Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

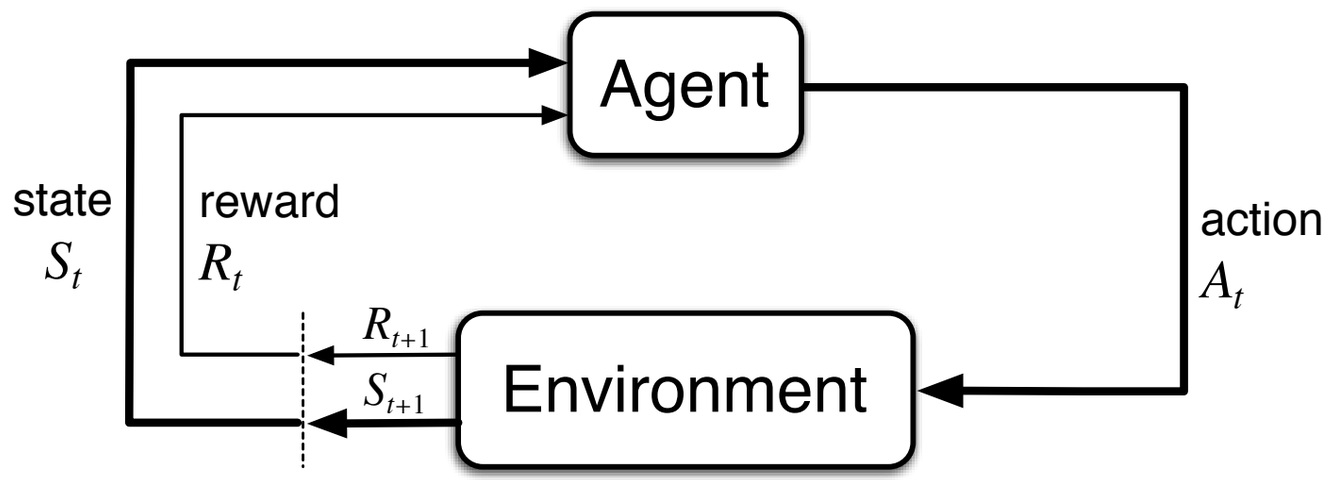
# Model Based Reinforcement Learning

Spring 2020, CMU 10-403

Katerina Fragkiadaki

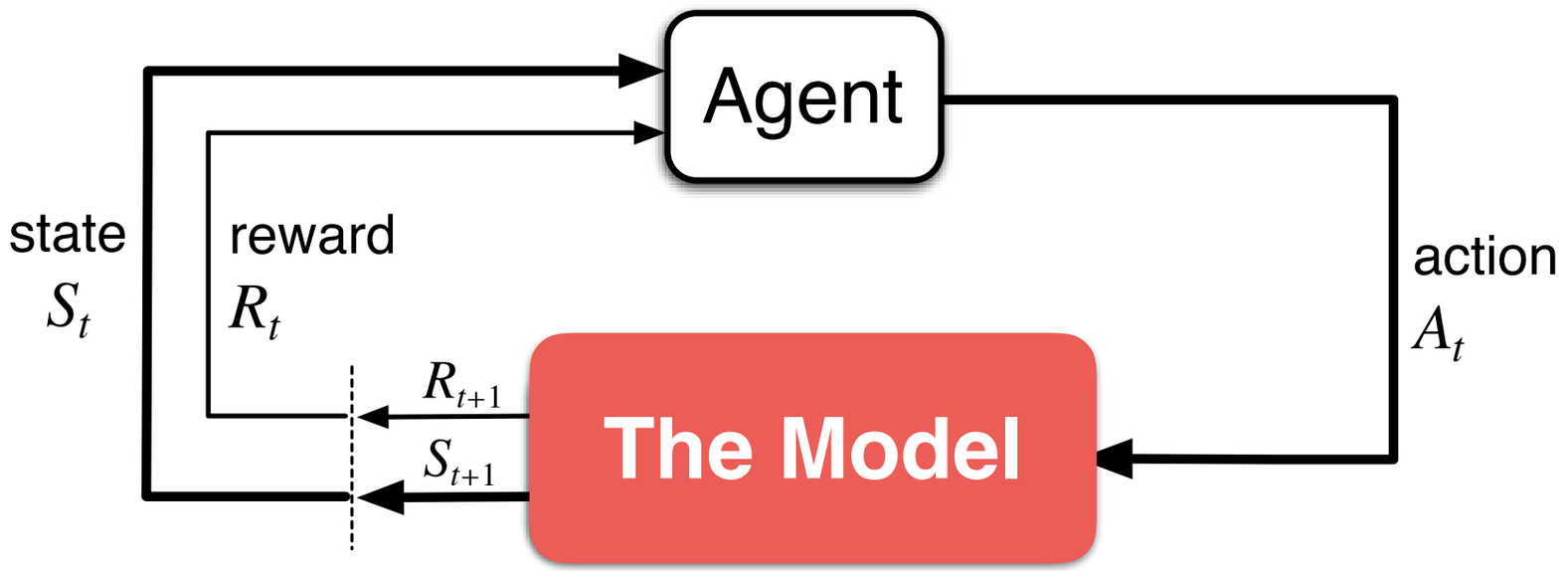
# Planning

**Planning**: any computational process that uses a model to create or improve a policy

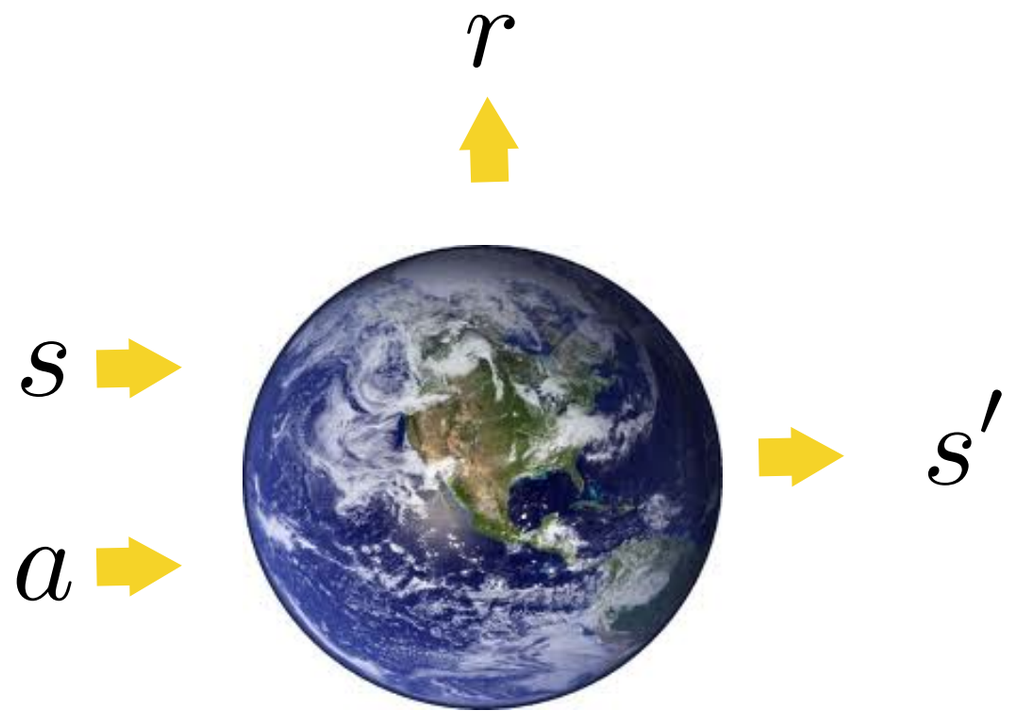
# Planning

**Planning**: any computational process that uses a model to create or improve a policy



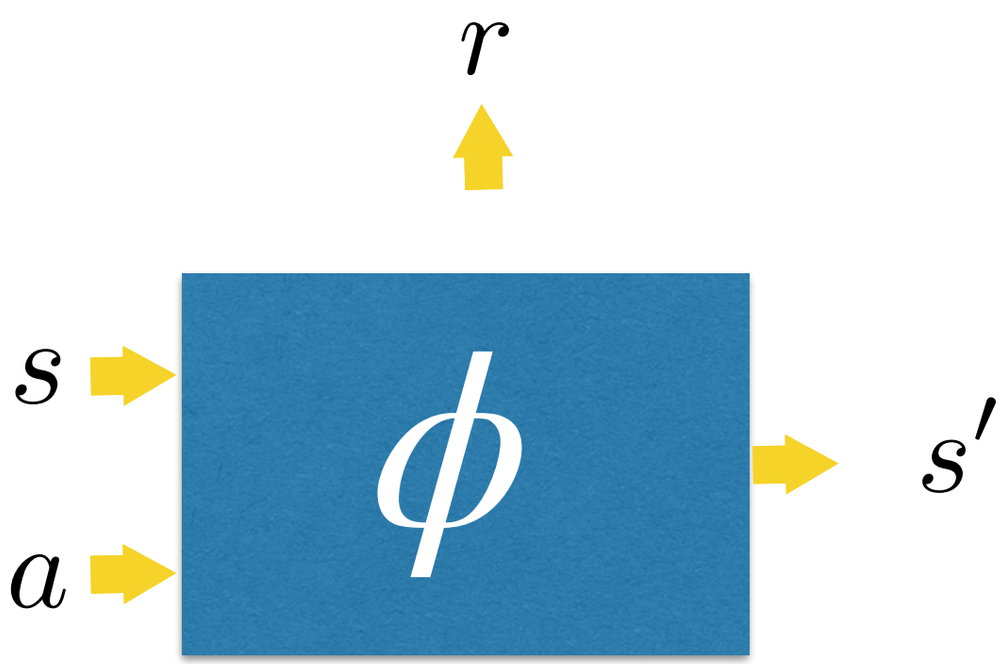
# Model

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



# Model learning

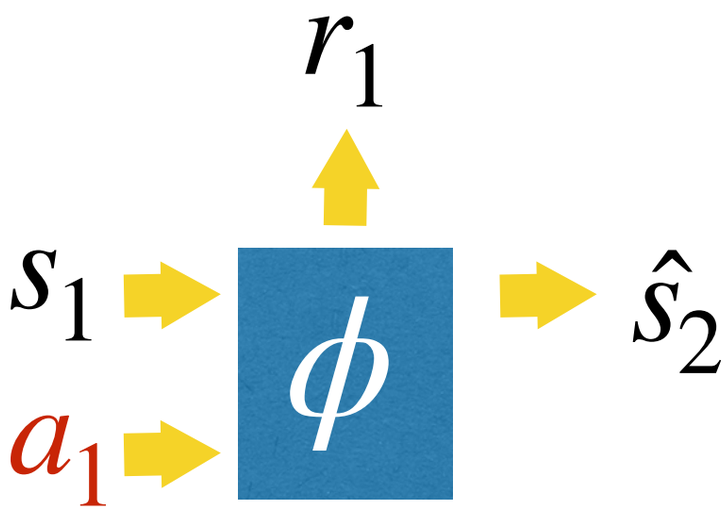
We will be learning the model using experience tuples. A supervised learning problem.




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

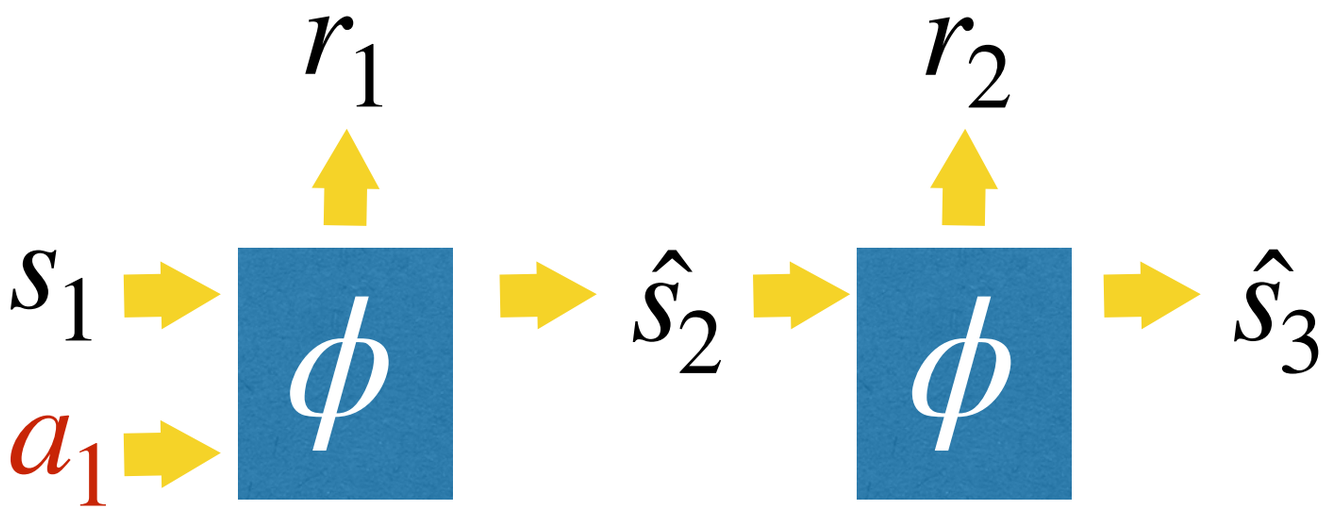
# Model unrolling

Predict long term effects of actions



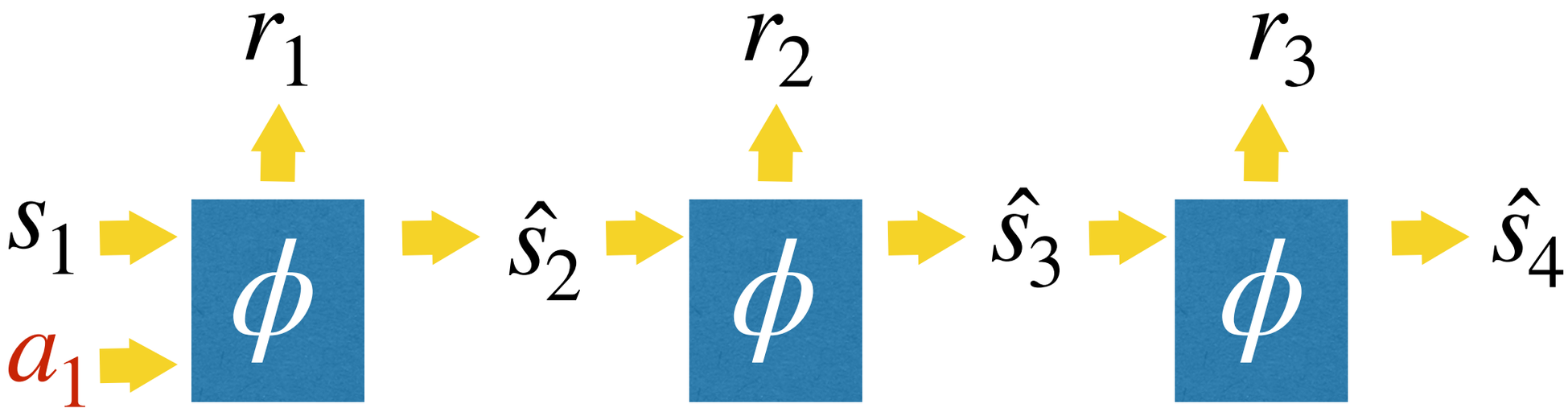
# Model unrolling

Predict long term effects of actions



# Model unrolling

Predict long term effects of actions



# Model learning

Newtonian Physics equations **VERSUS** general parametric forms

# Model learning

Newtonian Physics equations

VERSUS

general parametric forms

The dynamics equations are given and we only need to estimate few unknown parameters (a.k.a. system identification)

Using general neural networks: lots of unknown parameters, generic structure

# Model learning

Newtonian Physics equations

VERSUS

general parametric forms

The dynamics equations are given and we only need to estimate few unknown parameters (a.k.a. system identification)

Using general neural networks: lots of unknown parameters, generic structure

Easy to learn but suffers from under-modeling

Very flexible, very hard to get it to generalize

# Where can models be useful

1. In model-based control: given an initial state $s_0$, estimate a sequence of actions to reach a desired goal or maximize sum of rewards by unrolling the model forward in time.

2.In model-based RL: train policies by

    1. a model-free RL method that samples experience tuples using the model (simulated experience)
    2. imitating the model-based controller in (1).

3. In curiosity-driven exploration guided by model uncertainty (later lecture)

# Where can models be useful

1. In model-based control: given an initial state $s_0$, estimate a sequence of actions to reach a desired goal or maximize sum of rewards by unrolling the model forward in time.

2. In model-based RL: train policies by

    1. a model-free RL method that samples experience tuples using the model (simulated experience)
    2. imitating the model-based controller in (1).

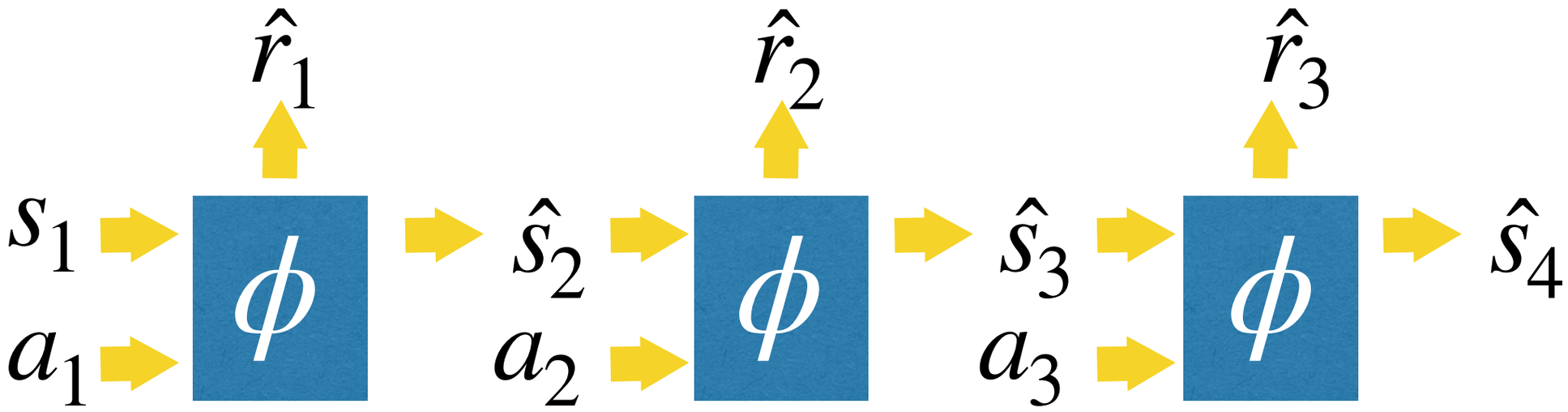
3. In curiosity-driven exploration guided by model uncertainty (later lecture)

# 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.

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

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

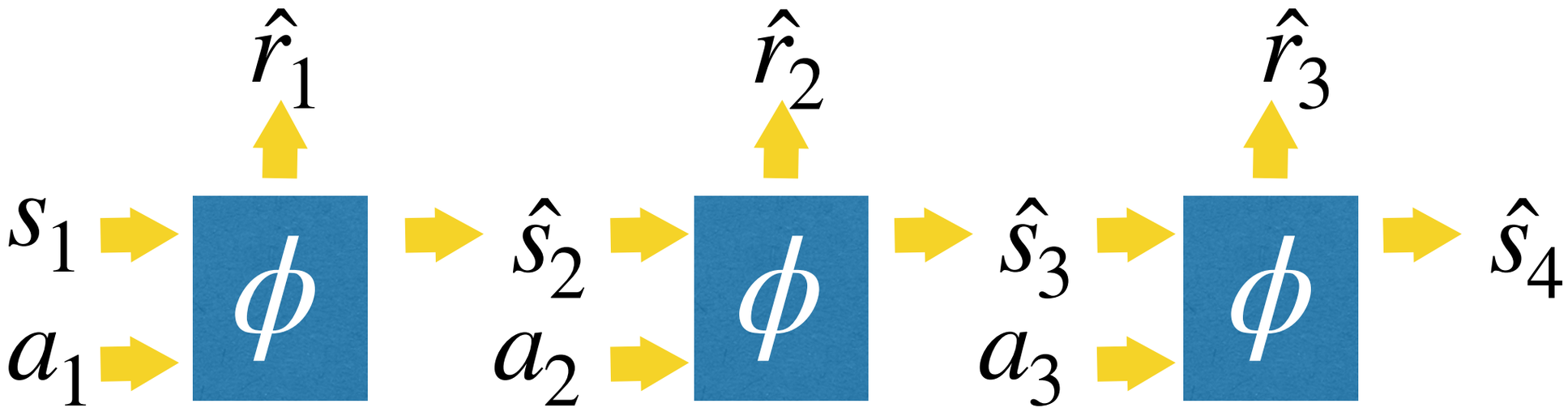


# Model-based control

$s_0$ 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.

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

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



# Model-based control

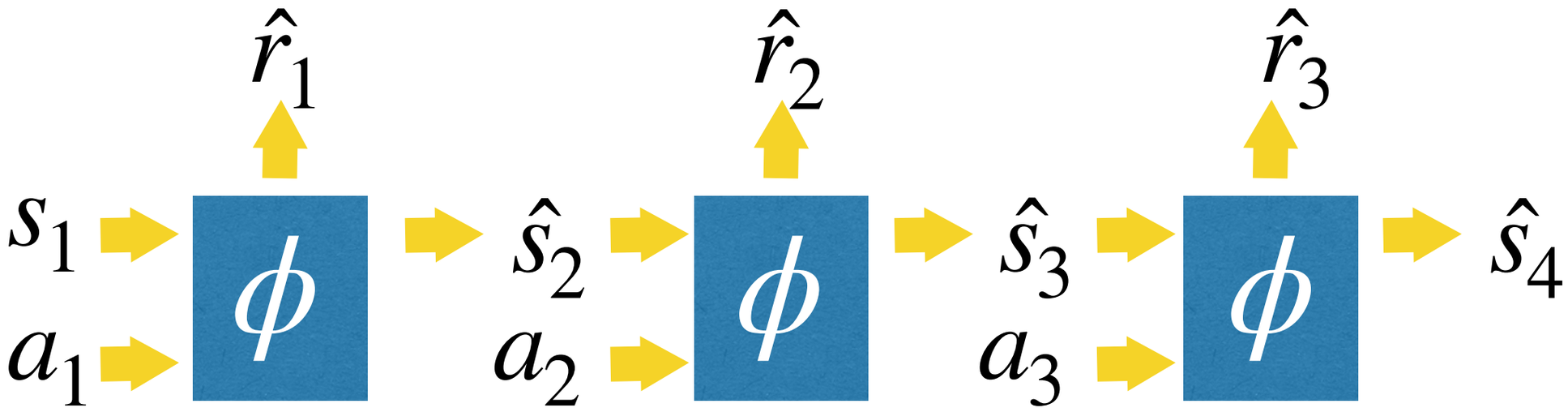
$s_0$ Given an initial state, estimate a sequence of actions to reach a desired goal or maximize sum of rewards by unrolling the model forward in time.

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

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

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

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



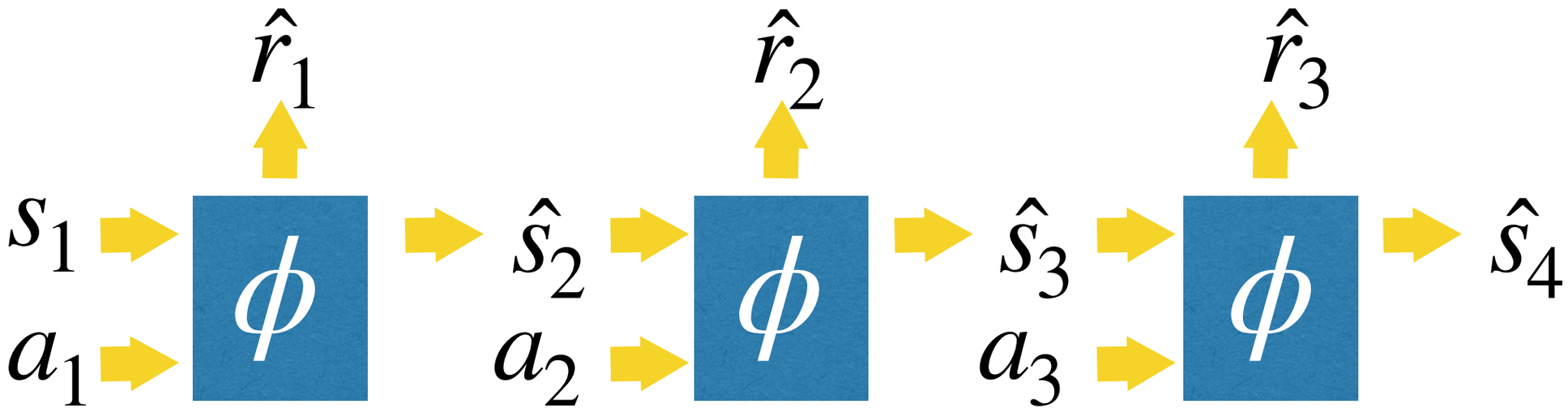
# Model-based control

$s_0$ 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.

$$\min_{a_1 \cdots a_T} . \ \|s_T - s_*\| \qquad \max_{a_1 \cdots a_T} . \ \sum_{t=1}^{T} r_t$$

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

If the dynamics are non-linear and the loss is not a quadratic, this optimization is difficult. We can use SGD or evolutionary search.



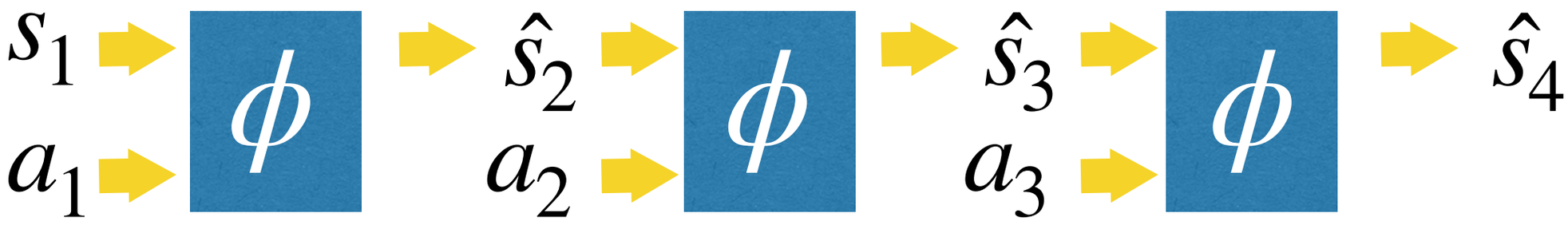
# Model-based control - SGD

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and computer error against a desired final state
4. Backpropagate the error to the action sequence

$a_1$ $a_2$ $a_3$

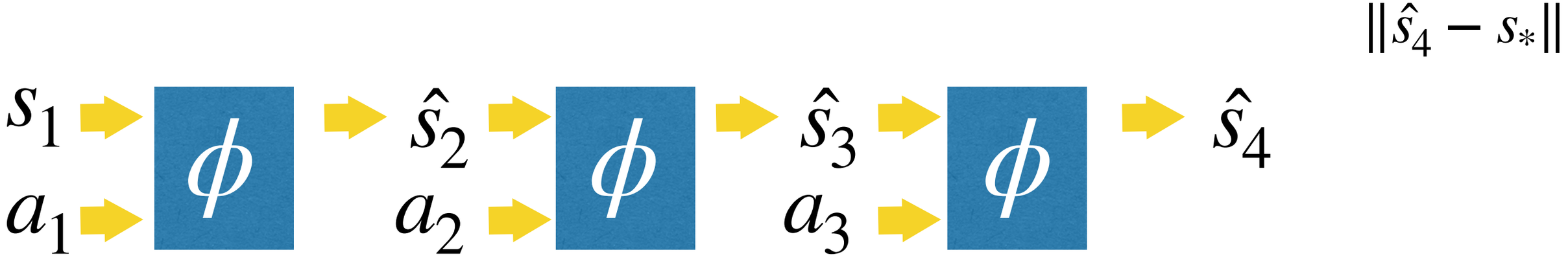
# Model-based control- SGD

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and computer error against a desired final state
4. Backpropagate the error to the action sequence



# Model-based control- SGD

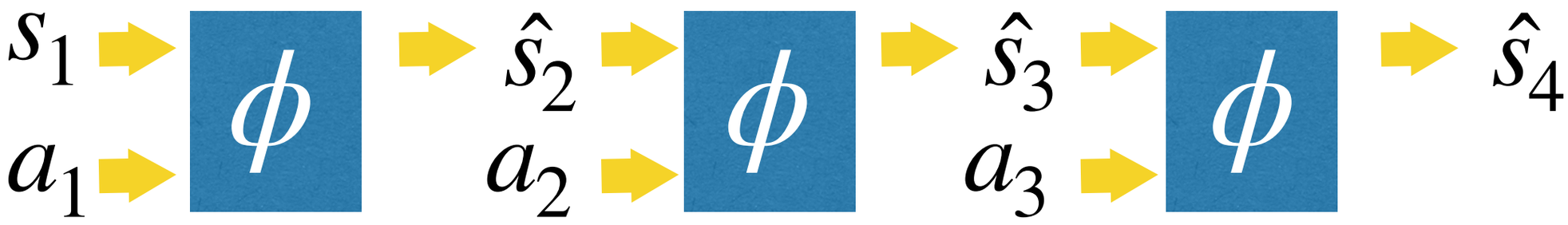
1.Given an initial action sequence
2.Unroll the model forward in time
3.Compare and computer error against a desired final state
4.Backpropagate the error to the action sequence



# Model-based control - SGD

1.Given an initial action sequence
2.Unroll the model forward in time
3.Compare and computer error against a desired final state
4.Backpropagate the error to the action sequence

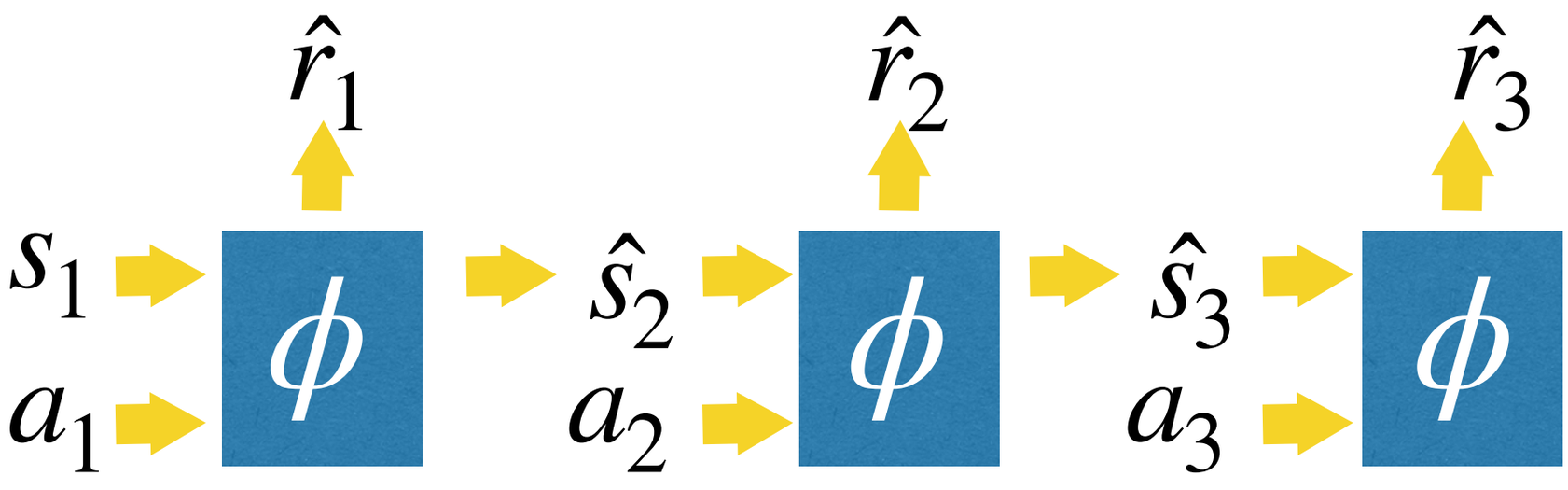
$$\min_{a_1 \cdots a_T} . \ \|s_T - s_*\|$$



# Model-based control - SGD

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compute sum of estimated rewards
4. Backpropagate the error to the action sequence

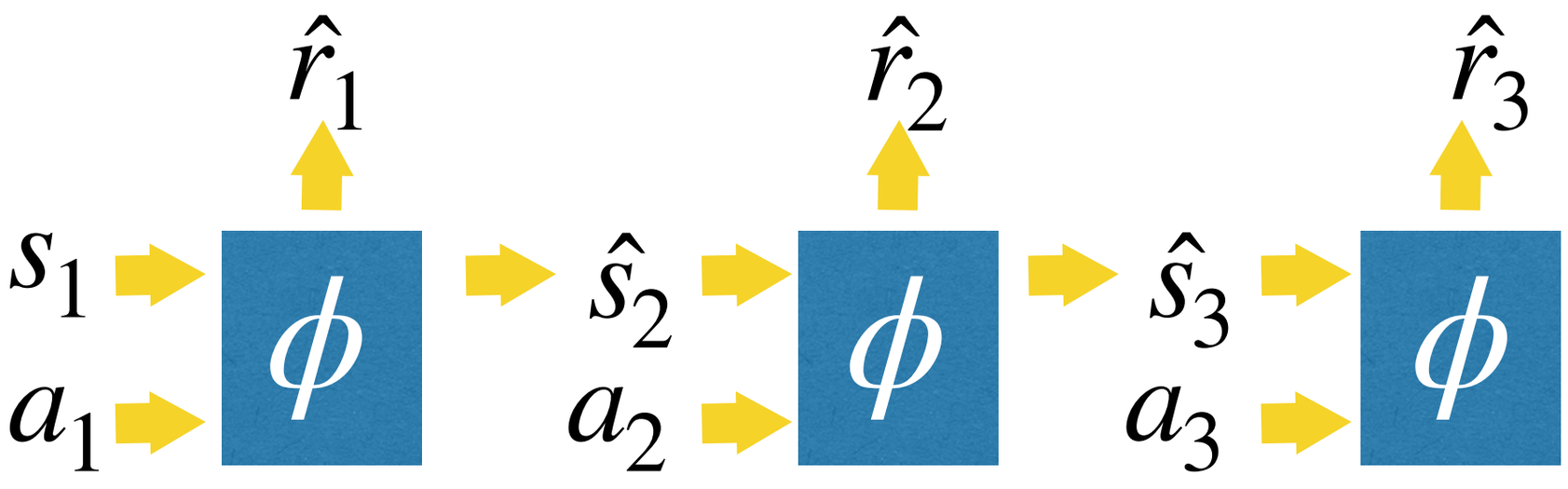
$$\max_{a_1 \cdots a_T} . \sum_{t=1}^{T} \hat{r}_t$$



# Model-based control - SGD

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compute sum of estimated rewards
4. Backpropagate the gradient to the action sequence

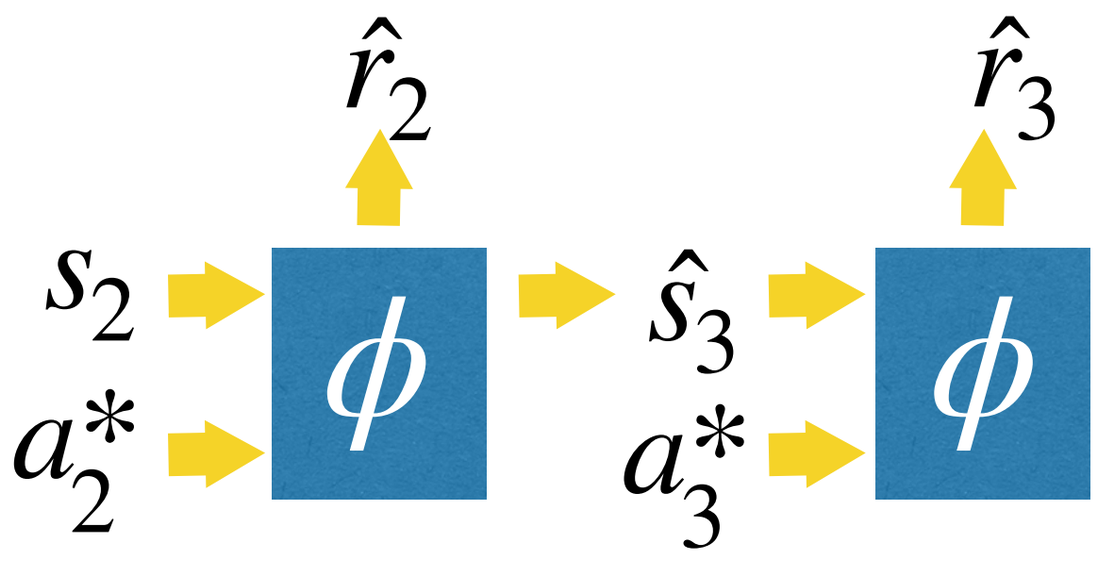
$$\max_{a_1 \cdots a_T} . \sum_{t=1}^{T} \hat{r}_t$$



# Model-predictive control

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compute sum of estimated rewards
4. Backpropagate the gradient to the action sequence
5. Execute **only the first action**
6. **GOTO 1**

$$\max_{a_1 \cdots a_T} \sum_{t=1}^{T} \hat{r}_t$$

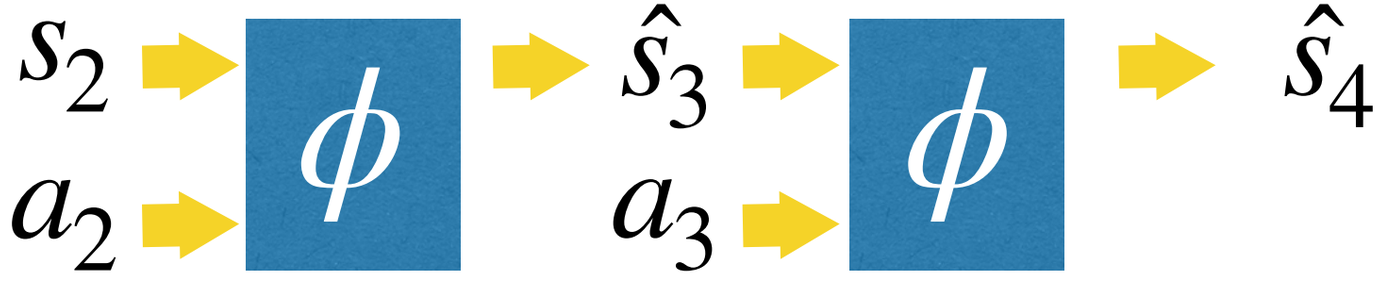



Q: Are the predicted rewards and future states the same as in the previous slide?

# Model-predictive control

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compute sum of estimated rewards
4. Backpropagate the gradient to the action sequence
5. Execute **only the first action**
6. **GOTO 1**

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



# Model-based control - derivative free

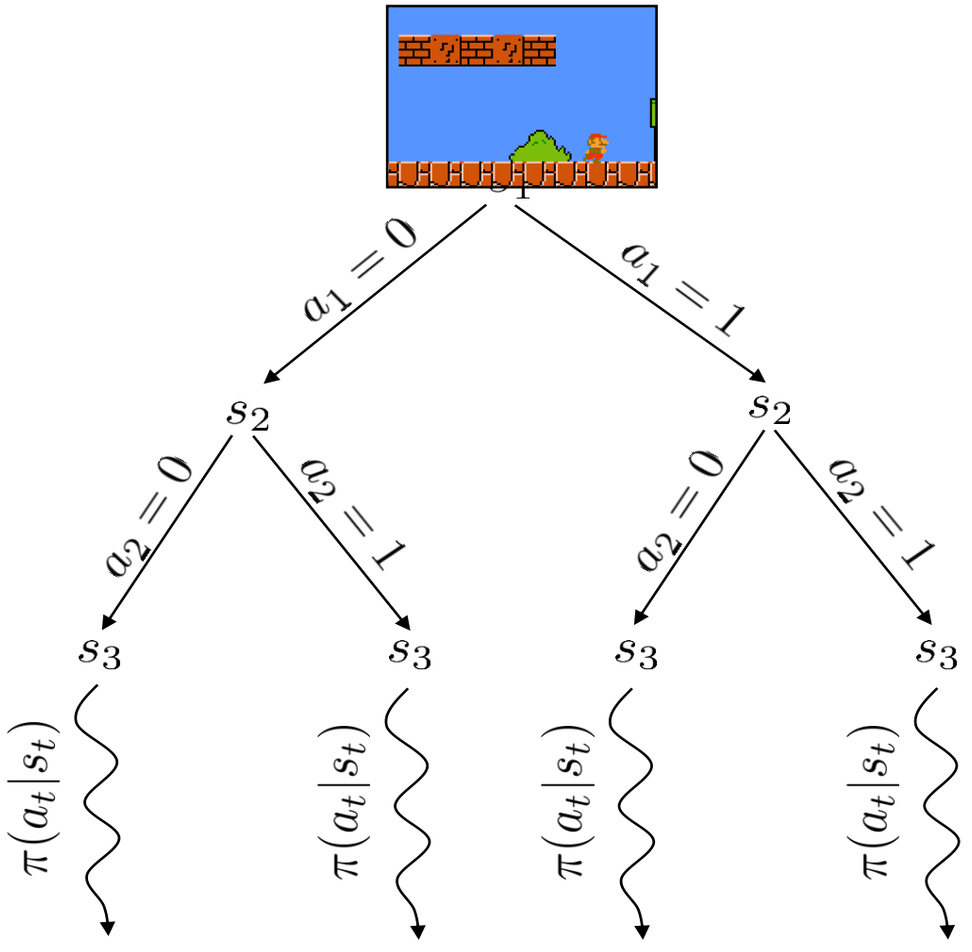
$$\min_{a_1 \cdots a_T} . \ \|\hat{s}_T - s_*\| \quad \text{s.t.} \ . \forall t, \ s_{t+1} = f(s_t, a_t; \phi)$$

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

Optimize over action selection using CMA-ES or CEM (sample actions, unroll, compute error, survival of the fittest, repeat) or discretize our actions and do tree search or MCTS.

Q: why would we use evolution instead of SGD?


$a_t$



# Why model learning

1. In model-based control: given an initial state $s_0$, estimate a sequence of actions to reach a desired goal or maximize sum of rewards by unrolling the model forward in time.

2. In model-based RL: train policies by

    1. a model-free RL method that samples experience tuples using the model (simulated experience)
    2. imitating the model-based controller in (1).

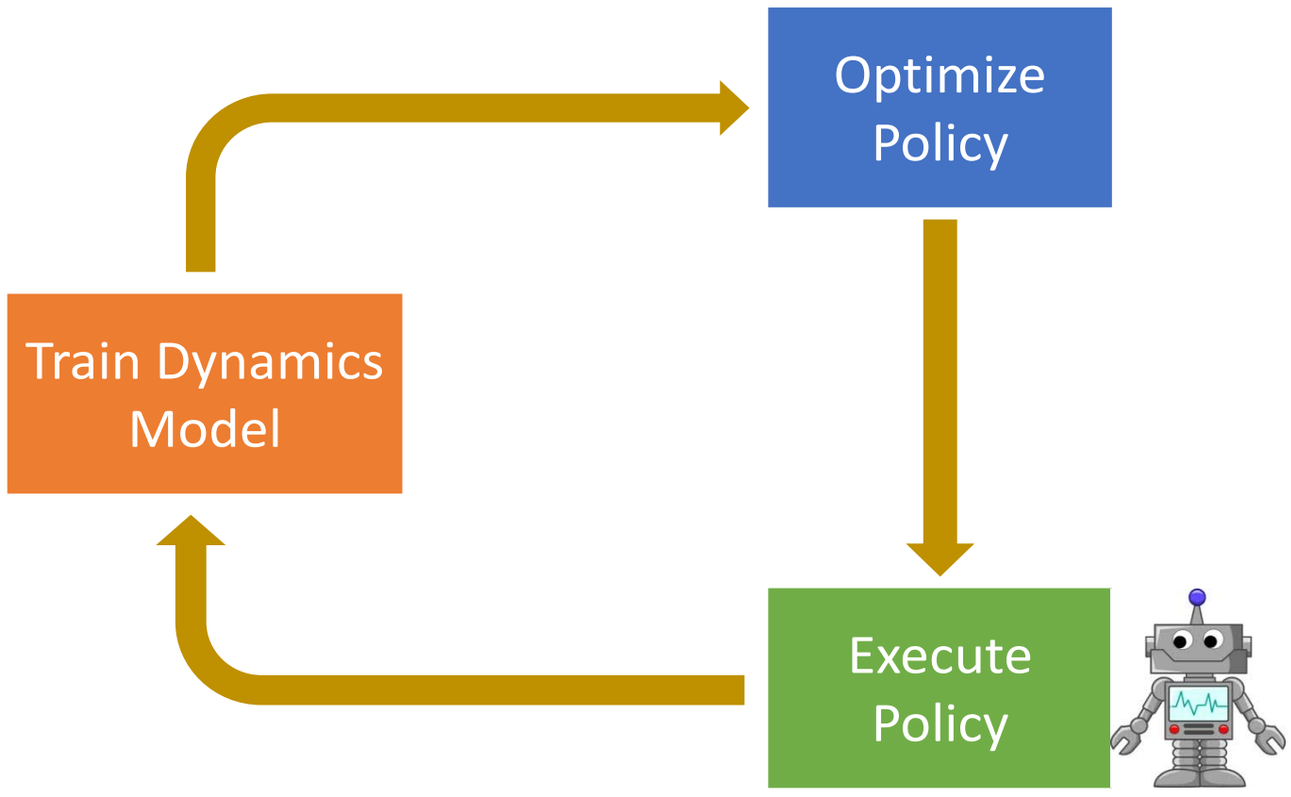
3. In curiosity-driven exploration guided by model uncertainty (later lecture)

# When models are learnt

# Alternating between model and policy learning

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

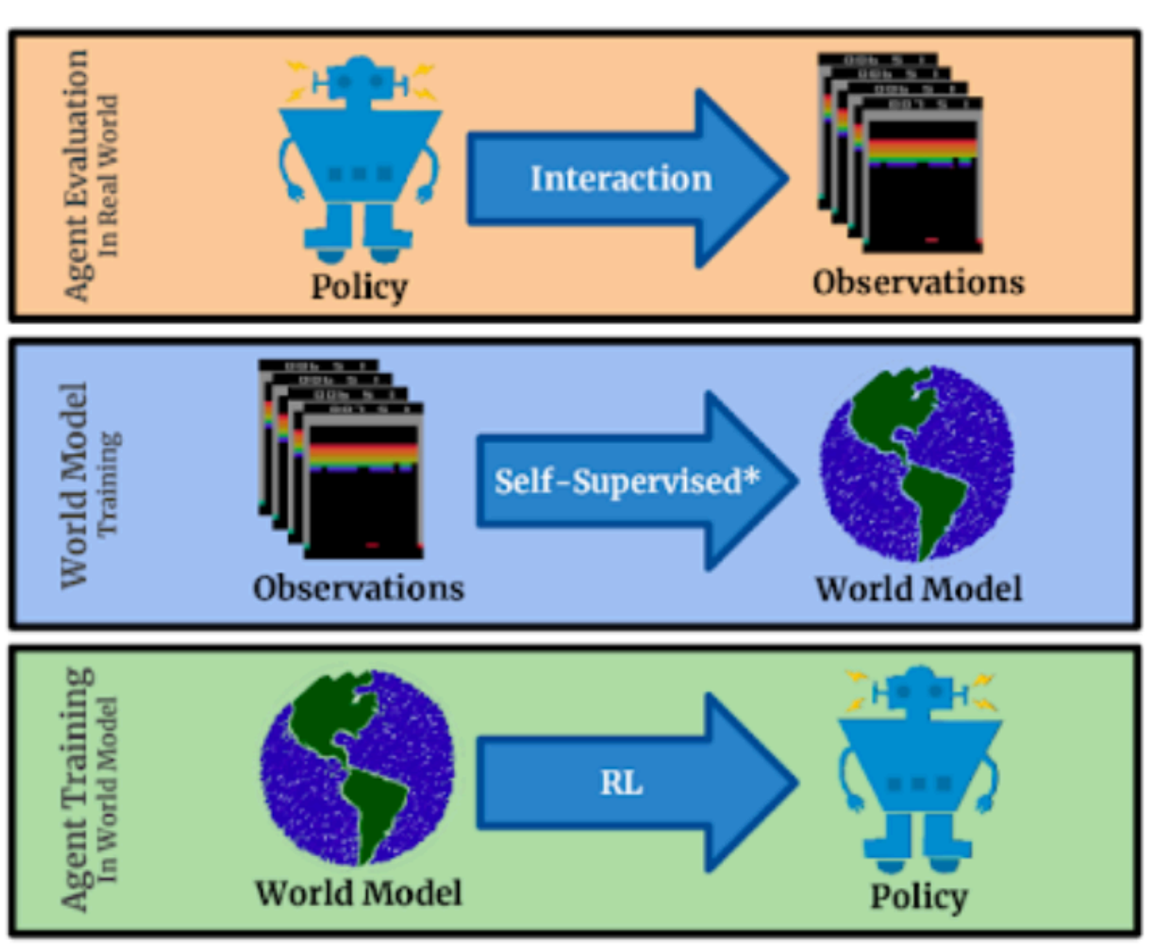
1. Run the policy and update experience tuples dataset $D$.
2. Train a dynamic model using $D$: $(s', r') = f(s, a; \phi)$
3. Update the policy by
    1. a model-free RL method on simulated experience sampled from the model
    2. Imitating a model-based controller
4. GOTO 1.



# Alternating between model and policy learning

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

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

# 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

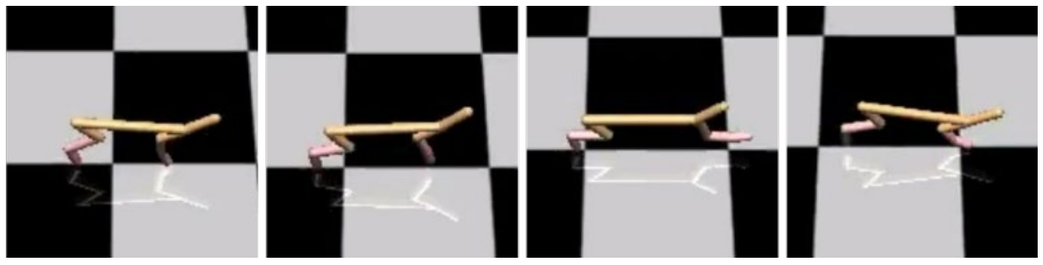
- Errors compound through unrolling

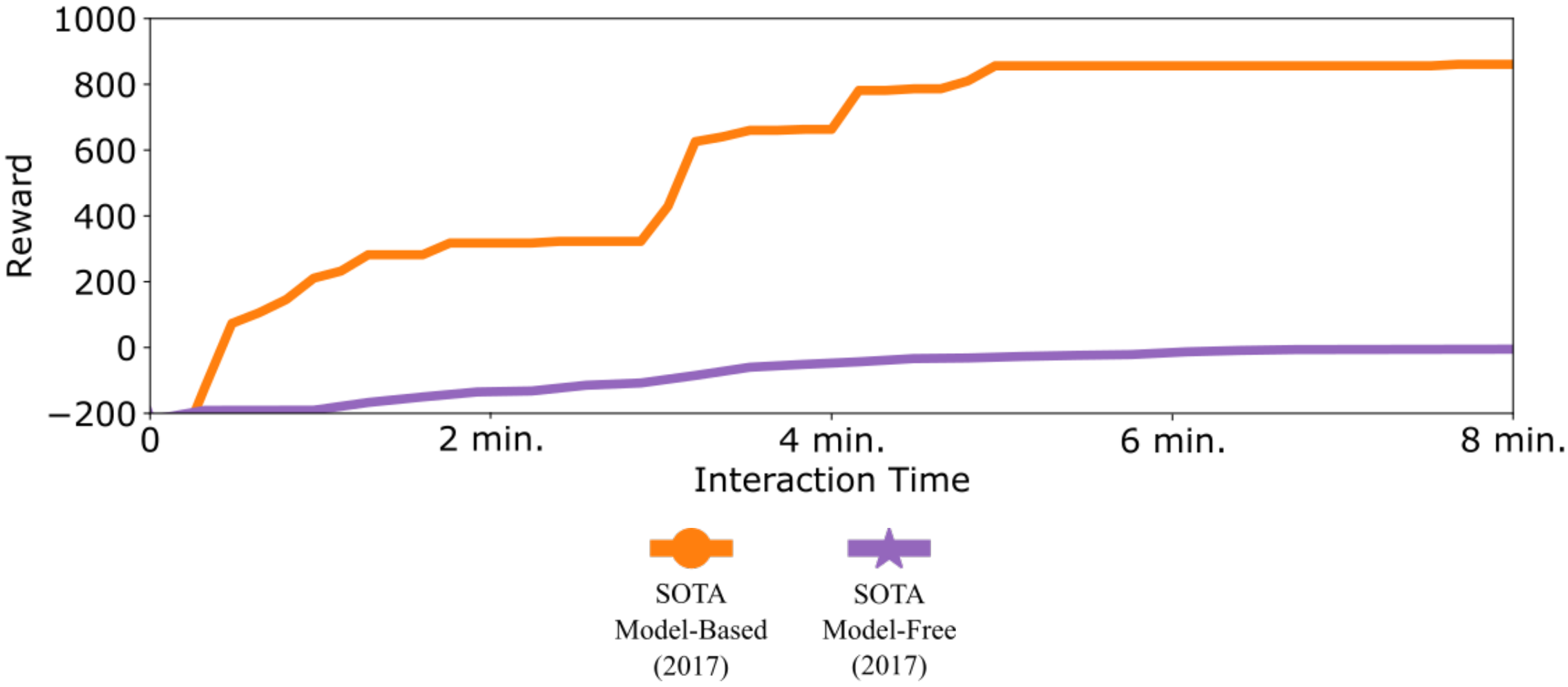
- Need to capture different futures (stochasticity of the environment)

- Need to represent uncertainty outside of the training data

- Action selection on top of model unrolling will surely exploit mistakes of the model.

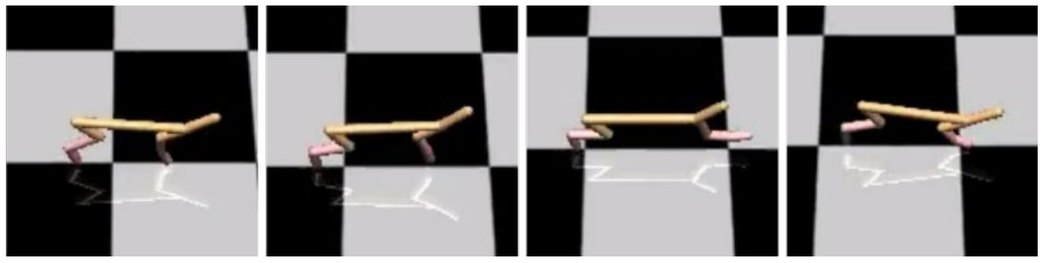
# Comparative Performance on HalfCheetah

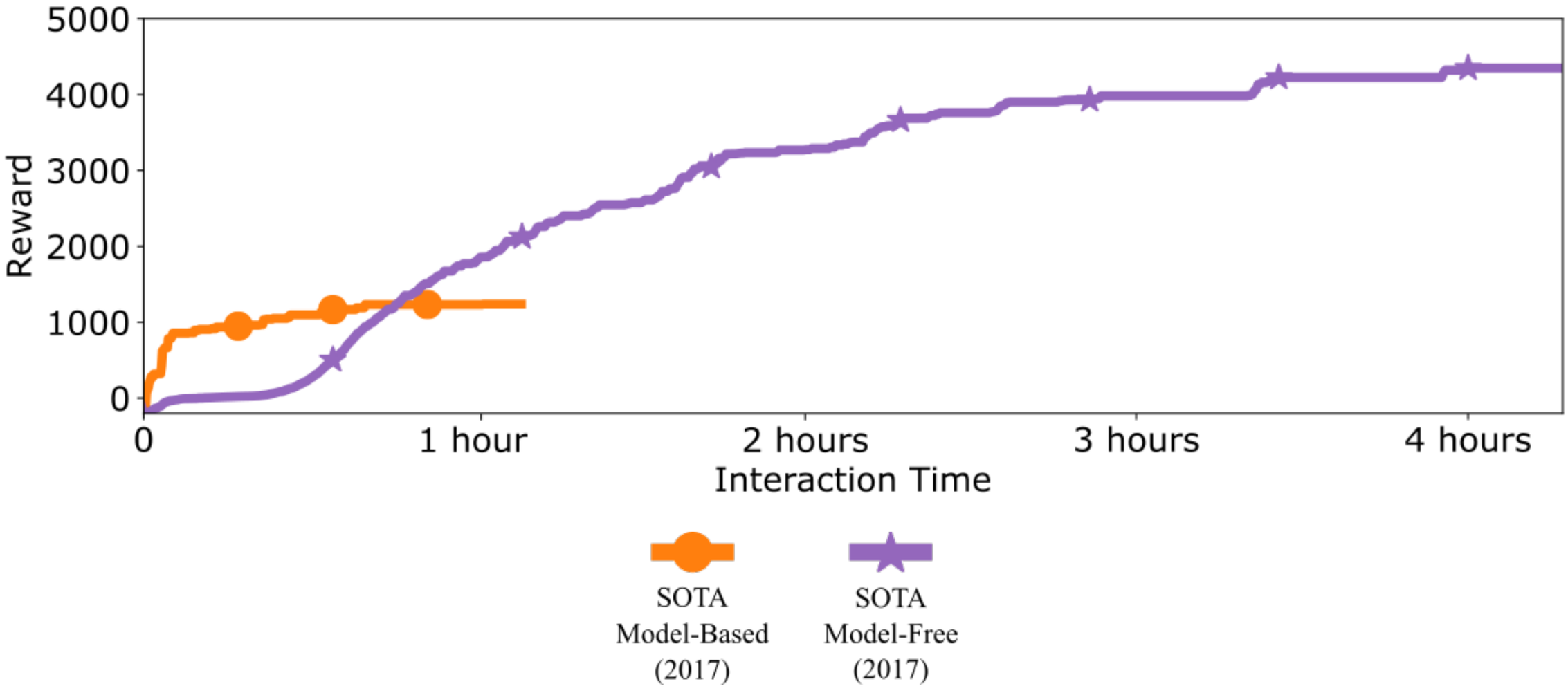





slide from Sergey Levine

# Comparative Performance on HalfCheetah

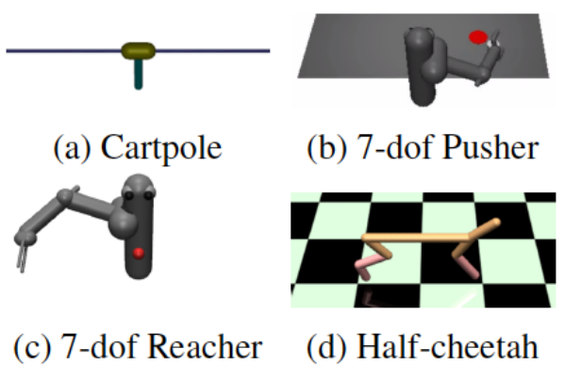


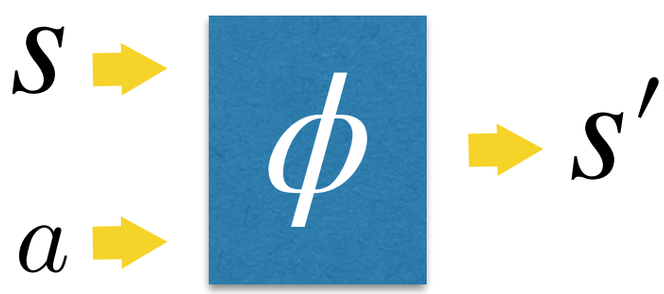



slide from Sergey Levine

# Model Learning

Where a low dimensional state is observed and given:

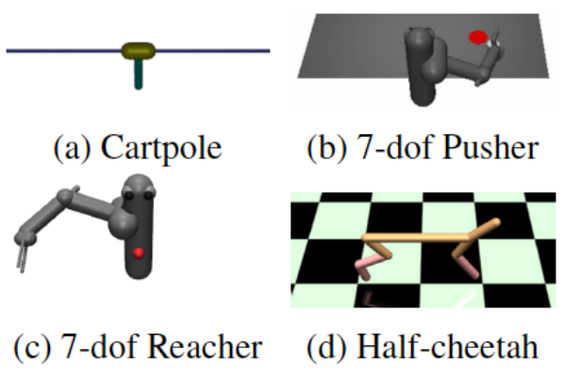



state can be 3D locations and 3D velocities of agent joints, actions can be torques

# Model Learning

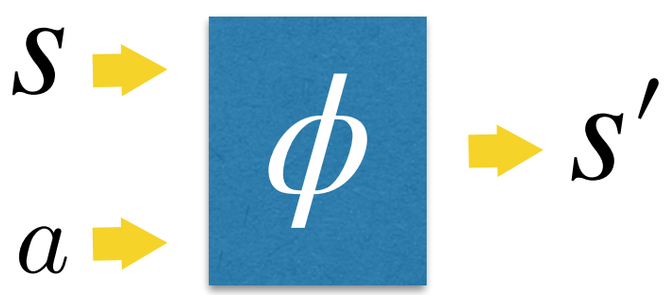
Where a low dimensional state is observed and given:



(a) Cartpole (b) 7-dof Pusher

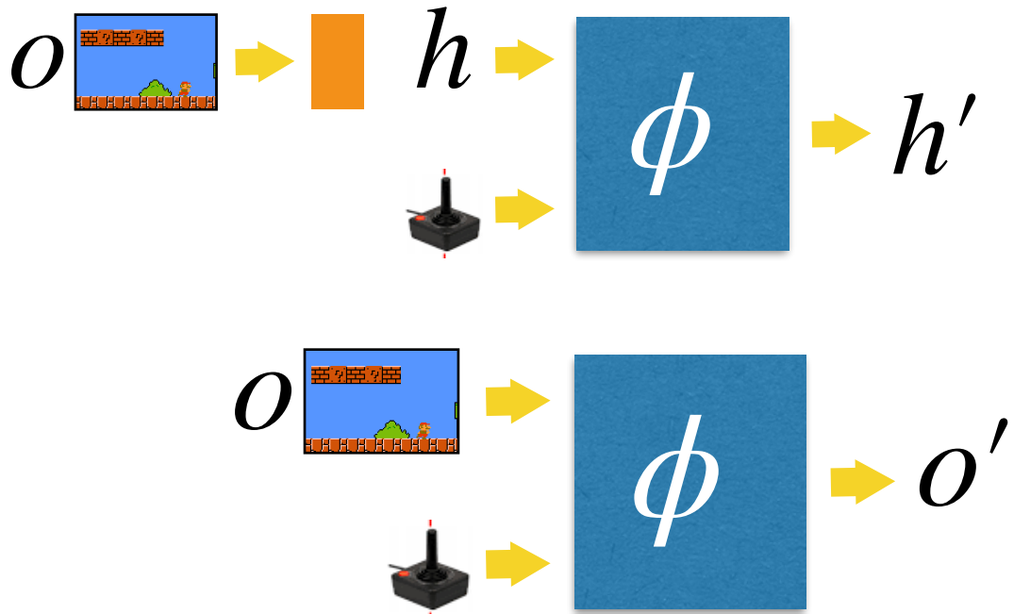
(c) 7-dof Reacher (d) Half-cheetah

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



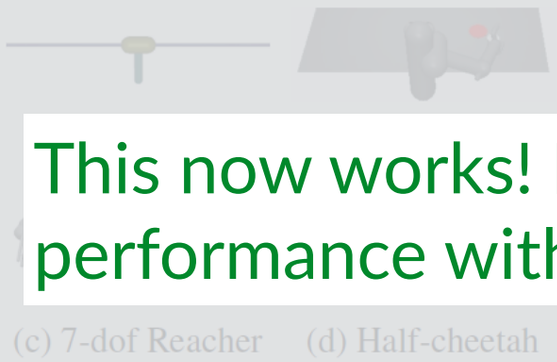

Where we only have access to (high dim) sensory input, e.g., images:

e.g., Atari game playing



# Model Learning

- Where a low dimensional state is observed and given:



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

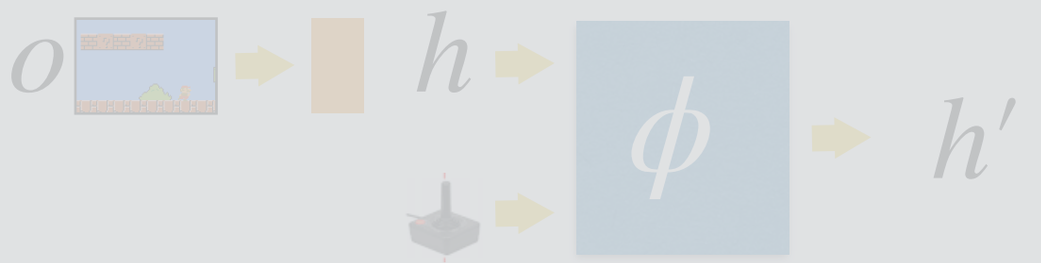
state can be 3D locations and 3D velocities of agent joints, actions can be torques

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

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

e.g., Atari game playing

Still an open problem :-(




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

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

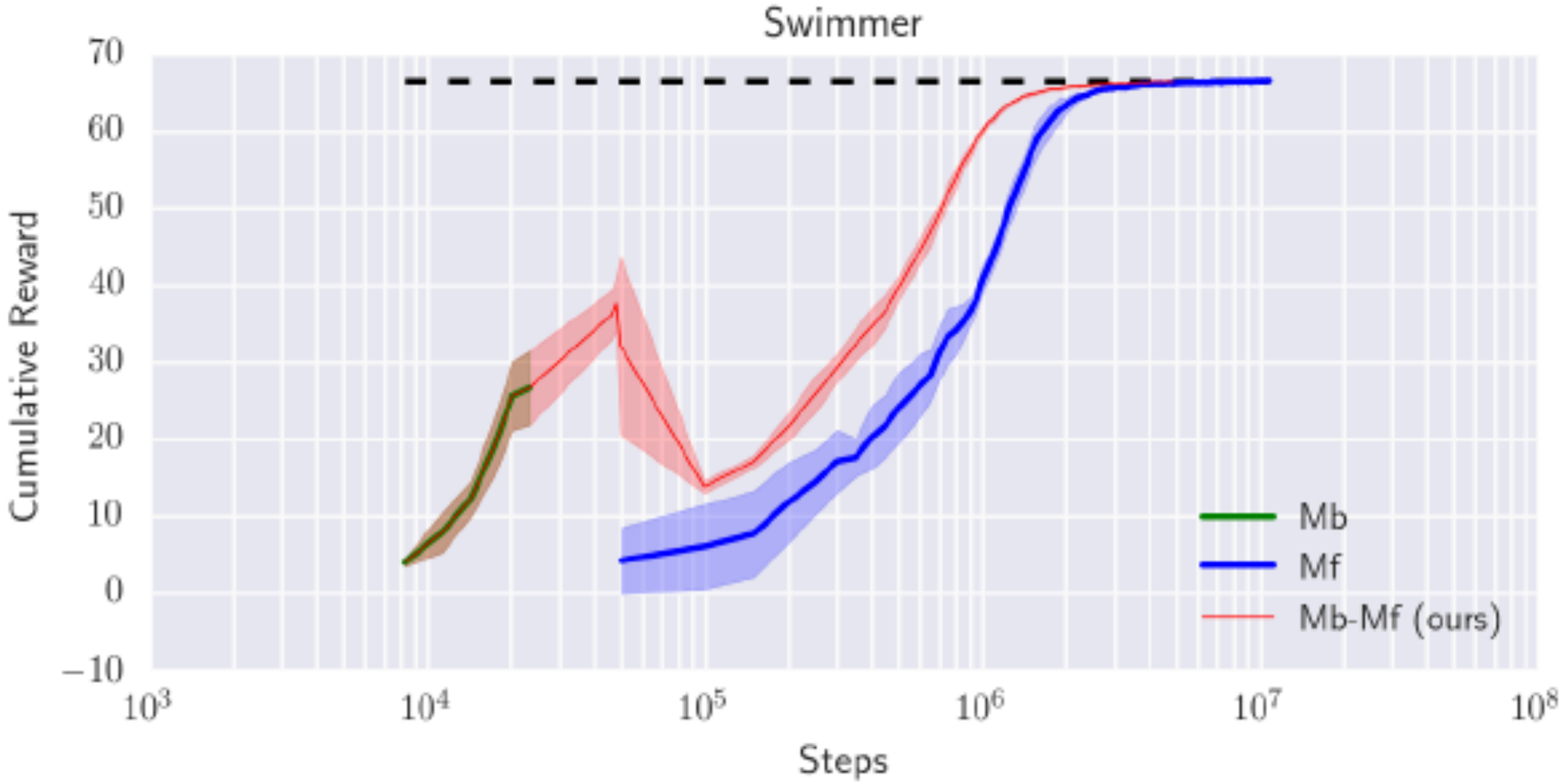
2.What are the right state representations?

3.How can we handle multimodality/ uncertainty?

# Model-based RL in a low-dim state space

# 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

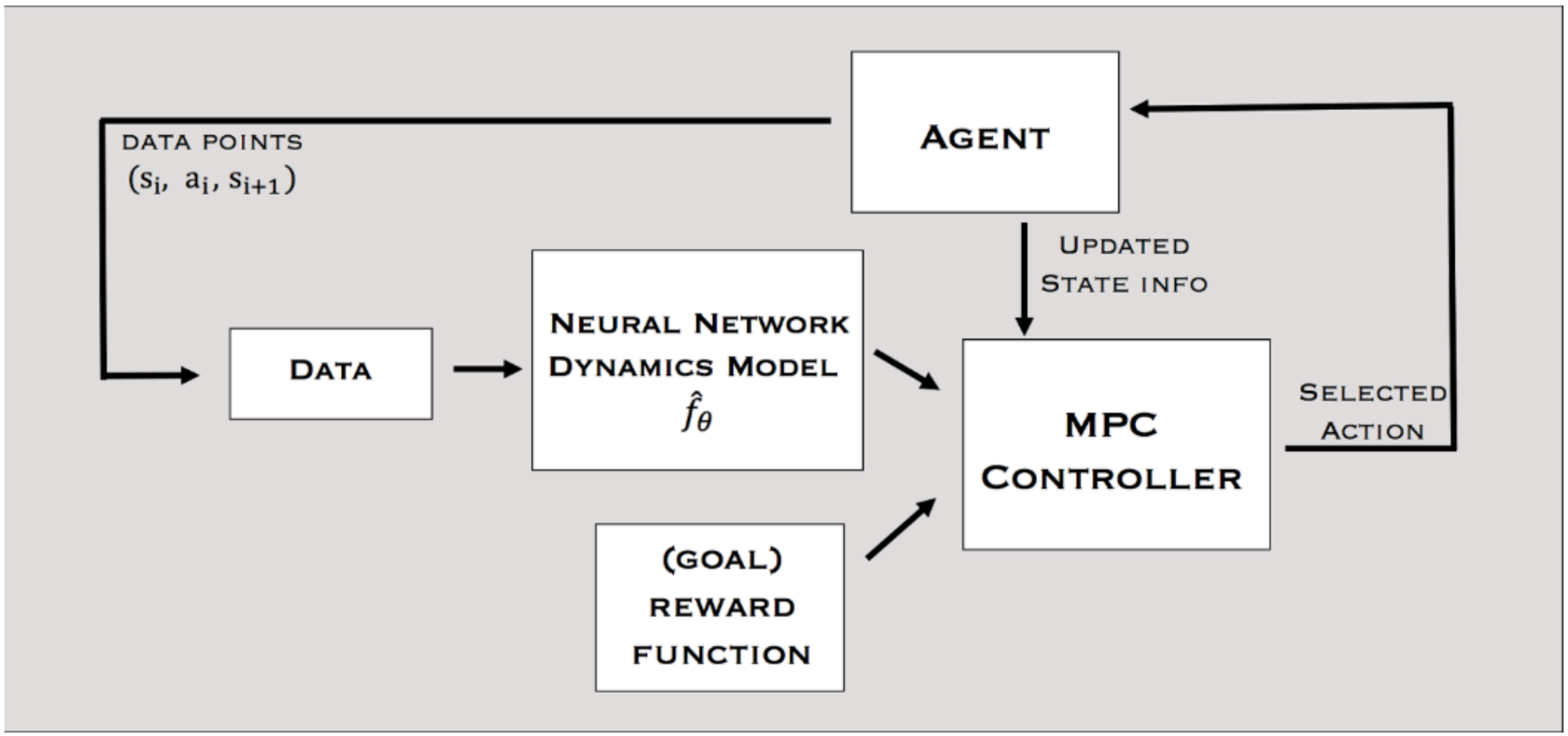



Neural network dynamics for model-based Deep RI with model-free finetuning, Nagabandi et al.

# Model-based RL

Collect a dataset D of random experience tuples $(s, a, s')$

1. Train transition dynamics $s' = s + f(s, a; \phi)$
2. Optimize action sequences using MPC with random search
3. **Aggregate** experience dataset with the inferred $(s, a)$ sequences
4. GOTO 1




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

# Model-based RL with model-free finetuning

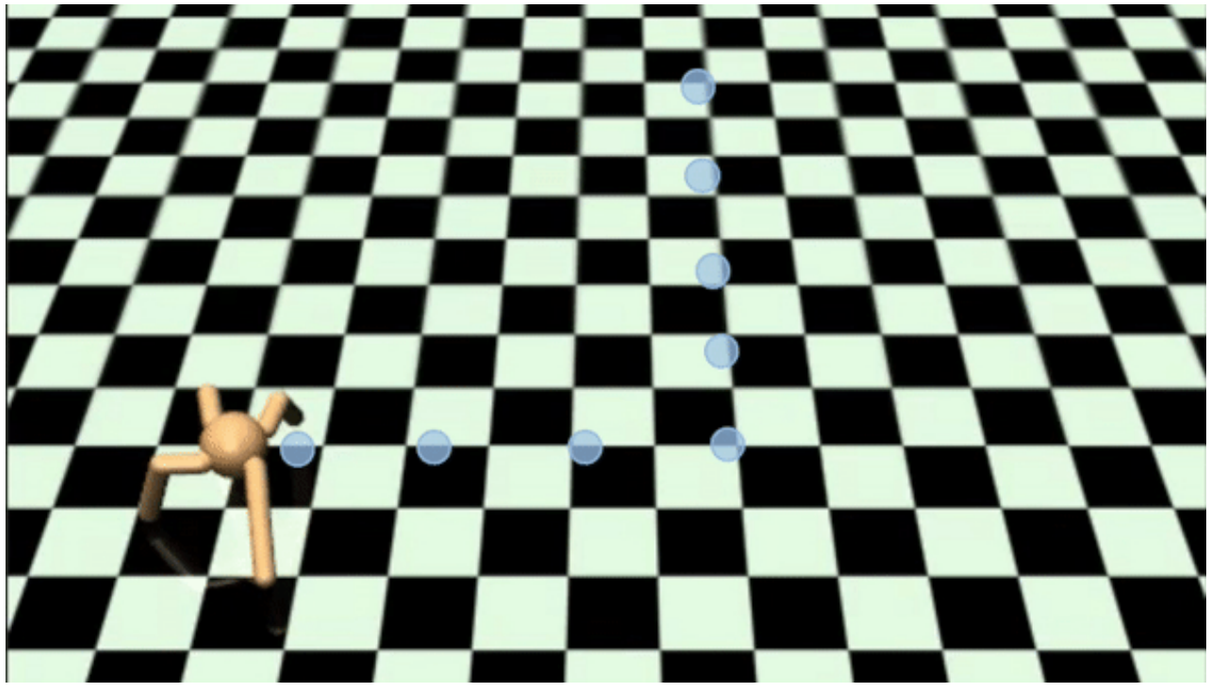
Collect a dataset D of random experience tuples $(s, a, s')$

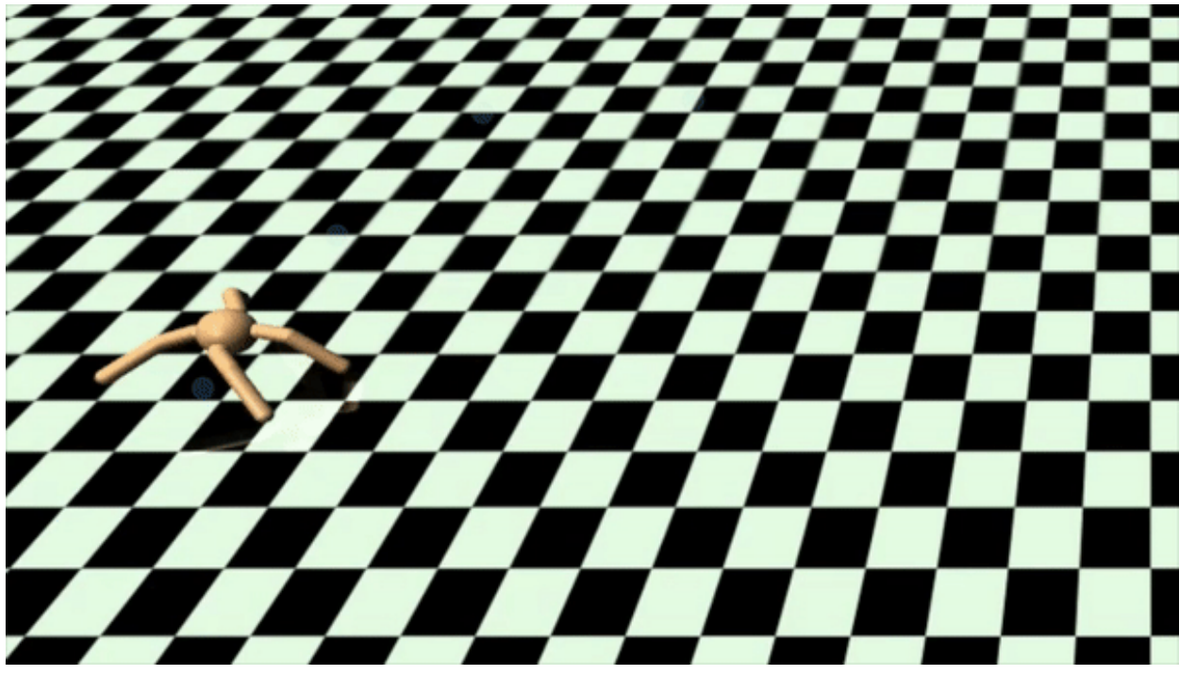
1. Train transition dynamics $s' = s + f(s, a; \phi)$
2. Optimize action sequences using MPC with random search
3. **Aggregate** experience dataset with the inferred $(s, a)$ sequences
4. GOTO 1

Initialize a policy $\pi(s; \theta)$ by imitating the MPC planner using DAGGER

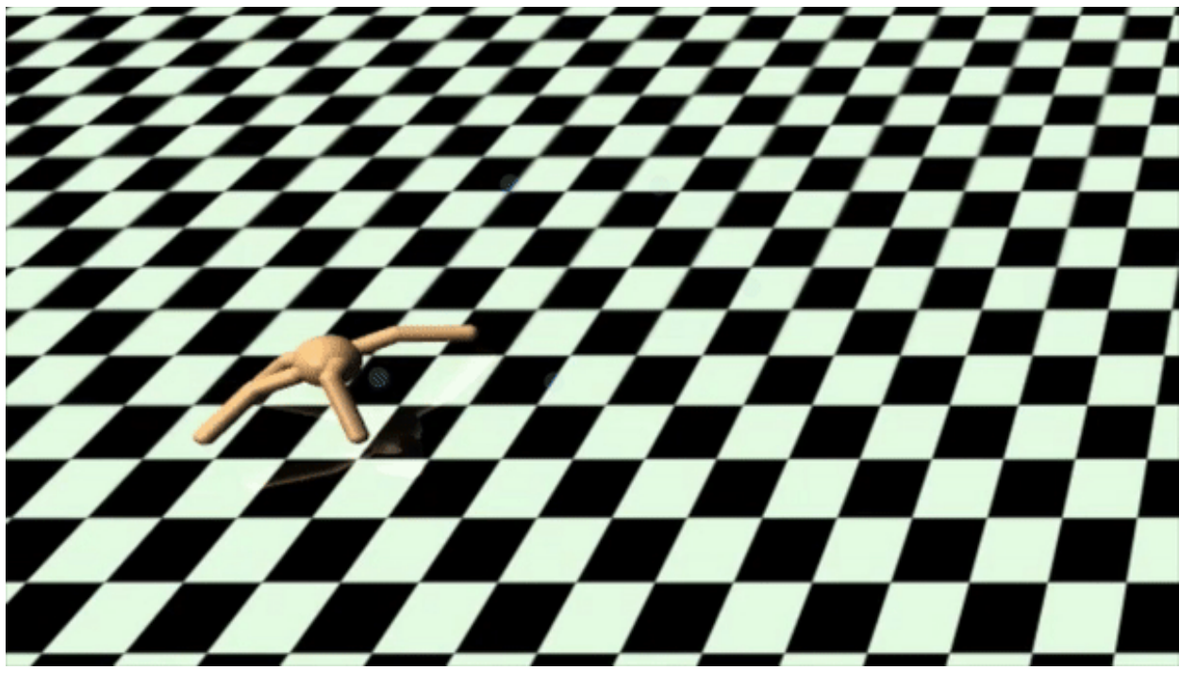
Finetune the policy using any model-free method, e.g., TRPO.

# 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.*

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

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

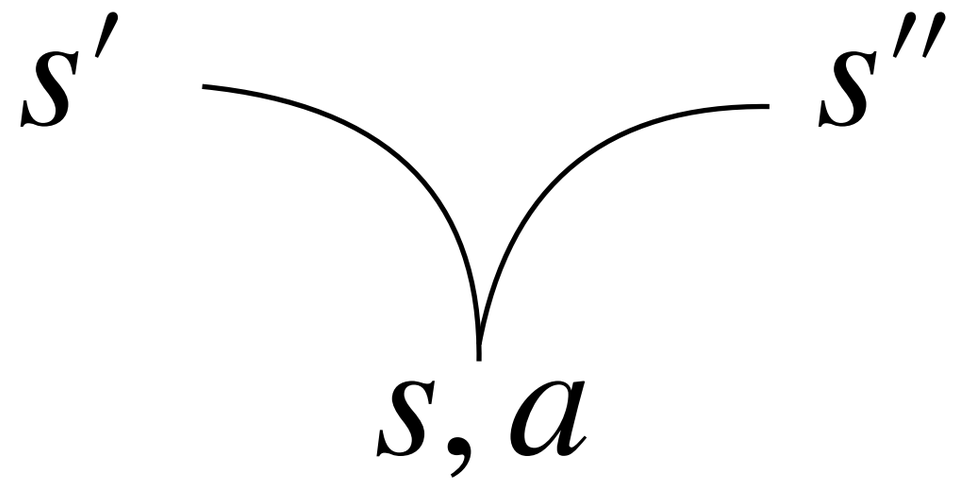
**Kurtland Chua** **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu

It's all about representing 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

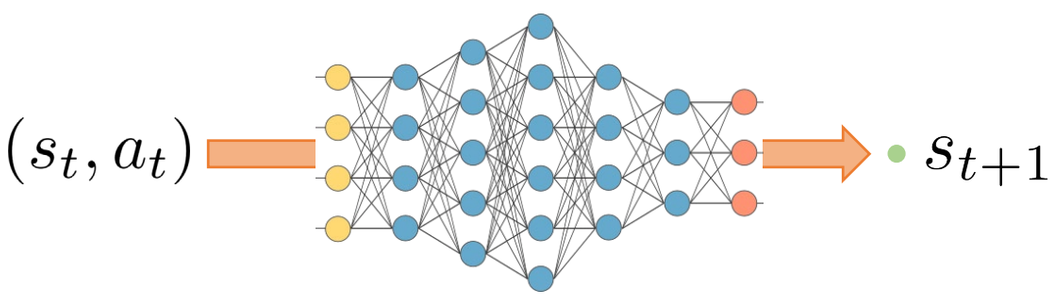


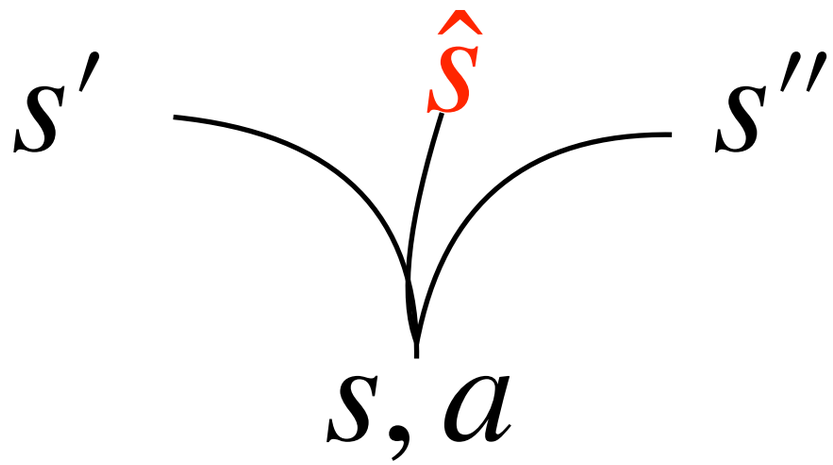

- This means our state does not capture enough information to help us delineate  the possible future outcomes.
- What is stochastic under one state representation, may not be stochastic under another. Is this true? Could we ever predict exactly what we will see in the TV when we switch the channel?
-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

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



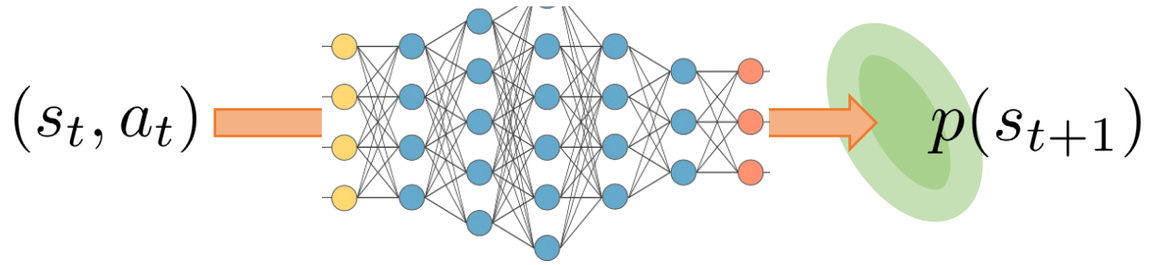




Failing means: not only we cannot capture the distribution, but we  output a solution that does not agree with any of its modes!

# Aleatoric uncertainty in model learning

- Consider experience tuples $\mathrm{D} = \{(s_i, a_i, s_i'), i = 1 \cdots N\}$.
- We will use a neural network that outputs a distribution over next states $s'$ given current state and action.
- Specifically, a Gaussian distribution, where the NN predicts the mean and the elements of the covariance matrix

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



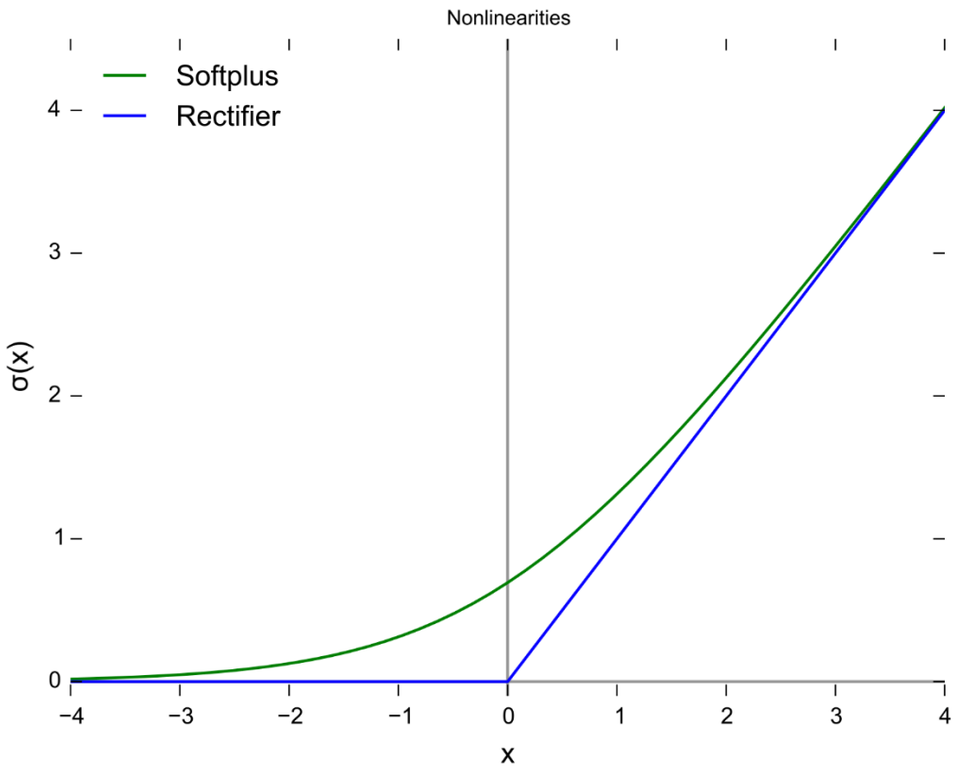

$$\mathscr{L}_\phi = -\frac{1}{N}\sum_{i=1}^{N} \log p_\phi(s_i' \mid s_i, a_i)$$

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

# Aleatoric uncertainty in model learning

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

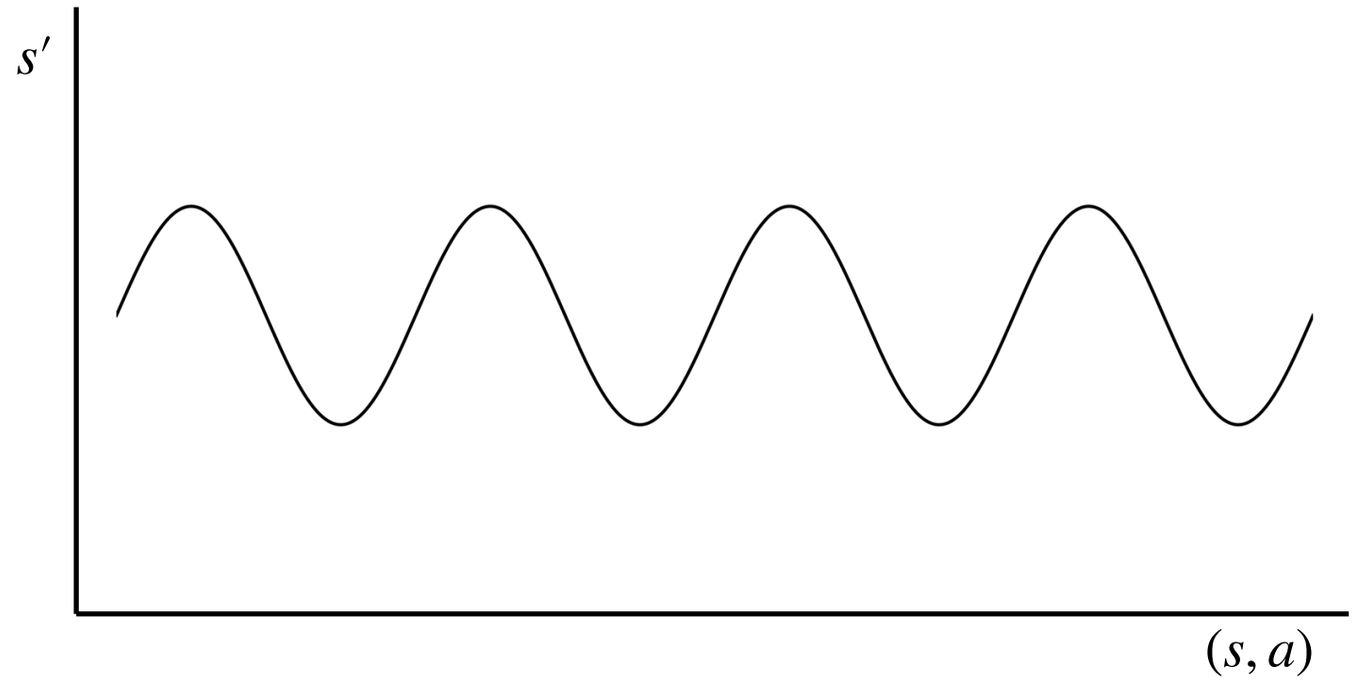
```
logvar = max_logvar - tf.nn.softplus(max_logvar - logvar)
logvar = min_logvar + tf.nn.softplus(logvar - min_logvar)
var = tf.exp(logvar)
```



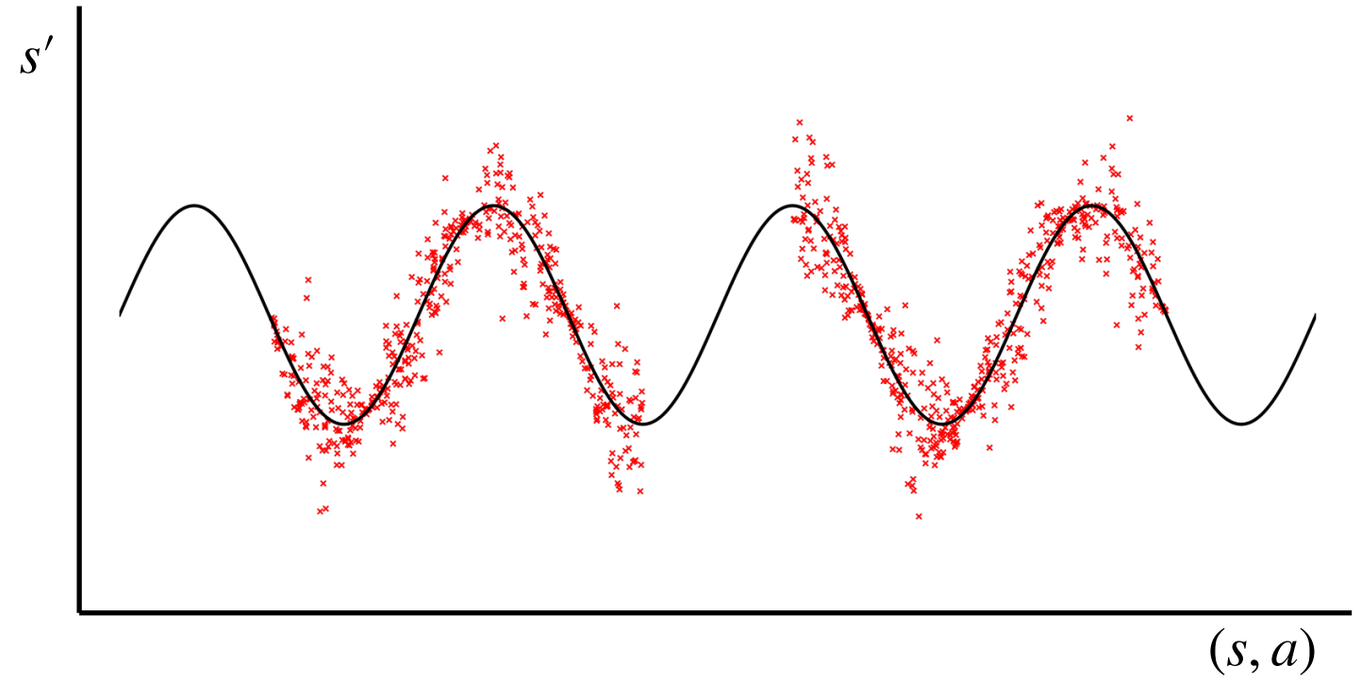

$\text{Rectifier}(x) = \max(0, x)$

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

# Epistemic uncertainty in Model Learning

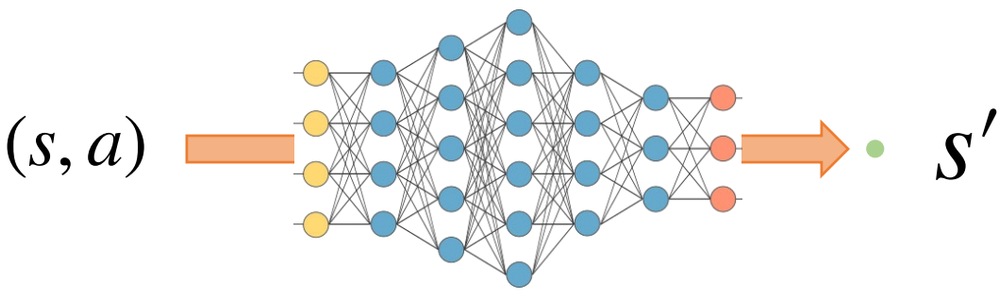


# Epistemic uncertainty in Model Learning

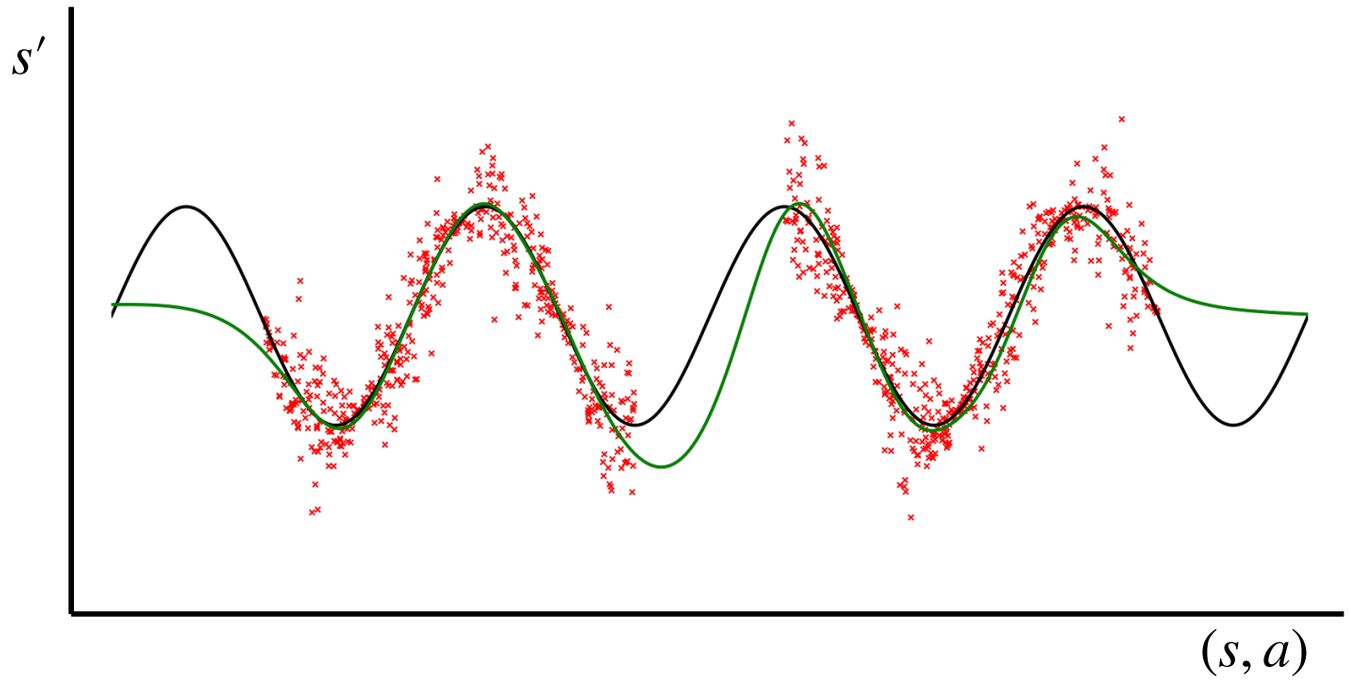



Red are observed data points (s,a,s')

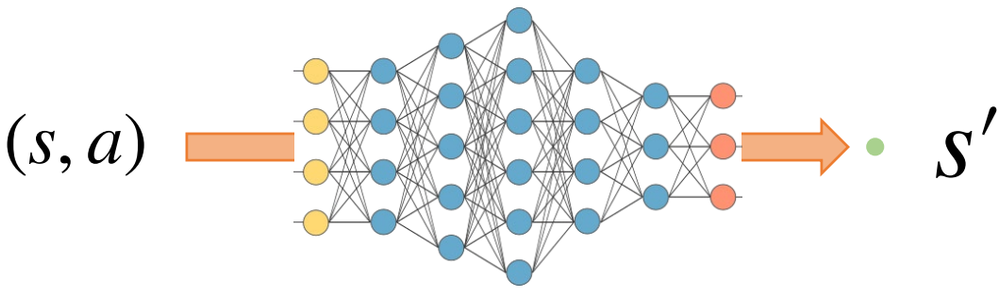
# Epistemic uncertainty in Model Learning




Fitting a deterministic neural network

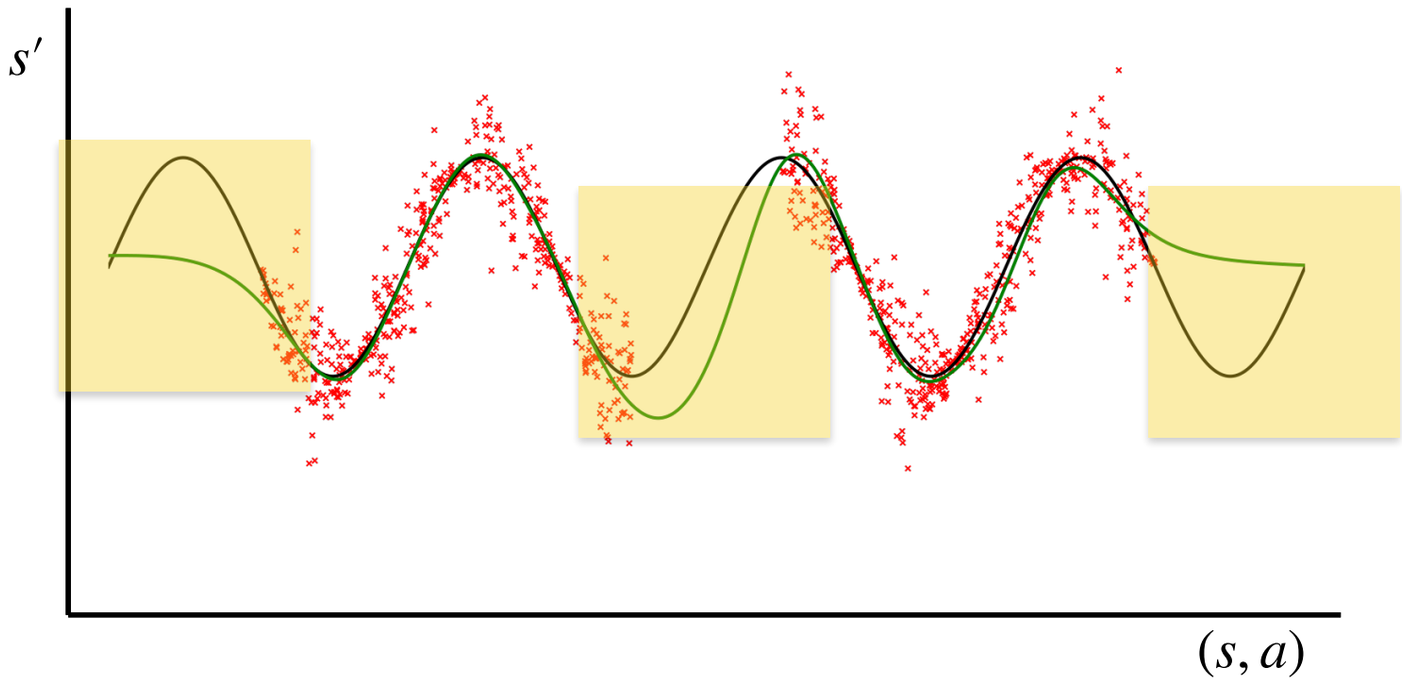


# Epistemic uncertainty in Model Learning



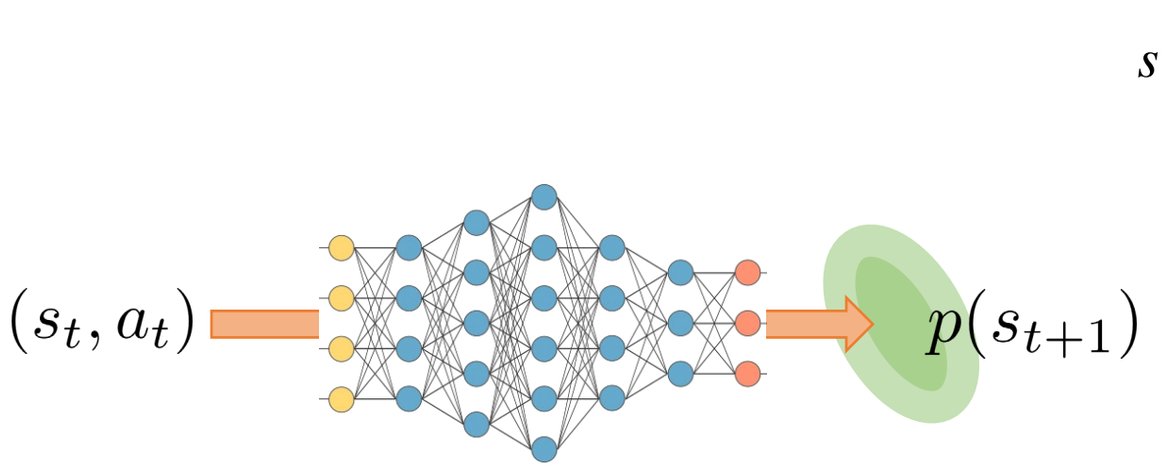

Fitting a deterministic neural network

Model errors!



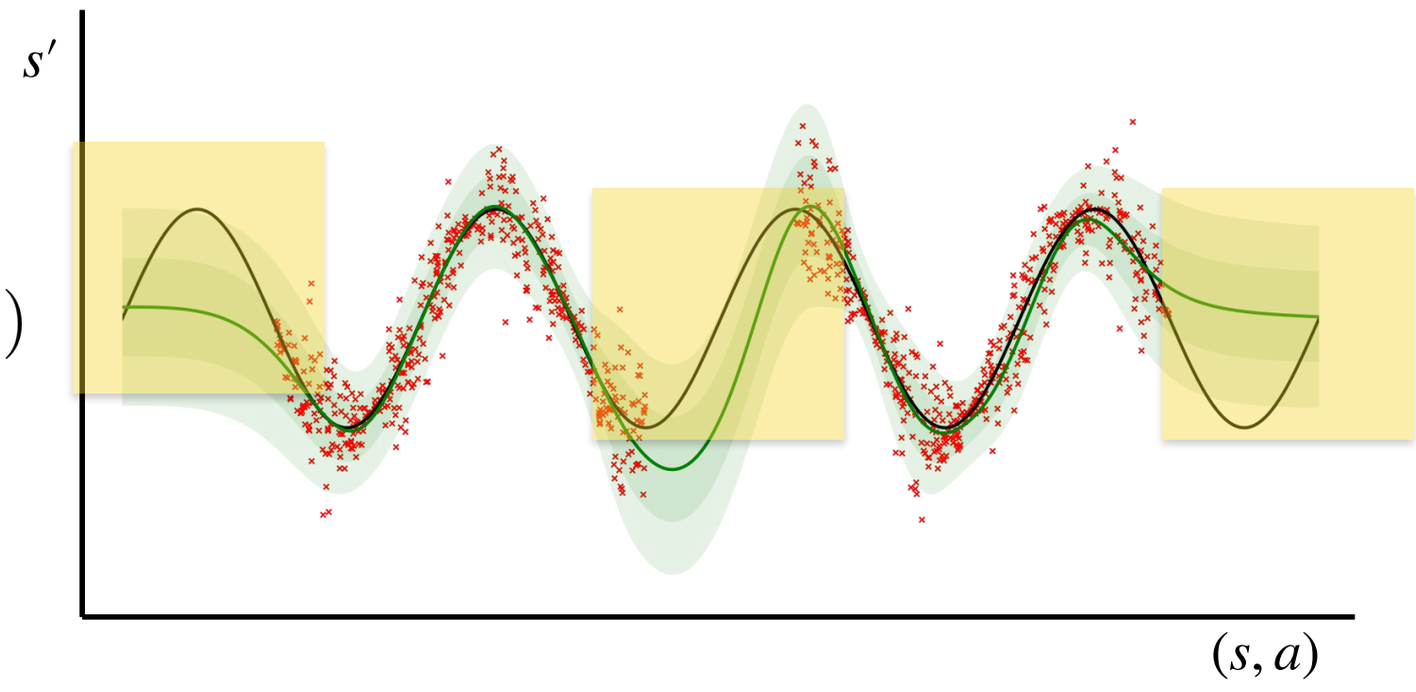

- There is a unique answer for s’ (no stochasticity) but I do not know it due to lack of data!

# Epistemic uncertainty in Model Learning



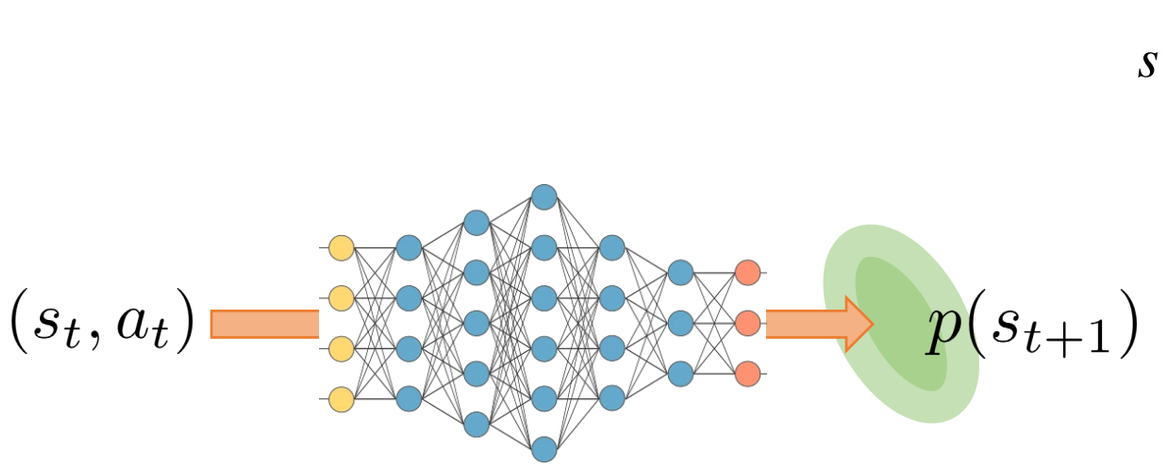

Fitting a neural network that parametrizes a distribution

Model errors!



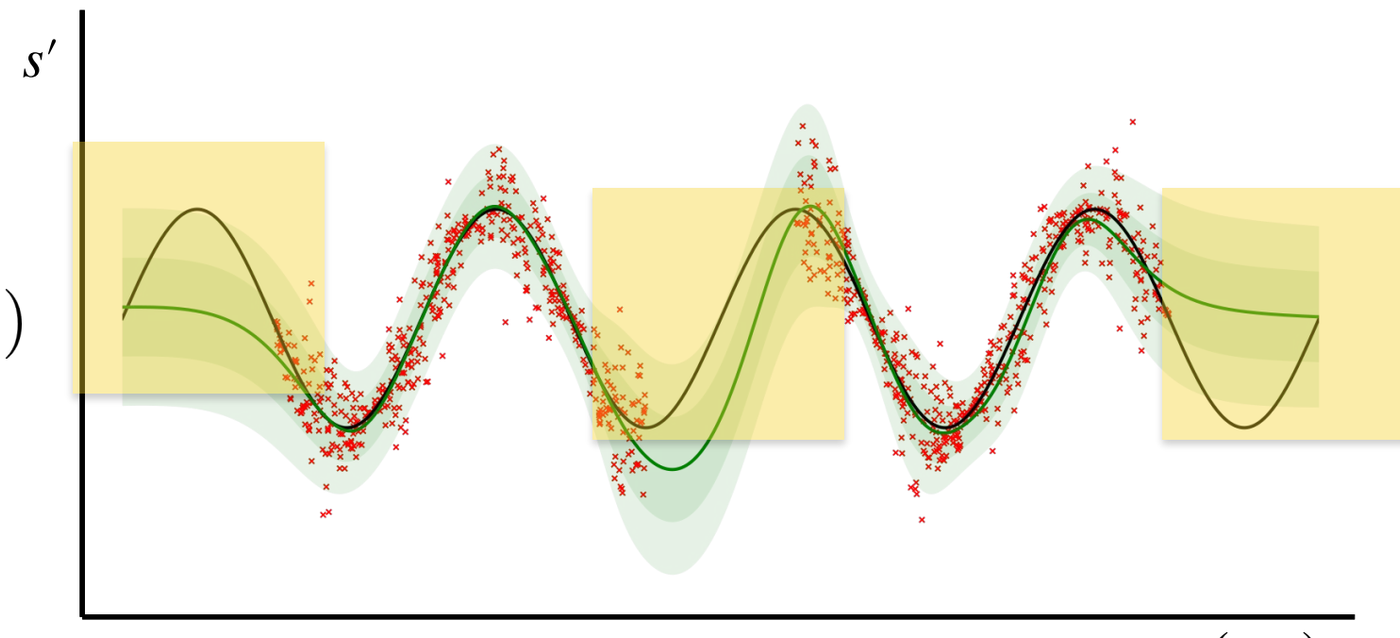

- There is a unique answer for s’ (no stochasticity) but I do not know it due to lack of data!
- Predicting a distribution won’t help! The predictions will suffer from lack of data and will be wrong.

# Epistemic uncertainty in Model Learning




Fitting a neural network that parametrizes a distribution

Model errors!




- 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 suffer from lack of data and will be wrong.
- How can I represent my uncertainty about my predictions? E.g., having high entropy when no data and low entropy close to data?

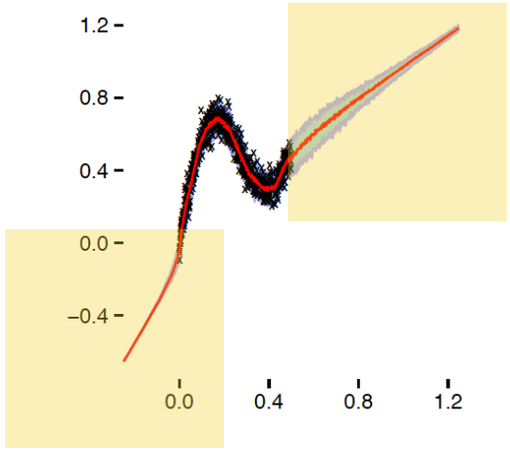
# Bayesian Inference!

# Bayes Rule

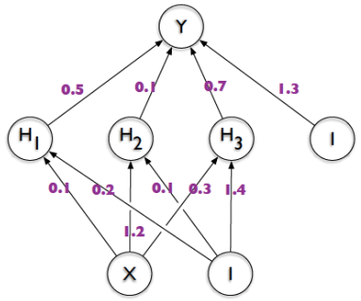
$$P(\text{ hypothesis } | \text{ data }) = \frac{P(\text{ hypothesis })P(\text{ data } | \text{ hypothesis })}{\sum_h P(h)P(\text{ data } | h)}$$

- 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.



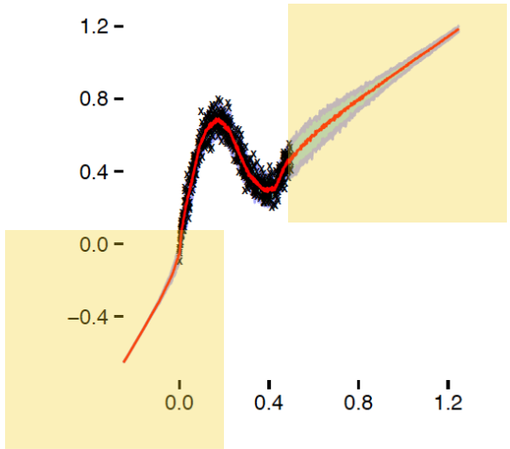

regression network



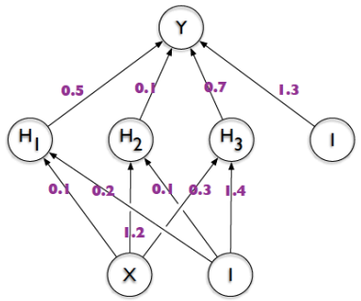

$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left( P(D \mid \phi) + \log P(\phi) \right)$$

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

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

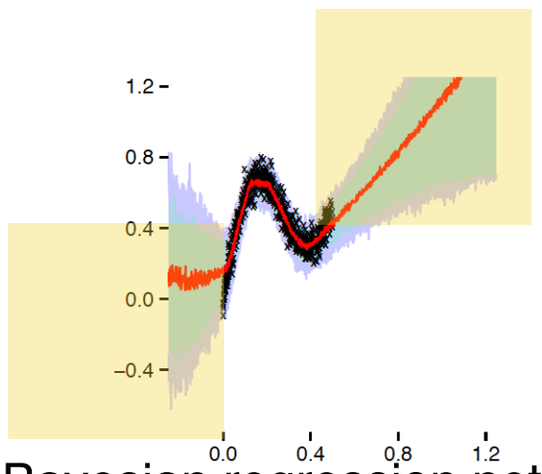



regression network



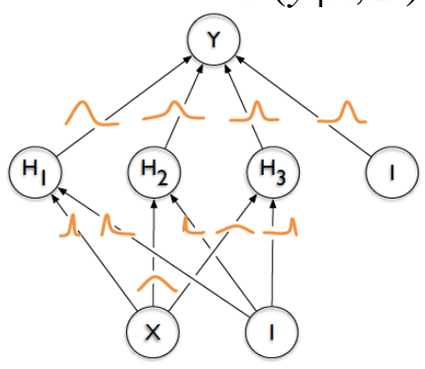

$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left( P(D \mid \phi) + \log P(\phi) \right)$$

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




Bayesian regression network

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

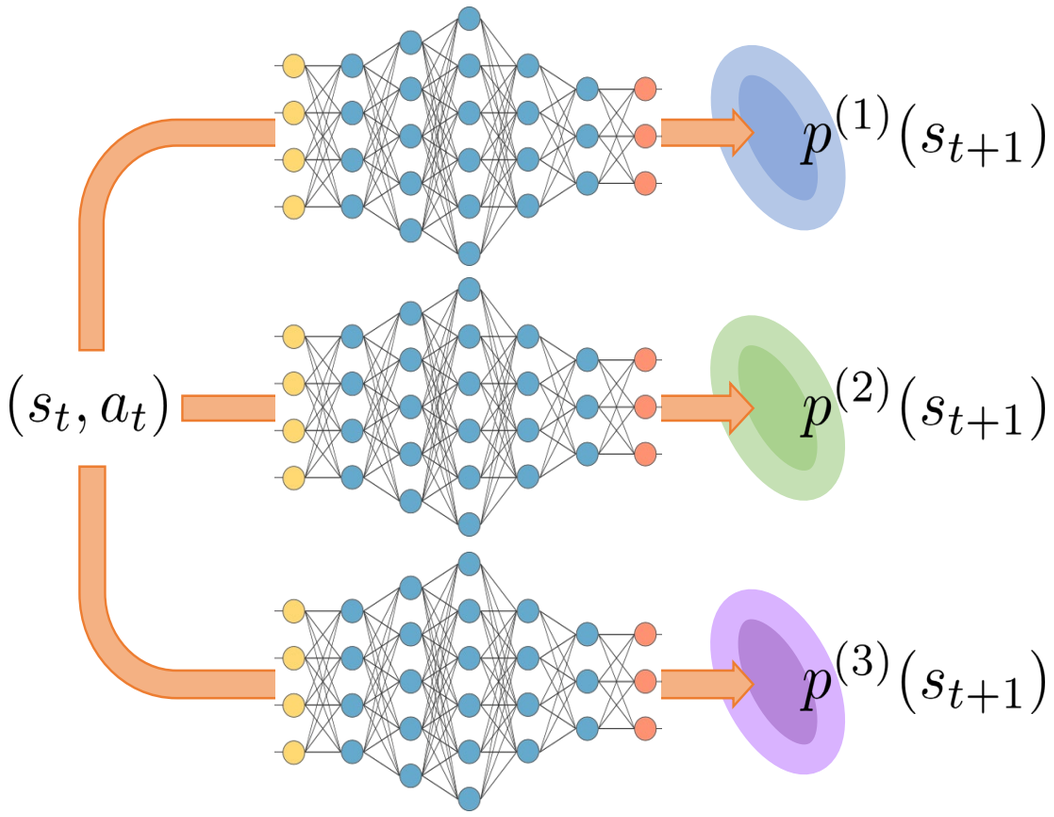


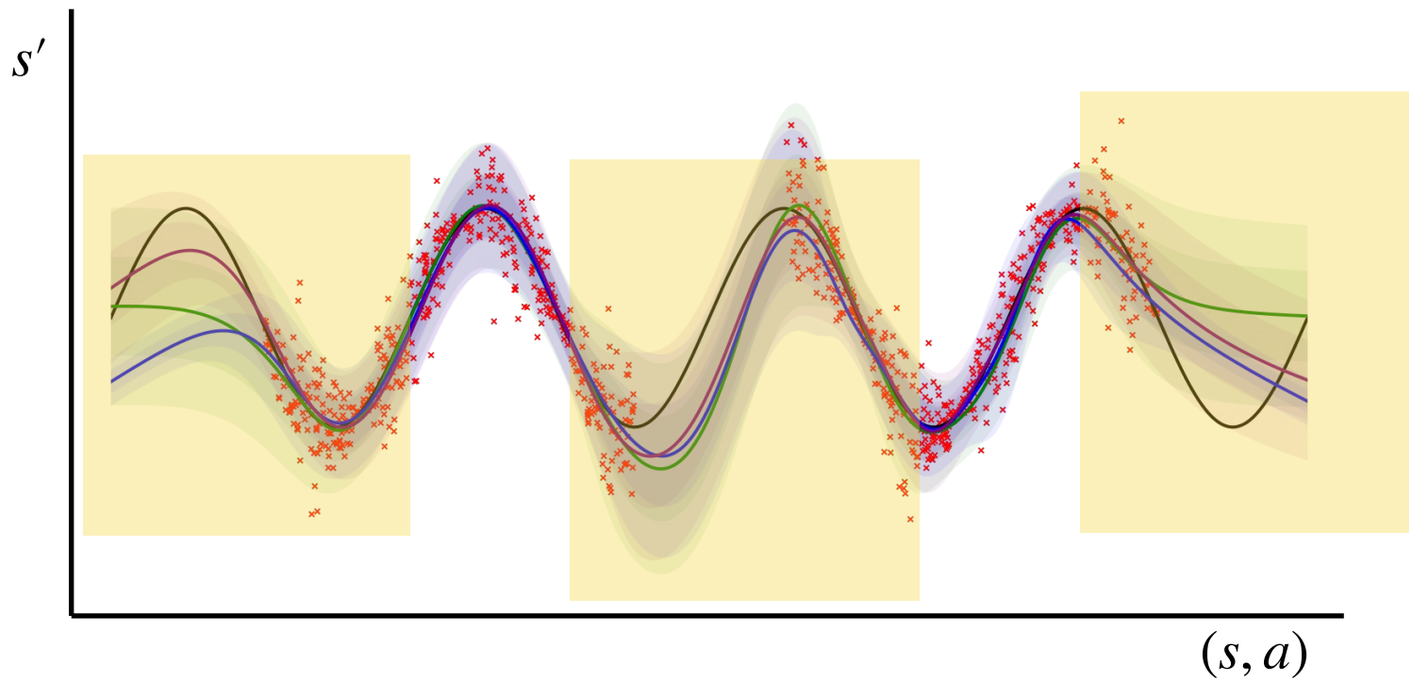

$P(\phi \mid D)$

- 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

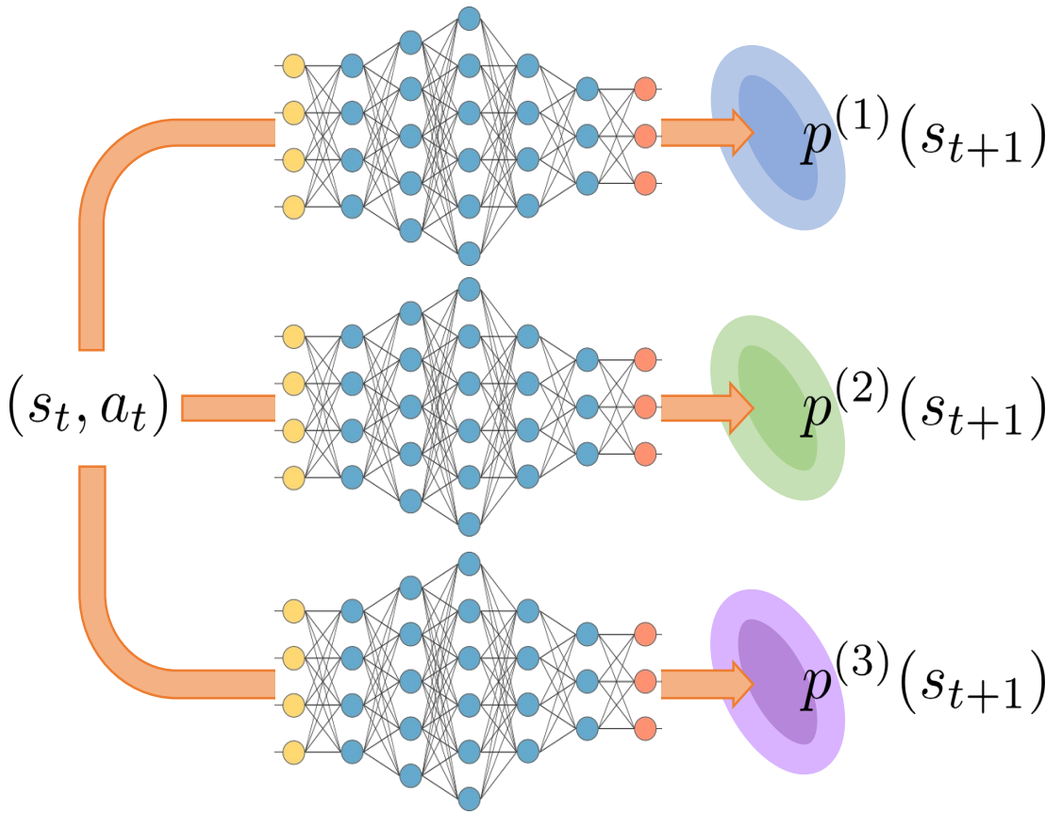
- Neural network Ensembles are a good approximation to Bayesian Nets.
- Instead of having explicit posteriors distributions for each neural net parameter, you just have a small set of neural nets, each trained on separate data.
  - On the data they have seen, they all agree (low entropy of predictions)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

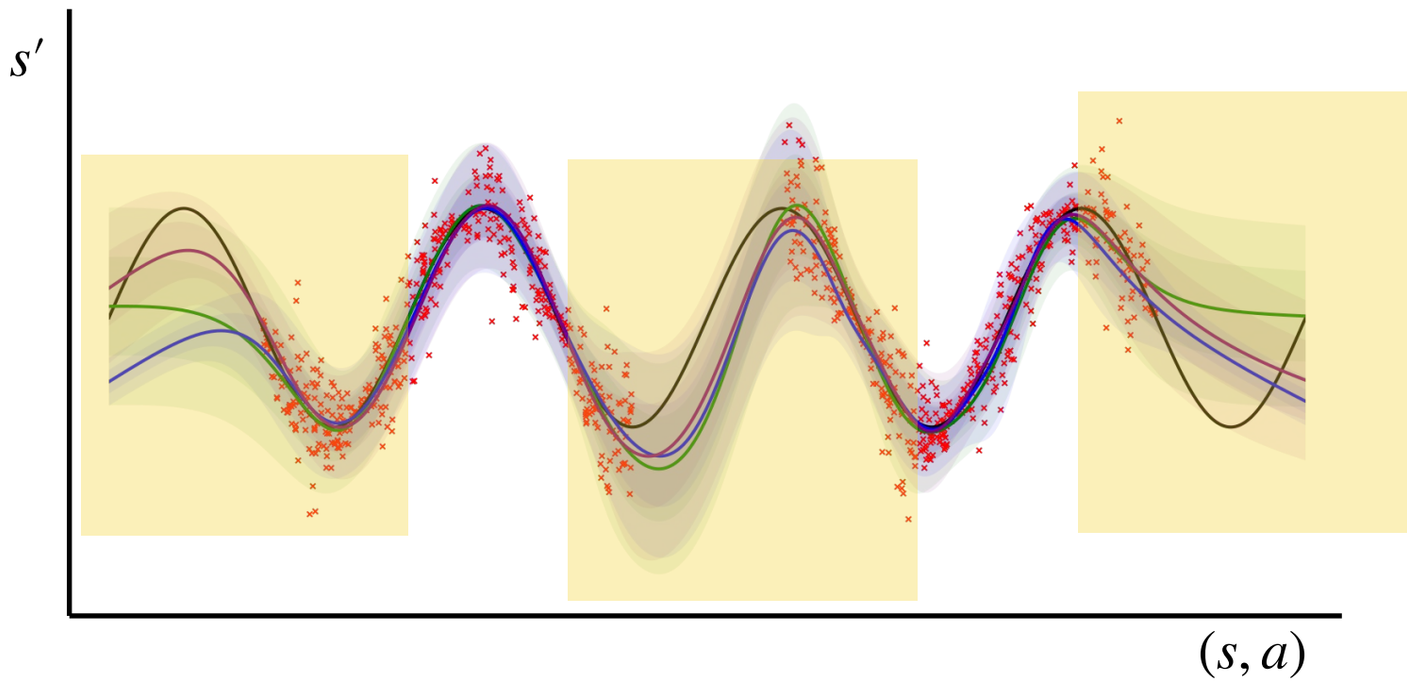


# NN Ensembles for representing Epistemic uncertainty

- 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



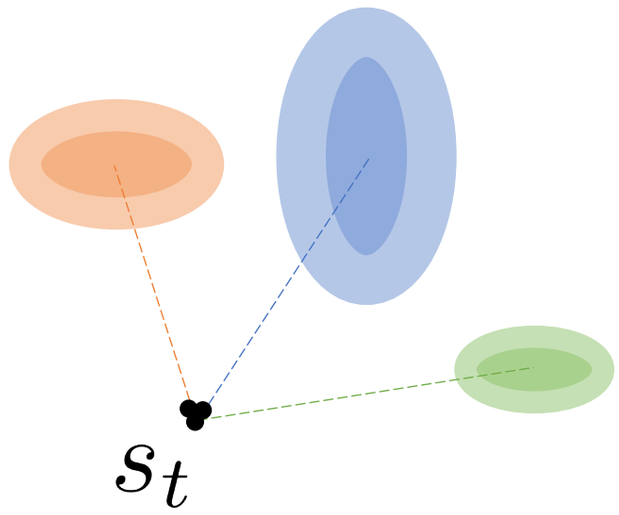
# Model Unrolling

$$\dot{s_t}$$

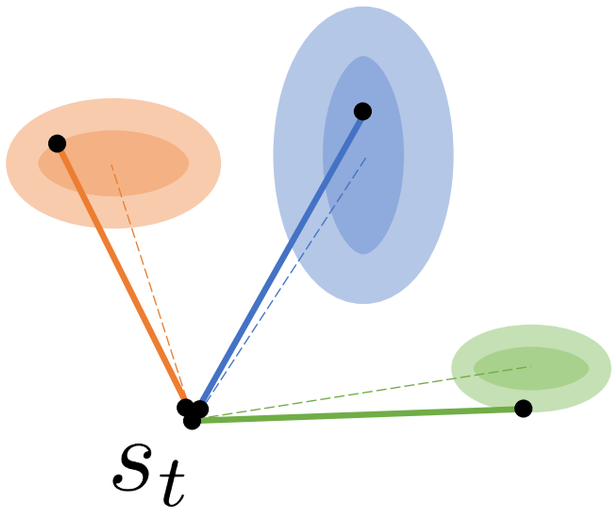
# Model Unrolling

$s_t$

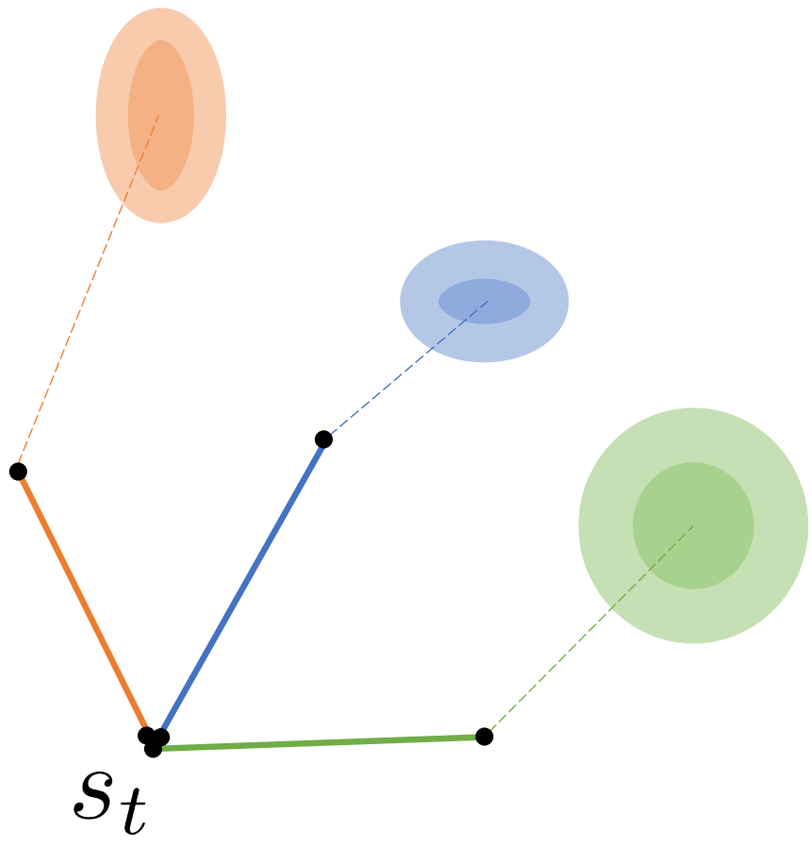
# Model Unrolling



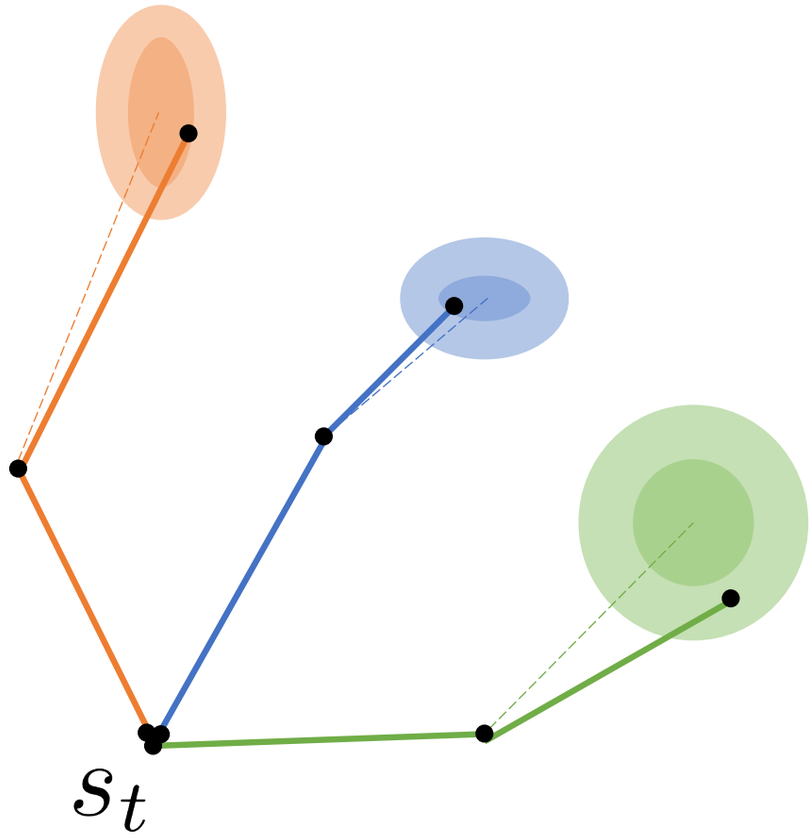
# Model Unrolling



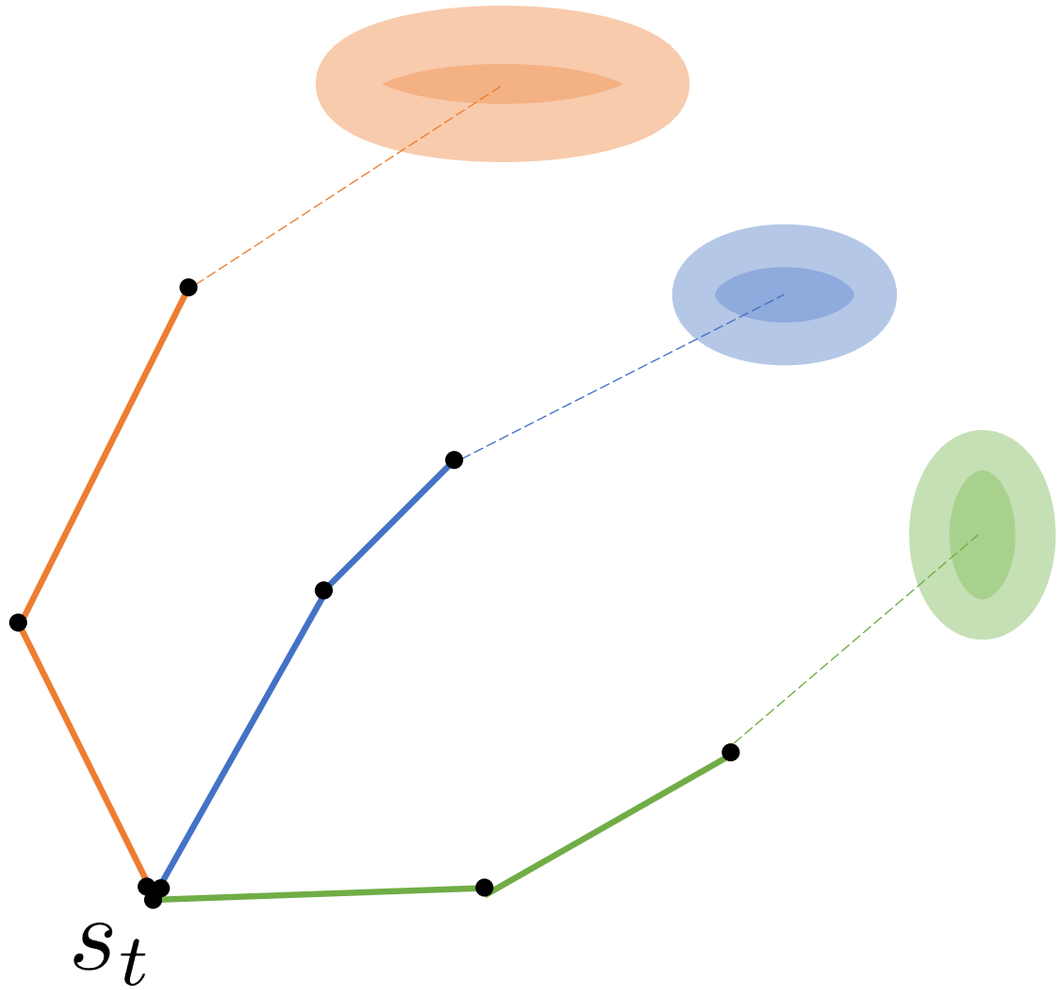
# Model Unrolling



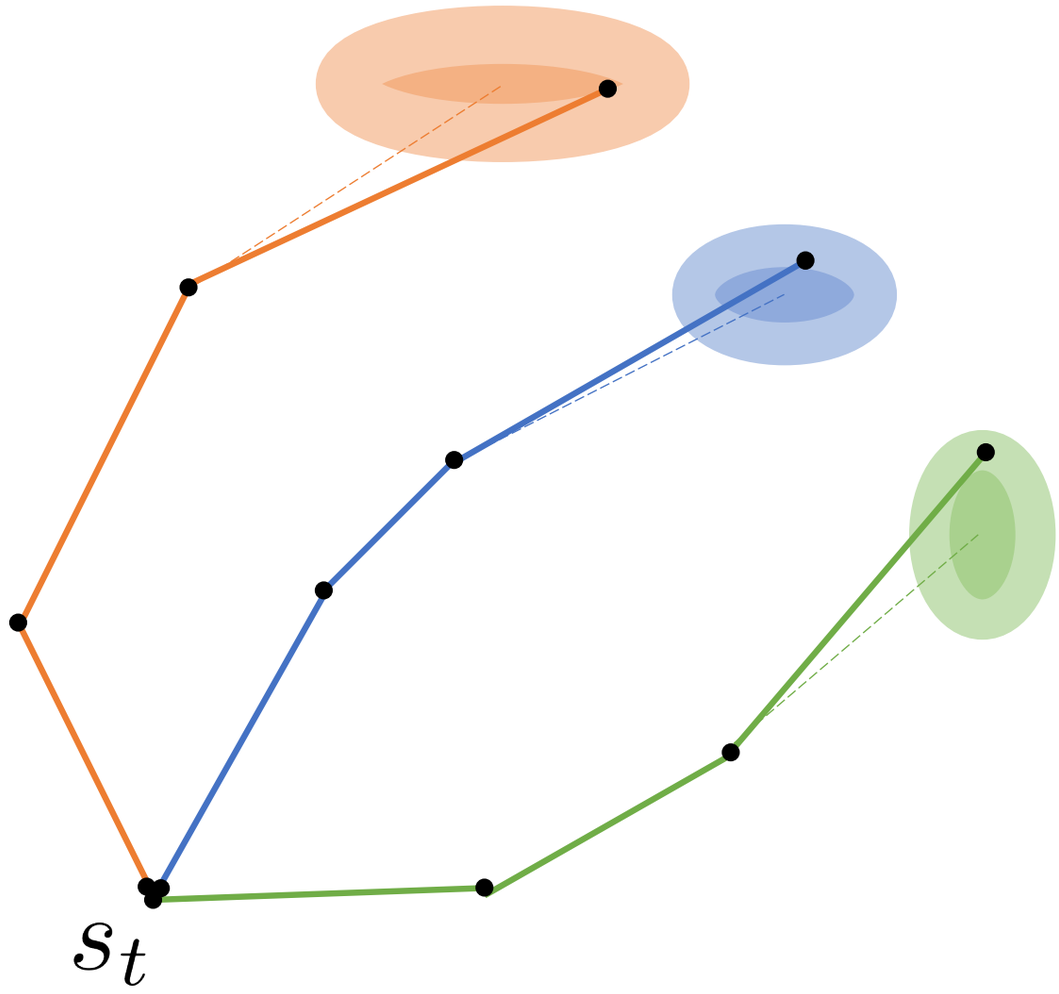
# Model Unrolling



# Model Unrolling

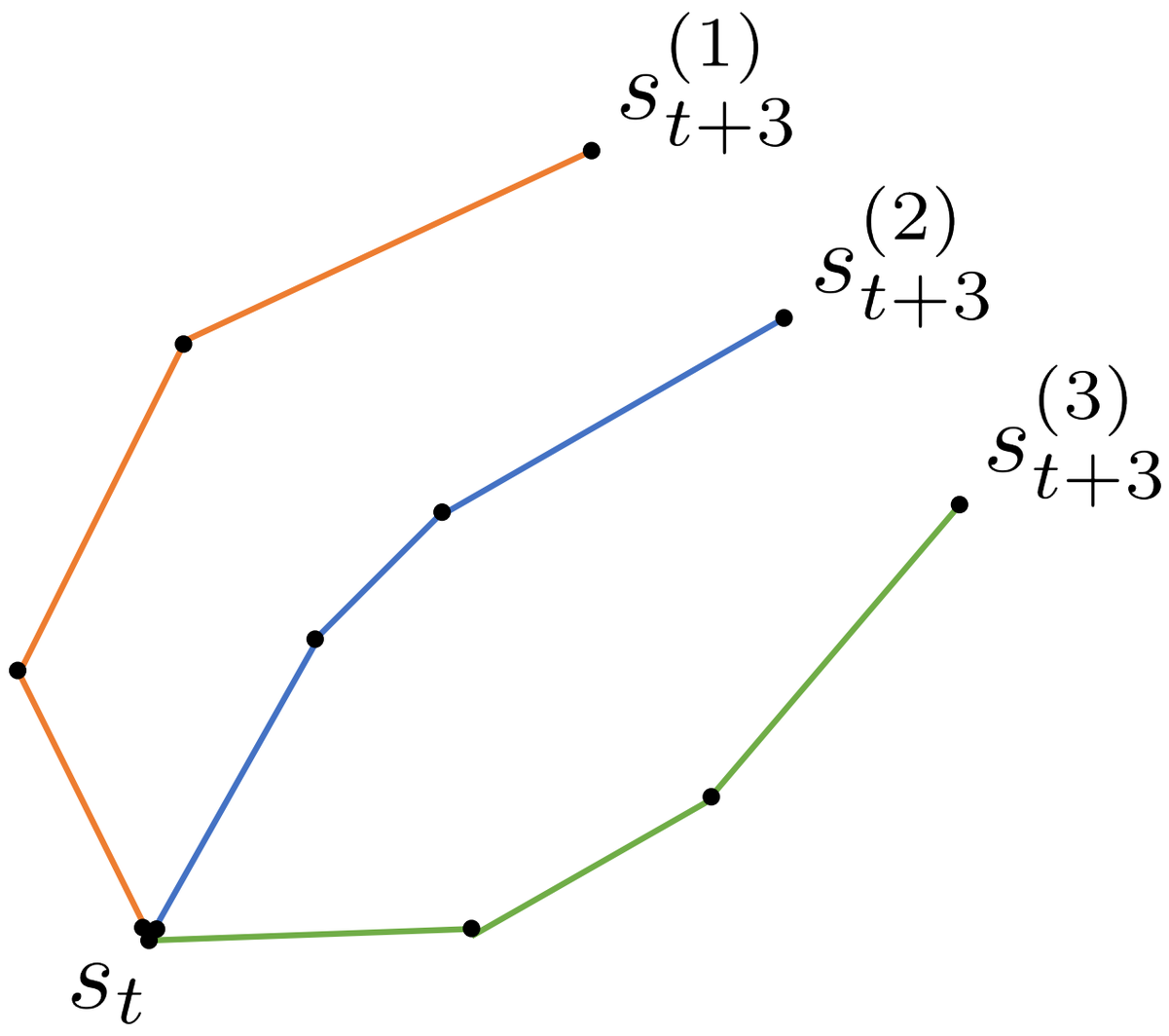


# Model Unrolling



# Model Unrolling

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




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

# Results

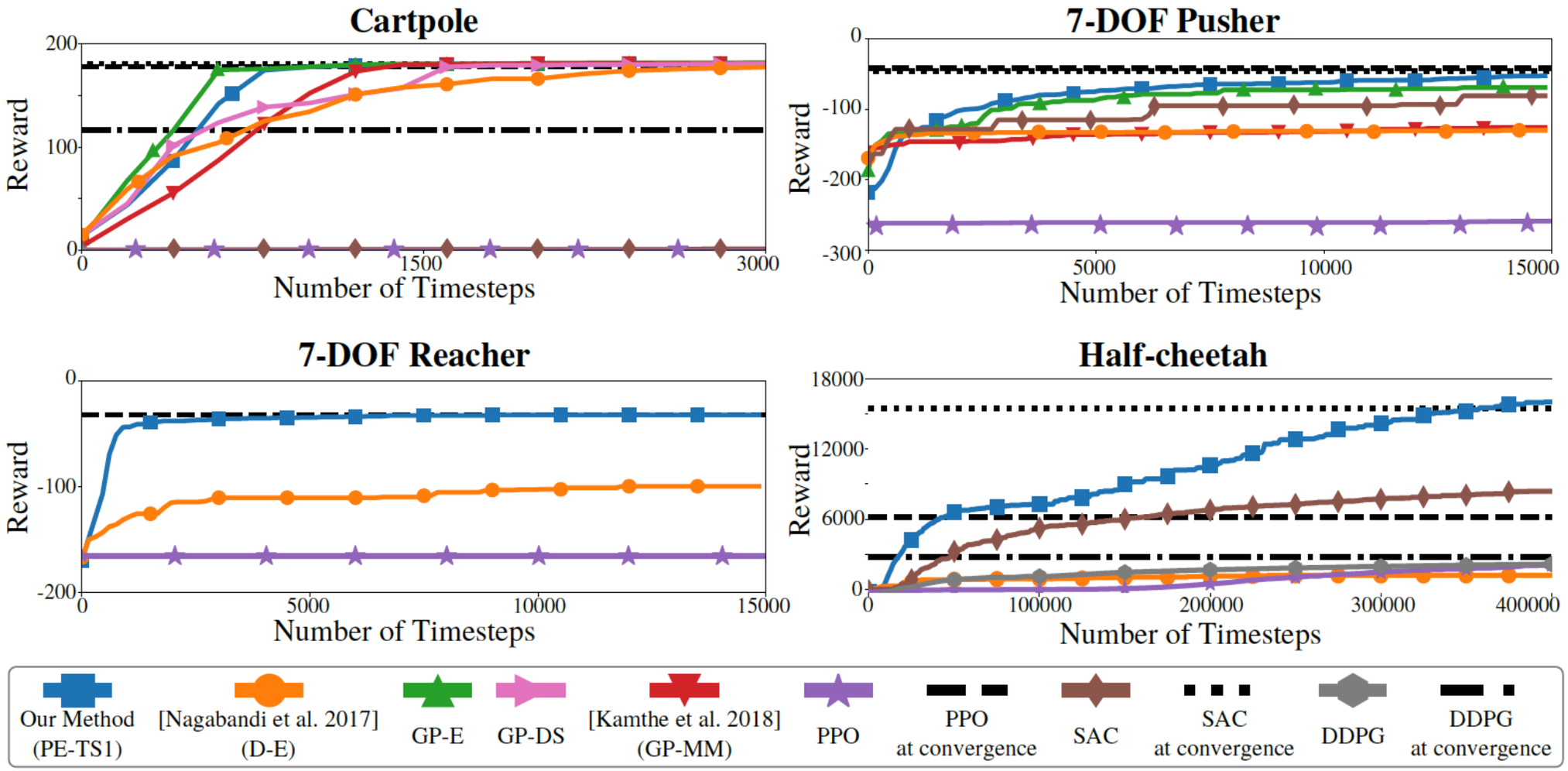



Figure 3: Learning curves for different tasks and algorithm. For all tasks, our algorithm learns in under 100K time steps or 100 trials. With the exception of Cartpole, which is sufficiently low-dimensional to efficiently learn a GP model, our proposed algorithm significantly outperform all other baselines. 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