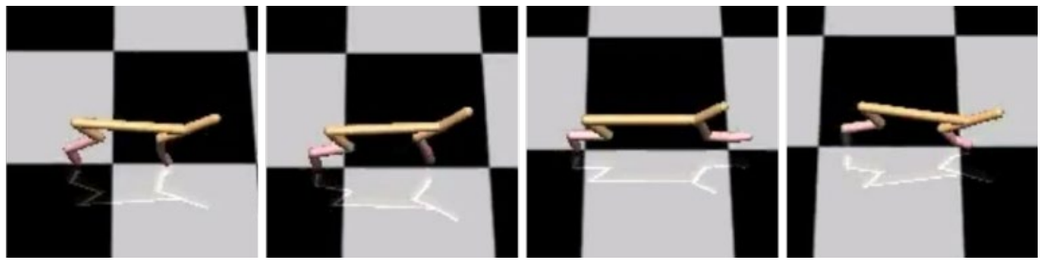
1.How can we learn models that are accurate and generalize across environments?

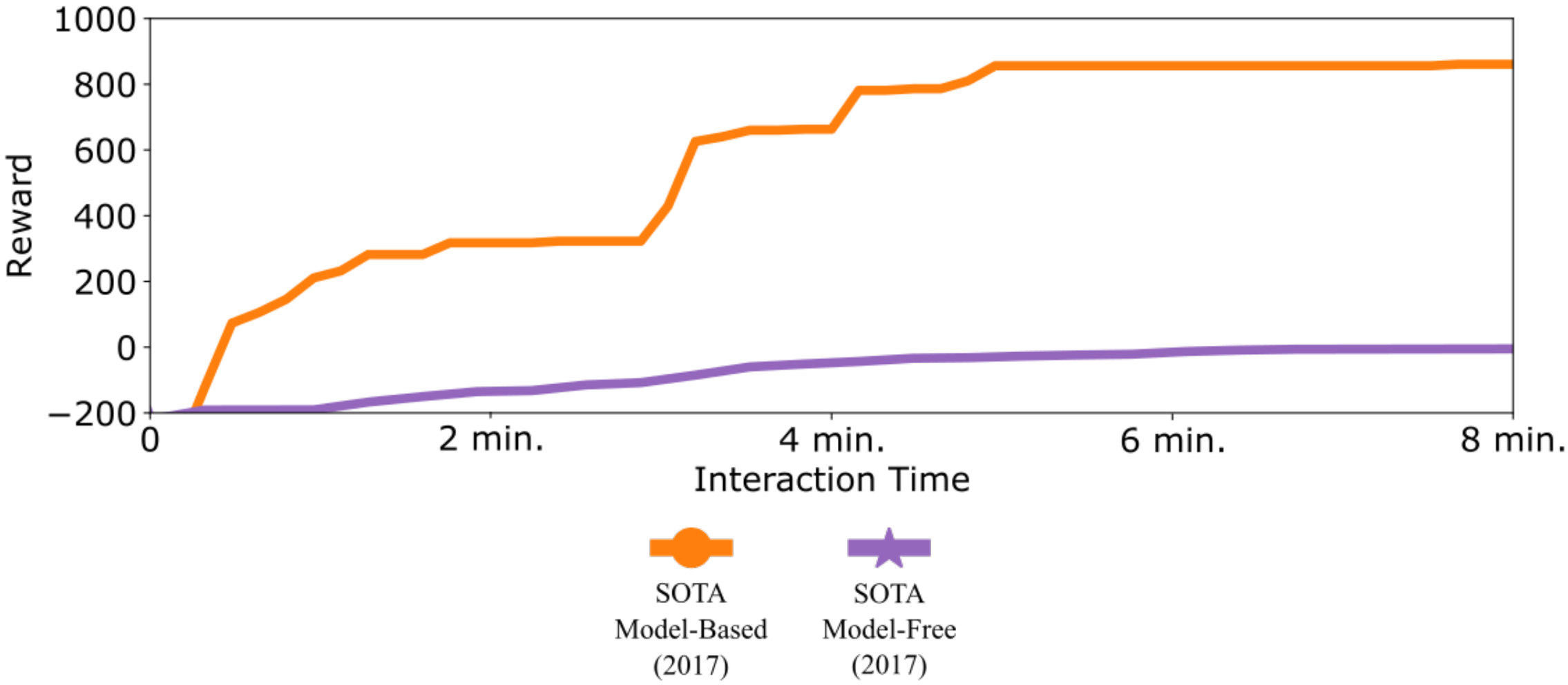
2.What are the right state representations?

3.How can we handle multimodality/ uncertainty?

Model-based RL in a low-dim state space

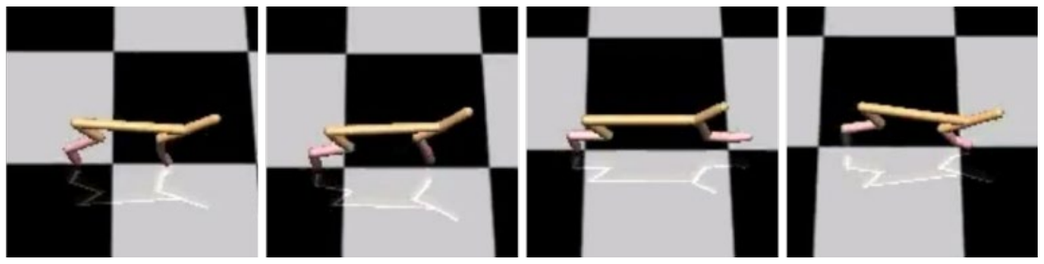
Comparative Performance on HalfCheetah

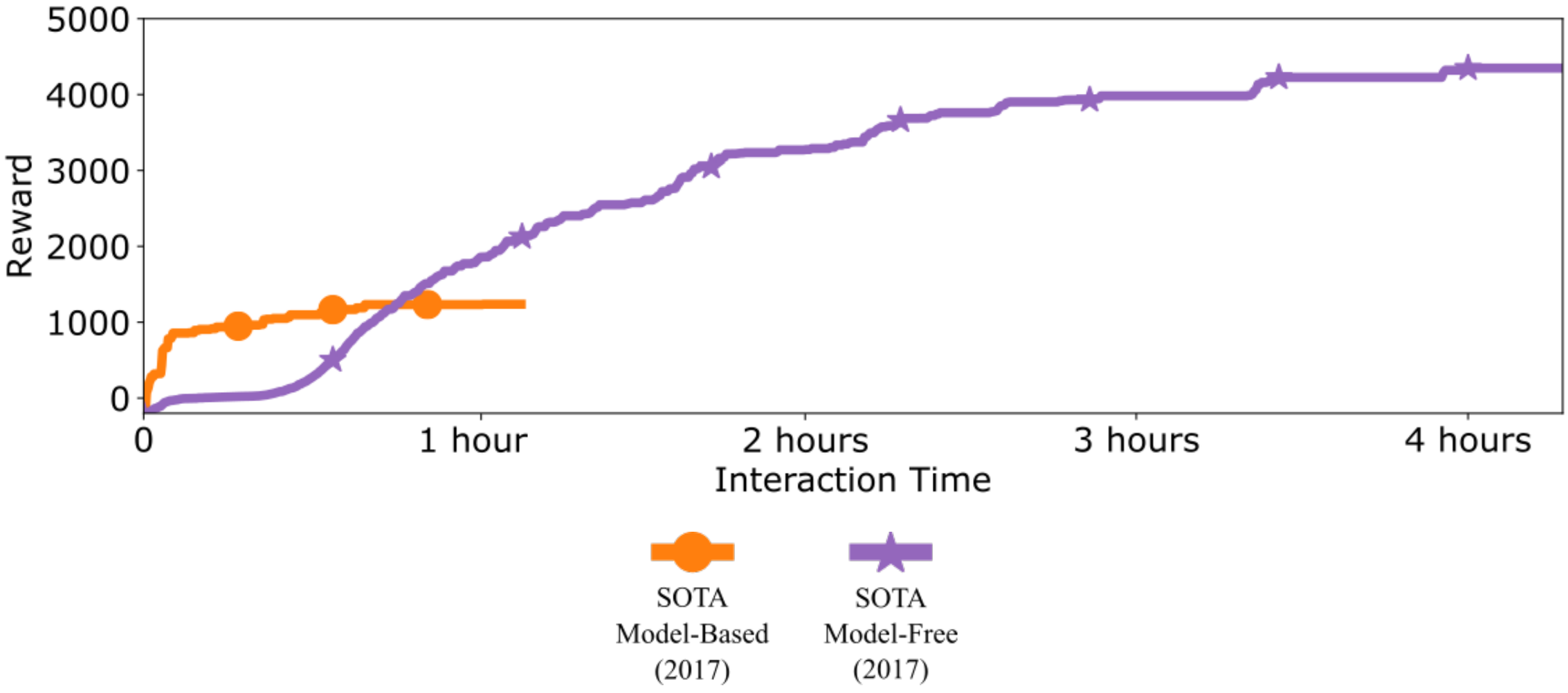




slide from Sergey Levine

Comparative Performance on HalfCheetah

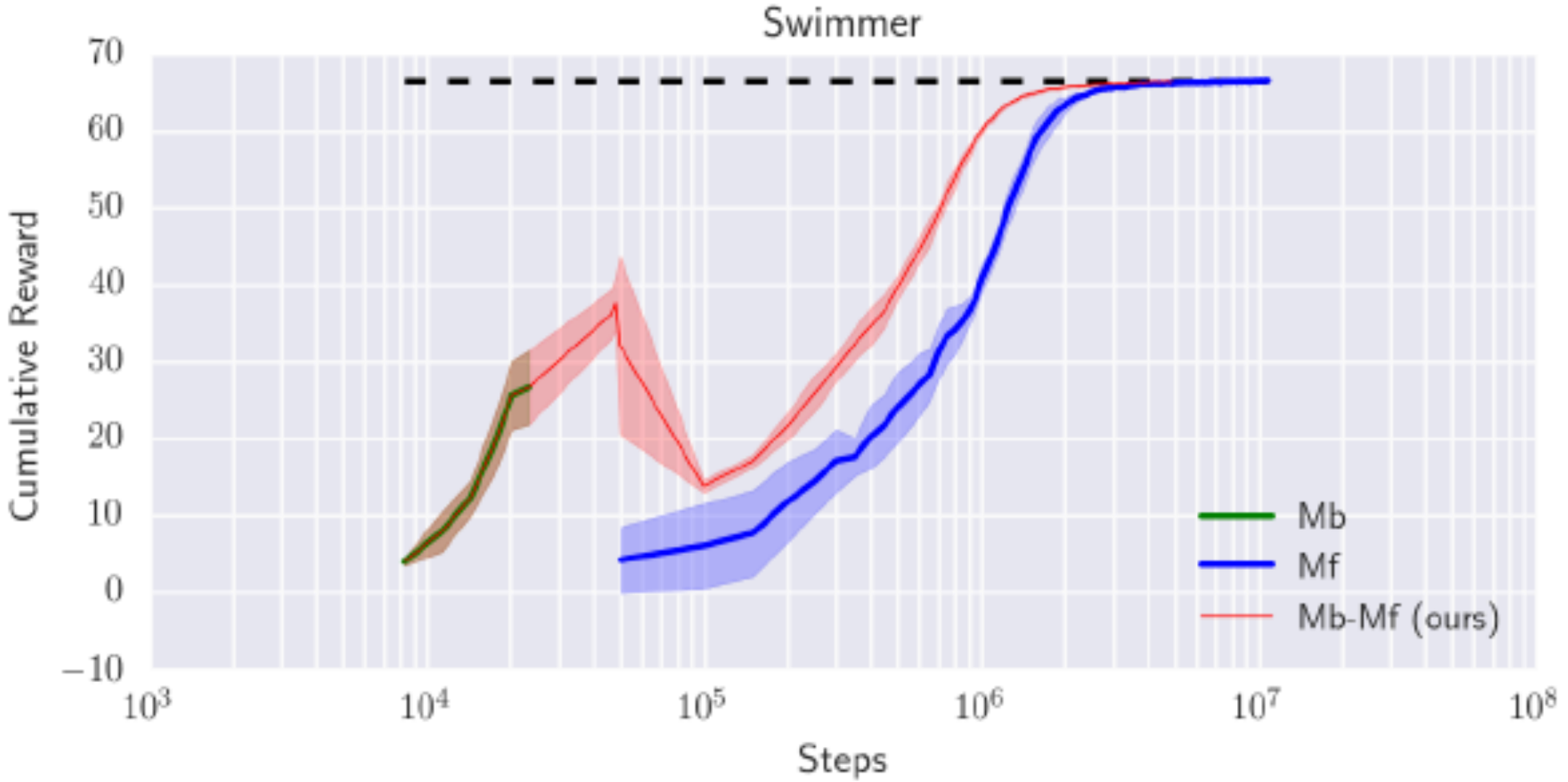




slide from Sergey Levine

Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, Sergey Levine
University of California, Berkeley



Neural network dynamics for model-based Deep RI with model-free finetuning, Nagabandi et al.

Model-based RL

Initialize D_{env} by acting randomly in the environment.

1. Train a dynamic model f using D_{env}: $(s', r') = f(s, a; \phi)$
2. Use model predictive control over f to estimate optimal actions from s_0.
3. Deploy the optimal actions in the environment and update D_{env}.
4. GOTO 1.

Note: we usually train transition dynamics to predict the state change:
$s' = s + f(s, a; \phi)$

How can I surpass the upper bound imposed by the accuracy of my model?

- Initialize a policy by imitating the MPC planner using DAGGER
- Finetune the policy using any model-free method, e.g., TRPO.

Neural network dynamics for model-based Deep RI with model-free finetuning, *Nagabandi et al.*

Can we skip the model-free finetuning step and still outperform model-free methods?

How can models help us learn to act?

- **Sample experience from the *learnt* model**. Then use a model-free method to learn policies at training time. At test time use only the learnt policy. (*MBPO[1], Model-ensemble TRPO[2], Model-based RL for Atari[10]*).
- **Sample experience from the known model**. Then use a model-free method to learn policies at training time. At test time use only the learnt policy. (All *model-free RL*).
- **Plan through the model at train time only**: Unroll the model forward in time and optimize for action sequences to maximize rewards at training time. Then, learn policies that predict the actions found by model unrolling. At test time use only the learnt policy.
  - Differentiable optimization: back propagate the error through the dynamics function over time.(Linear quadratic regulator(LQR)[3]: linear models, dream-to-control[4]: non-linear models)
  - Evolutionary search for actions, e.g., CMA-ES. (PETS[5])
  - Monte Carlo tree search: combination of learnt policies and model unrolling. (*Playing Atari with offline MCTS[6]*)
- **Plan through the model at both train and test time**:Unroll the model forward in time and optimize for action sequences to maximize rewards at training time. Then, learn policies that predict the actions found by model unrolling. At test time, use both model unrolling and the learnt policy. (*AlphaGoZero[7]*, *MuZero[8]*)
- **Plan through the model**:Use model rollouts to supply better targets to Q functions (Stochastic Ensemble Value Expansion[9]). Learn a policy that maximizes the Q function, and use it at test time.

[1]When to Trust Your Model-Model-Based Policy Optimization (MBPO)
[2]Model-ensemble trust region policy optimization
[4]Dream-to-control:learning behaviors via latent imagination
[5]Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models (PETS)
[6]Deep Learning for Real-Time Atari Game Play Using Offline MCTS
[7]Mastering the game of Go without human knowledge(AlphaGoZero)
[8]Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model (MuZero)
[9]Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion
[10]Model-baser RL for Atari, Kaiser et al.

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

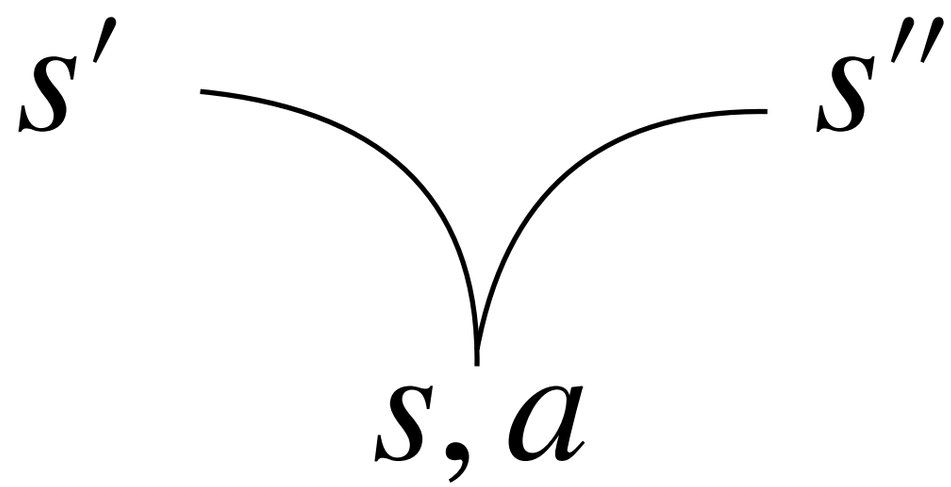
Kurtland Chua **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu

It's all about representing model uncertainty. Two types of uncertainty:

1. Epistemic uncertainty: uncertainty due to lack of data (that would permit to uniquely determine the underlying system)
2. Aleatoric uncertainty: uncertainty due to inherent stochasticity of the system

Aleatoric uncertainty in model learning

The environment can be stochastic

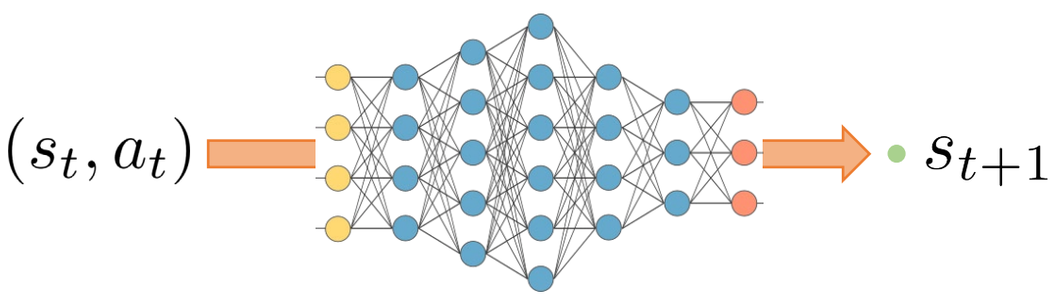


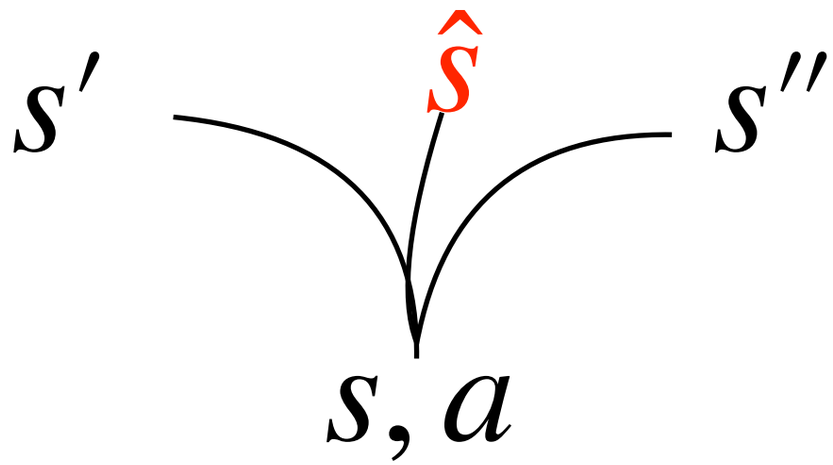
- This means our state does not capture enough information to help us delineate the possible future outcomes.
- What is stochastic under one state representation, may not be stochastic under another. Is this true? Could we ever predict exactly what we will see in the TV when we switch the channel?
- We will always have part of the information hidden, so stochasticity will always be there

Aleatoric uncertainty in model learning

If the environment is stochastic, regression fails.

$$\mathcal{L}_\phi = \sum_{i=1}^{N} \|f(s_i, a_i; \phi) - s_i'\|$$



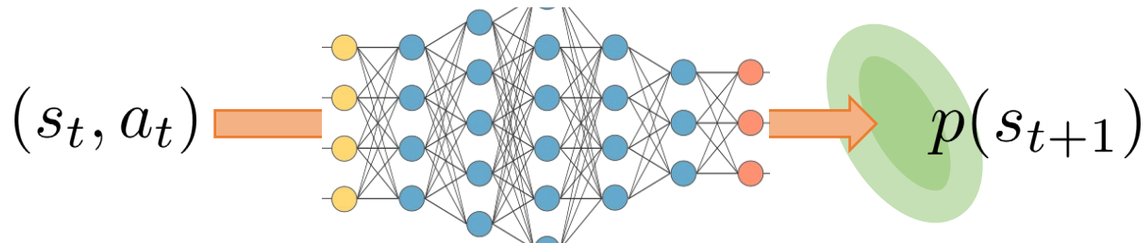


Failing means: not only we cannot capture the distribution, but we output a solution that does not agree with any of its modes!

Aleatoric uncertainty in model learning

- Our model will output a Gaussian distribution over next states s' given current state and action.
- A NN will predicts the mean and the elements of the covariance matrix. (We have seen this before)

$$p_\phi(s'|s,a) = \frac{\exp\left(-\frac{1}{2}(s'-\mu(s,a;\phi)^\top(\mathbf{\Sigma}(s,a;\phi))^{-1}(s'-\mu(s,a;\phi)\right)}{\sqrt{(2\pi)^d \det\mathbf{\Sigma}(s,a;\phi)}}$$



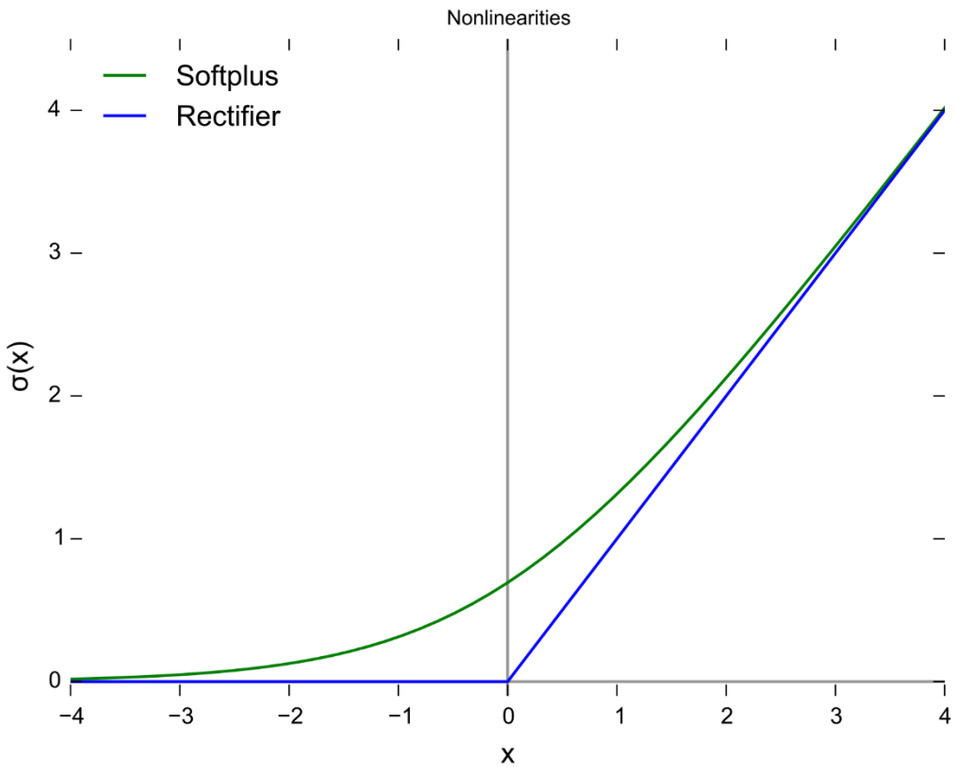
$$\mathscr{L}_\phi = -\frac{1}{N}\sum_{i=1}^{N} \log p_\phi(s'_i|\, s_i, a_i)$$

$$= \frac{1}{2}(s'_i - \mu(s_i,a_i;\phi))^T\mathbf{\Sigma}(s_i,a_i;\phi)^{-1}(s'_i - \mu(s_i,a_i;\phi)) + \frac{1}{2}\log(\det\mathbf{\Sigma}(s_i,a_i;\phi)) + \text{ const.}$$

Aleatoric uncertainty in model learning

Variance should be always positive, what do we do?
We output log(variance) and we exponentiate.

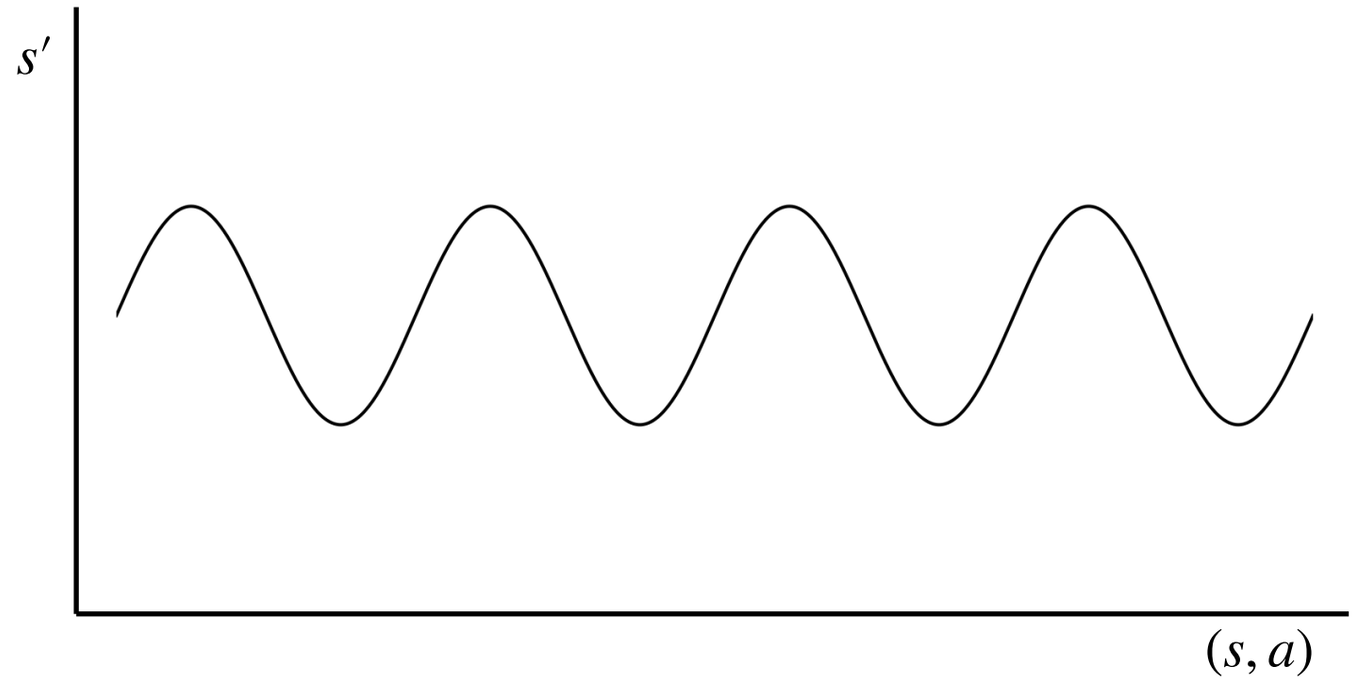
```
logvar = max_logvar - tf.nn.softplus(max_logvar - logvar)
logvar = min_logvar + tf.nn.softplus(logvar - min_logvar)
var = tf.exp(logvar)
```



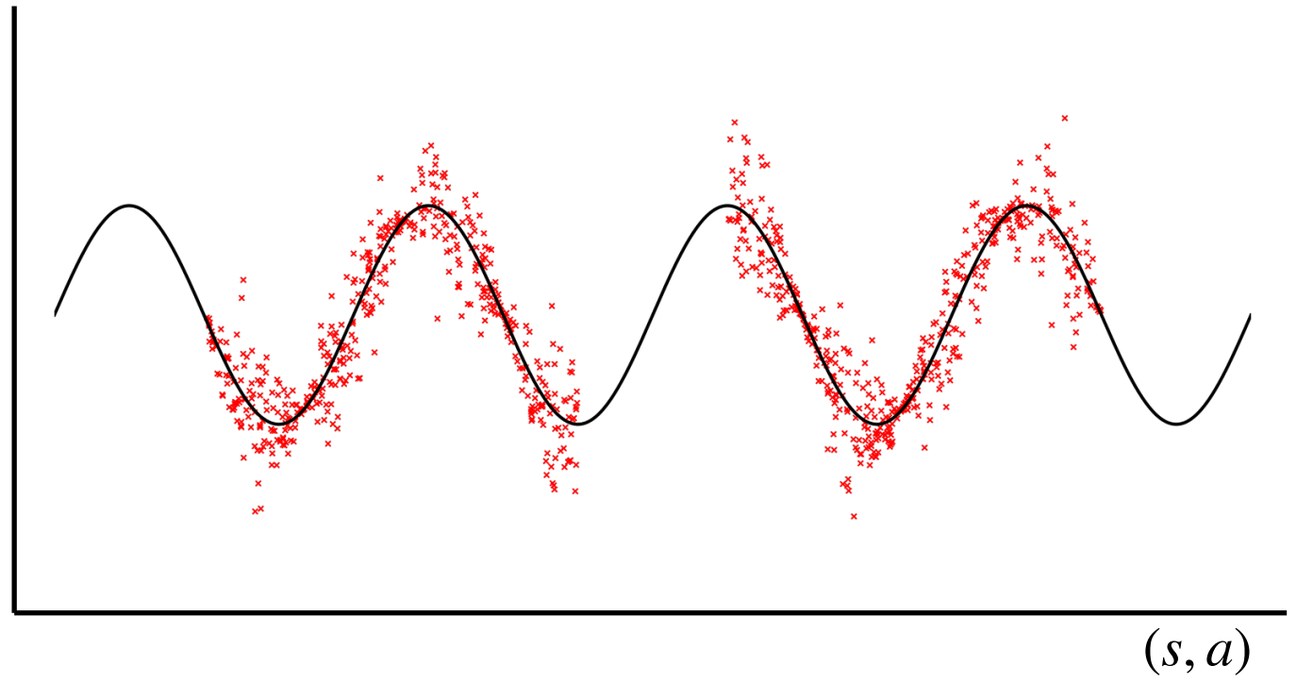
$\text{Rectifier}(x) = \max(0, x)$

$\text{Softplus}(x) = \log(1 + e^x)$

Epistemic uncertainty in Model Learning

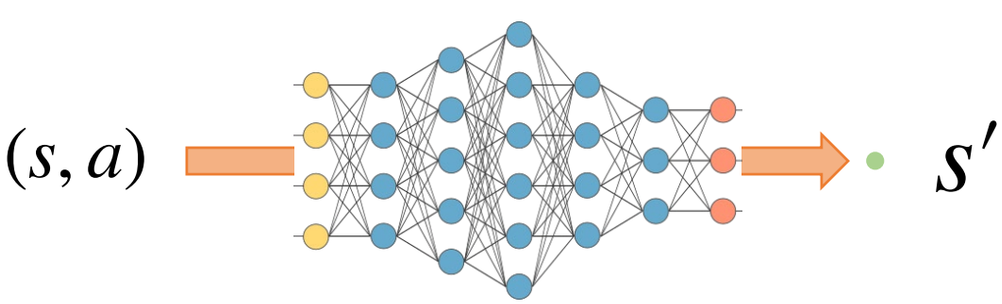


Epistemic uncertainty in Model Learning

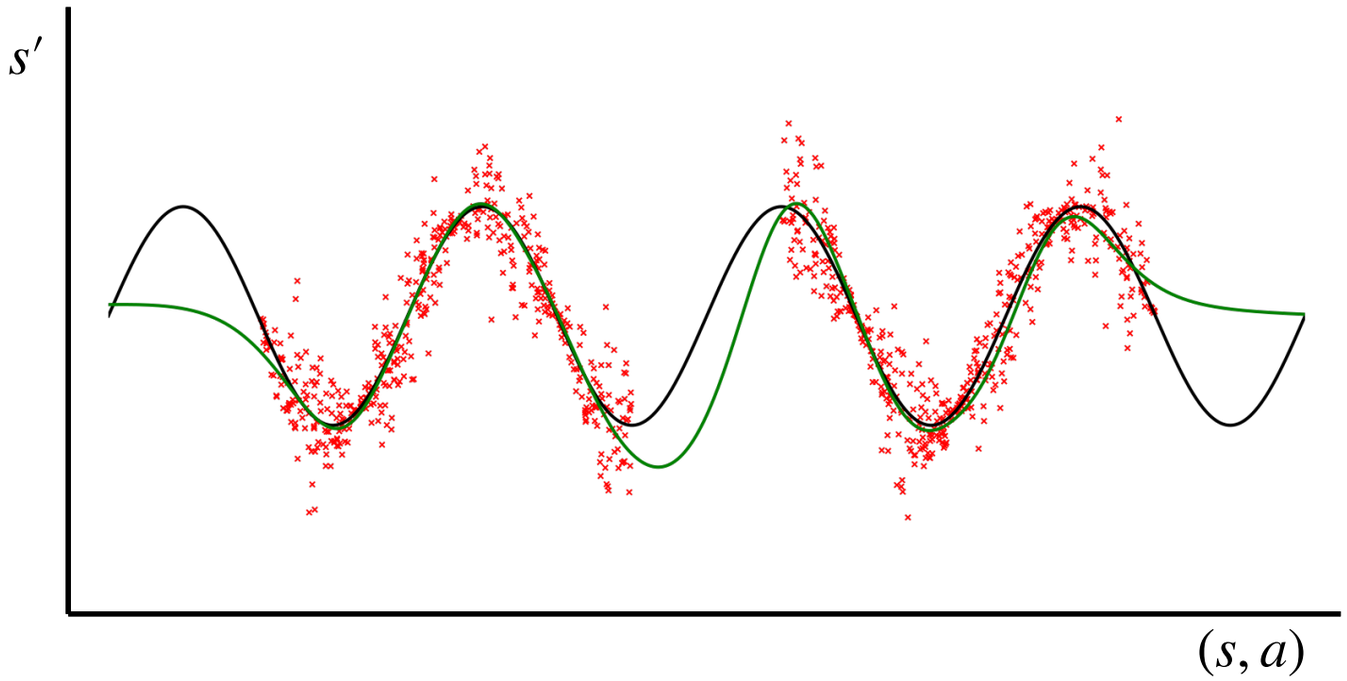


Red are observed data points (s,a,s')

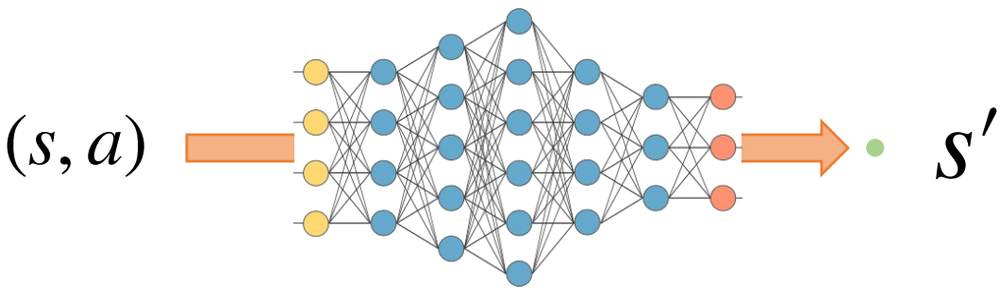
Epistemic uncertainty in Model Learning



Fitting a deterministic neural network

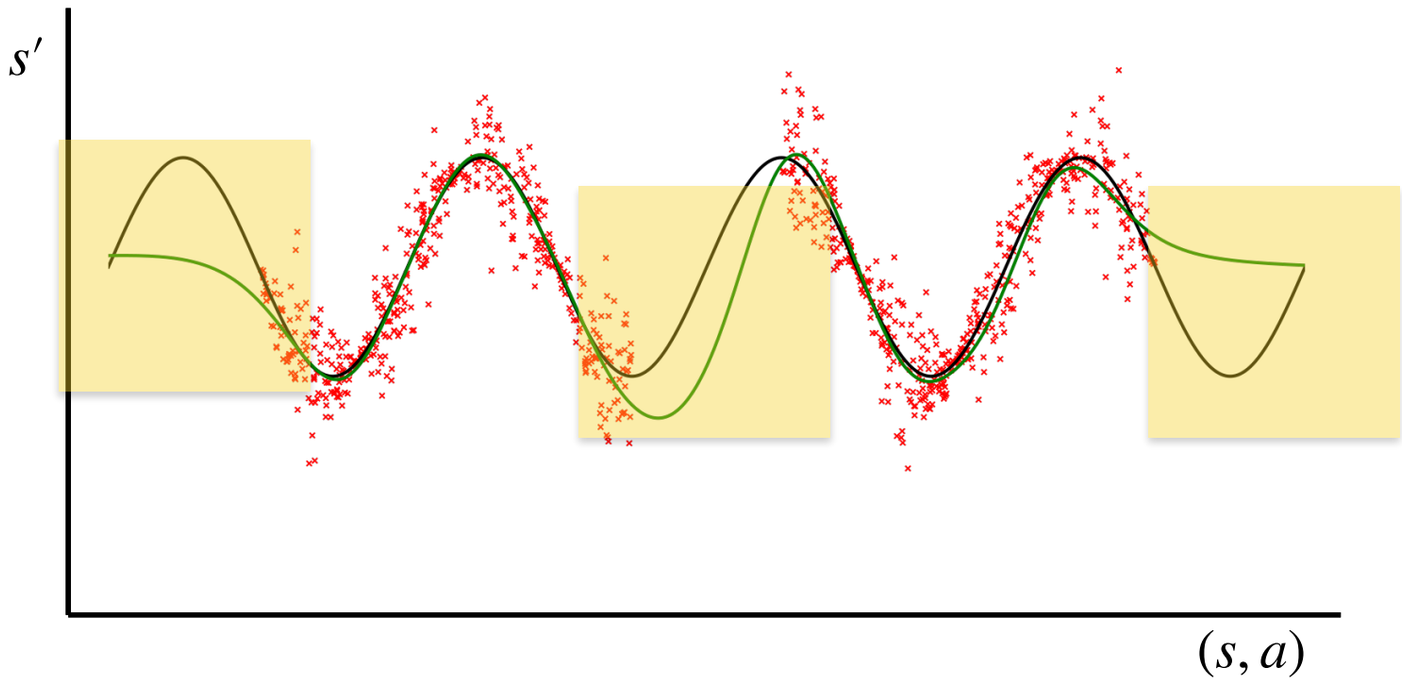


Epistemic uncertainty in Model Learning



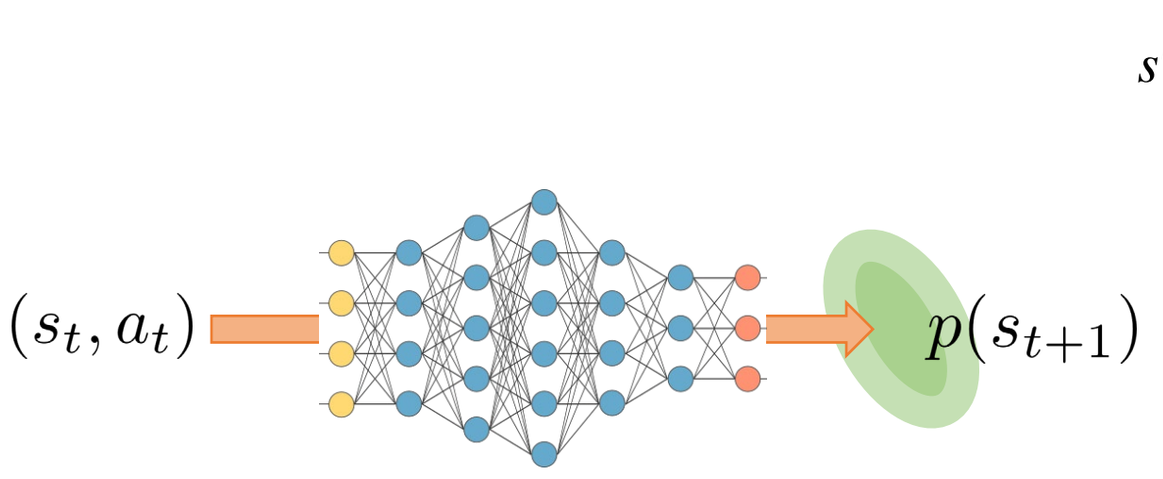
Fitting a deterministic neural network

Model errors!

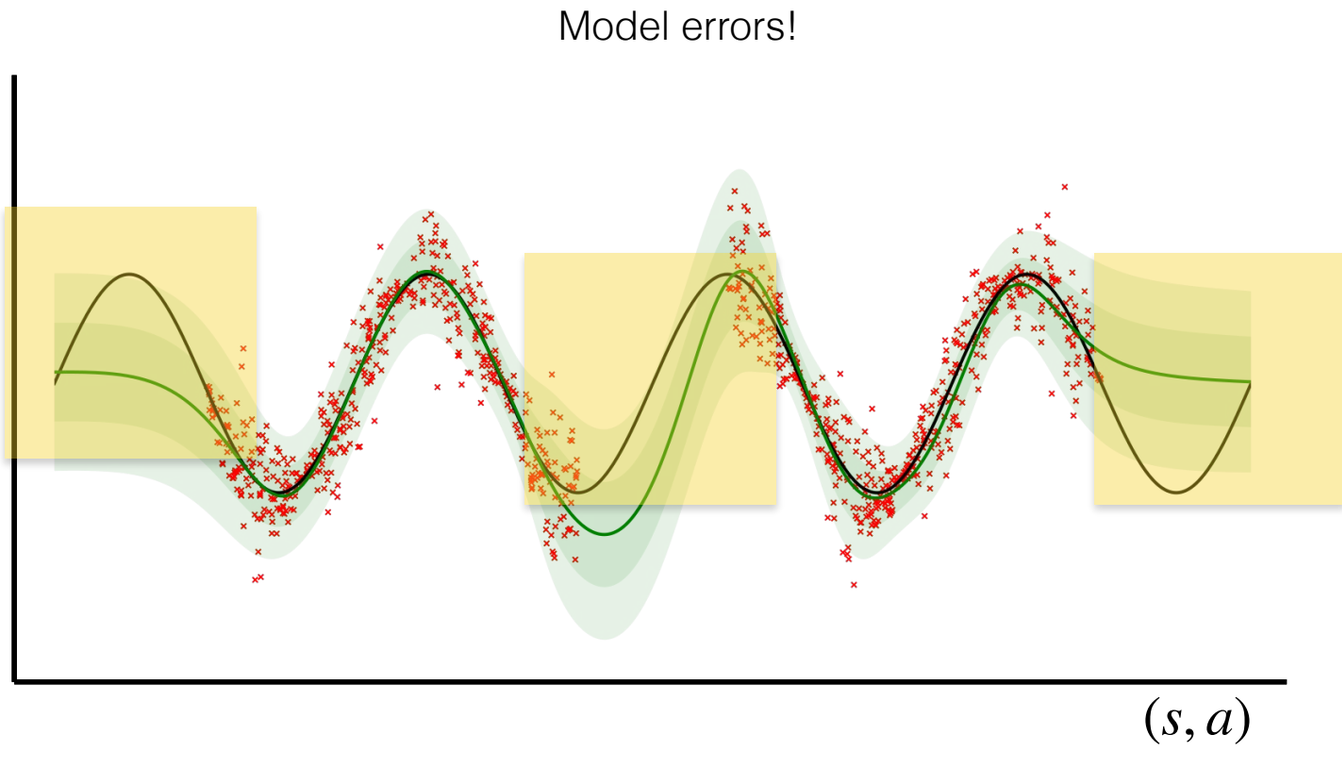


- There is a unique answer for s’ (no stochasticity) but I do not know it due to lack of data.

Epistemic uncertainty in Model Learning

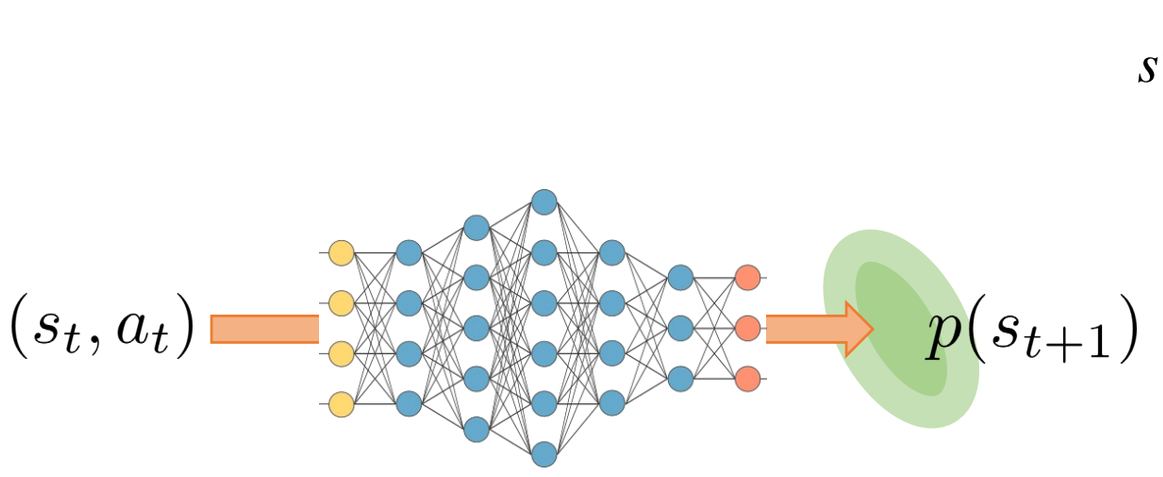


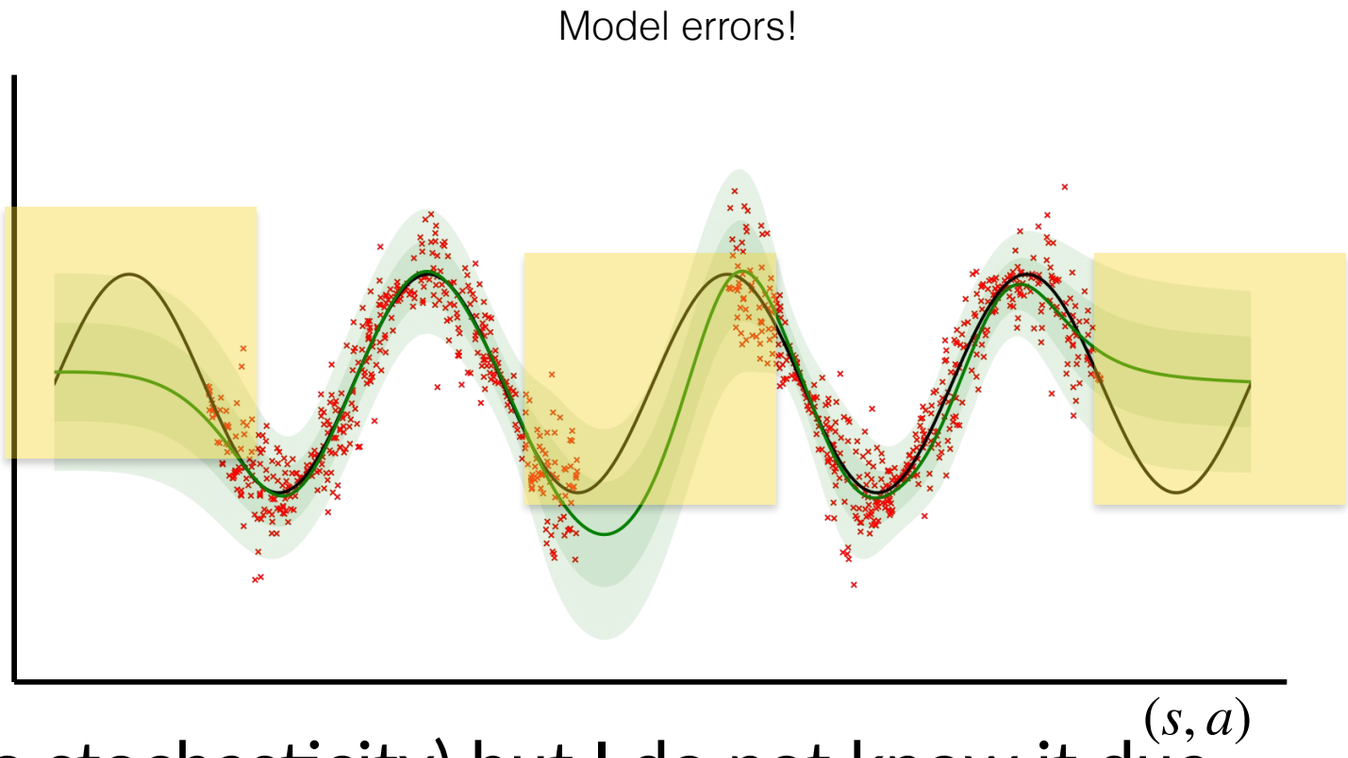
Fitting a neural network that parametrizes a distribution



- There is a *unique* answer for s' (no stochasticity) but I do not know it due to lack of data.
- Predicting a distribution won't help. The predictions will be inaccurate due to lack of data.

Epistemic uncertainty in Model Learning





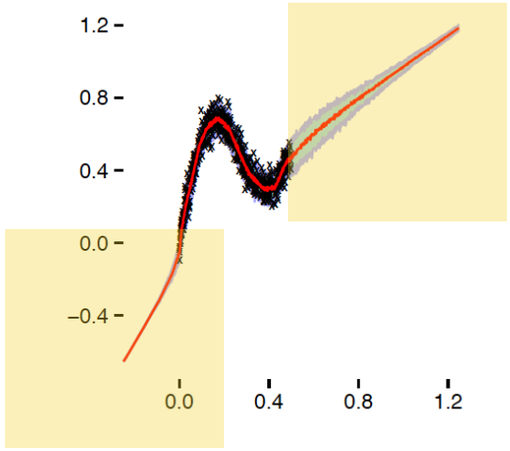
- There is a *unique* answer for s' (no stochasticity) but I do not know it due to lack of data.
- Predicting a distribution won't help. The predictions will be inaccurate due to lack of data.
- How can I represent my uncertainty about my predictions? E.g., having high entropy when no data and low entropy close to data?

Bayesian Inference

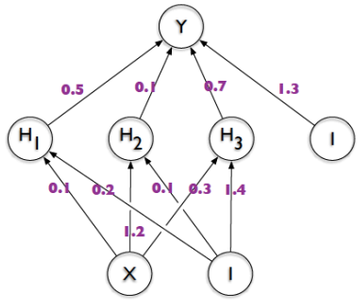
Bayes Rule

$$P(\text{ hypothesis } | \text{ data }) = \frac{P(\text{ hypothesis })P(\text{ data } | \text{ hypothesis })}{\sum_h P(h)P(\text{ data } | h)}$$

- Q: What are the hypotheses here?
- A: Hypotheses here are weights for our learning model, i.e., weights of our neural networks that learns the transition dynamics
- Q: Is this still useful when our prior over parameters is uniform?
- A: Yes. The point is to keep all the hypotheses that fit equally well the training set instead of committing to one, so that I can represent my uncertainty.



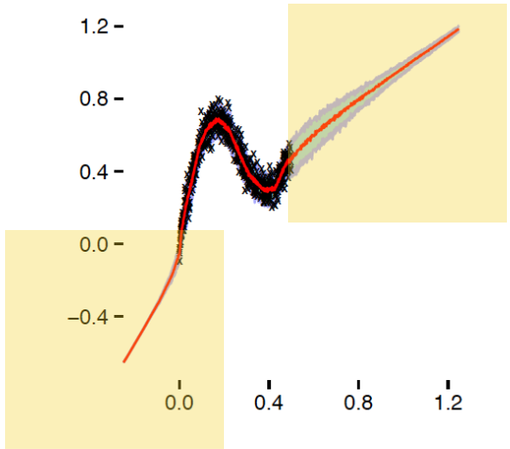
regression network



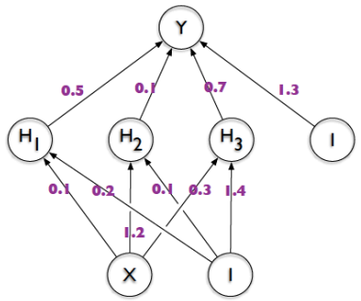
$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left(P(D \mid \phi) + \log P(\phi) \right)$$

Committing to a **single** solution for my neural weights

I cannot quantify my uncertainty away from the training data :-(

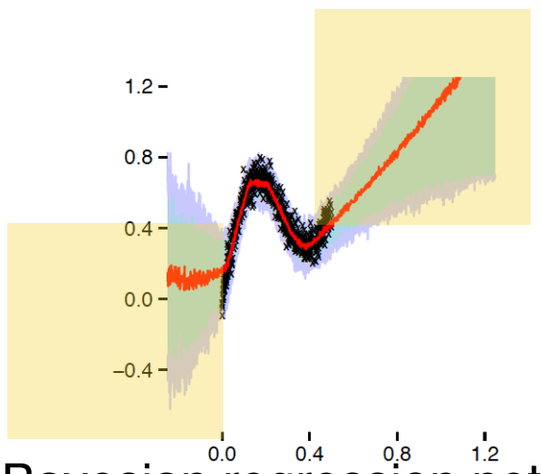


regression network



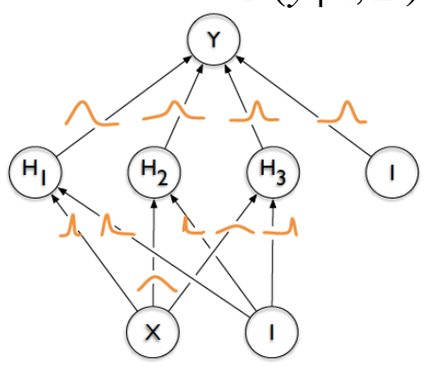
$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left(P(D \mid \phi) + \log P(\phi) \right)$$

Committing to a **single** solution for my neural weights
I cannot quantify my uncertainty away from the training data :-(



Bayesian regression network

$$P(y \mid x, \mathrm{D}) = \int P(y \mid x, \phi) P(\phi \mid \mathrm{D}) d\phi$$

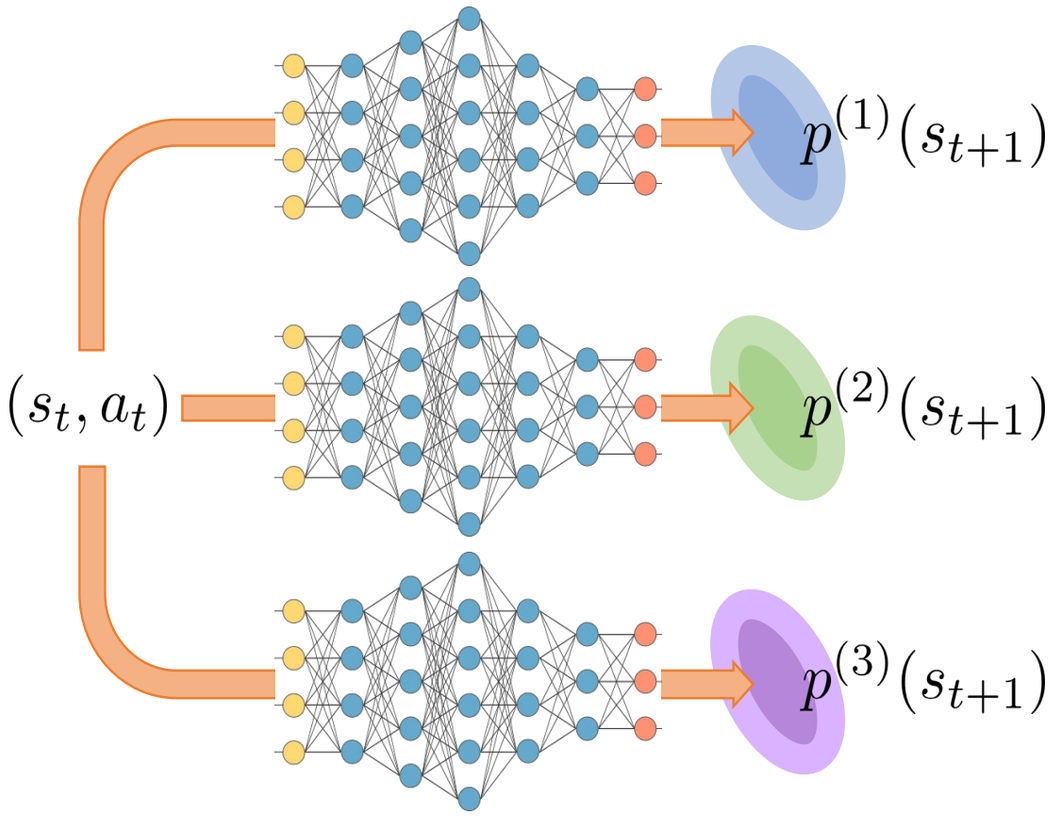


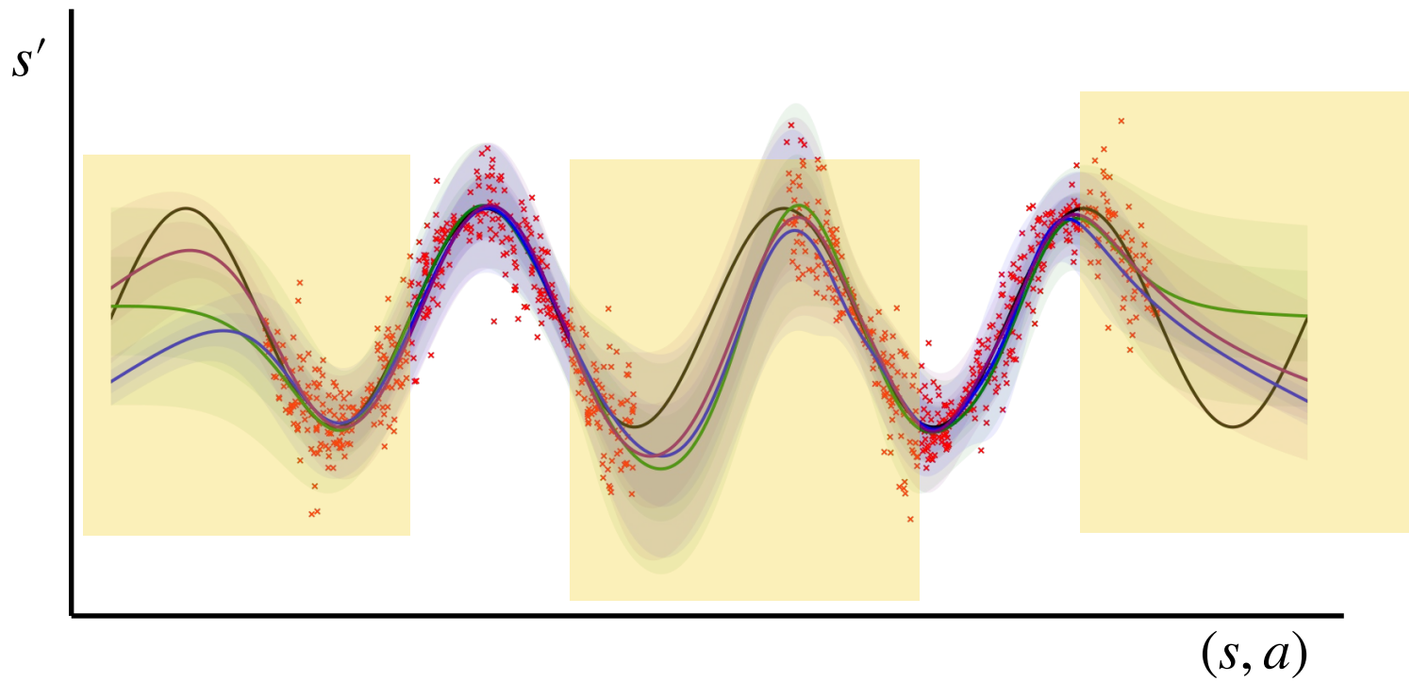
$P(\phi \mid D)$

- Having a posterior distribution over my neural weights.
- I can quantify my uncertainty by sampling networks and measuring the entropy of their predictions :-)
- Inference of such posterior is intractable :-(but there are some nice recent variational approximations

NN Ensembles for representing Epistemic uncertainty

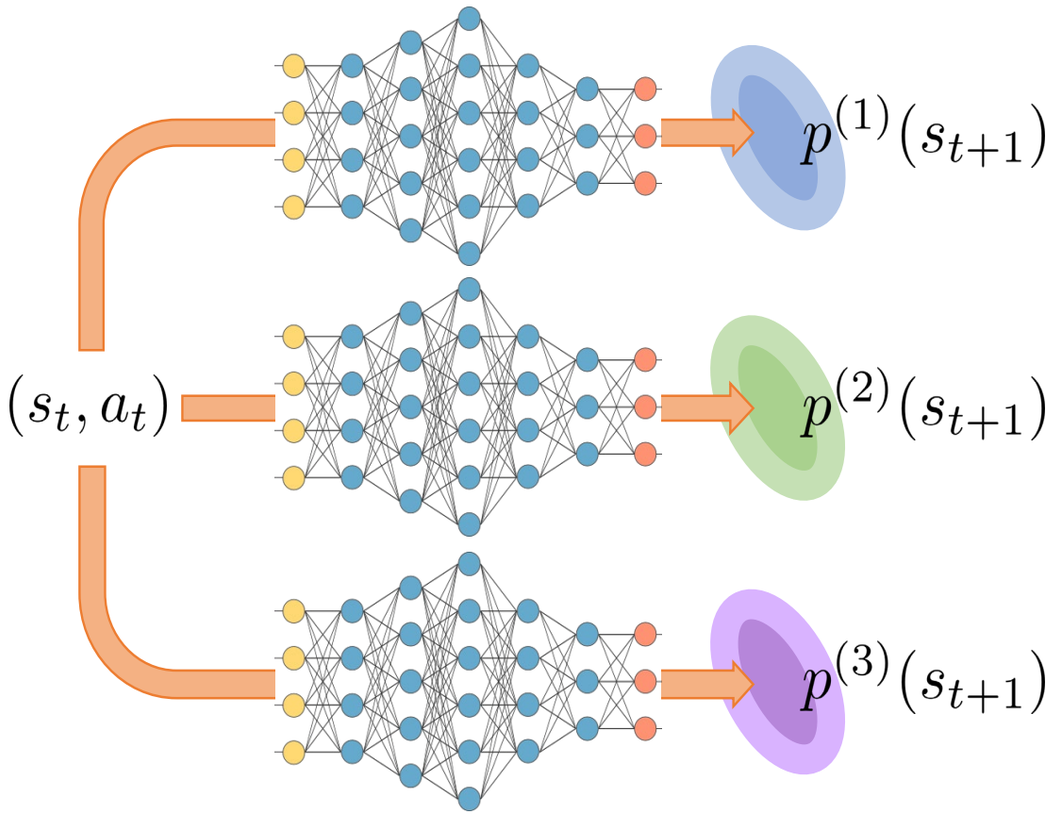
- Neural network Ensembles are a good approximation to Bayesian Nets.
- Instead of having explicit posteriors distributions for each neural net parameter, you just have a small set of neural nets, each trained on separate data.
  - On the data they have seen, they all agree (low entropy of predictions)
  - On the data they have not seen, each fails in its own way (high entropy of predictions)

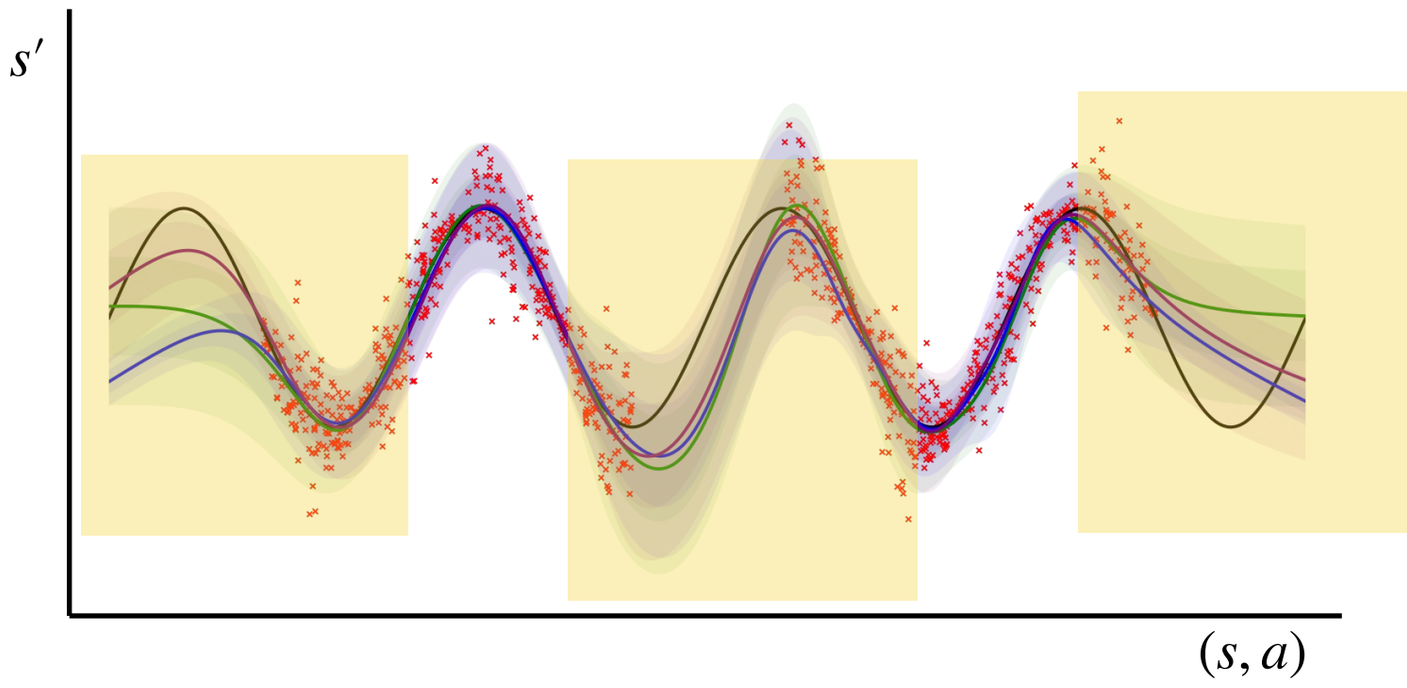




NN Ensembles for representing Epistemic uncertainty

- Neural network Ensembles are a good approximation to Bayesian Nets.
  - How do we train such neural network ensembles given a dataset of interactions?
  - The most popular way is to train bunch of network with different initializations and on different subsets of the data.
  - Check also this cool paper: HyperGAN: A Generative Model for Diverse, Performant Neural Networks





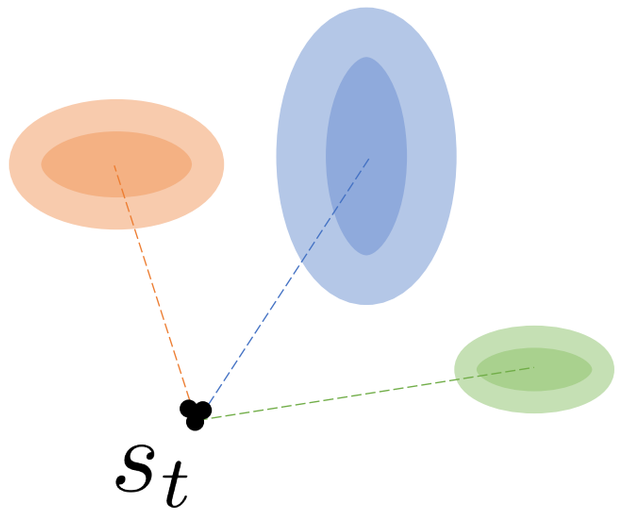
Model Unrolling

$$\dot{s_t}$$

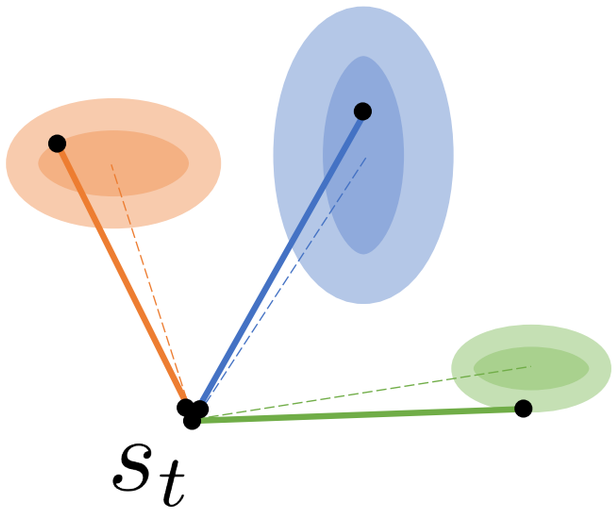
Model Unrolling

s_t

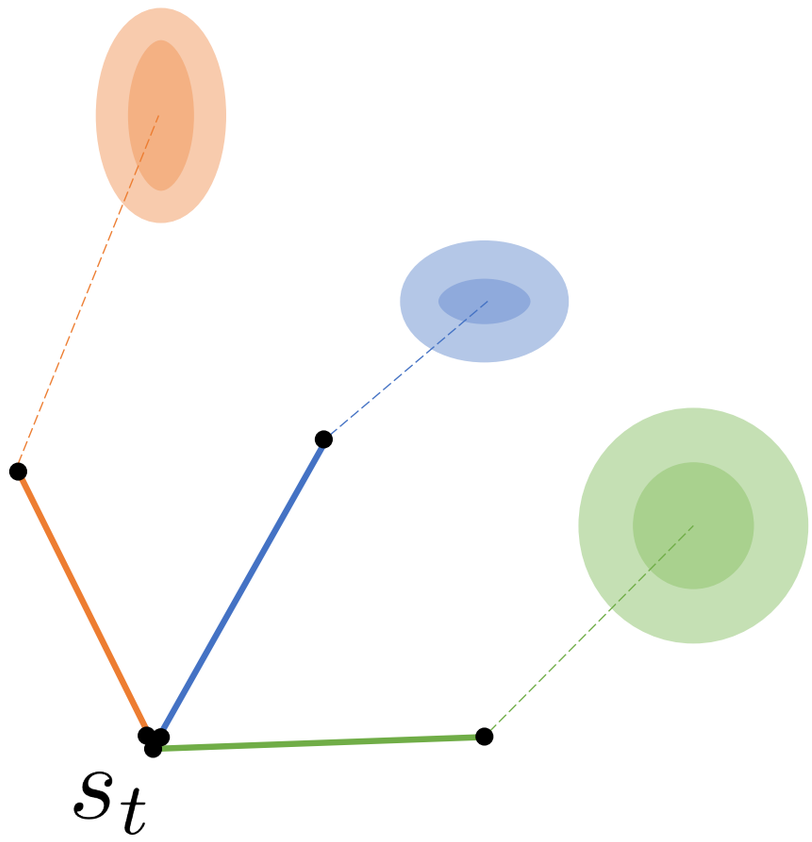
Model Unrolling



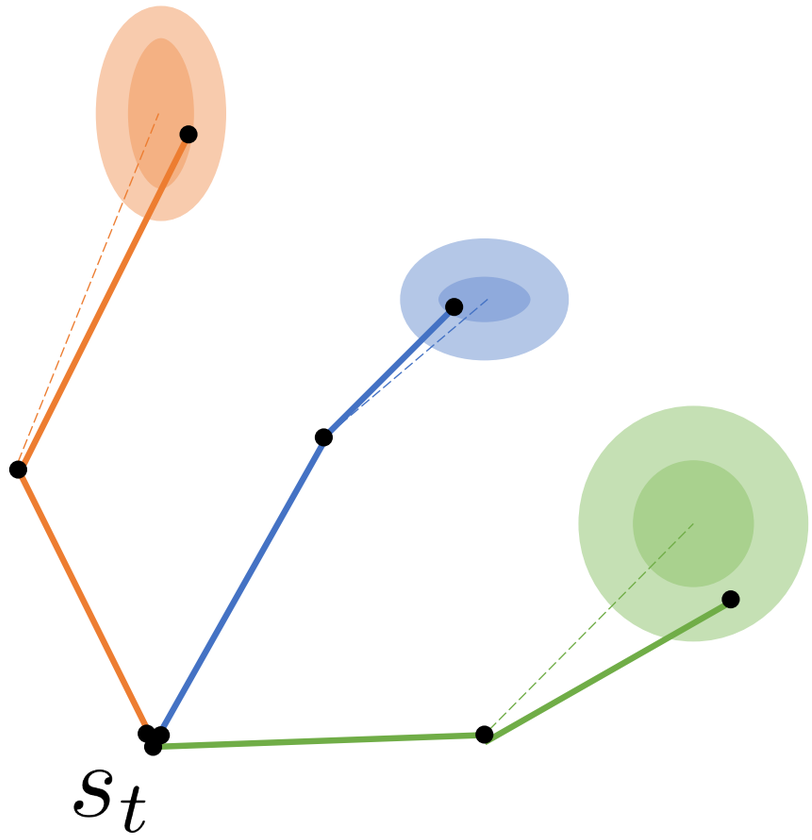
Model Unrolling



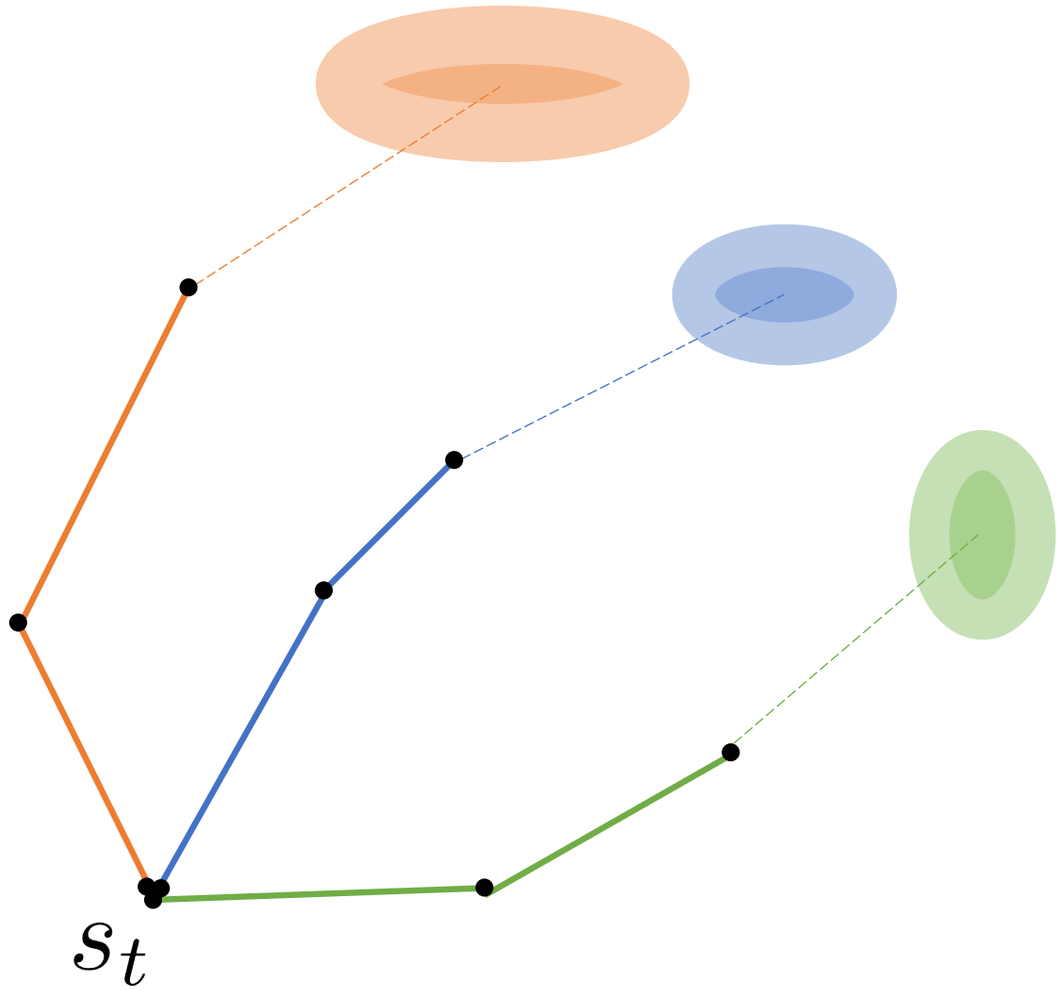
Model Unrolling



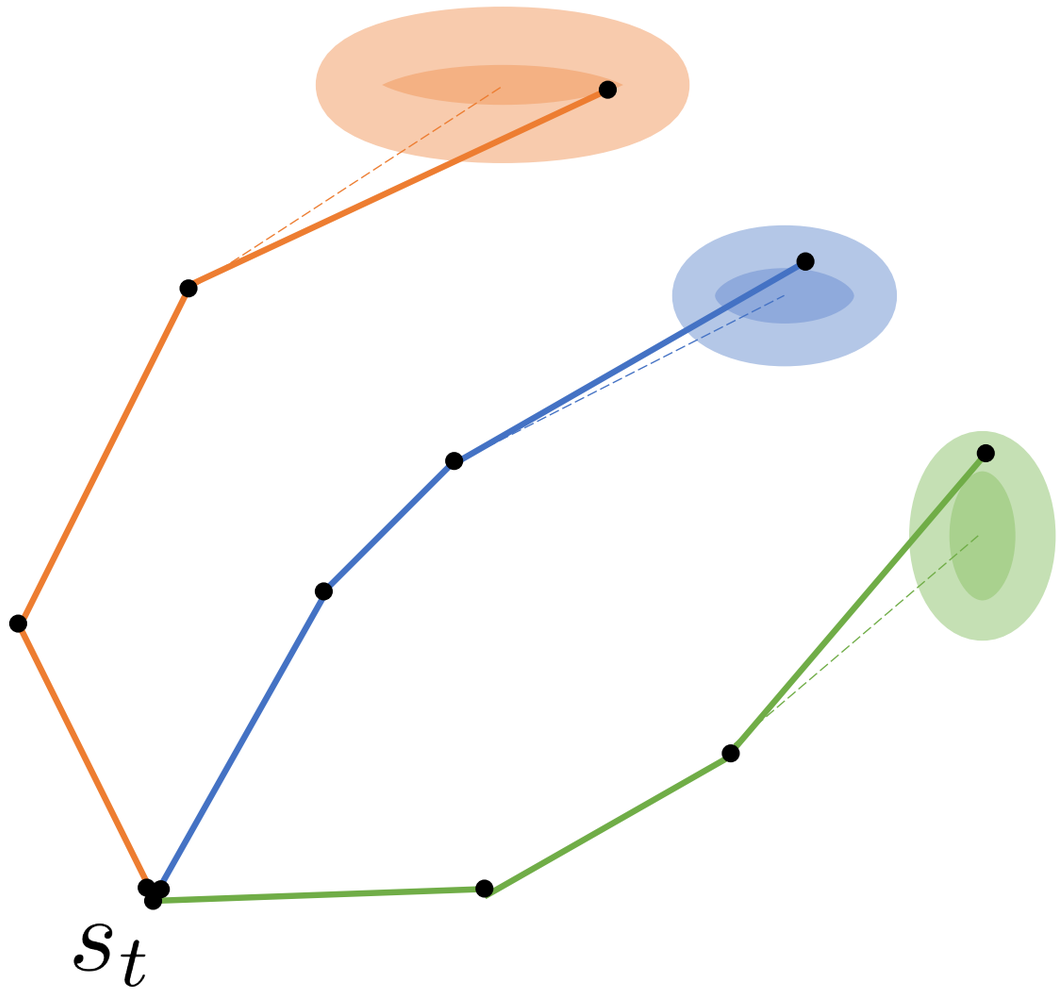
Model Unrolling



Model Unrolling

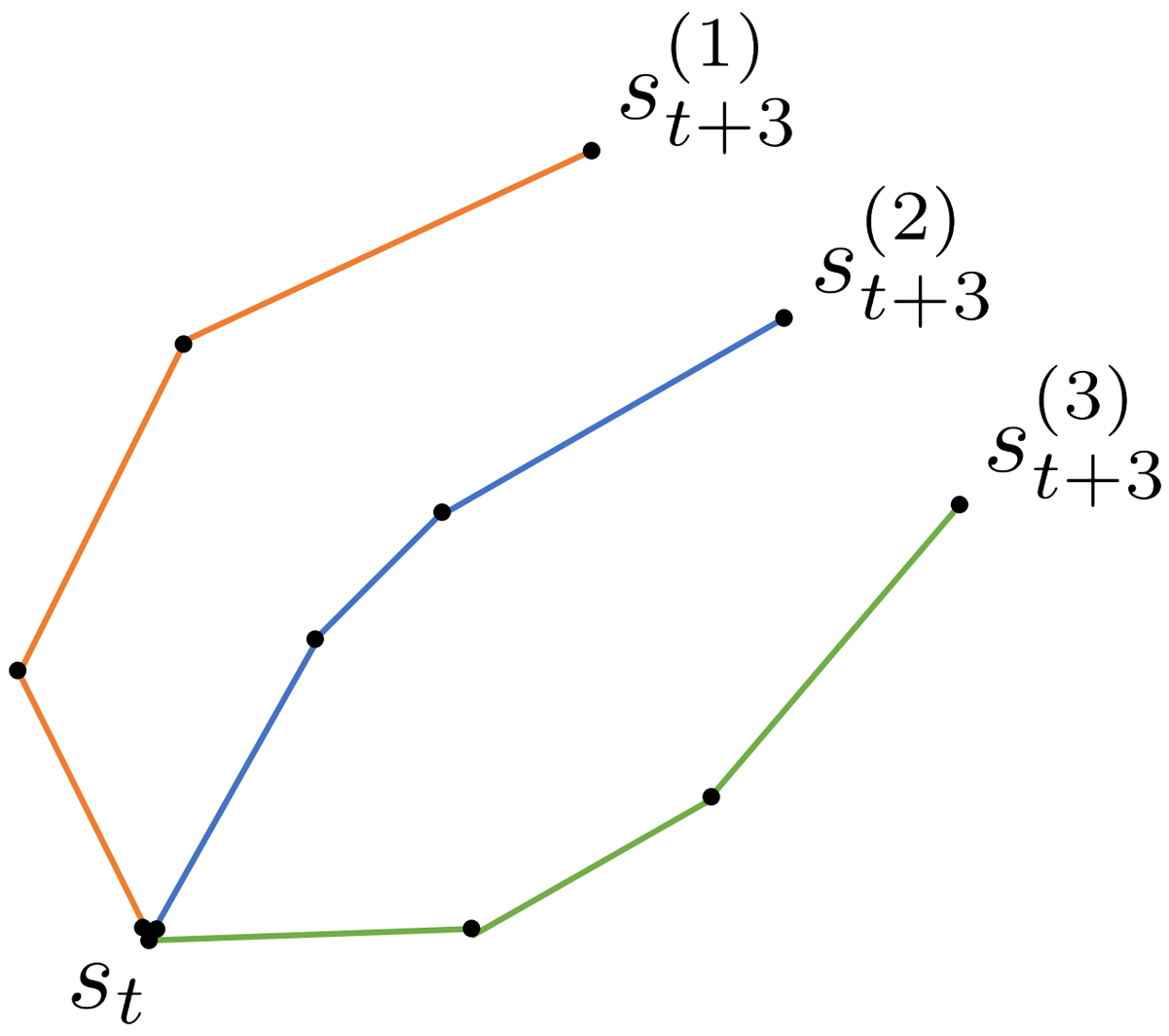


Model Unrolling



Model Unrolling

I compute the reward of an action sequence by averaging across particles



https://www.youtube.com/watch?time_continue=86&v=3d8ixUMSiL8

Probabilistic Ensembles Trajectpry Sampling (PETS)

Initialize D_{env} using experience from random actions.

1. Train probabilistic ensemble dynamics model using D_{env}.
2. For t=1..TaskHorizon
    1. Use Cross-entropy Method (CEM) to select actions $a^*_{t...T}$ by unrolling the model
    2. Execute first action a^*_t.
    3. Update $D_{env} \leftarrow D_{env} \cup \{s_t, a^*_t, s_{t+1}\}$
3. GOTO 2.

Results

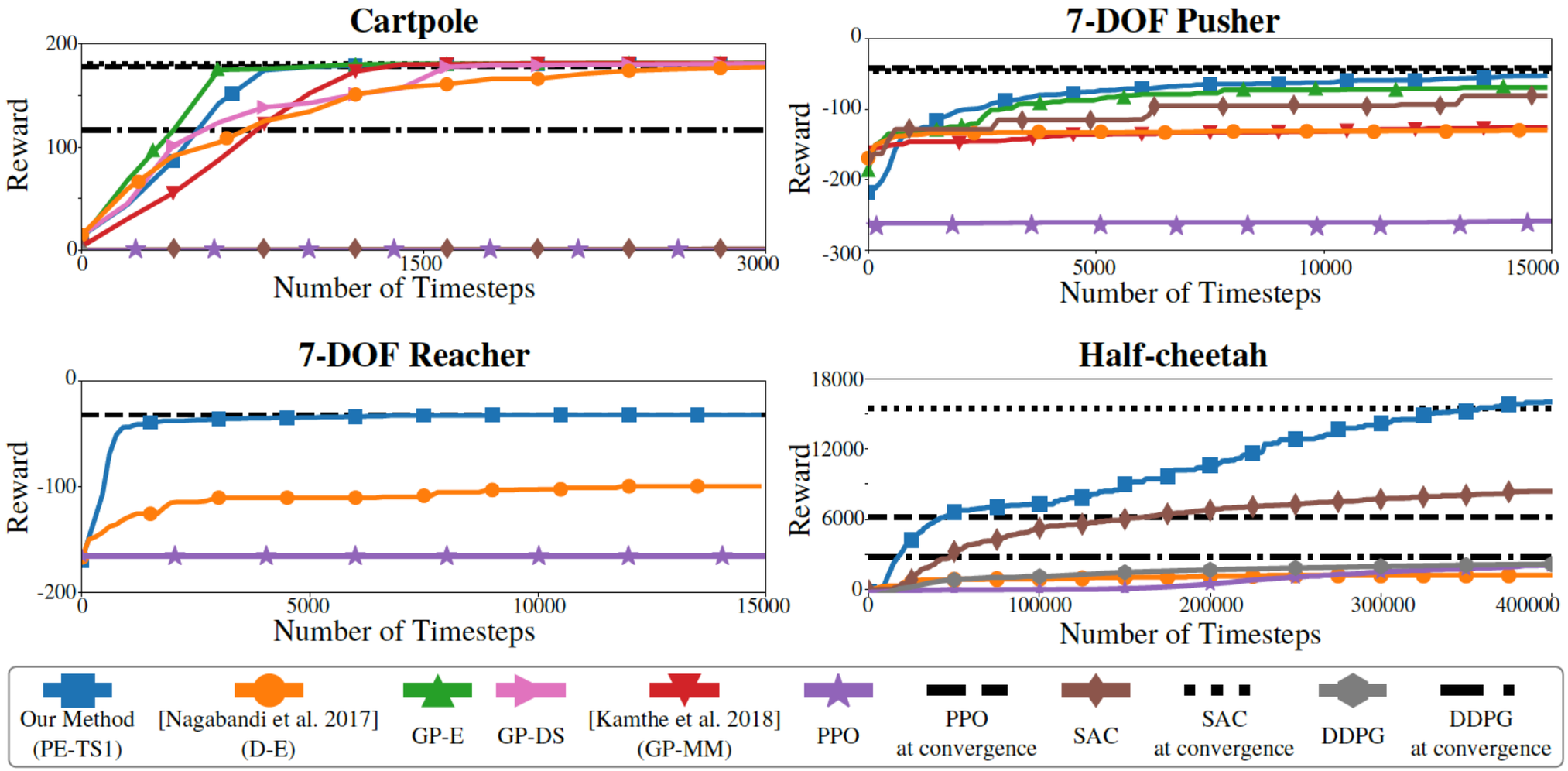


Figure 3: Learning curves for different tasks and algorithm. For all tasks, our algorithm learns in under 100K time steps or 100 trials. With the exception of Cartpole, which is sufficiently low-dimensional to efficiently learn a GP model, our proposed algorithm significantly outperform all other baselines. For each experiment, one time step equals 0.01 seconds, except Cartpole with 0.02 seconds. For visual clarity, we plot the average over 10 experiments of the maximum rewards seen so far.

https://www.youtube.com/watch?time_continue=86&v=3d8ixUMSiL8

Visiting next

- **Sample experience** from the *learnt* model. Then use a model-free method to learn policies at training time. At test time use only the learnt policy. (*MBPO[1]*, *Model-ensemble TRPO[2]*).
- **Sample experience** from the known environment model. Then use a model-free method to learn policies at training time. At test time use only the learnt policy. (All *model-free RL*).
- **Plan through the model**: Unroll the model forward in time and optimize for action sequences to maximize rewards at training time. Then, learn policies that predict the actions found by model unrolling. At test time use only the learnt policy.
  - Differentiable optimization: back propagate the error through the dynamics function over time.(Linear quadratic regulator(LQR)[3]: linear models, dream-to-control[4]: non-linear models)
  - Evolutionary search for actions, e.g., CMA-ES. (PETS[5])
  - Monte Carlo tree search: combination of learnt policies and model unrolling. (*Playing Atari with offline MCTS[6]*)
- **Plan through the model**:Unroll the model forward in time and optimize for action sequences to maximize rewards at training time. Then, learn policies that predict the actions found by model unrolling. At test time, use both model unrolling and the learnt policy. (*AlphaGoZero[7]*, *MuZero[8]*)
- **Plan through the model**:Use model rollouts to supply better targets to Q functions (Stochastic Ensemble Value Expansion[9]). Learn a policy that maximizes the Q function, and use it at test time.

[1]When to Trust Your Model-Model-Based Policy Optimization (MBPO)
[2]Model-ensemble trust region policy optimization
[4]Dream-to-control:learning behaviors via latent imagination
[5]Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models (PETS)
[6]Deep Learning for Real-Time Atari Game Play Using Offline MCTS
[7]Mastering the game of Go without human knowledge(AlphaGoZero)
[8]Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model (MuZero)
[9]Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion

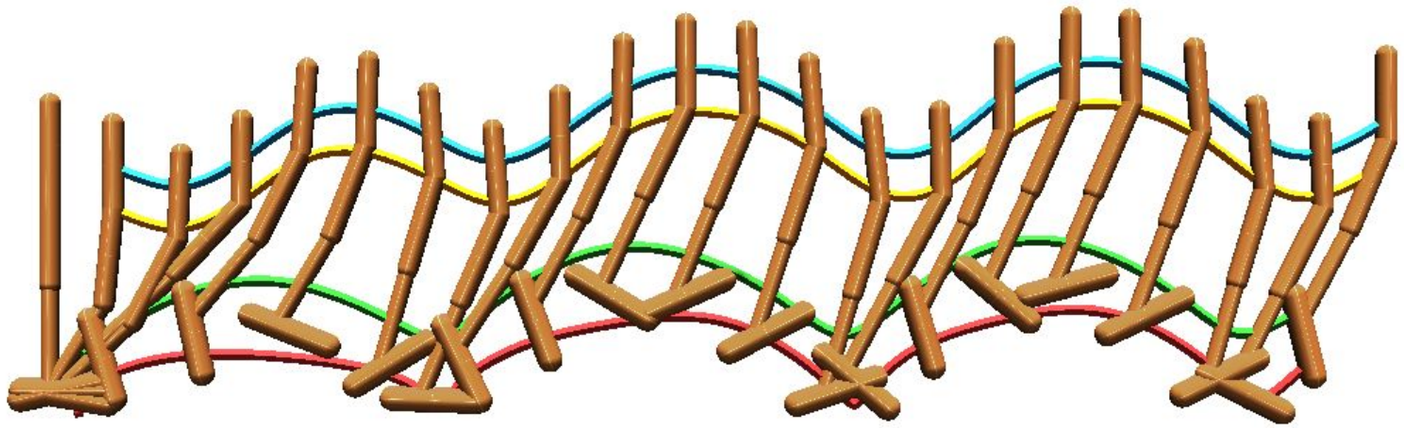
Model-ensemble trust region policy optimization

Initialize policy $\pi(s;\theta)$ and $D_{env} = \{\}$.

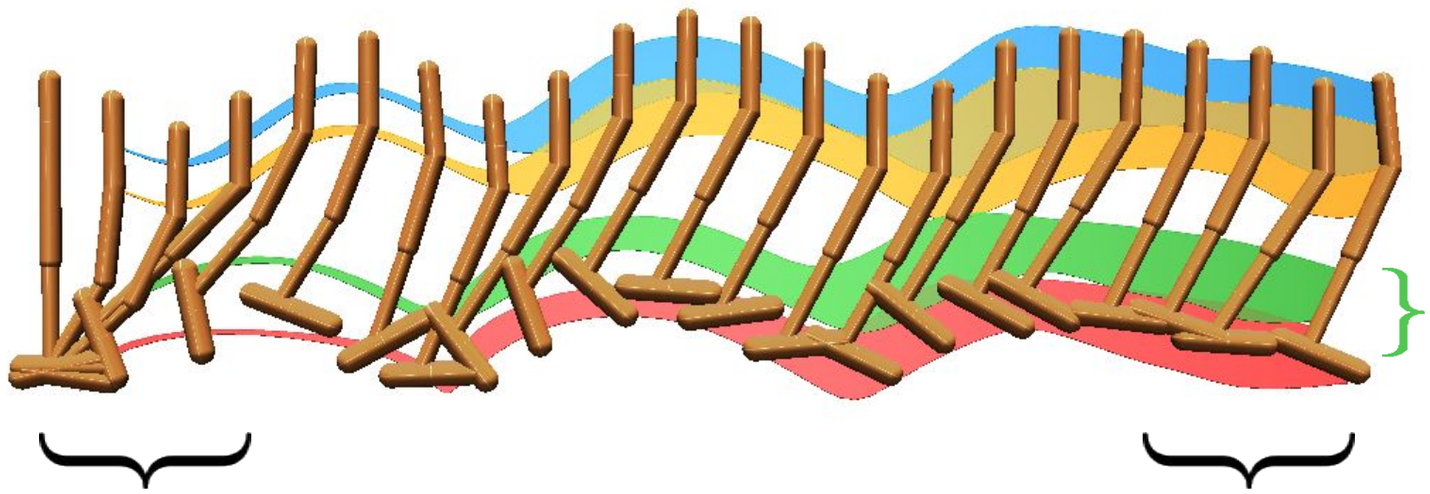
1. Run the policy and update experience tuples dataset D_{env}.
2. Train probabilistic ensemble dynamics model using D_{env}
   $(s', r') = f^i(s, a; \phi), i = 1..N$
3. **Repeat**
   1. Collect simulated experience D_{model} by sampling from the model ensemble using the policy to select actions, starting from s_0.
   2. Update the policy using TRPO on D_{model}.
4. **Until** performance across all model ensembles stops improving
5. GOTO 1.

I use the models just to obtain simulated experience.
I update the policy so that it does well across **all** members of the model ensemble: the policy cannot exploit inaccuracies of any one of them.

The problem with long rollouts



Environment rollout



Model rollout (x1000)

} mean prediction +/- std dev

accurate, low variance

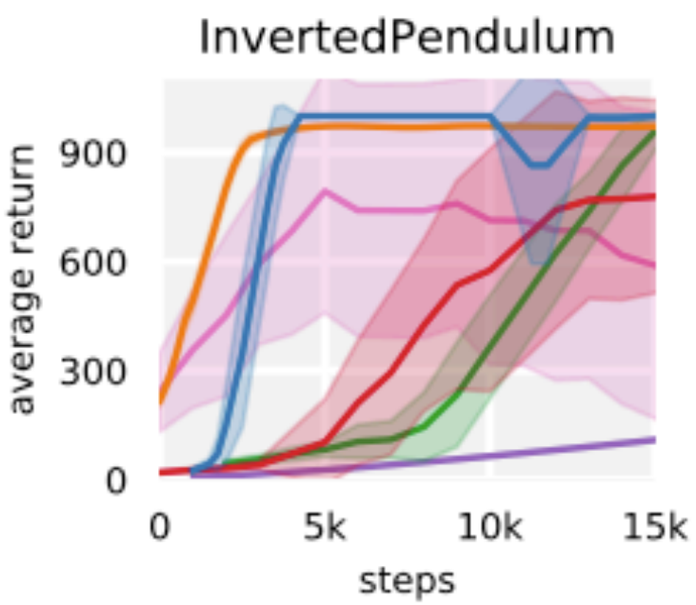
low accuracy, high variance

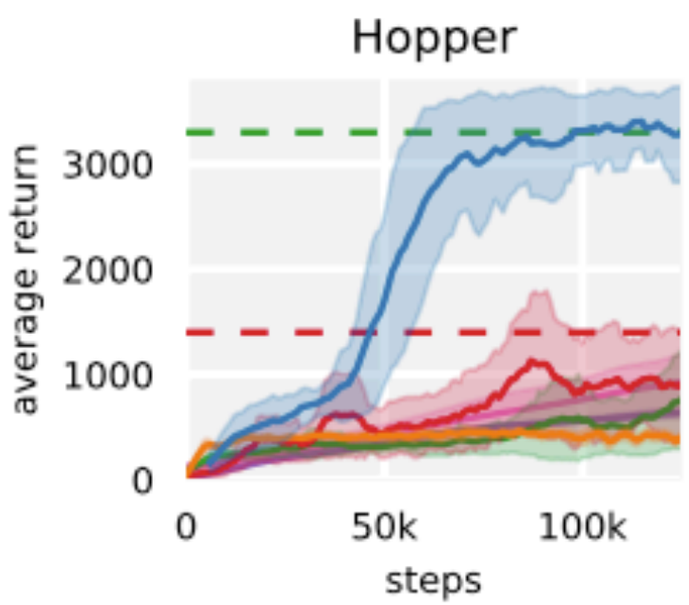
When to Trust Your Model: Model-Based Policy Optimization

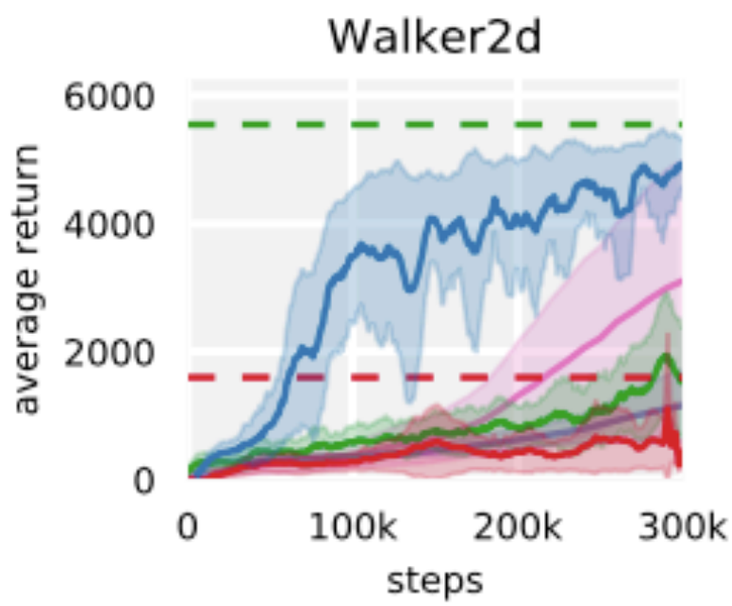
Initialize policy $\pi(s;\theta)$ and $D_{env} = \{\}$.

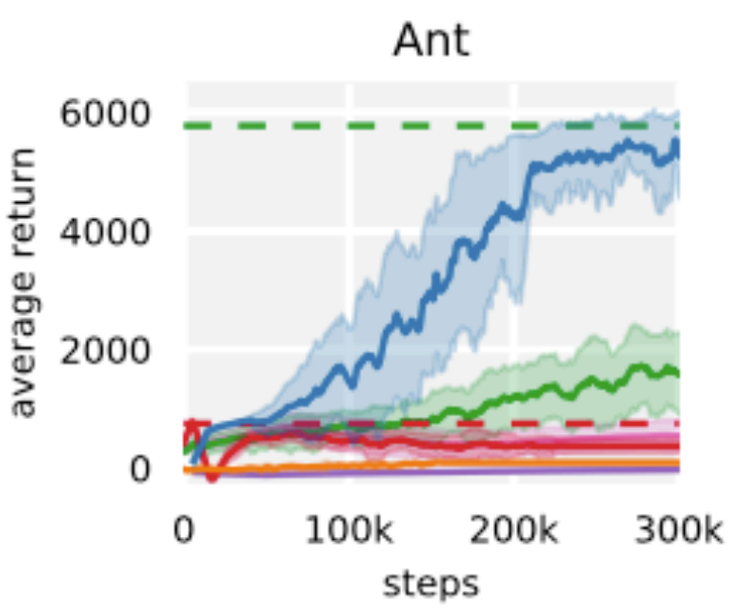
1. Run the policy and update experience tuples dataset D_{env}.
2. Train probabilistic ensemble dynamics model using D_{env}
   $(s', r') = f^i(s, a; \phi), i = 1..N$
3. For M model rollouts
   1. Sample s_t and then collect simulated experience D_{model} by sampling from the model for k time steps using the policy π_θ to select actions.
4. Update the policy using SAC on D_{model}.
5. GOTO 1.

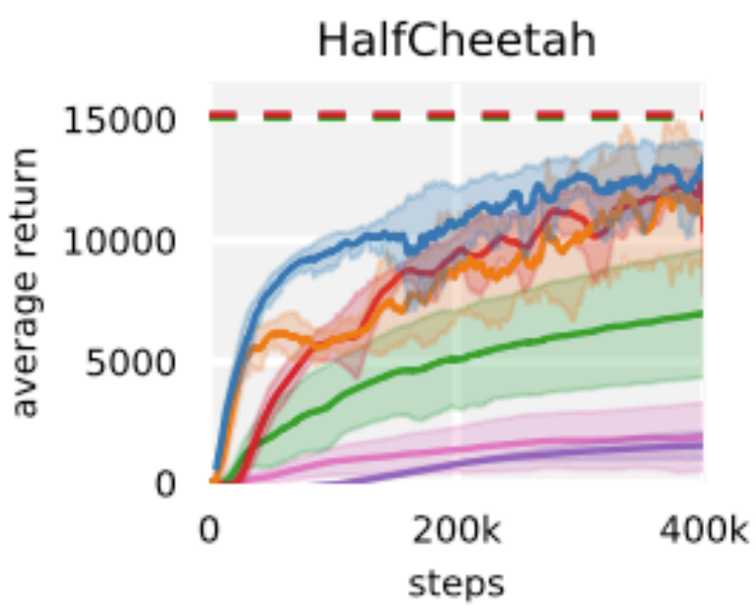
- The model rollout can be shorter than the task horizon.
- a combination of model ensembles with short model rollouts is sufficient to prevent model exploitation
- different transitions along a single model rollout to be sampled from different dynamics models

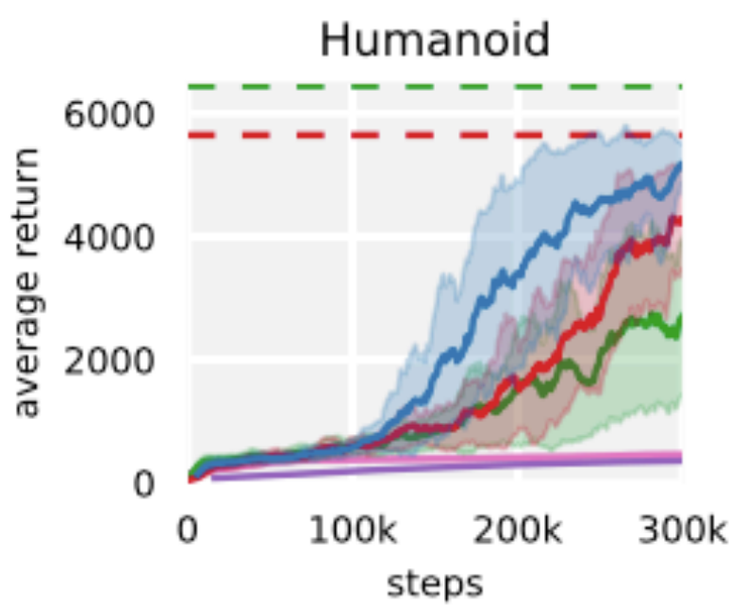

InvertedPendulum
average return
900
600
300
0
0
5k
10k
15k
steps


Hopper
average return
3000
2000
1000
0
0
50k
100k
steps


Walker2d
average return
6000
4000
2000
0
0
100k
200k
300k
steps


Ant
average return
6000
4000
2000
0
0
100k
200k
300k
steps


HalfCheetah
average return
15000
10000
5000
0
0
200k
400k
steps


Humanoid
average return
6000
4000
2000
0
0
100k
200k
300k
steps

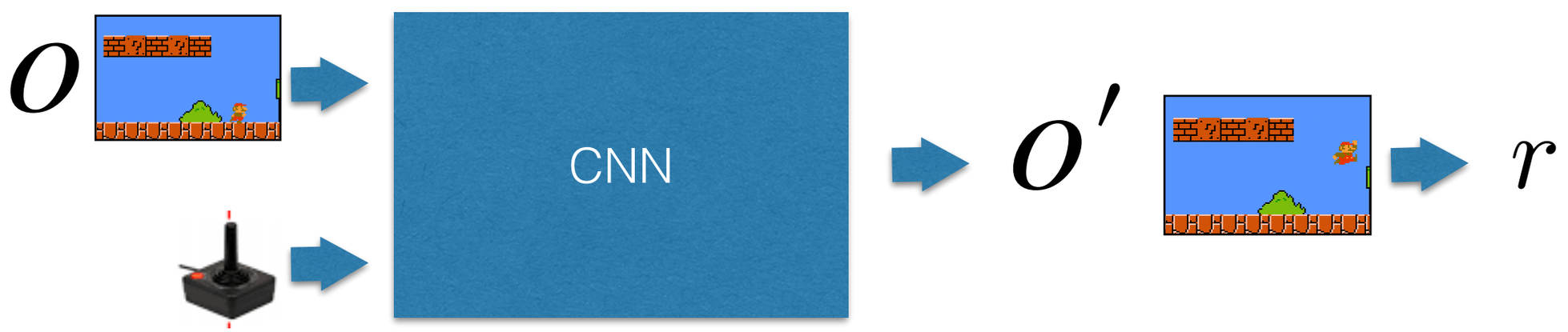
MBPO
SAC
PPO
PETS
STEVE
SLBO
convergence

Learning models from videos as opposed to low-dim states

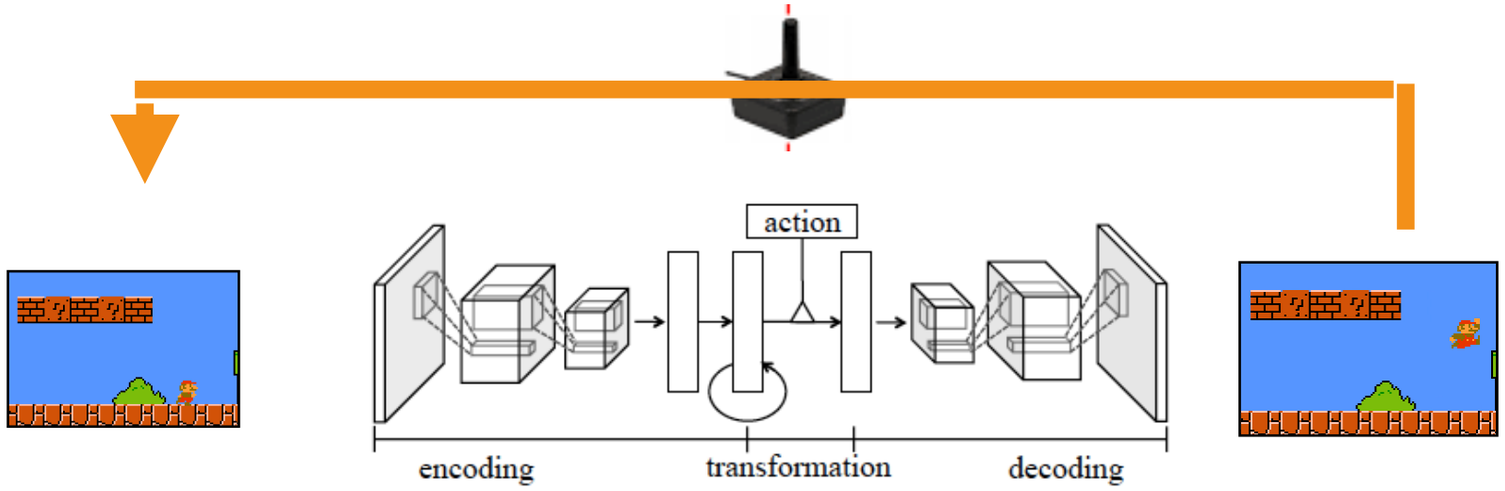
Action-Conditional Video Prediction using Deep Networks in Atari Games

Junhyuk Oh Xiaoxiao Guo Honglak Lee Richard Lewis Satinder Singh

- Train a neural network that given an image (sequence) and an action, predict the pixels of the next frame
- Unroll it forward in time to predict multiple future frames



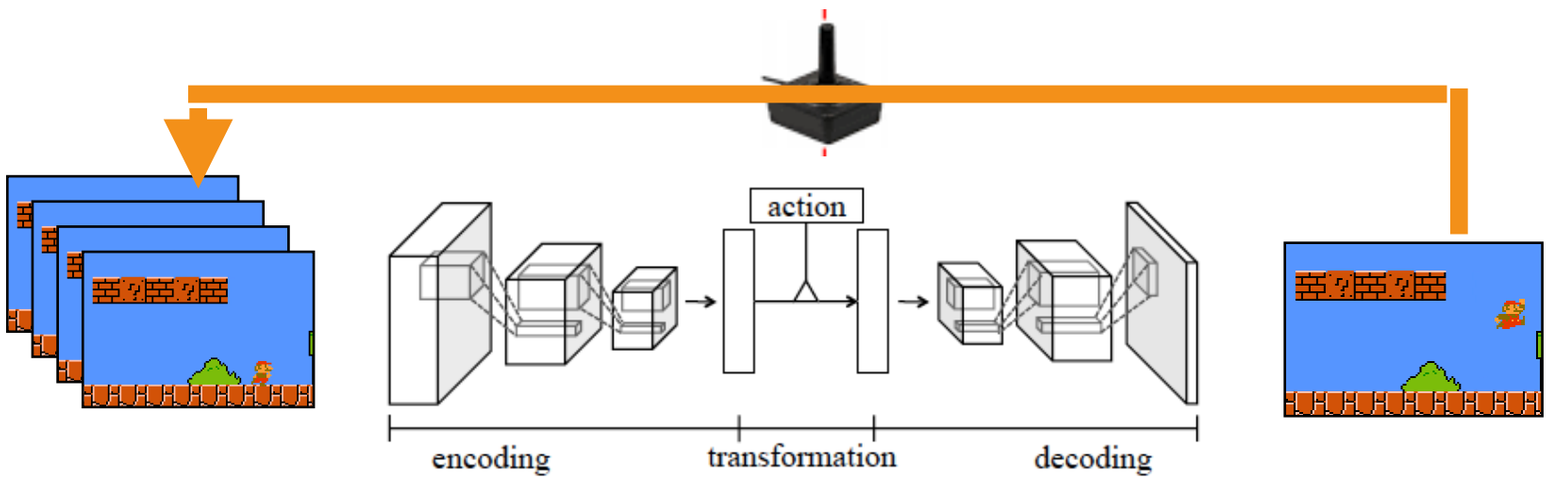
Minimizing error accumulation during unrolling



Unroll the model by feeding the prediction back as input!

Action-Conditional Video Prediction using Deep Networks in Atari Games, *Oh et al.*

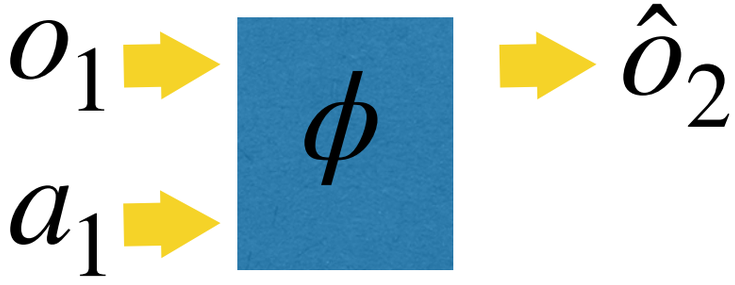
Minimizing error accumulation during unrolling



Q: Can I train my model using tuples (o, a, o') and at test time unroll it over time?
A: No, we will have distribution shift, same as in imitation learning: tiny mistakes will soon cause the model to drift

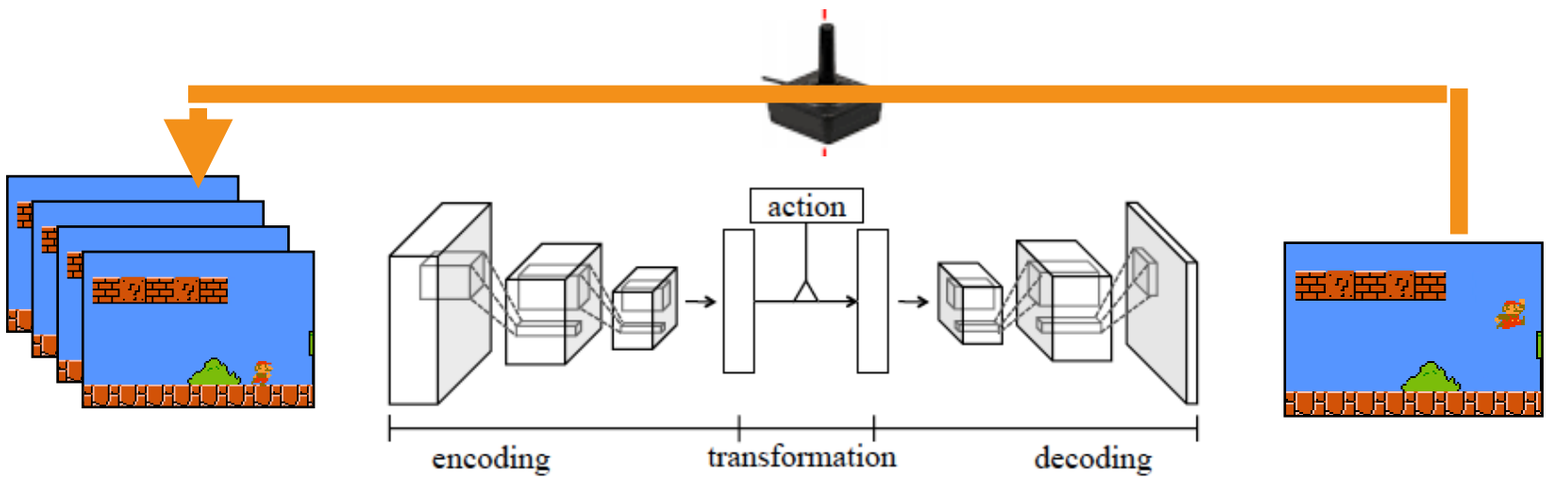
Solution: Progressively increase the unrolling horizon k at training time so that the model learns to handle its mistakes:

$$\mathscr{L}(\phi) = \frac{1}{N}\sum_{i=1}^{N} \|f(a_1^i, o_1^i; \phi) - o_2^i\|$$



Action-Conditional Video Prediction using Deep Networks in Atari Games, *Oh et al.*

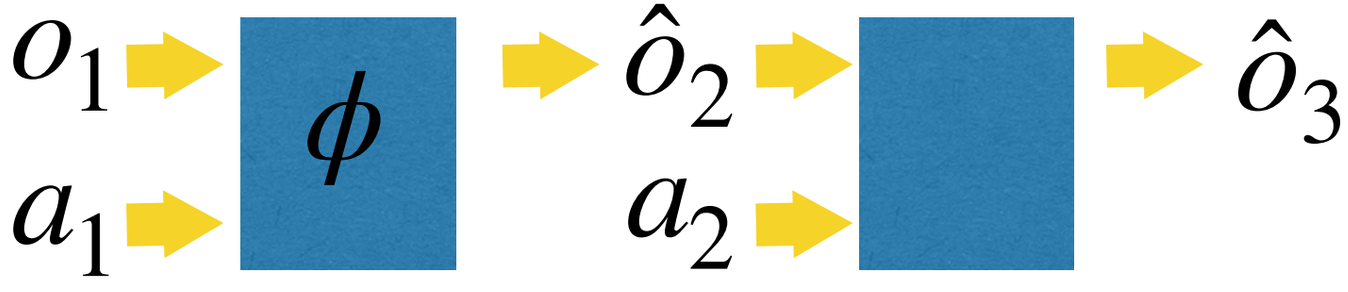
How to train our model so that unrolling works



Q: Can I train my model using tuples (o, a, o') and at test time unroll it over time?
A: No, we will have distribution shift, same as in imitation learning: tiny mistakes will soon cause the model to drift

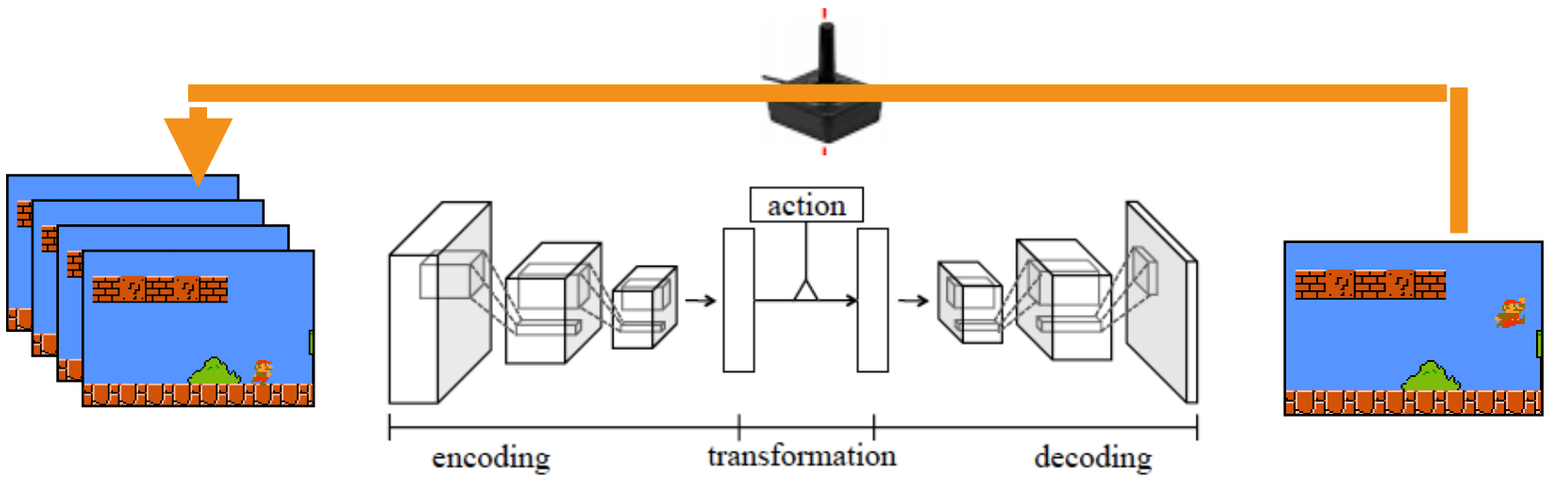
Solution: Progressively increase the unrolling horizon k at training time so that the model learns to handle its mistakes:

$$\mathscr{L}(\phi) = \frac{1}{N}\sum_{i=1}^{N} \|f(a_2^i, f(a_1^i, o_1^i; \phi); \phi) - o_3^i\| + \|f(a_1^i, o_1^i; \phi) - o_2^i\|$$



Action-Conditional Video Prediction using Deep Networks in Atari Games, *Oh et al.*

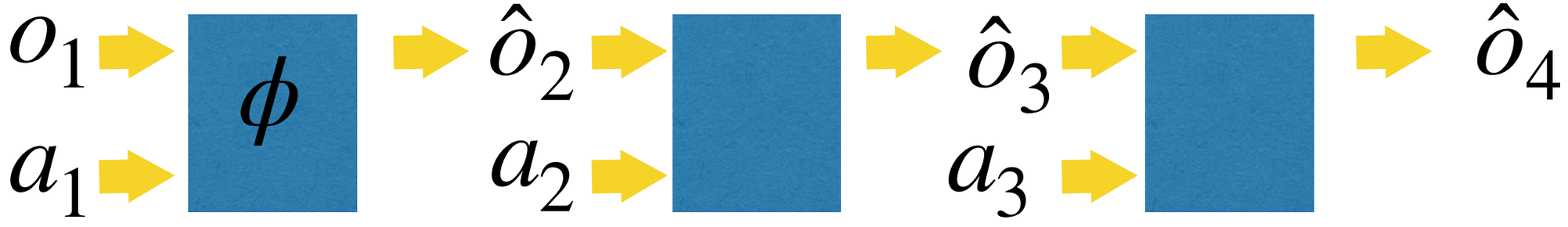
How to train our model so that unrolling works



Q: Can I train my model using tuples (o, a, o') and at test time unroll it over time?
A: No, we will have distribution shift, same as in imitation learning: tiny mistakes will soon cause the model to drift

Solution: Progressively increase the unrolling horizon k at training time so that the model learns to handle its mistakes:

$$\mathscr{L}(\phi) = \frac{1}{N}\sum_{i=1}^{N} \|f(a_3^i, f(a_2^i, f(a_1^i, o_1^i; \phi); \phi); \phi) - o_4^i\| + \|f(a_2^i, f(a_1^i, o_1^i; \phi); \phi) - o_3^i\| + \|f(a_1^i, o_1^i; \phi) - o_2^i\|$$



Action-Conditional Video Prediction using Deep Networks in Atari Games, *Oh et al.*

Small objects are missed, e.g., the bullets.

Q: Why?

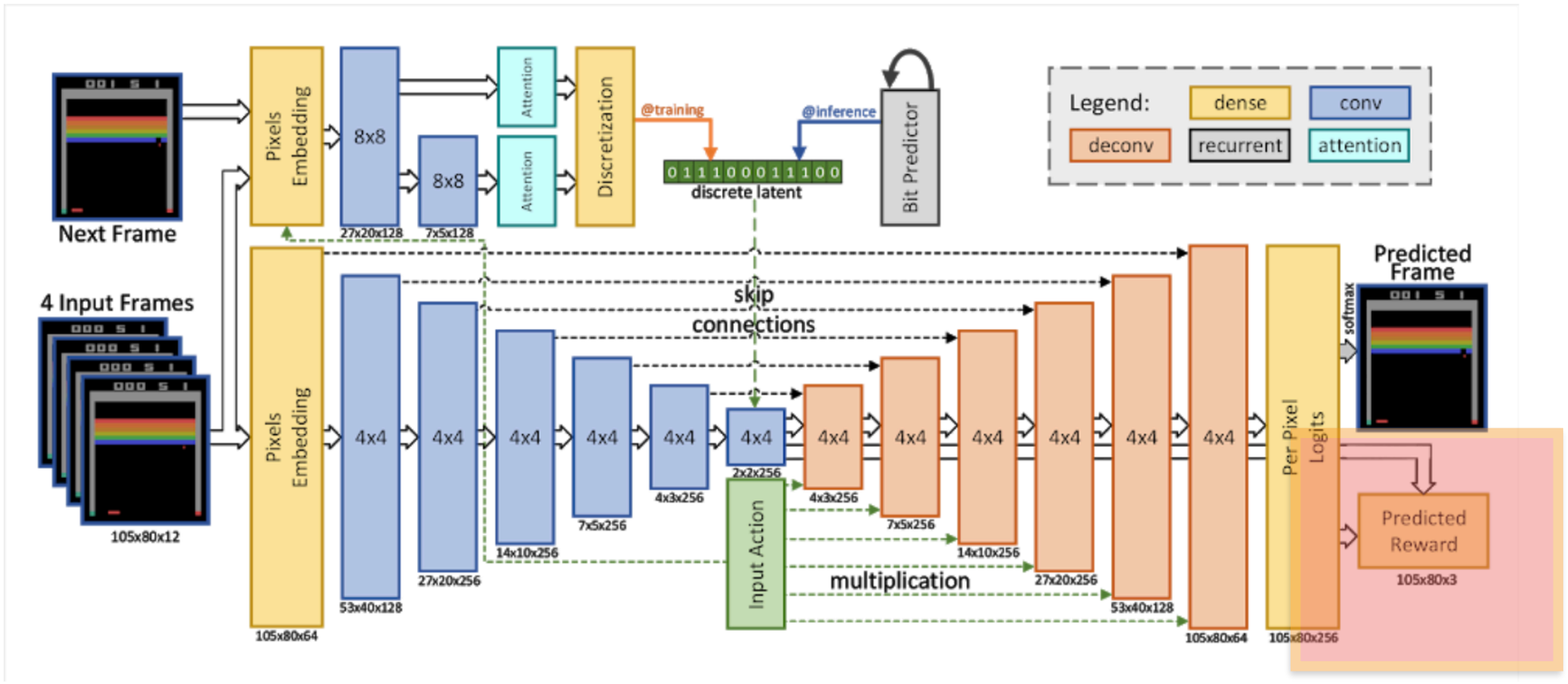
A:They induce a tiny mean pixel prediction loss (despite the fact they may be very task-relevant!)

How can we get the model error to predict accurately the part of the observation relevant for the task and neglect irrelevant details?

Model-Based Reinforcement Learning for Atari

Łukasz Kaiser **Ryan Sepassi** Google Brain
Henryk Michalewski **Piotr Miłoś** University of Warsaw
Błażej Osiński deepsense.ai

Similar architecture as before but..we also predict rewards!



Model-baser RL in Atari

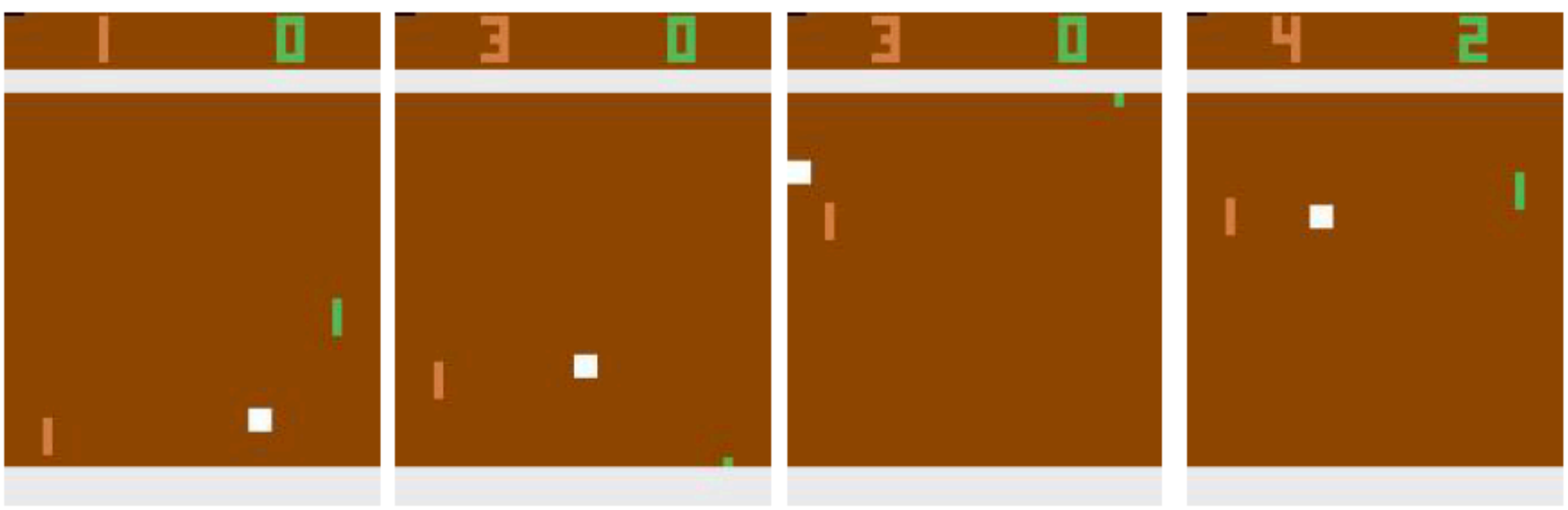
Initialize policy $\pi(s;\theta)$ and $D_{env} = \{\}$.

1. Run the policy and update experience tuples dataset D_{env}.
2. Train the dynamics model using D_{env} $(o_{t+1}, r_{t+1}) = f(o_{t-4..t}, a_t, z; \phi)$
3. Sample $o_{t-4} \ldots o_t$ from the experience buffer and then collect simulated experience D_{model} by unrolling the model for 50 time steps using the policy π_θ to select actions.
4. Update the policy using PPO on trajectories in $D_{model} + D_{env}$.
5. GOTO 1.

- The model is stochastic with discrete latent variables: DISCRETE AUTOENCODERS FOR SEQUENCE MODELS
- $D_{model} >> D_{env}$
- Note the short rollouts from sampled states from the experience buffer just like in MBPO

Reward-aware model learning loss

- We train the dynamics model to generate a future sequence so that the rewards obtained from the simulated sequence agree with the rewards obtained in the ``real" (videogame) world. I put L2 loss on the rewards as opposed to just on pixels. This encourages to focus on objects that are too small and incur a tiny L2 pixel loss, but may be important for the game.
- (Nonetheless, they made the ball larger :-( )



results

Results

- Number of frames required to reach human performance

	PPO	Model-based
Breakout	800K	120K
Pong	1000K	500K
Freeway	200K	10K

results