Carnegie Mellon

School of Computer Science

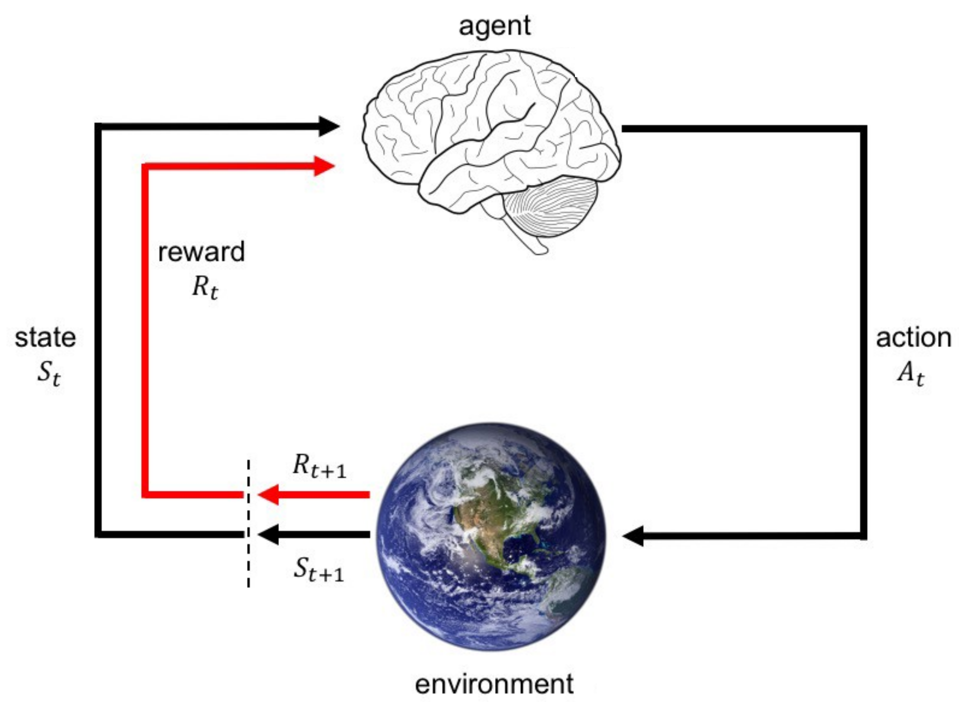
Deep Reinforcement Learning and Control

# Model Based Reinforcement Learning: Low-dimensional models, Explicit models

Fall 2021, CMU 10-703

Instructors
Katerina Fragkiadaki
Ruslan Salakhutdinov

# Model-free Reinforcement learning



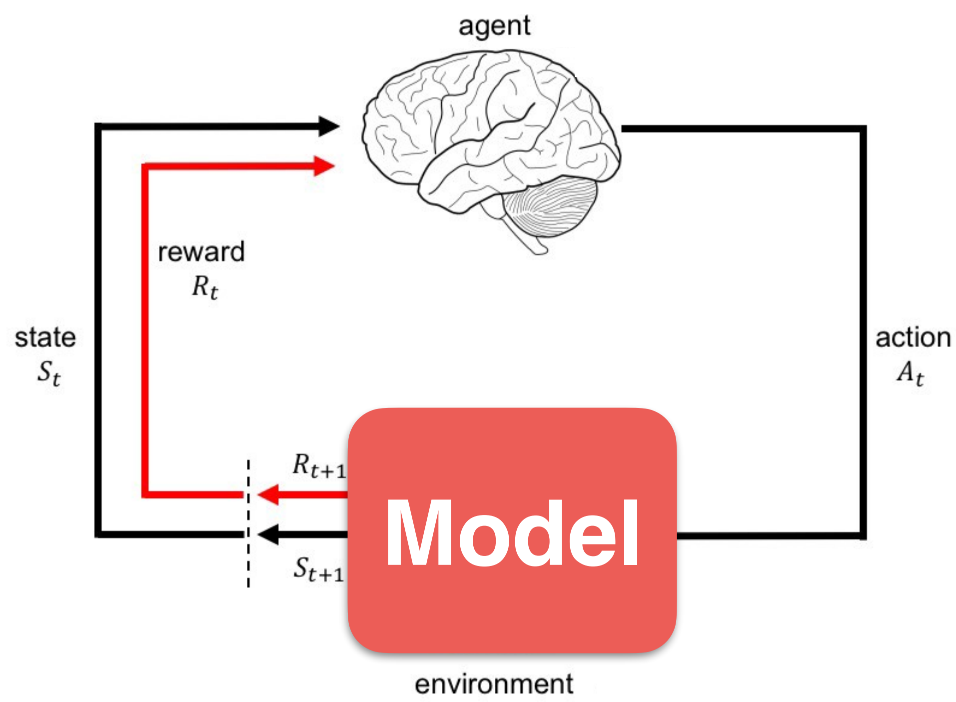

$$a_t \sim \pi_\theta(a_t \mid s_t)$$

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

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

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

# The model: transition+reward function




$$a_t \sim \pi_\theta(a_t \,|\, s_t)$$

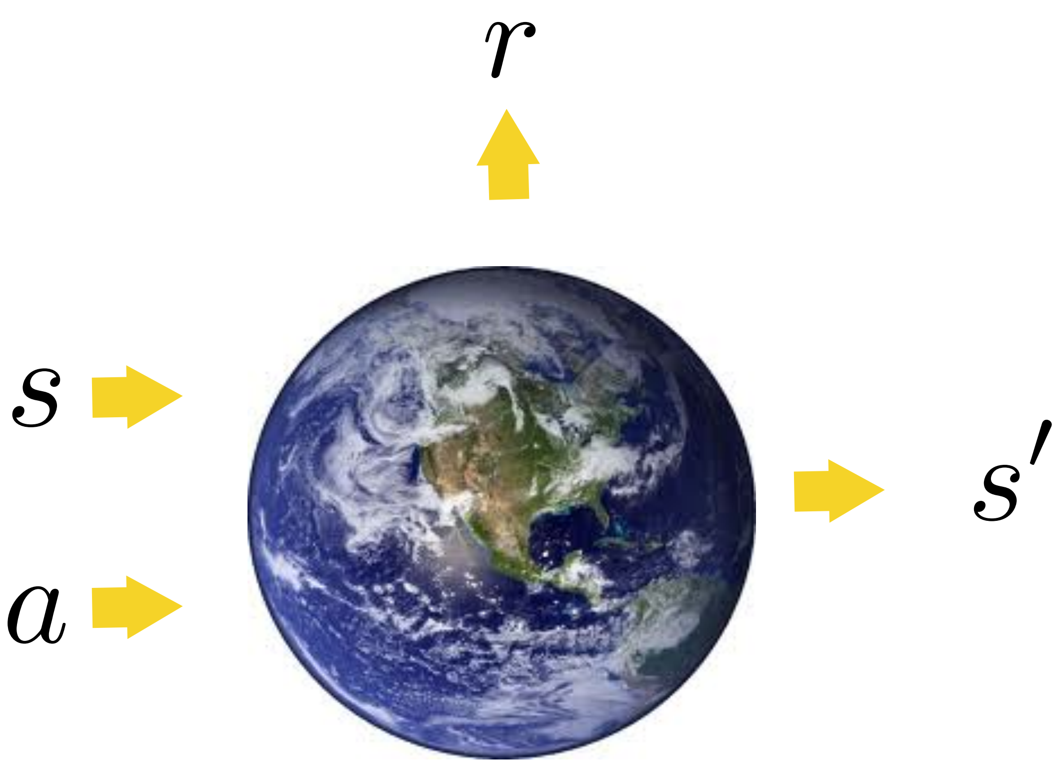
$$p(s_{t+1} \,|\, s_t, a_t)$$
$$p(r_{t+1} \,|\, s_t, a_t)$$

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

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

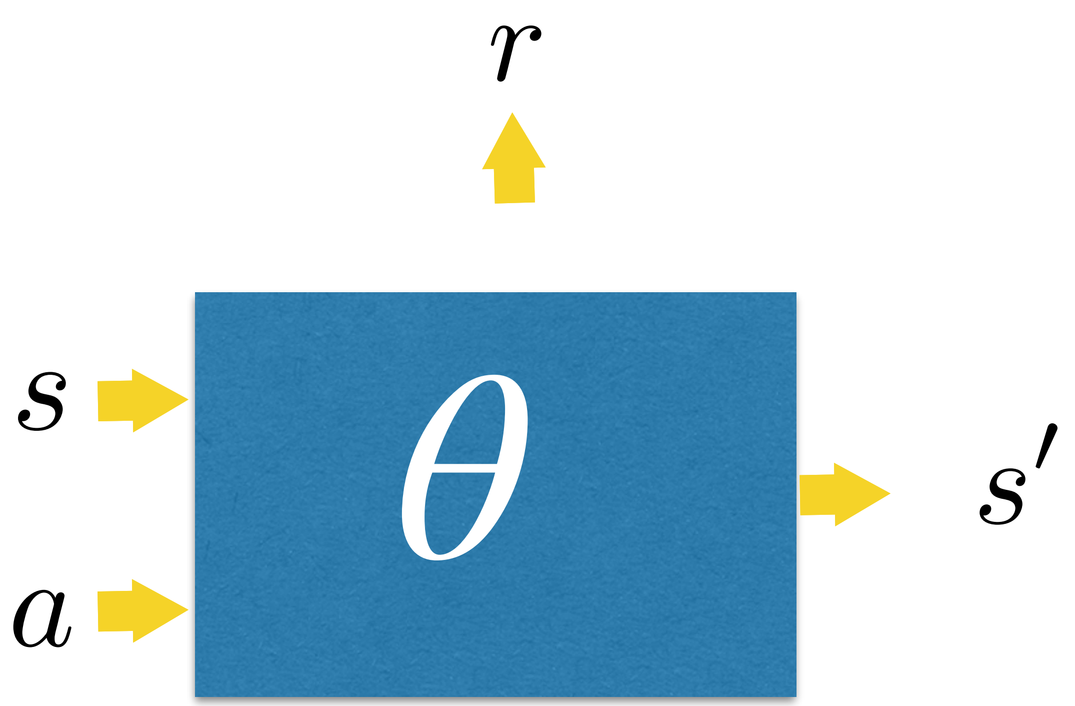
# The model: transition+reward function

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 and  use it to select actions and/or learn policies.

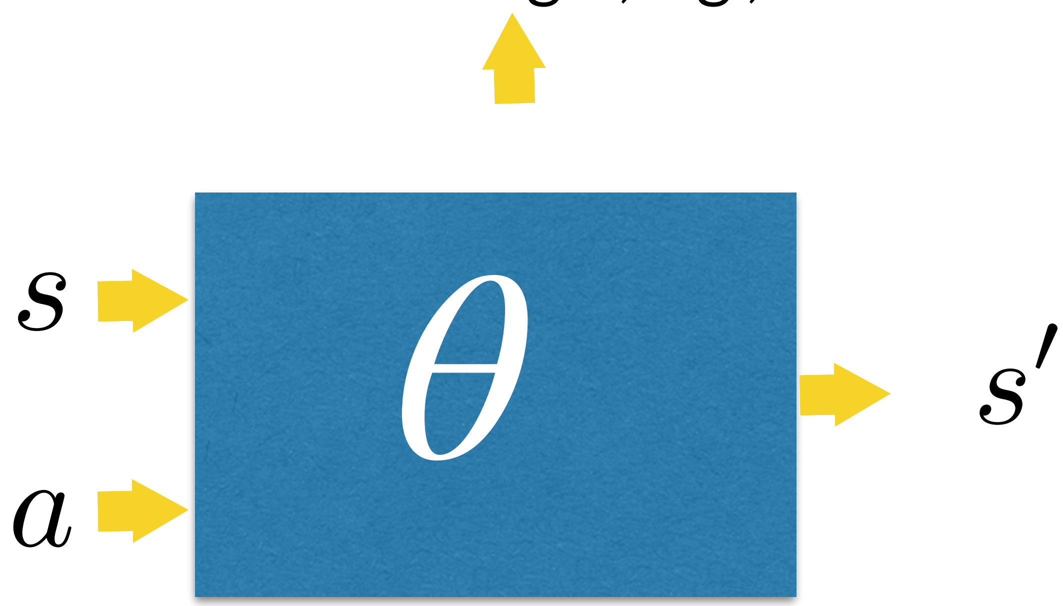



gaussian process, random forest, deep neural network, linear function

# Model-based Reinforcement learning

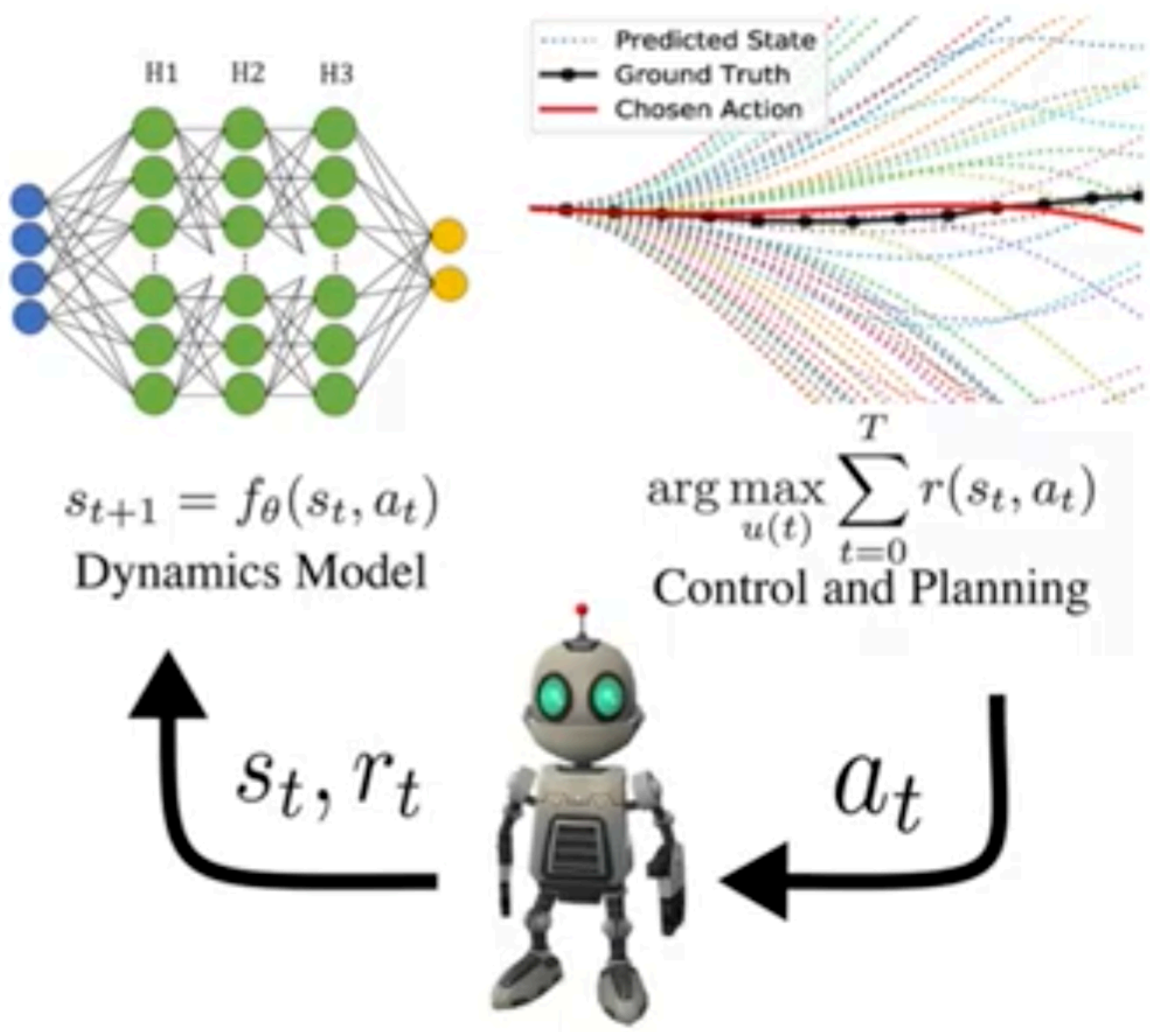
Models can be useful for:

- Action selection by lookahead (model forward unrolling) at train or test time
- Synthetic experience generation that augments real experience
- Auxiliary task for representation learningm, e.g., add a forward prediction loss




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

# Model-based Reinforcement learning




While improving:

1. Agent acts in environment
2. Learn model of dynamics

$$p_\theta = \arg\max_\theta \sum_{i=1}^{N} \log\, p_\theta(s_{t+1}|s_t, a_t)$$

3. Plan actions to maximize reward

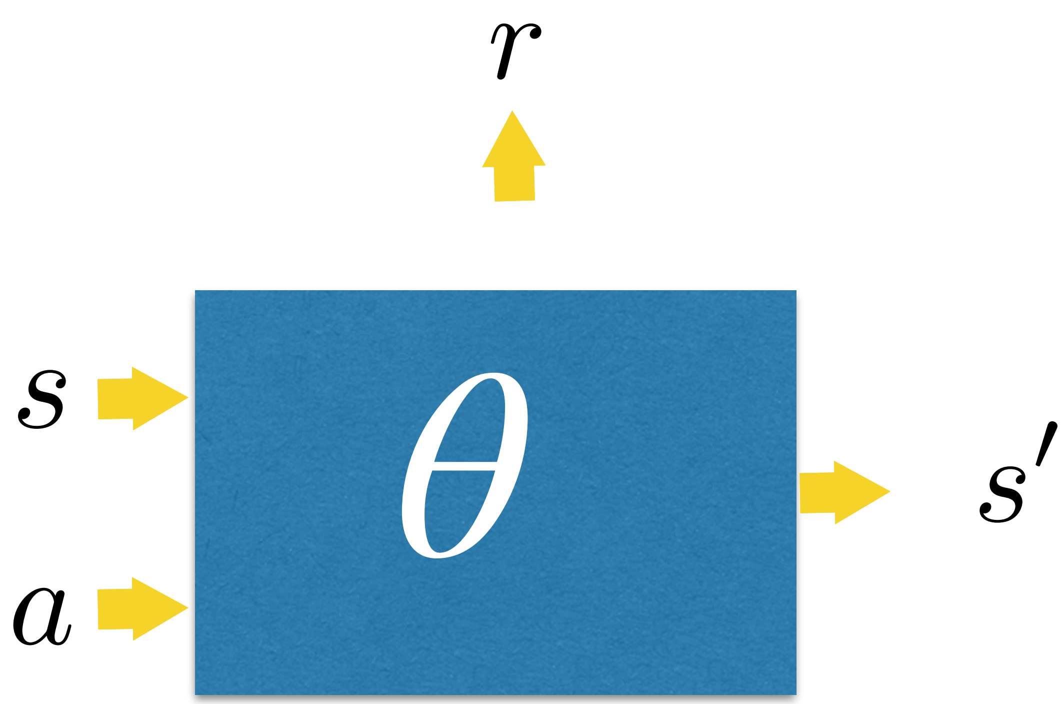
$$a^* = \arg\max_a \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$$

$$s.t.\ s_{t+1} \sim p_\theta(s_{t+1}|s_t, a_t)$$

# Model-based Reinforcement learning

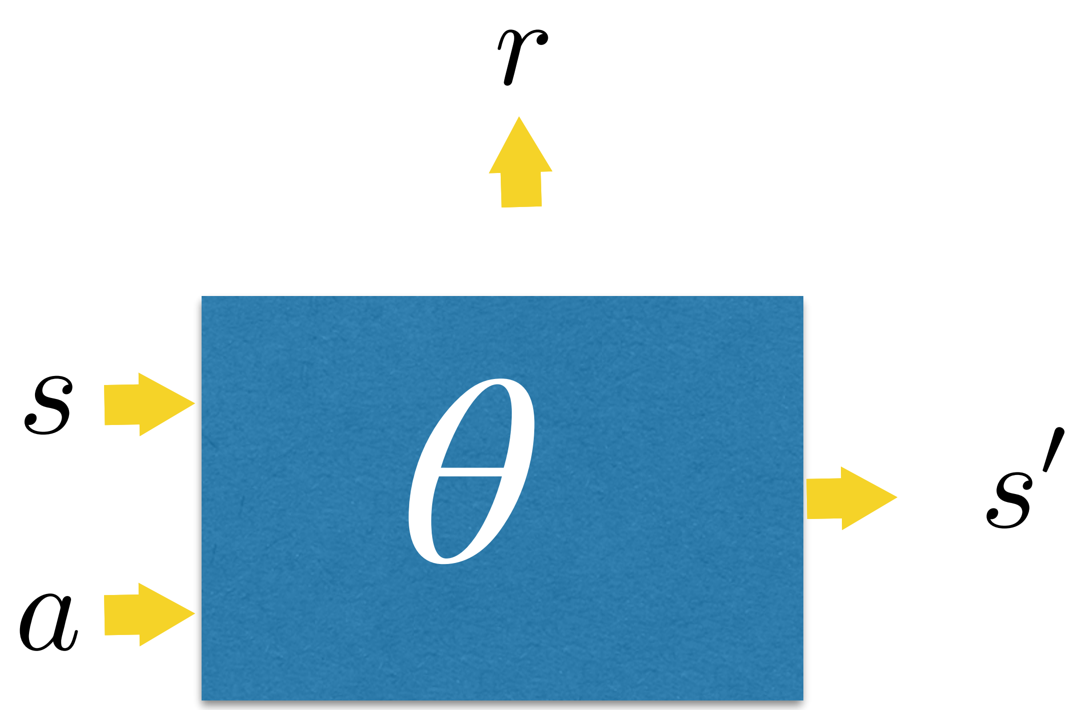
Models can be useful for:

- Action selection by lookahead (model forward unrolling) at train or test time
- Synthetic experience generation that augments real experience
- Auxiliary task for representation learningm, e.g., add a forward prediction loss




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

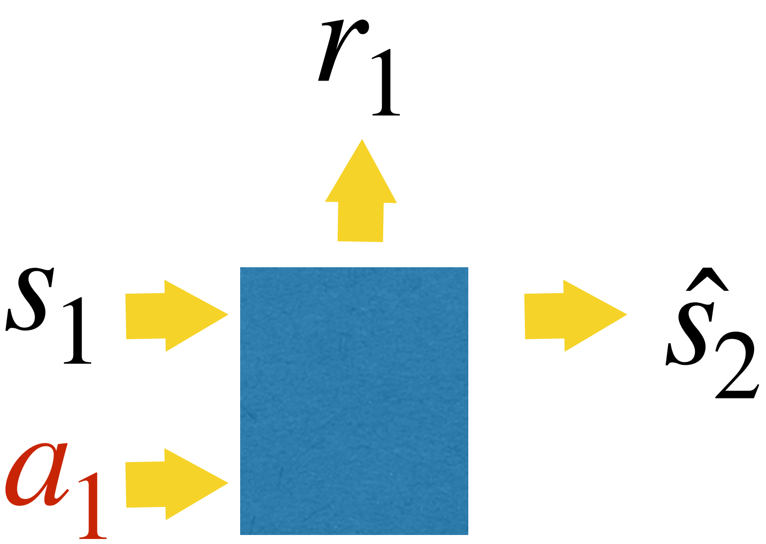
# Consider we have a model in hand..




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

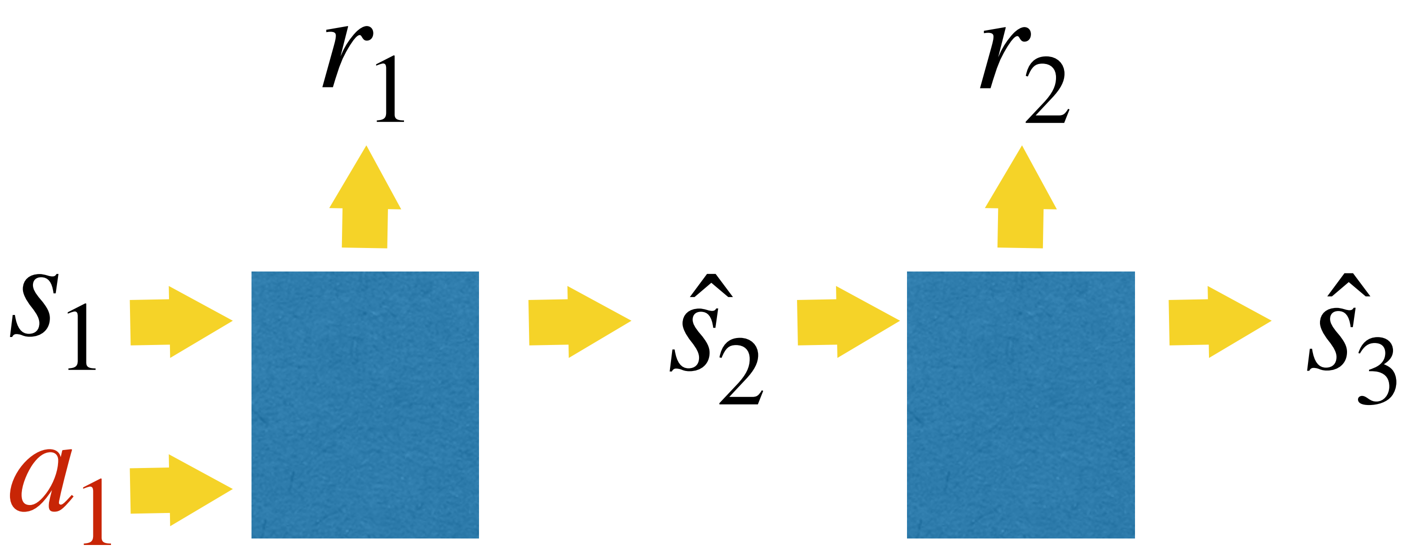
We can use it to look ahead far in the future and compute consequences of our actions.

# Planning: Model unrolling for action selection



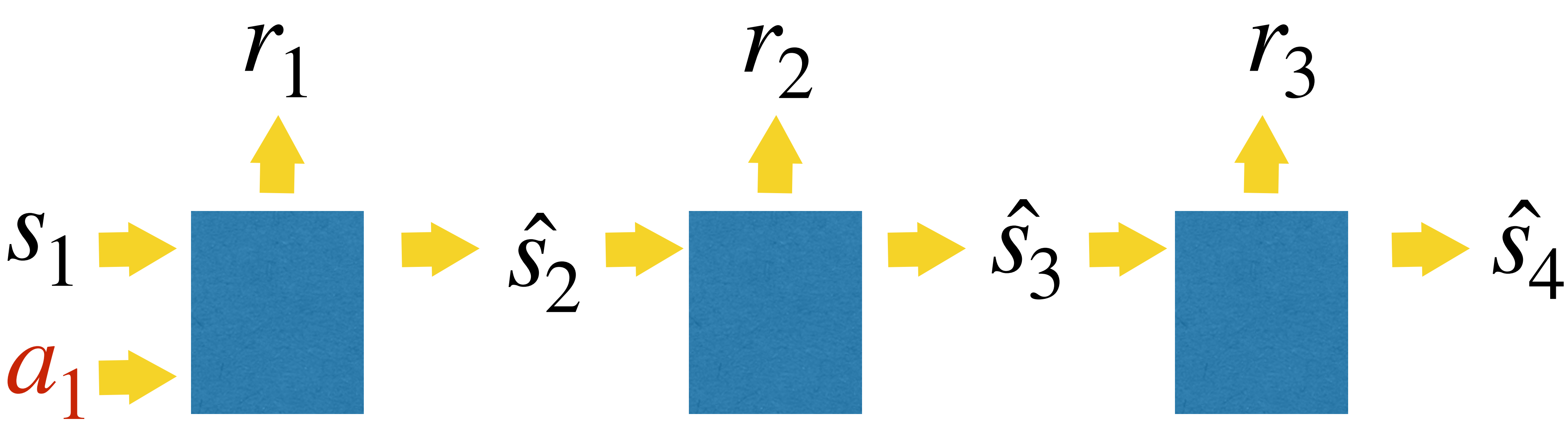
# Planning: Model unrolling for action selection

Feeding the predictions back as input to the model.



# Planning: Model unrolling for action selection

Feeding the predictions back as input to the model.

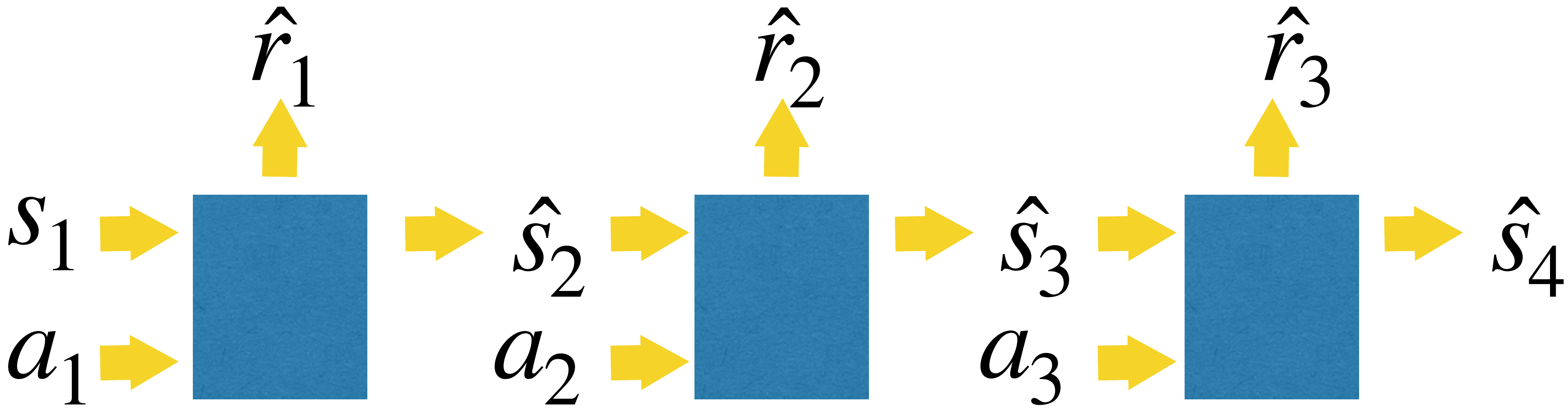


# Planning: Model unrolling for action selection

$s_0$ 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.

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

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



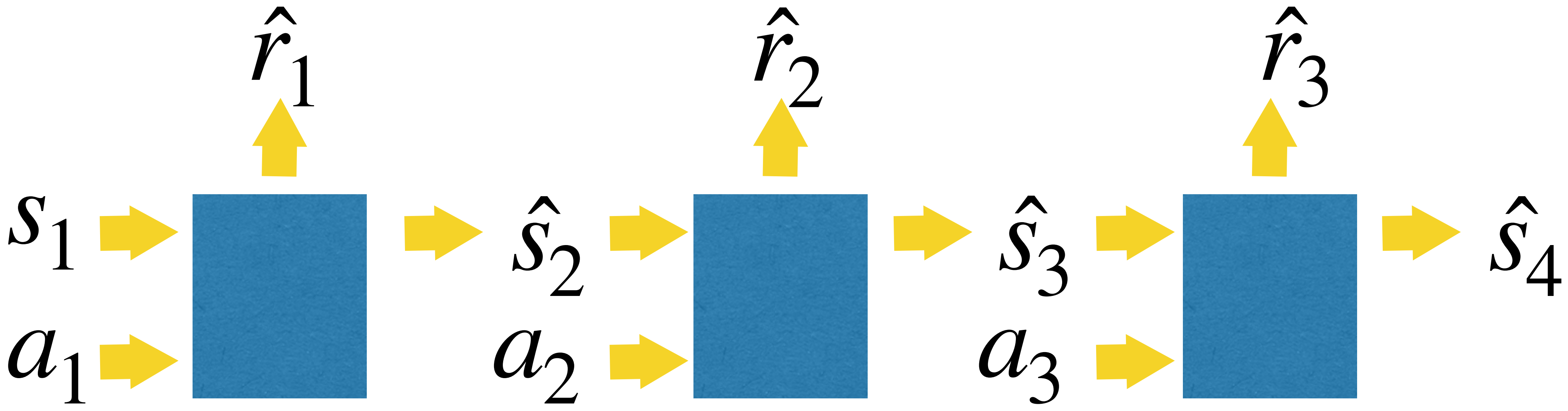
# Planning: Model unrolling for action selection

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.

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

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

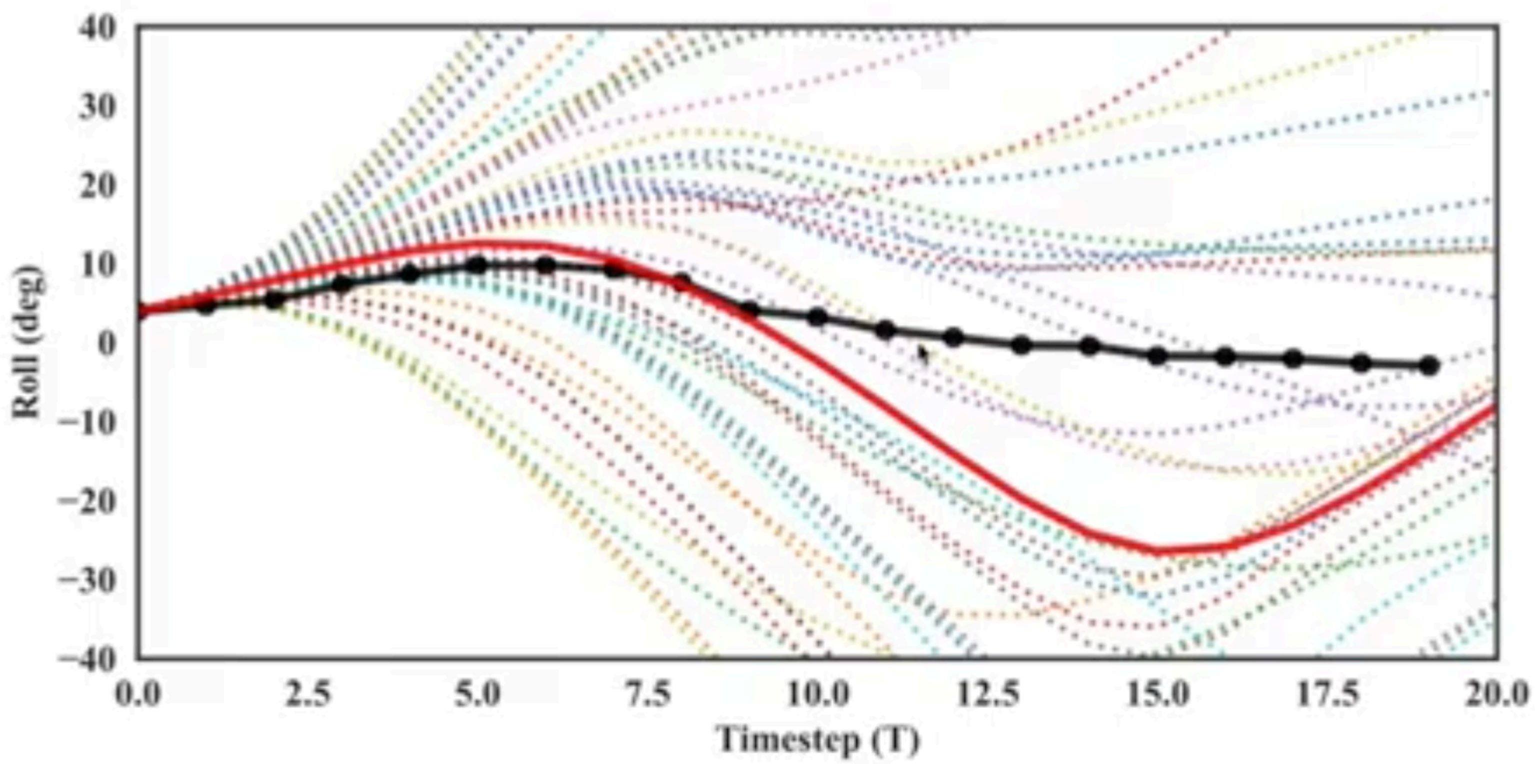
If the dynamics are non-linear and the loss is not a quadratic, this optimization is difficult. We can use gradient descent optimization, evolutionary search, MCTS, etc..



# Trajectory prediction through model unrolling

$$s_T = f_\theta\Big(f_\theta(\cdots f_\theta(s_i, a_i)\cdots)\Big)$$

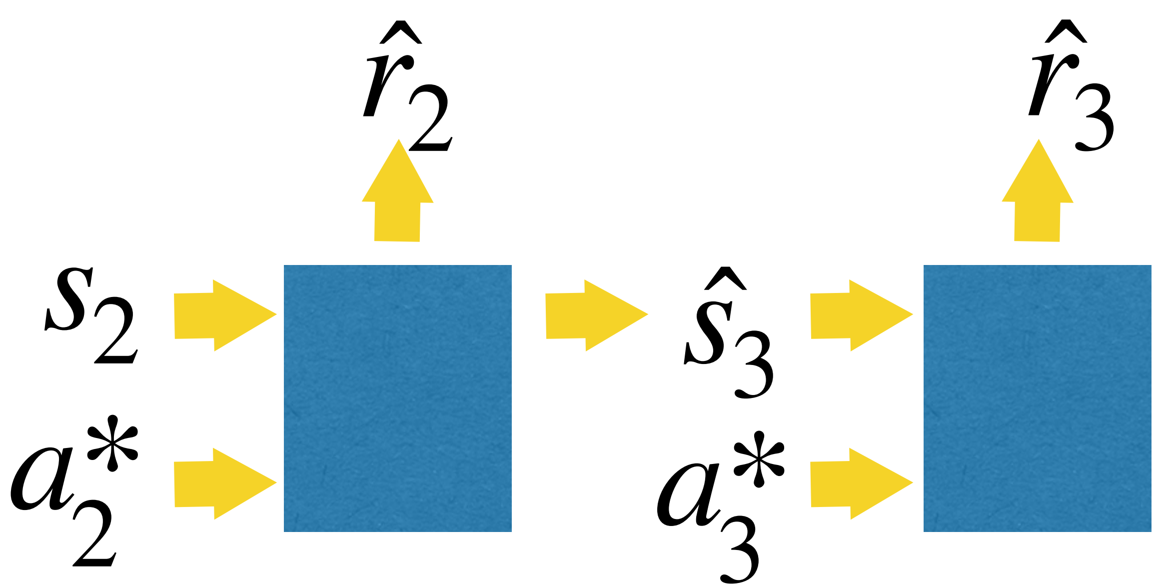
Many compounded network passes!



# 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 timestep helps fight model error accumulation through unrolling.

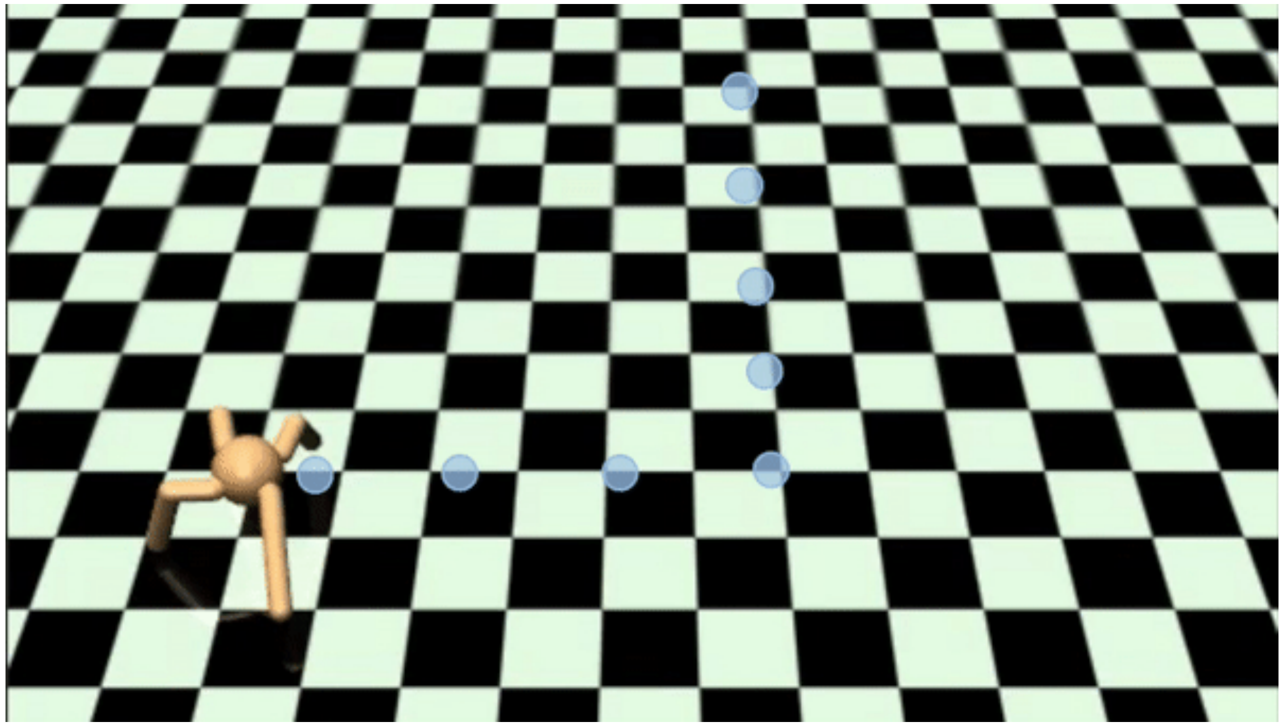
$$\max_{a_2 \cdots a_T} . \sum_{t=2}^{T} \hat{r}_t$$

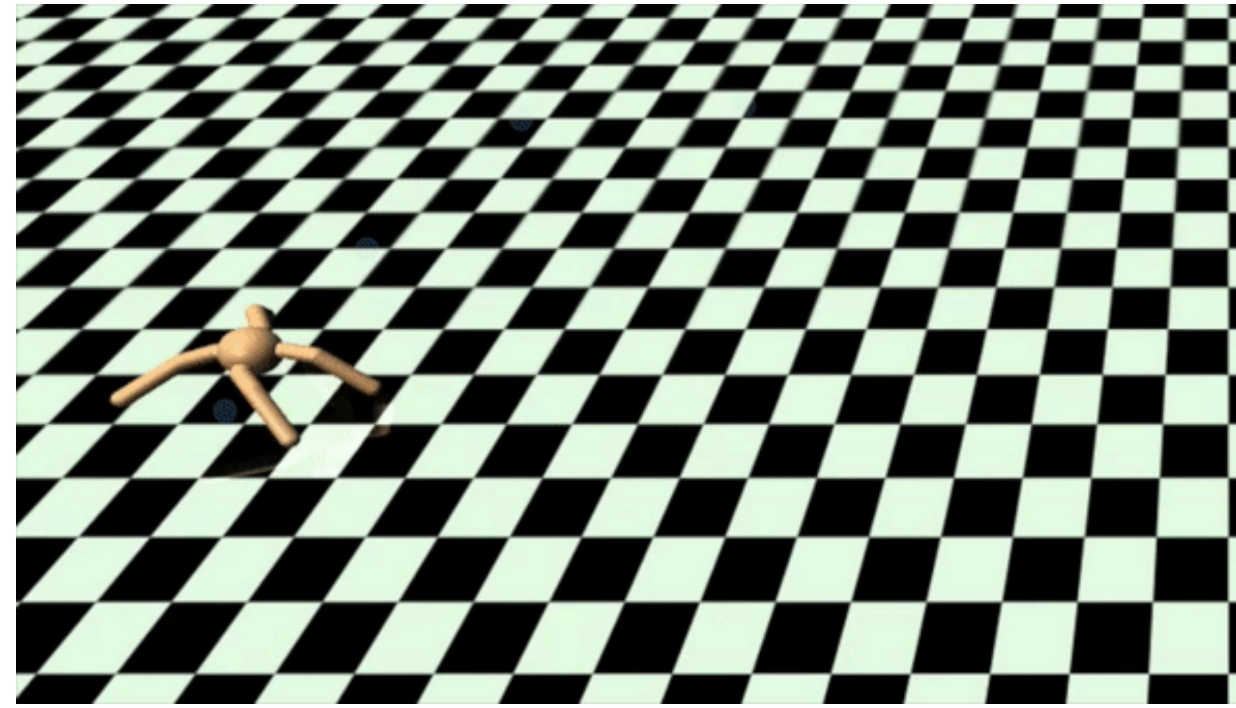


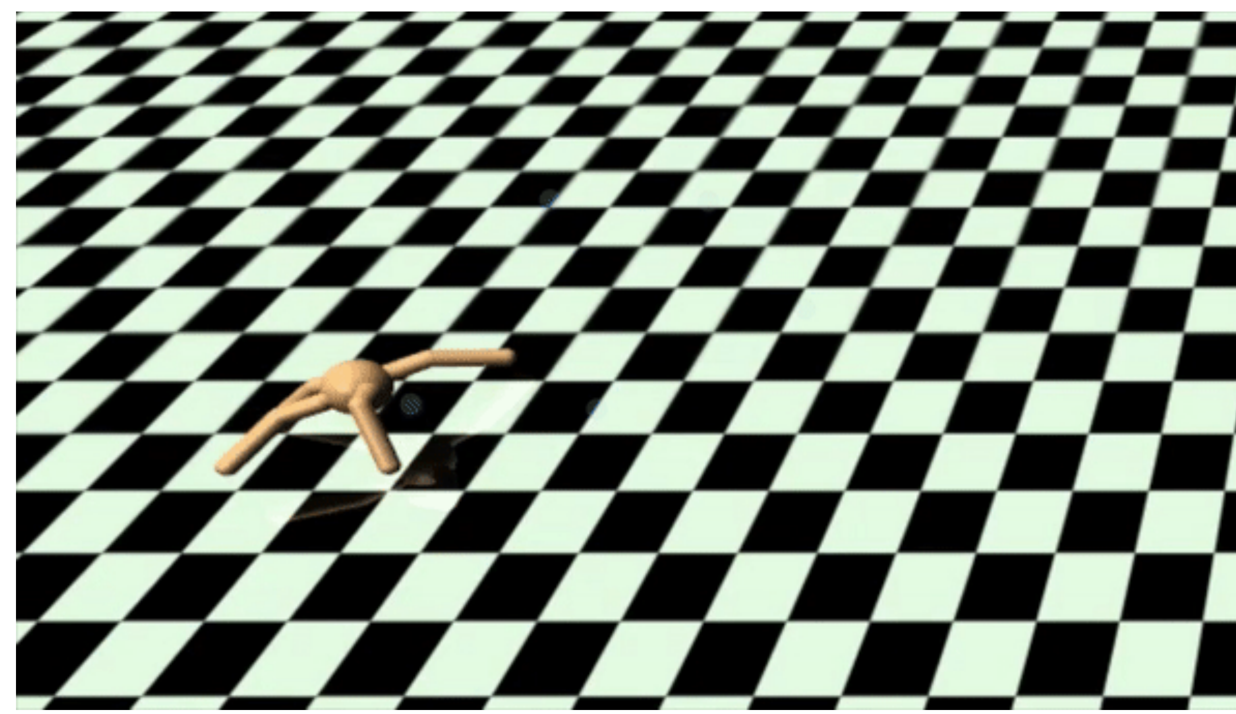
# Benefits of Model-based Reinforcement learning

- 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







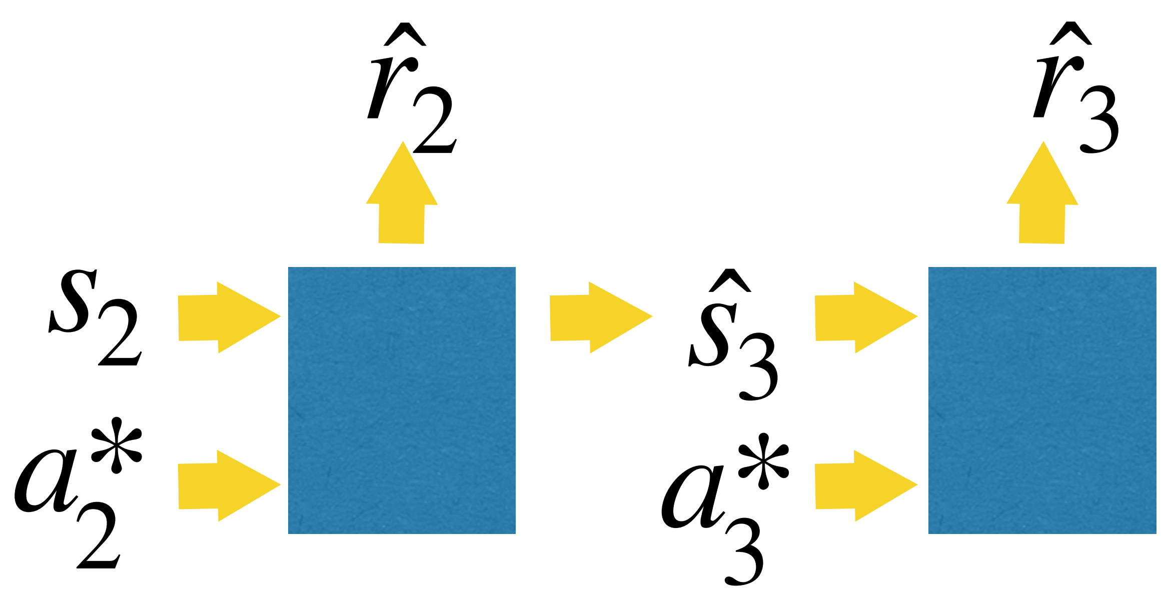
Training a model based controller allows to follow arbitrary trajectories at test time: the model allows you to optimize different reward function for different tasks, without any retraining.

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

# When models are learnt

# MBRL as a two step process

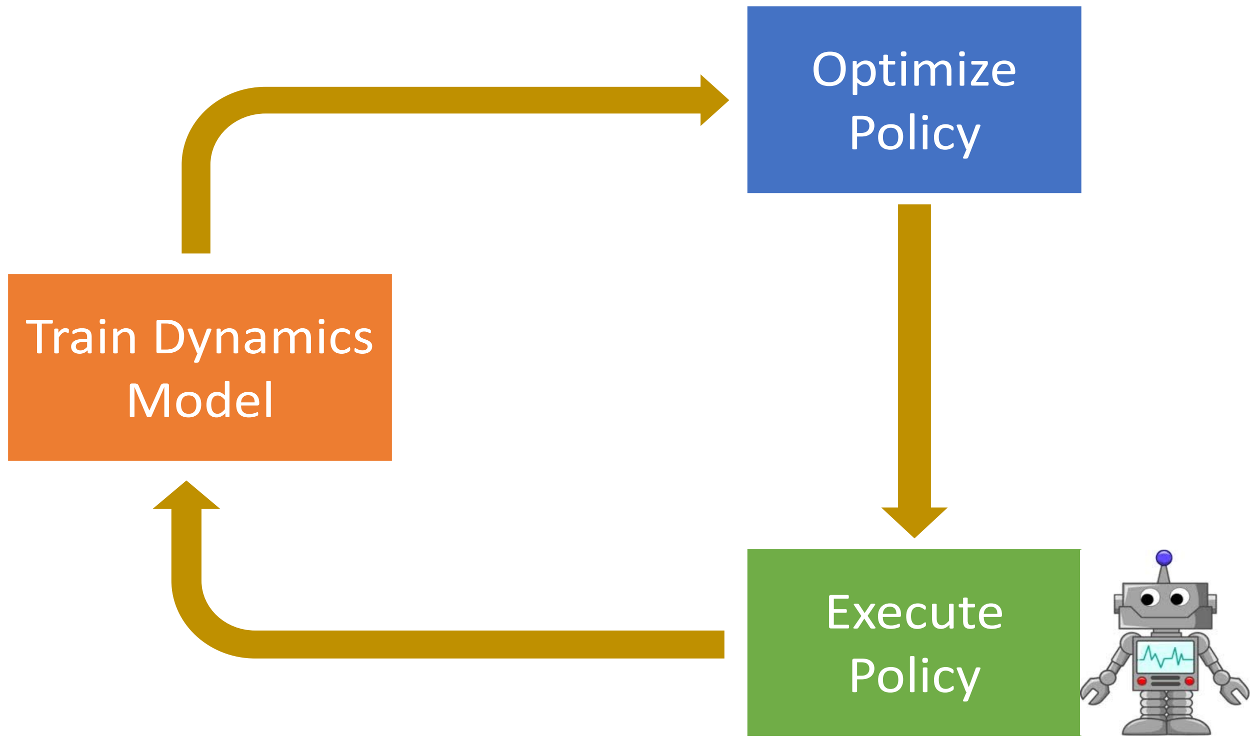
1. Initialize policy $\pi(s;\theta)$ and obtain (random) experience tuples $D_{env}$.
2. Train a dynamic model using $D_{env}$ using single step or multi-step maximum likelihood: $\max_{\theta} \sum_{i=1}^{N} \log p_\theta(s_i', r_i' \mid s_i, a_i)$
3. Update the policy by
    1. a model-free RL method on simulated experience $D_{model}$ sampled from the model.
    2. Imitating a model-based controller (planner). $\max_{a_1 \cdots a_T} . \ \mathbb{E}_{p_\theta} \sum_{t=1}^{T} \hat{r}_t$



# Alternating between model and policy learning

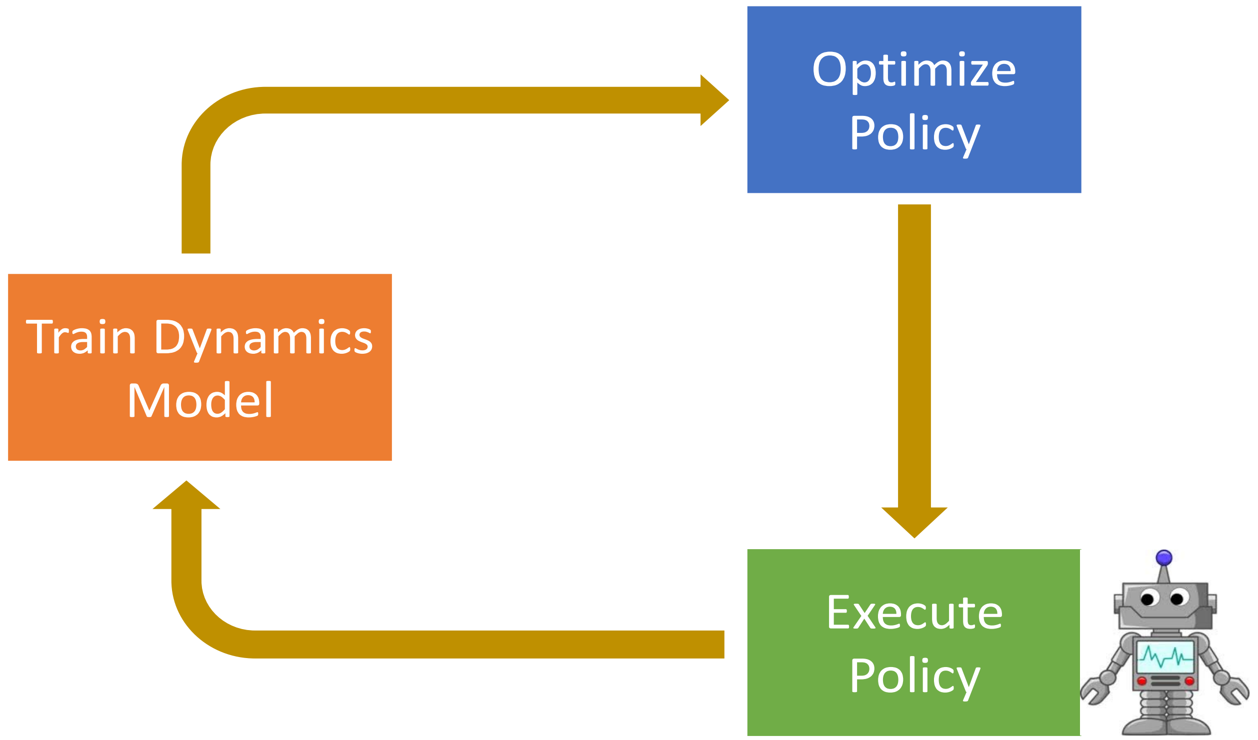
Initialize policy $\pi(s;\theta)$ and $D_{env} = \{\}$.

1. Run the policy and update experience tuples dataset $D_{env}$.
2. Train a dynamic model using $D_{env}$: $(s', r') = f(s, a; \theta)$
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 between model learning and policy learning (action selection) ?

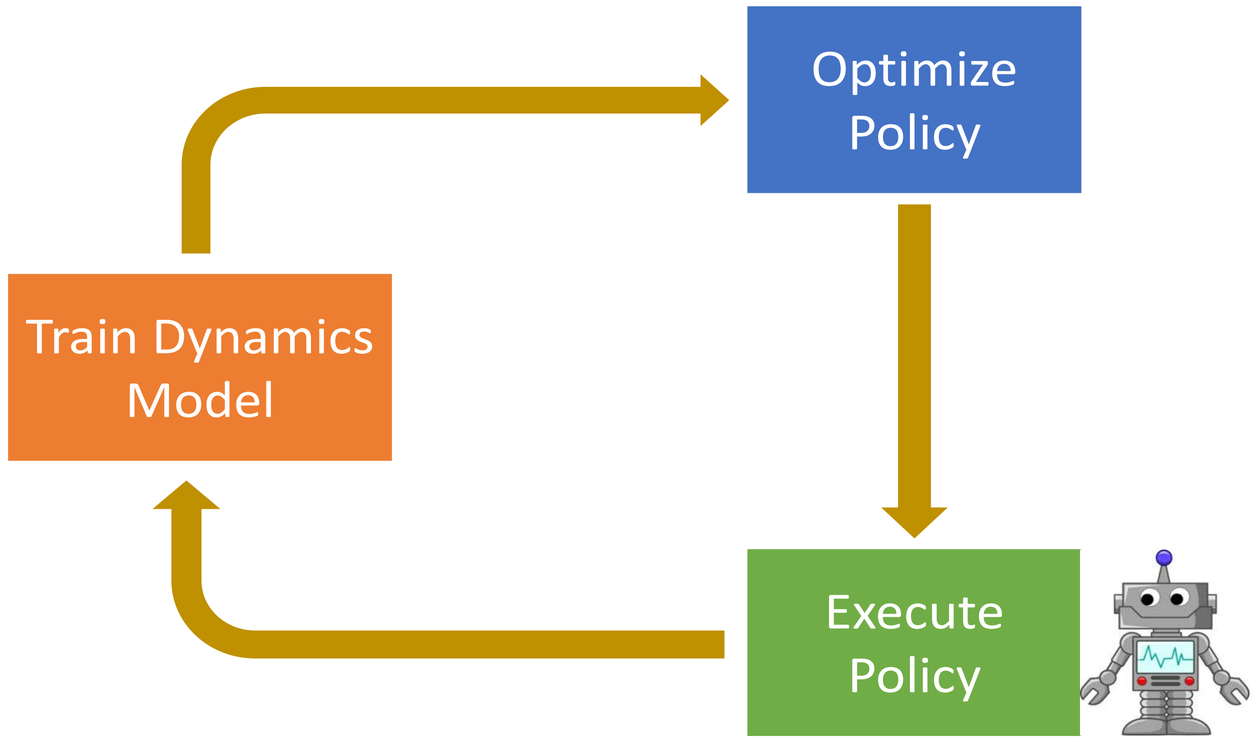
Model dynamics are only accurate close to the data used to train the model. If we plan through the model or if we sample from it, we may land far from the model's training data distribution. Deploying the predicted actions in the environment and updating the modelwith the newly collected experience tries to bring the data distribution the model is trained on with the state distribution visited by the selected actions (the policy learnt).



# Alternating between model and policy learning

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

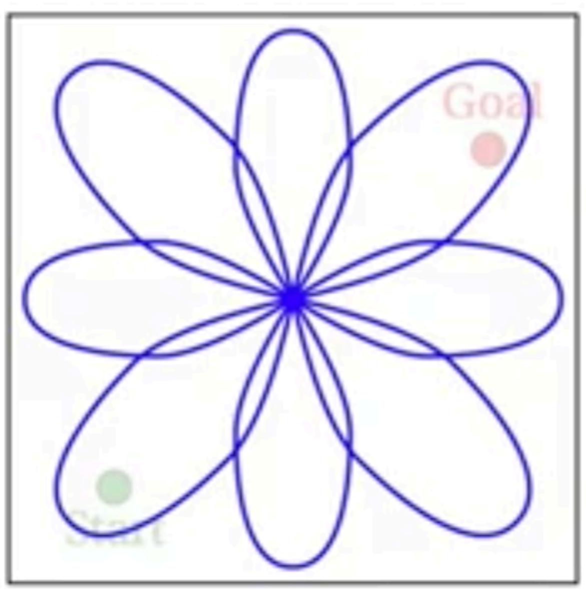
1. Run the policy and update experience tuples dataset $D_{env}$.
2. Train a dynamic model using $D_{env}$: $(s', r') = f(s, a; \theta)$
3. Update the policy by
   1. a model-free RL method on simulated experience $D_{model}$ sampled from the model.
   2. Imitating a model-based controller (planner)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



# Two step process versus alternation

**System Identification**

- Obtain a task-agnostic (sometimes global) model
- Then learn control




**Reinforcement learning**

- Observe task-specific data subset
- Iteratively learn model, control

# 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).
- **Disconnect between model learning objectives and reward optimization objectives.**

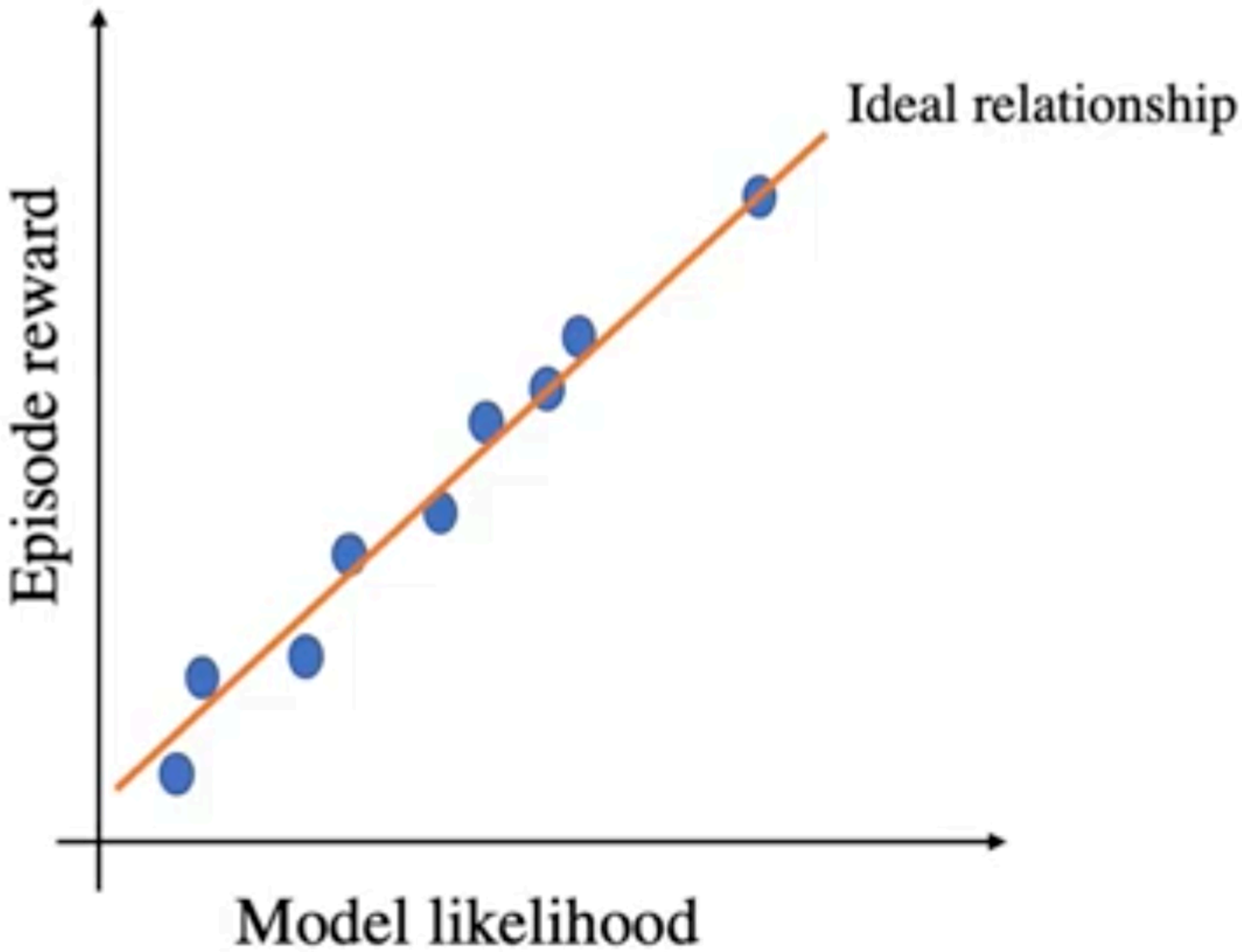
# Objective Mismatch in MBRL

$$\textbf{Training:}\ \arg\max_{\theta} \sum_{i=1}^{N} \log p_\theta(s_i'|s_i, a_i), \quad \textbf{Control:}\ \arg\max_{a_{t:t+T}} \mathbb{E}_{\pi_\theta(s_t)} \sum_{i=t}^{t+T} r(s_i, a_i)$$

Models are useful if they yield good policies, but they are trained to maximize likelihood of transitions, rather than the performance of the policies that result from them.

A model that makes small mistakes in critical states cancause a policy to take suboptimal actions. Alternatively, a model with large errors may yield a policy that attains high return if the model errors occur in states that the policy never visits.

# Ideally



# In practise


(*a*) CP Expert ($\rho = 0.59$) (*b*) CP On-Policy ($\rho = 0.34$) (*c*) CP Grid ($\rho = -0.06$)

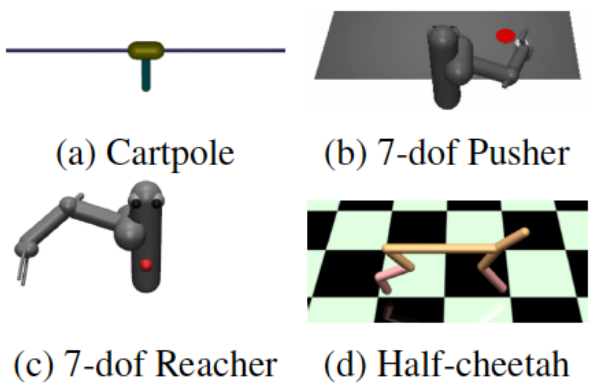
(*d*) HC Expert ($\rho = 0.07$) (*e*) HC On-Policy ($\rho = 0.46$) (*f*) HC Sampled ($\rho = 0.19$)

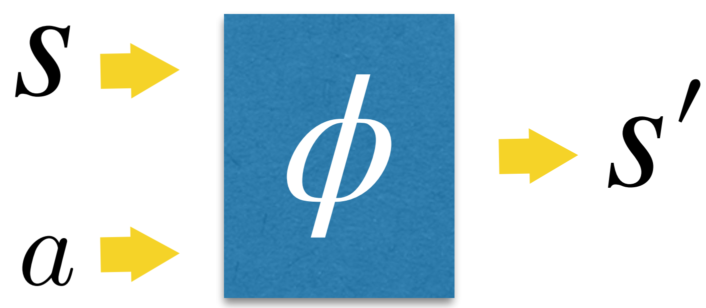
We will get back to this topic next week.

*Objective mismatch in MBRL, Lambert et al.*

# Model Learning

Where a low dimensional state is observed and given:

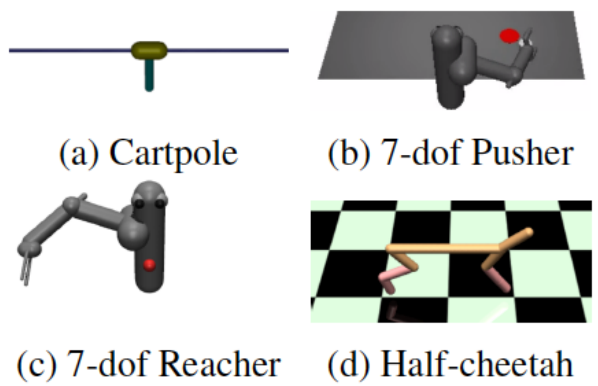


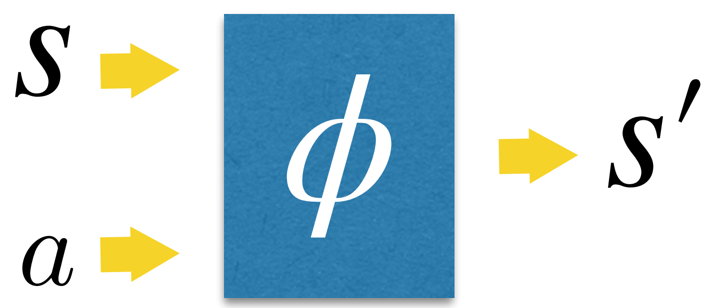




state can be 3D locations and 3D velocities of agent joints, actions can be torques

# Model Learning

## Where a low dimensional state is observed and given:

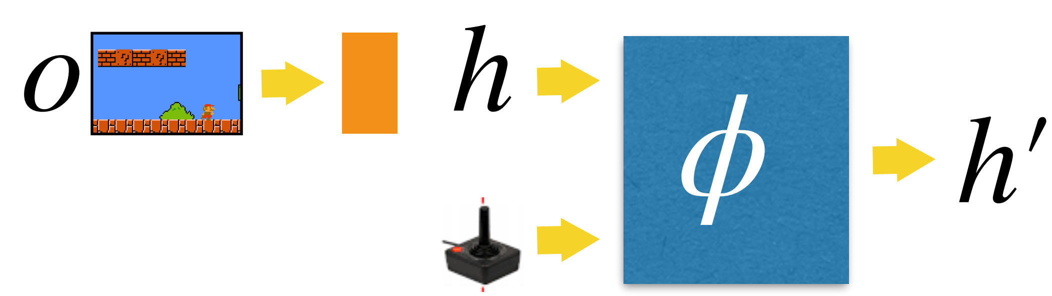


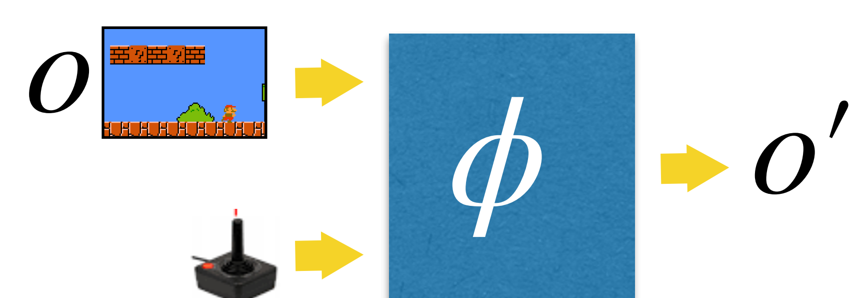




state can be 3D locations and 3D velocities of agent joints, actions can be torques

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

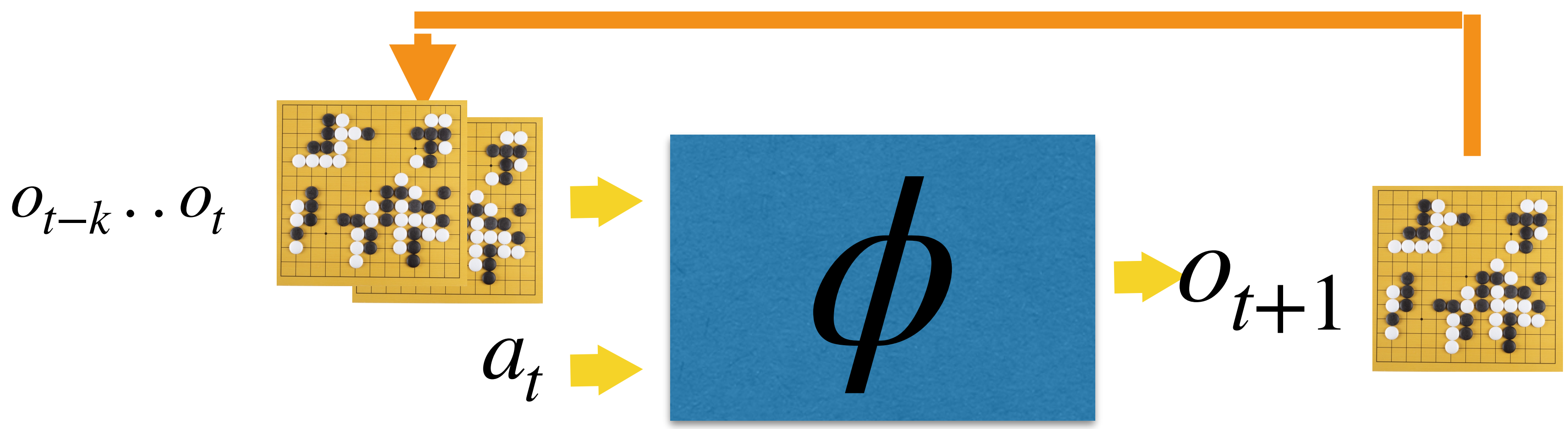
e.g., Atari game playing



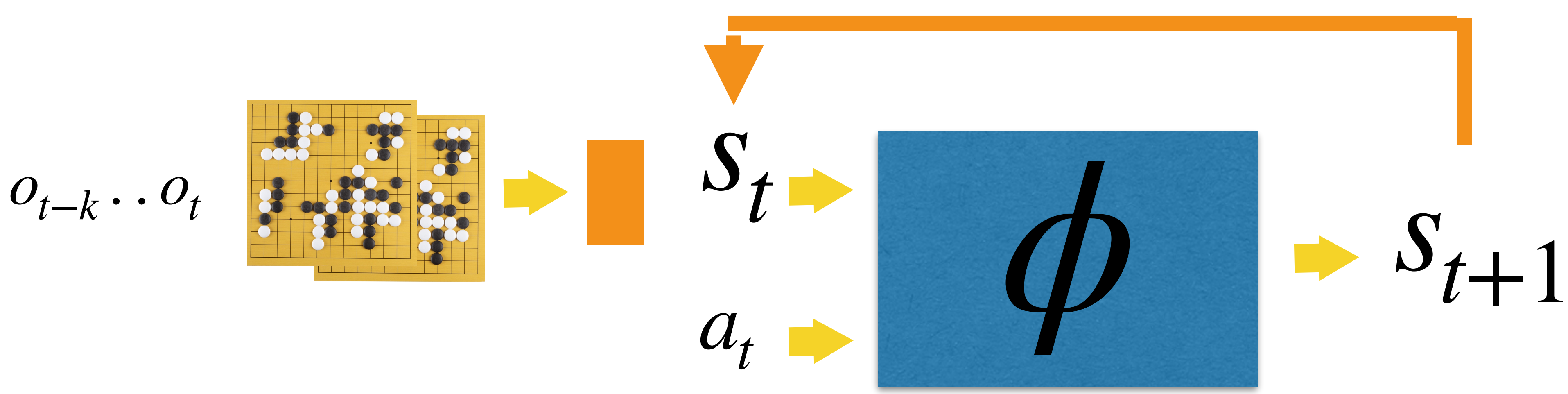



# Model learning from sensory input

Unrolling in the observation space: the model is trained to predict future observations.

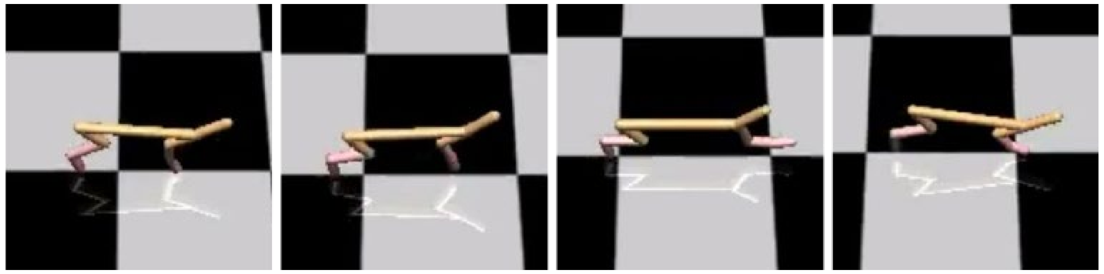


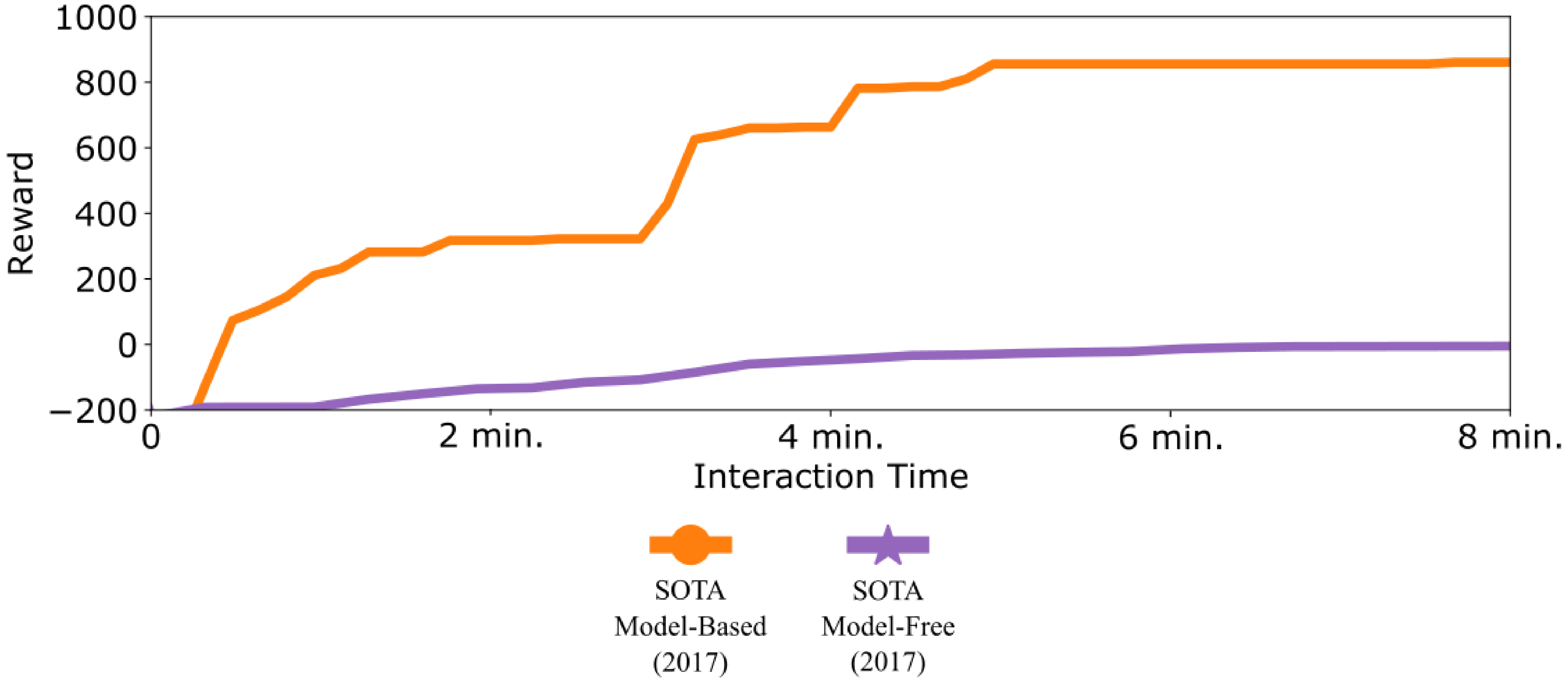

Unrolling in the latent space: optionally use observation reconstuction as an auxiliary loss.



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

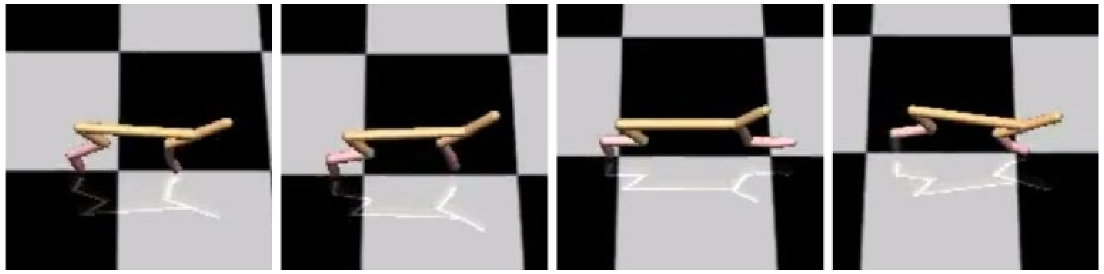
# Comparative Performance on HalfCheetah

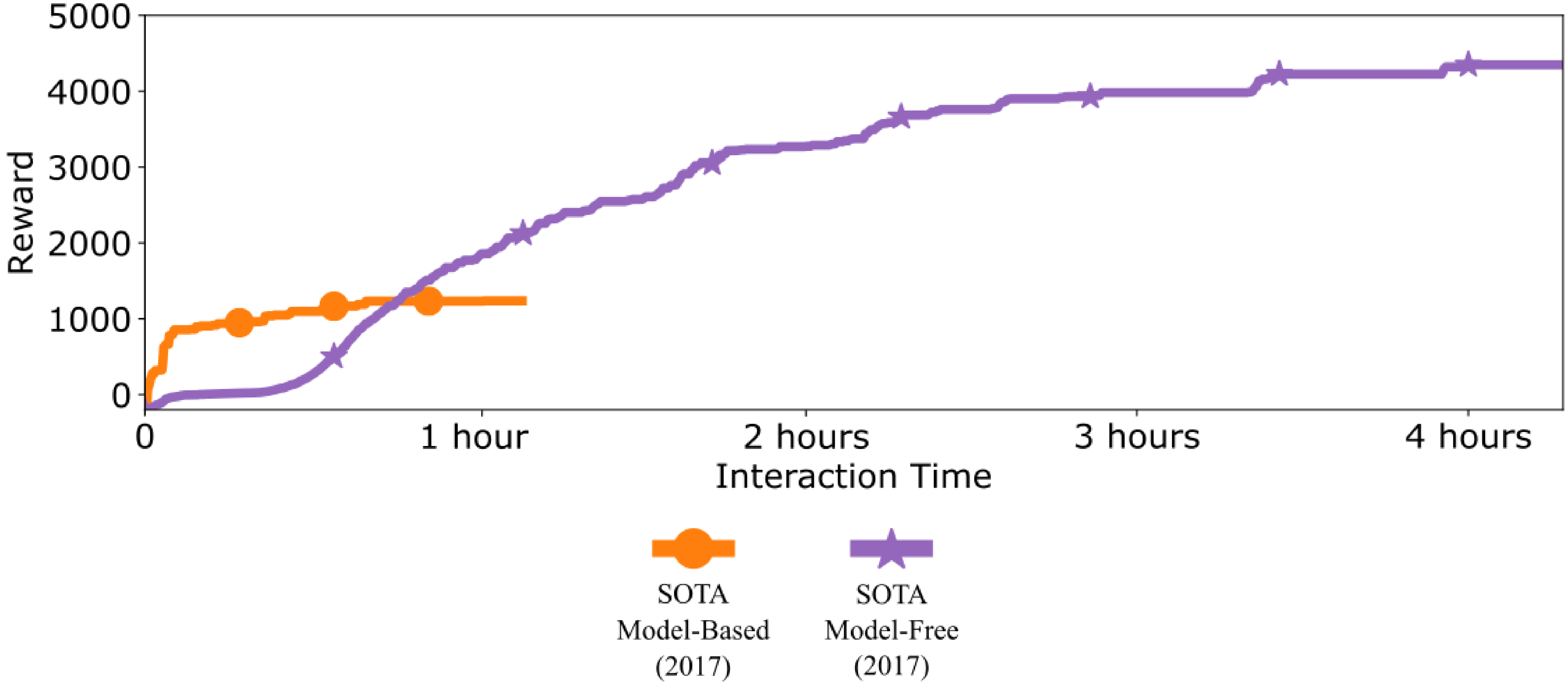





slide from Sergey Levine

# Comparative Performance on HalfCheetah

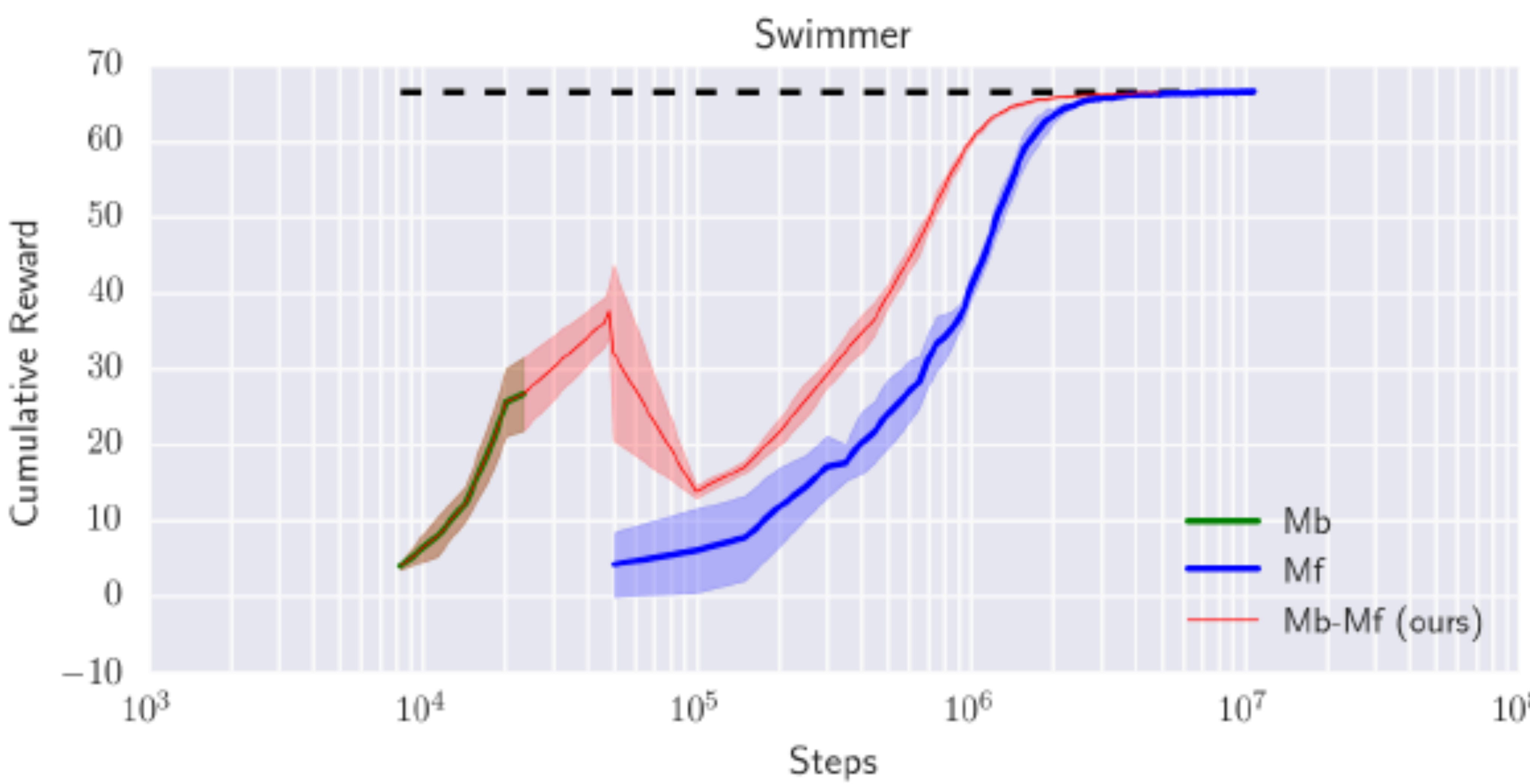





slide from Sergey Levine

# 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: $s' = s + f(s$

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 Rl 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

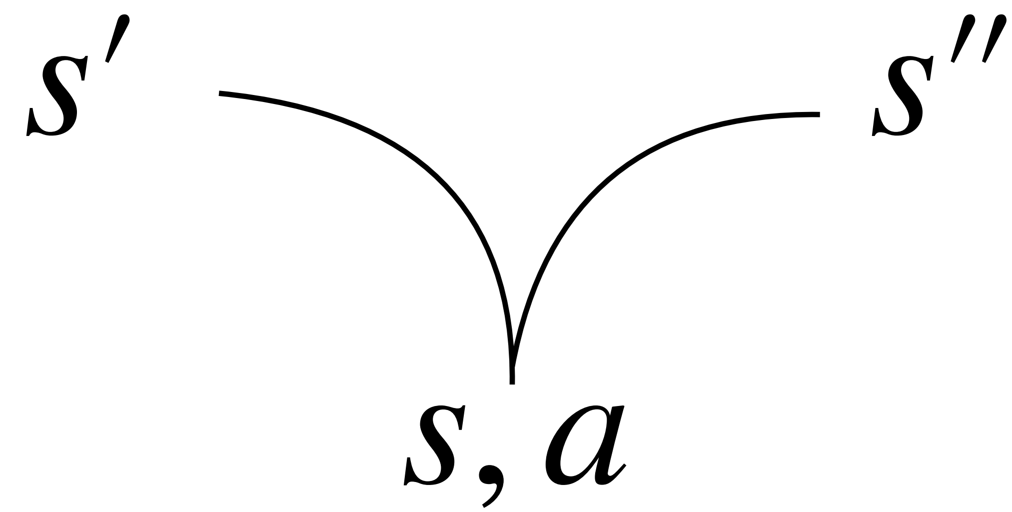
**Kurtland Chua** **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu

## Represent model's 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



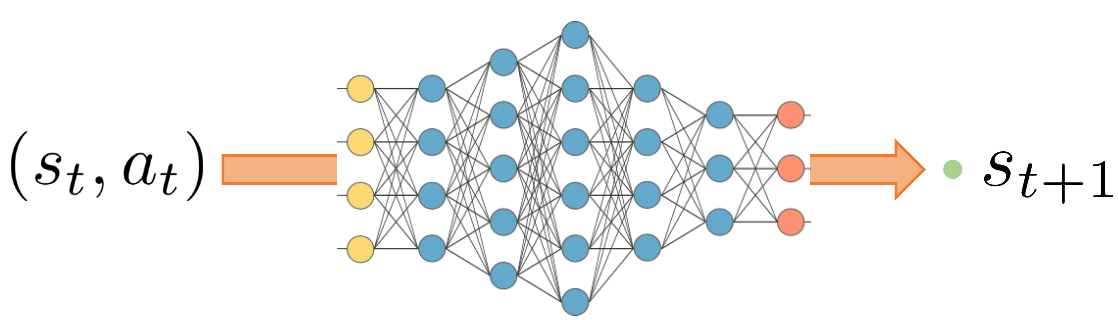

The environment can be stochastic

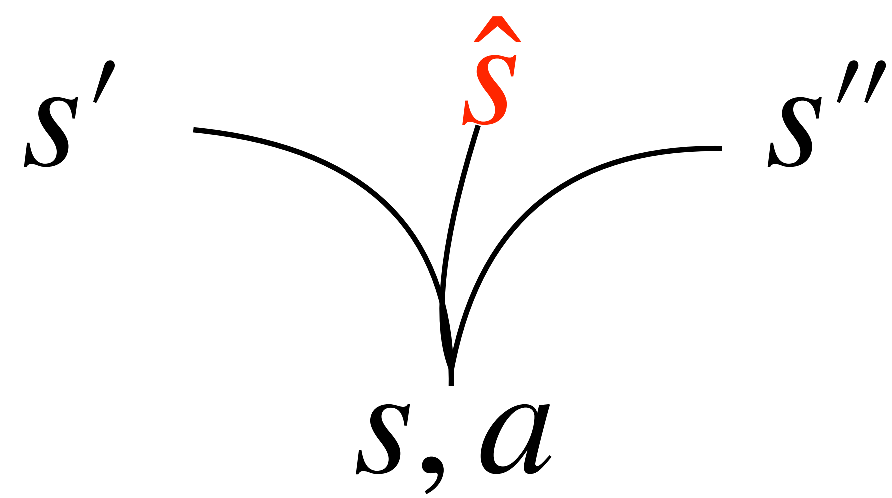
- This means our state does not capture enough information to help us delineate the possible future outcomes.
- What is stochastic under one state representation, may not be stochastic under another. Is this true? Could we ever predict exactly what we will see in the TV when we switch the channel?
-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

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



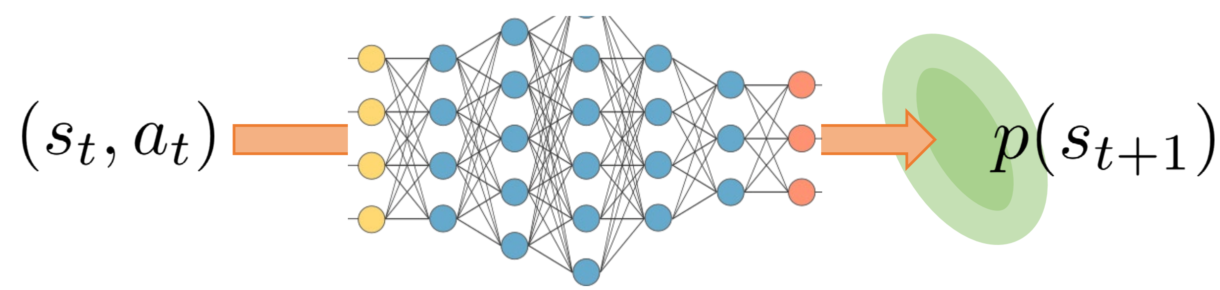




Failing means: not only we cannot capture the distribution, but we output a solution that does not agree with any of its modes!

# 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)

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



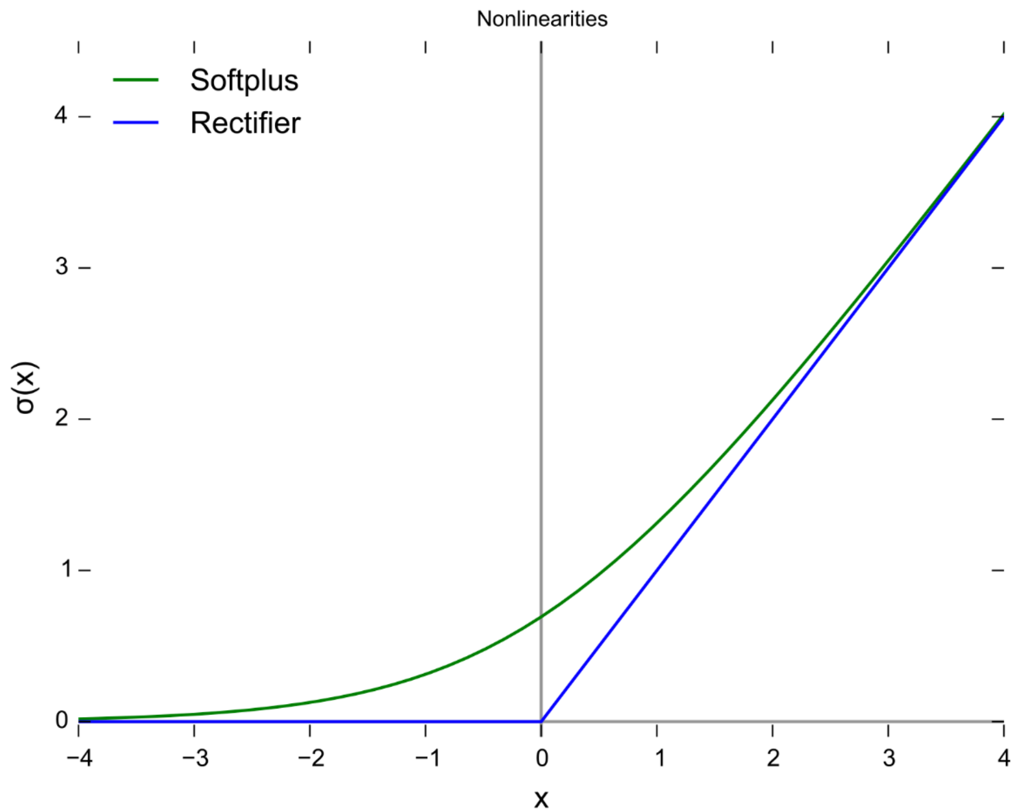

$$\mathcal{L}_\phi = -\frac{1}{N}\sum_{i=1}^{N} \log p_\phi(s_i'|s_i, a_i)$$

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

# Aleatoric uncertainty in model learning

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

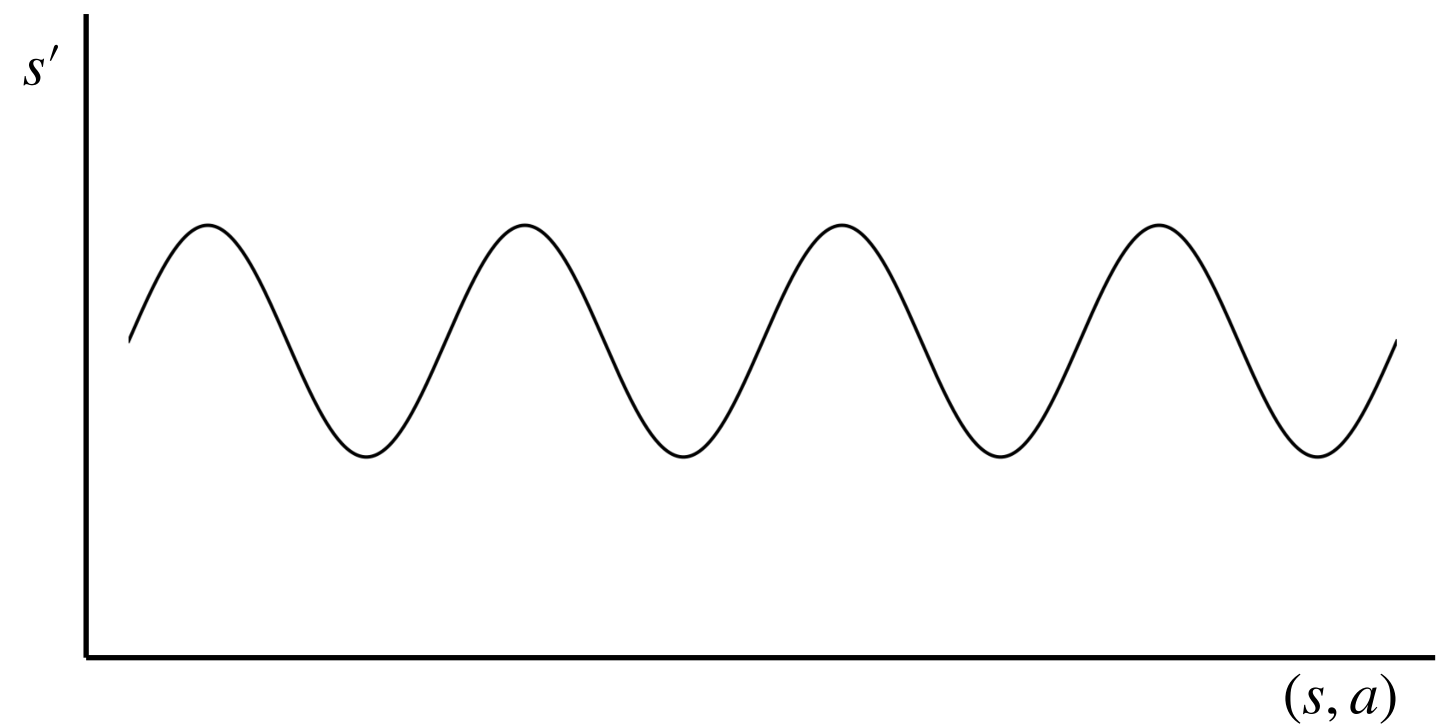
```
logvar = max_logvar - tf.nn.softplus(max_logvar - logvar)
logvar = min_logvar + tf.nn.softplus(logvar - min_logvar)
var = tf.exp(logvar)
```



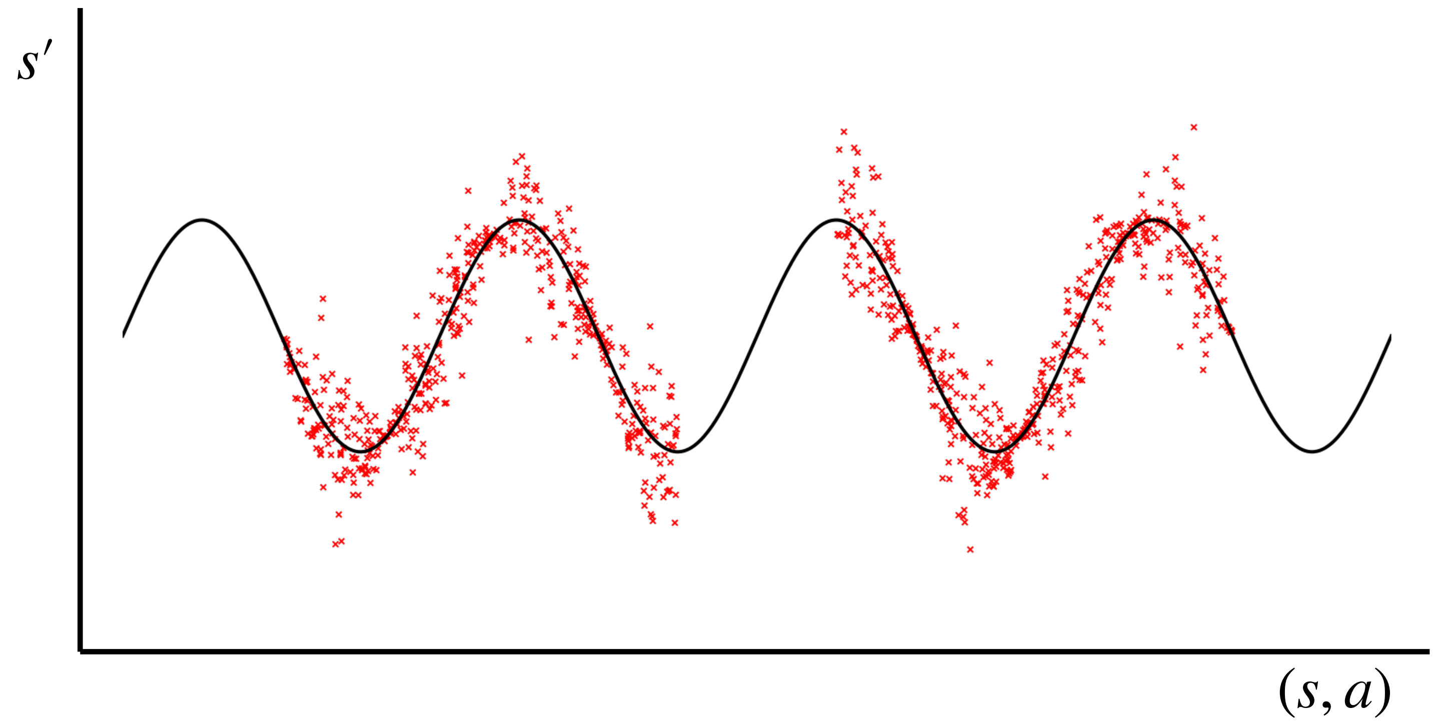

$\text{Rectifier}(x) = \max(0,x)$

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

# Epistemic uncertainty in Model Learning

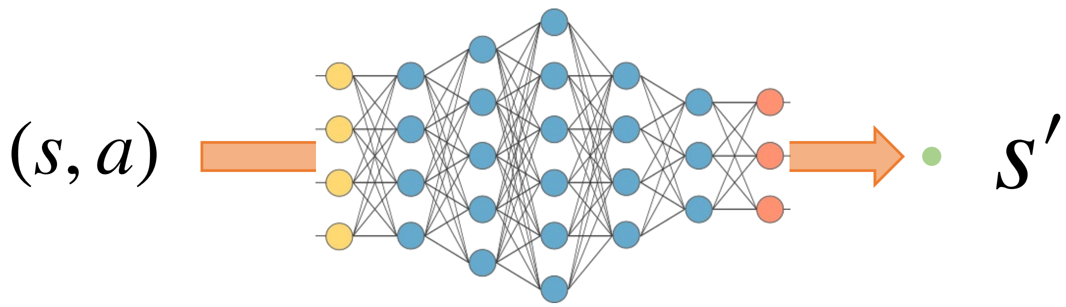


# Epistemic uncertainty in Model Learning

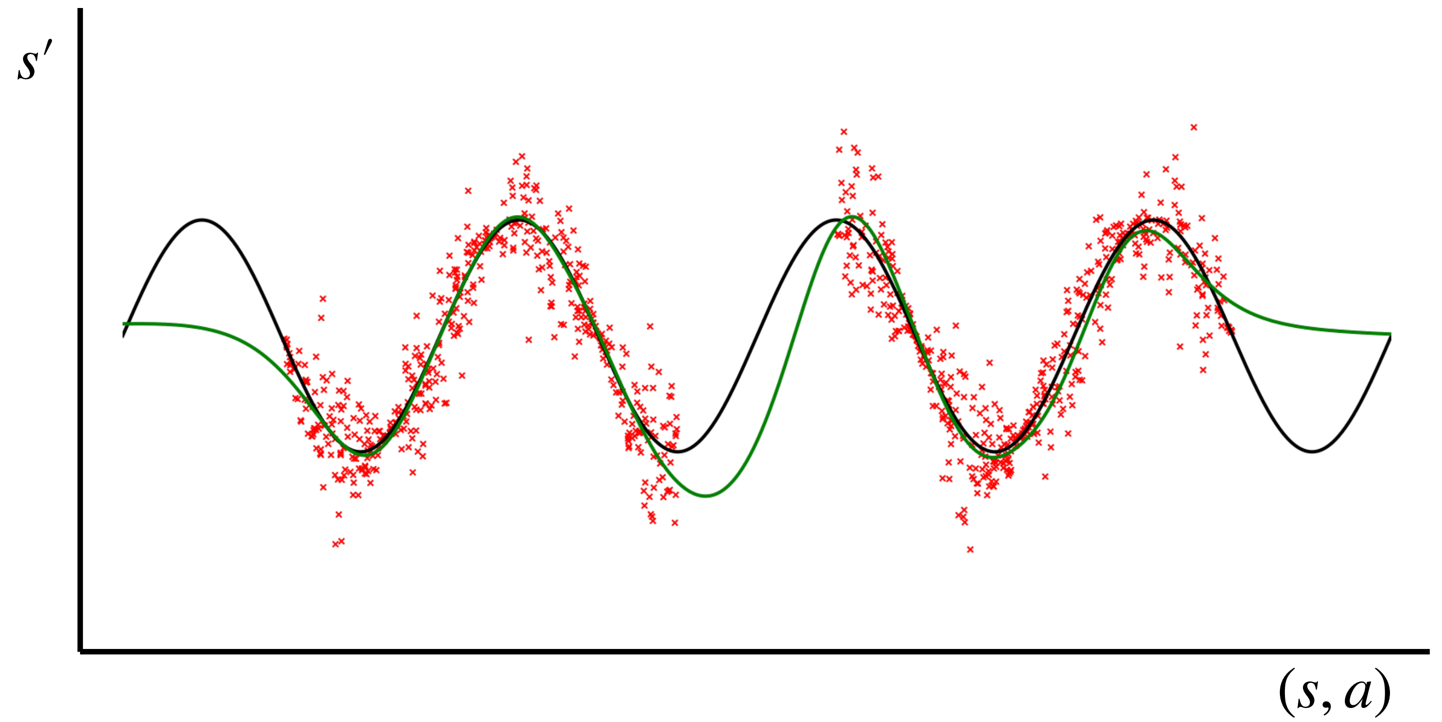



Red are observed data points (s,a,s')

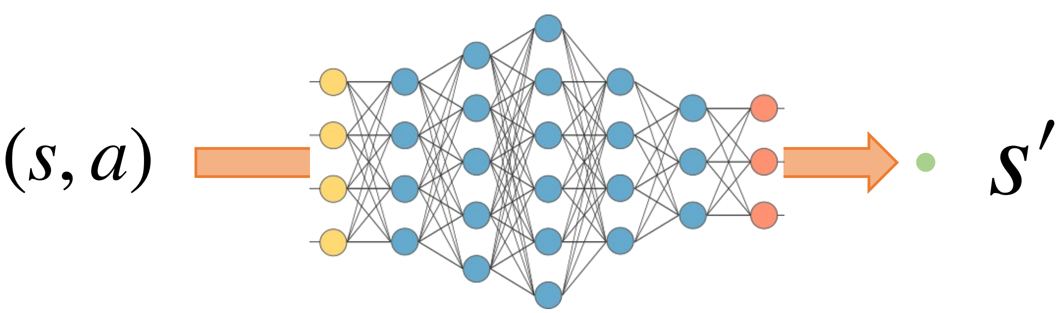
# Epistemic uncertainty in Model Learning



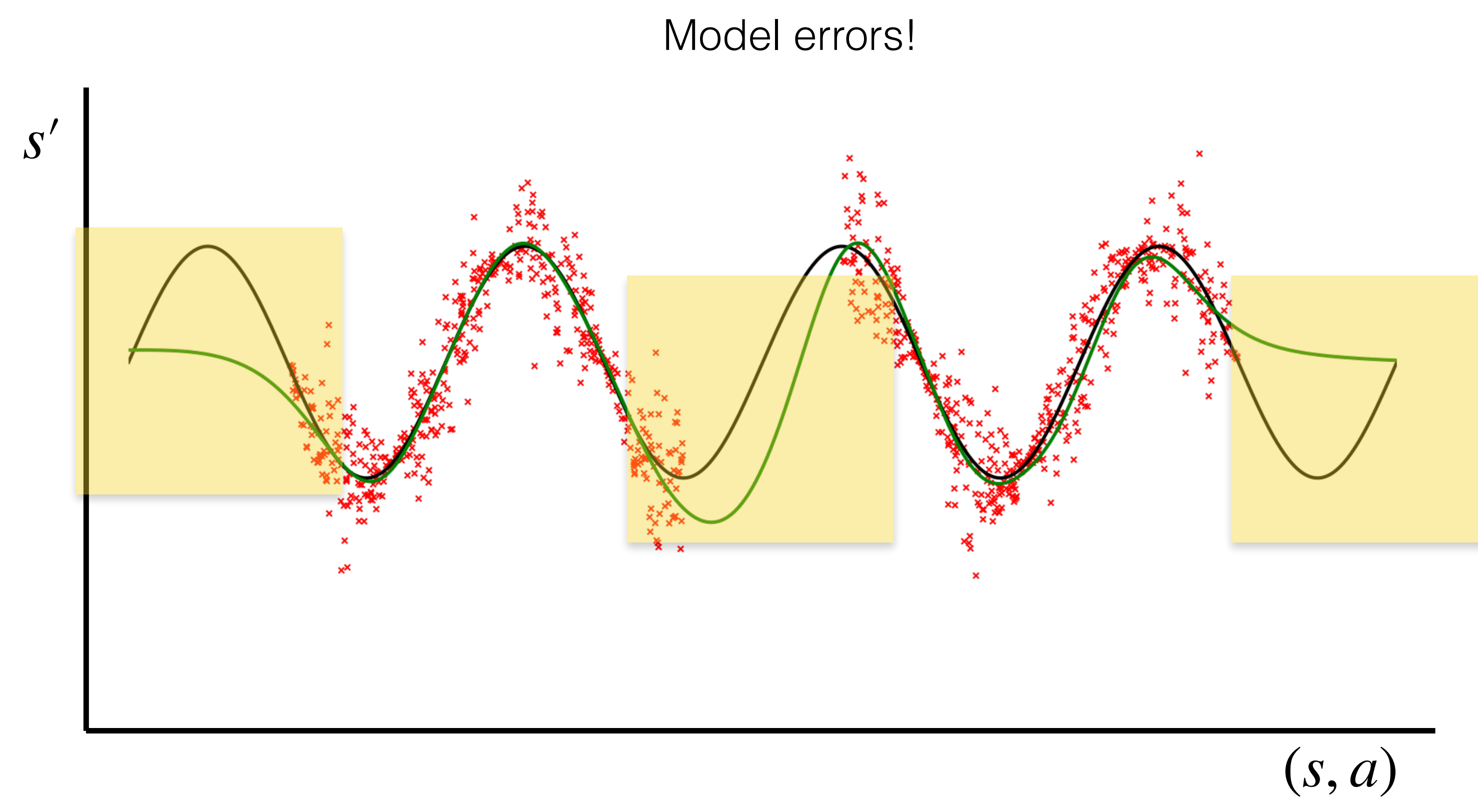

Fitting a deterministic neural network



# Epistemic uncertainty in Model Learning

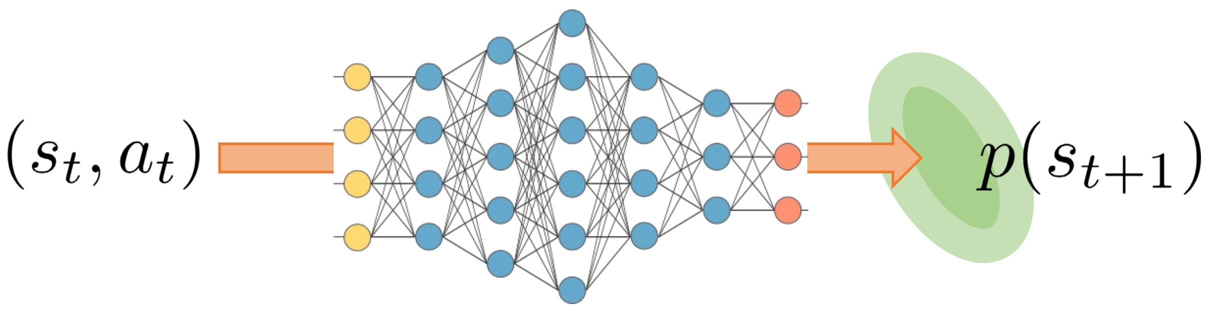



Fitting a deterministic neural network

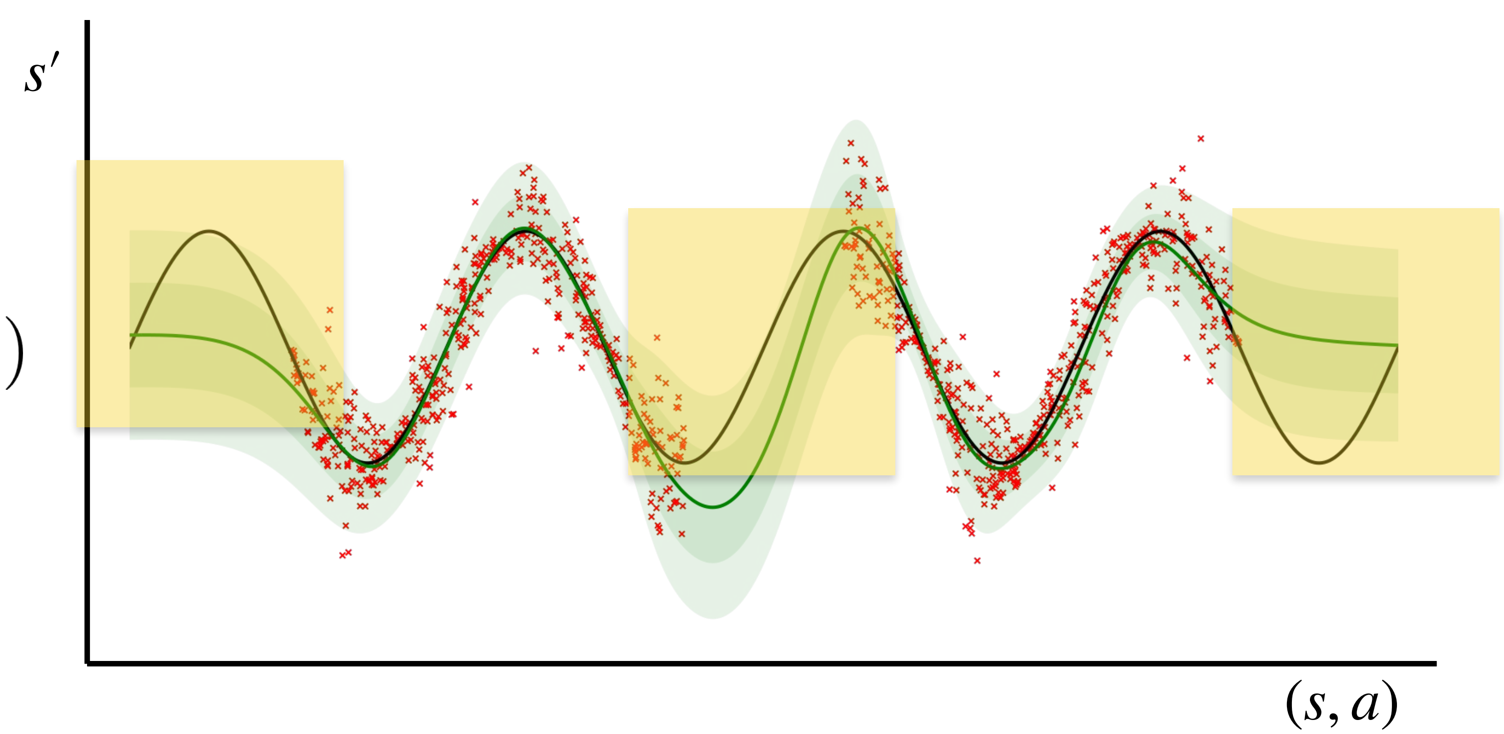



- There is a unique answer for s’ (no stochasticity) but I do not know it due to lack of data.

# Epistemic uncertainty in Model Learning

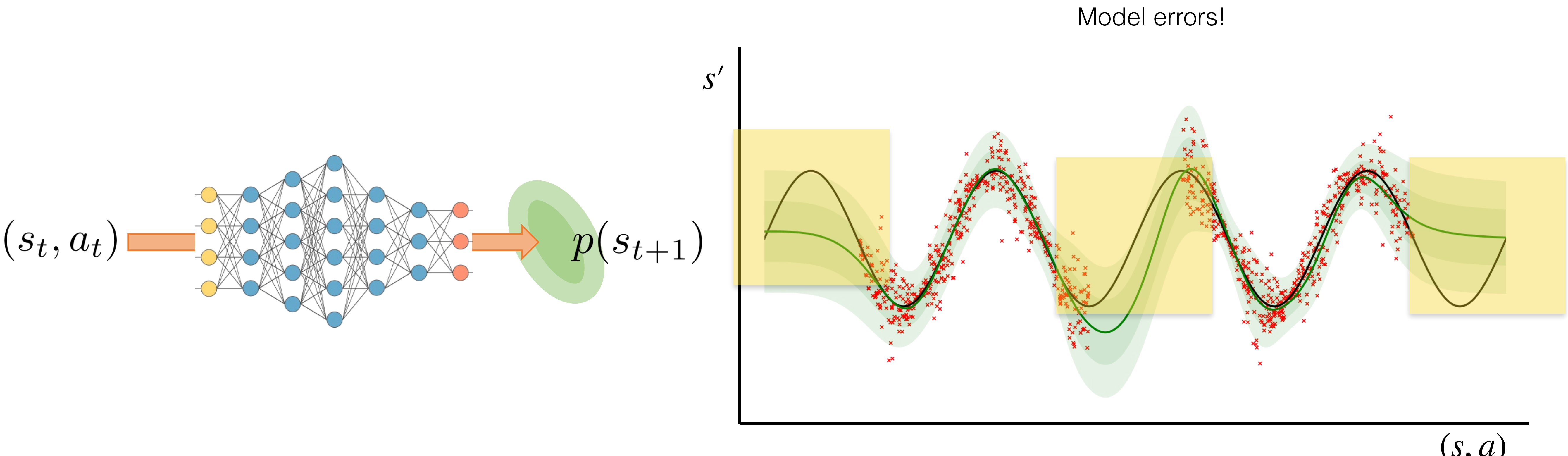



Fitting a neural network that parametrizes a distribution




- 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



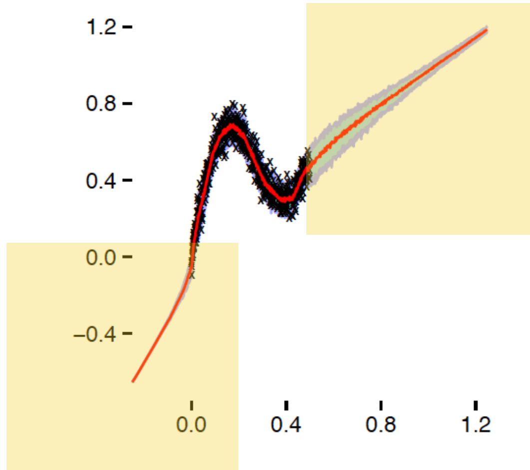

- There is a *unique* answer for s' (no stochasticity) but I do not know it due to lack of data.
- Predicting a distribution won't help. The predictions will be inaccurate due to lack of data.
- How can I represent my uncertainty about my predictions? E.g., having high entropy when no data and low entropy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

# Bayesian Inference

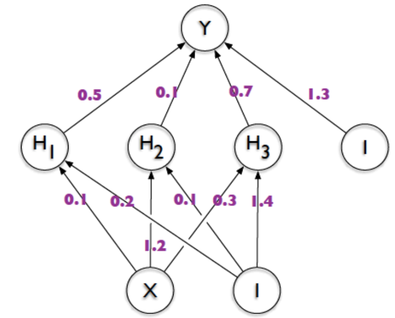
# Bayes Rule

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

- 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.



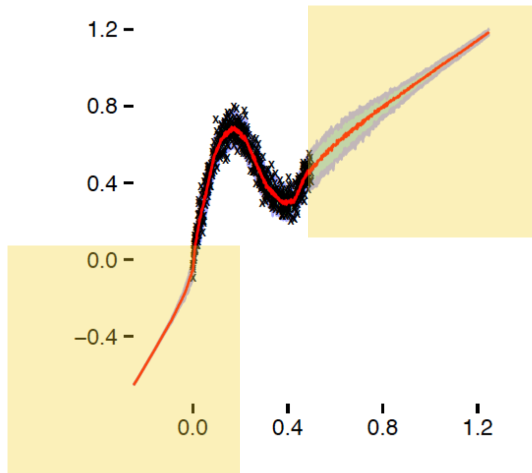

regression network



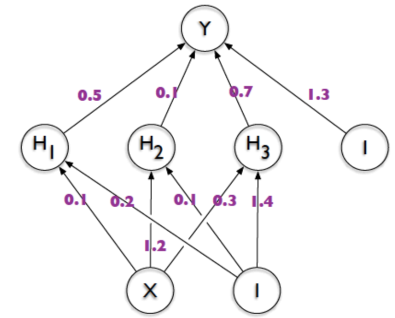

$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left(P(D \mid \phi) + \log P(\phi)\right)$$

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

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

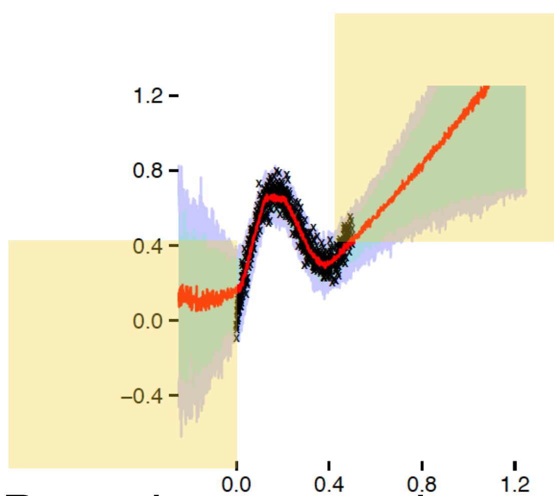



regression network



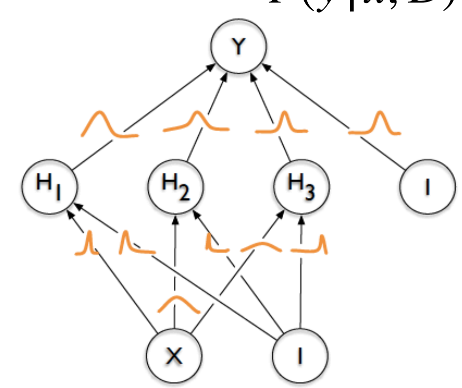

$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \,|\, D) = \arg\max_{\phi} \left(P(D \,|\, \phi) + \log P(\phi)\right)$$

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




Bayesian regression network

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

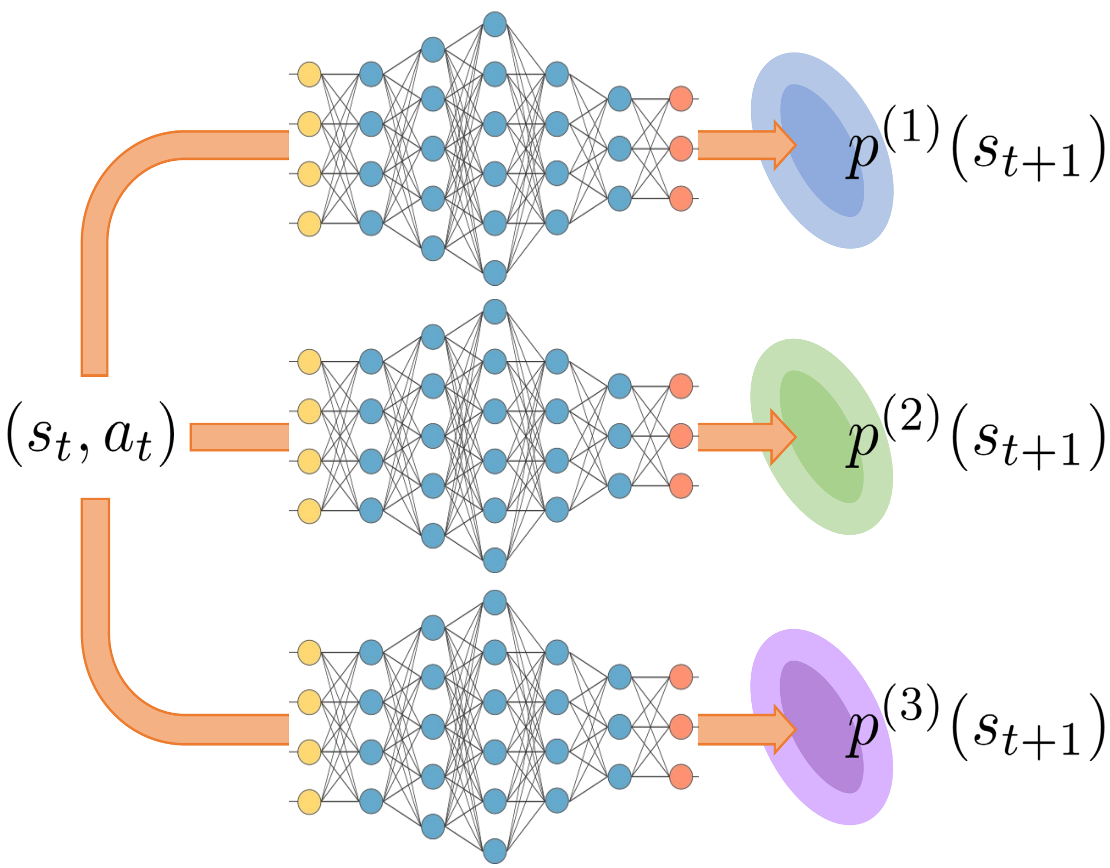


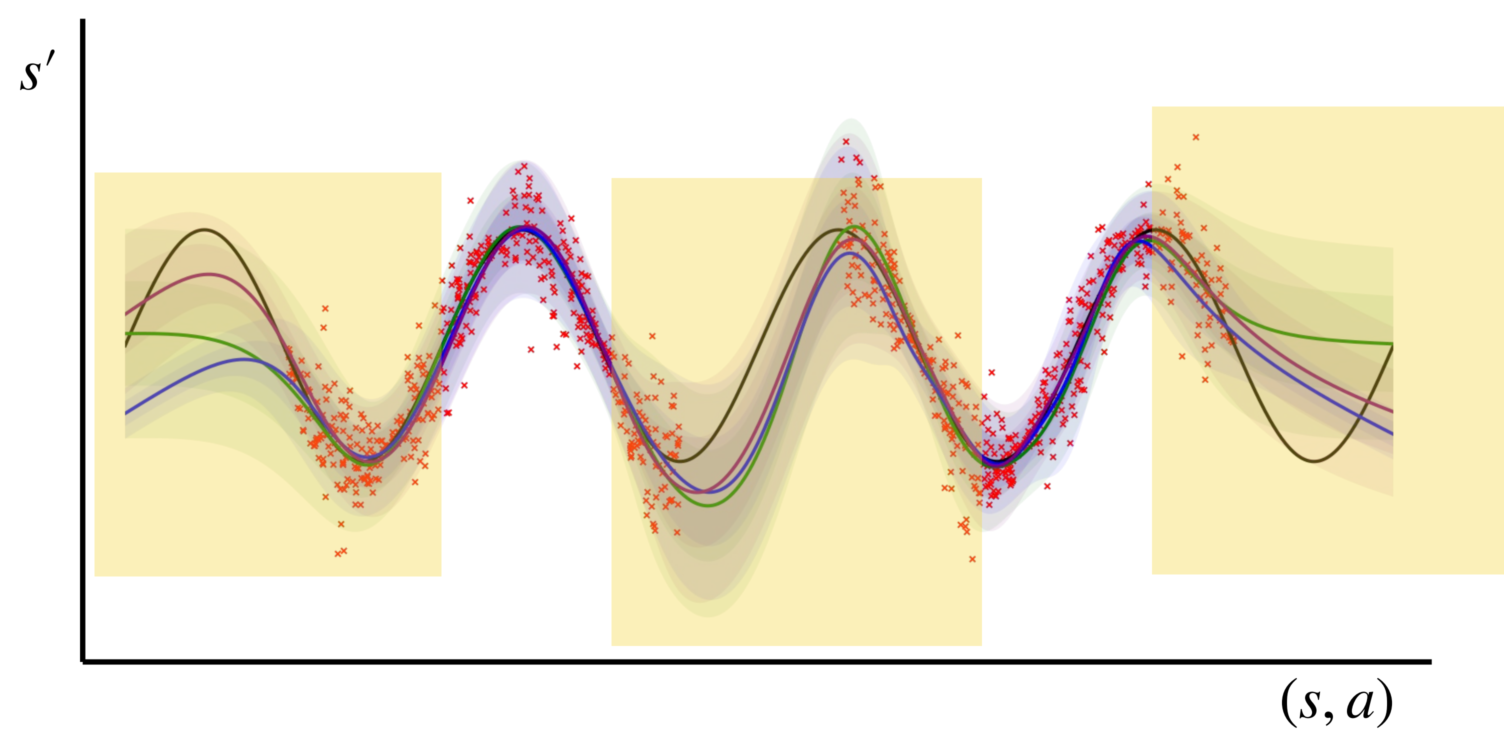

$P(\phi \,|\, D)$

- 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

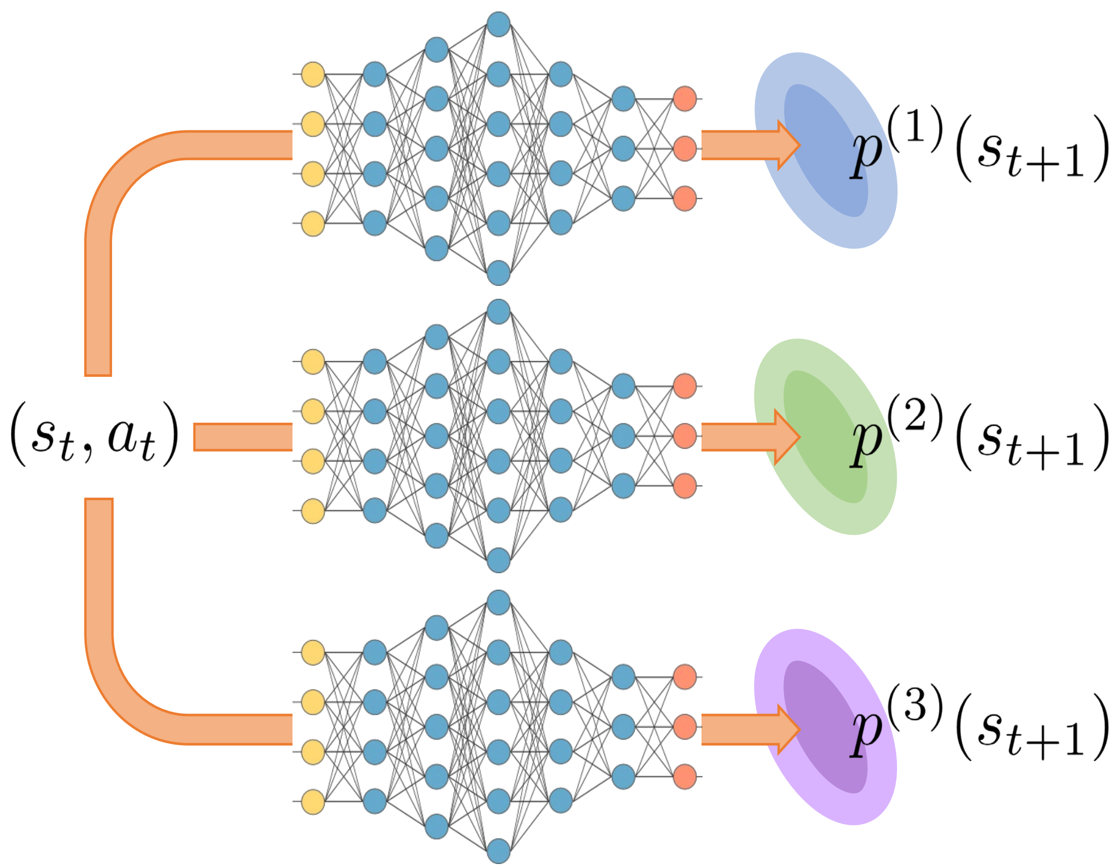
- Neural network Ensembles are a good approximation to Bayesian Nets.
- Instead of having explicit posteriors distributions for each neural net parameter, you just have a small set of neural nets, each trained on separate data.
  - On the data they have seen, they all agree (low entropy of predictions)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

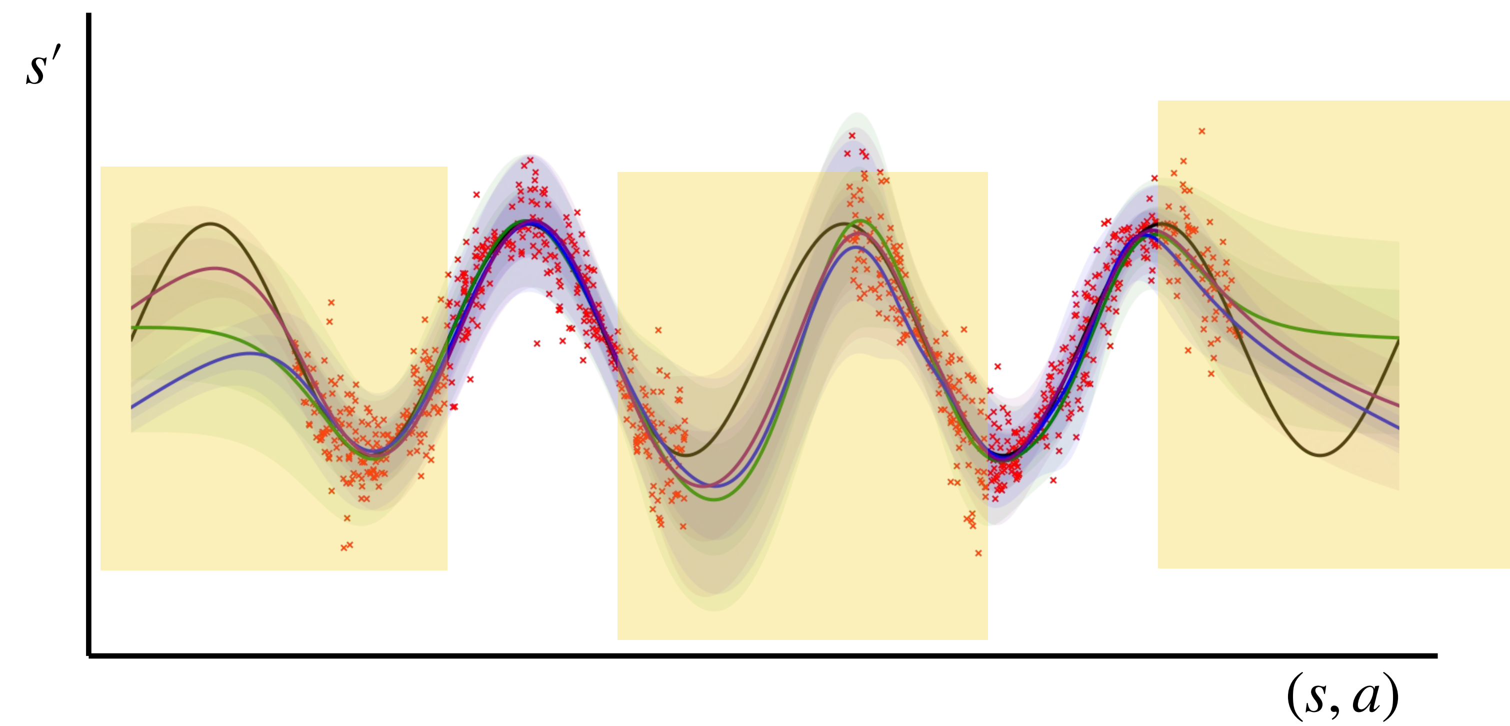


# NN Ensembles for representing Epistemic uncertainty

- 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



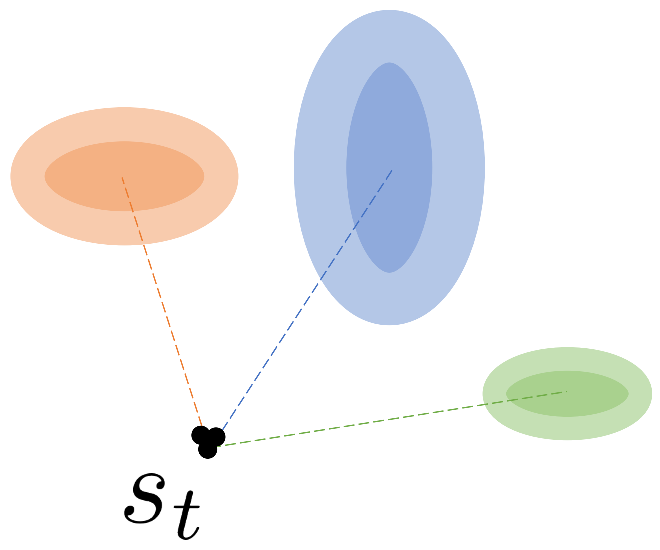
# Model Unrolling

$s_t$

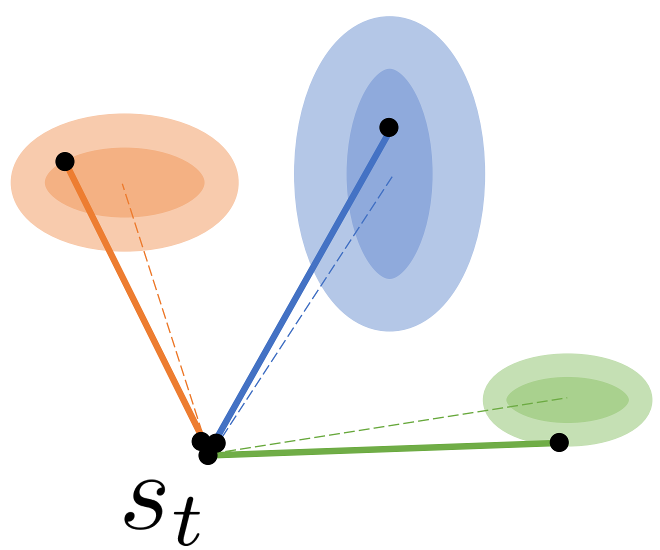
# Model Unrolling

$s_t$

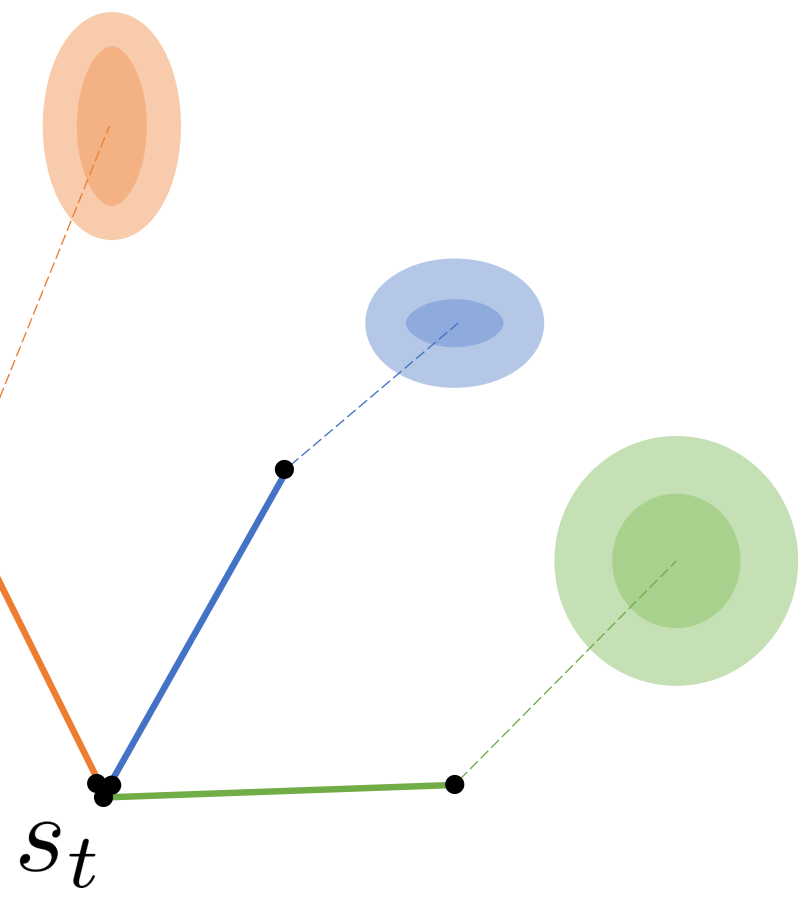
# Model Unrolling



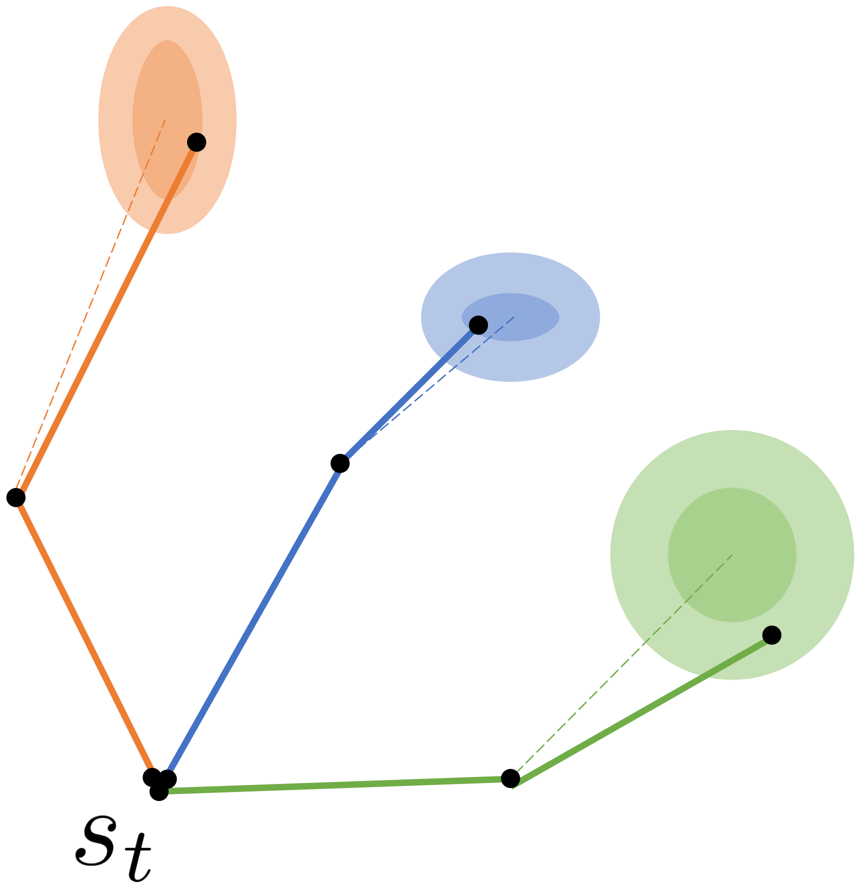
# Model Unrolling



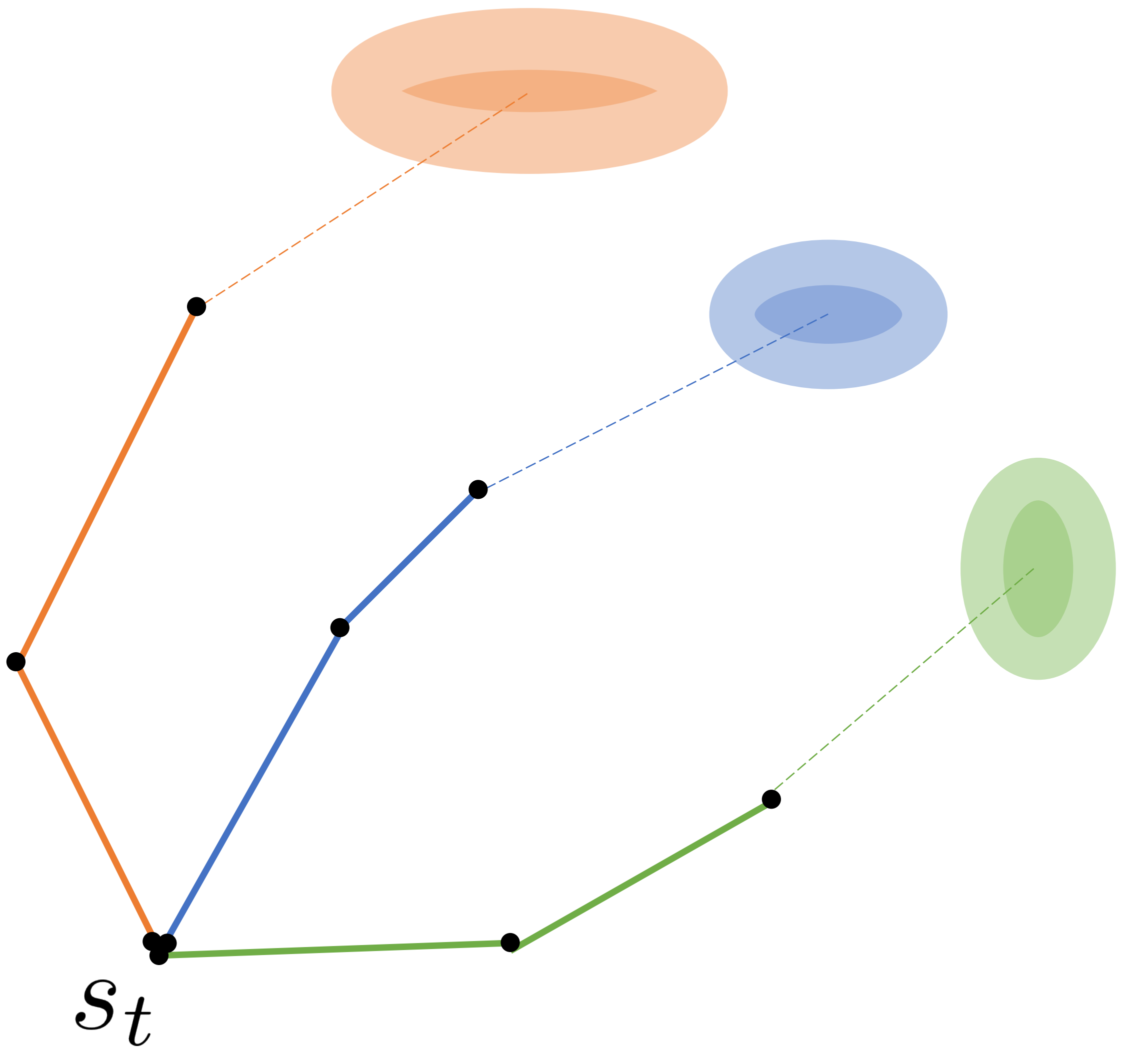
# Model Unrolling



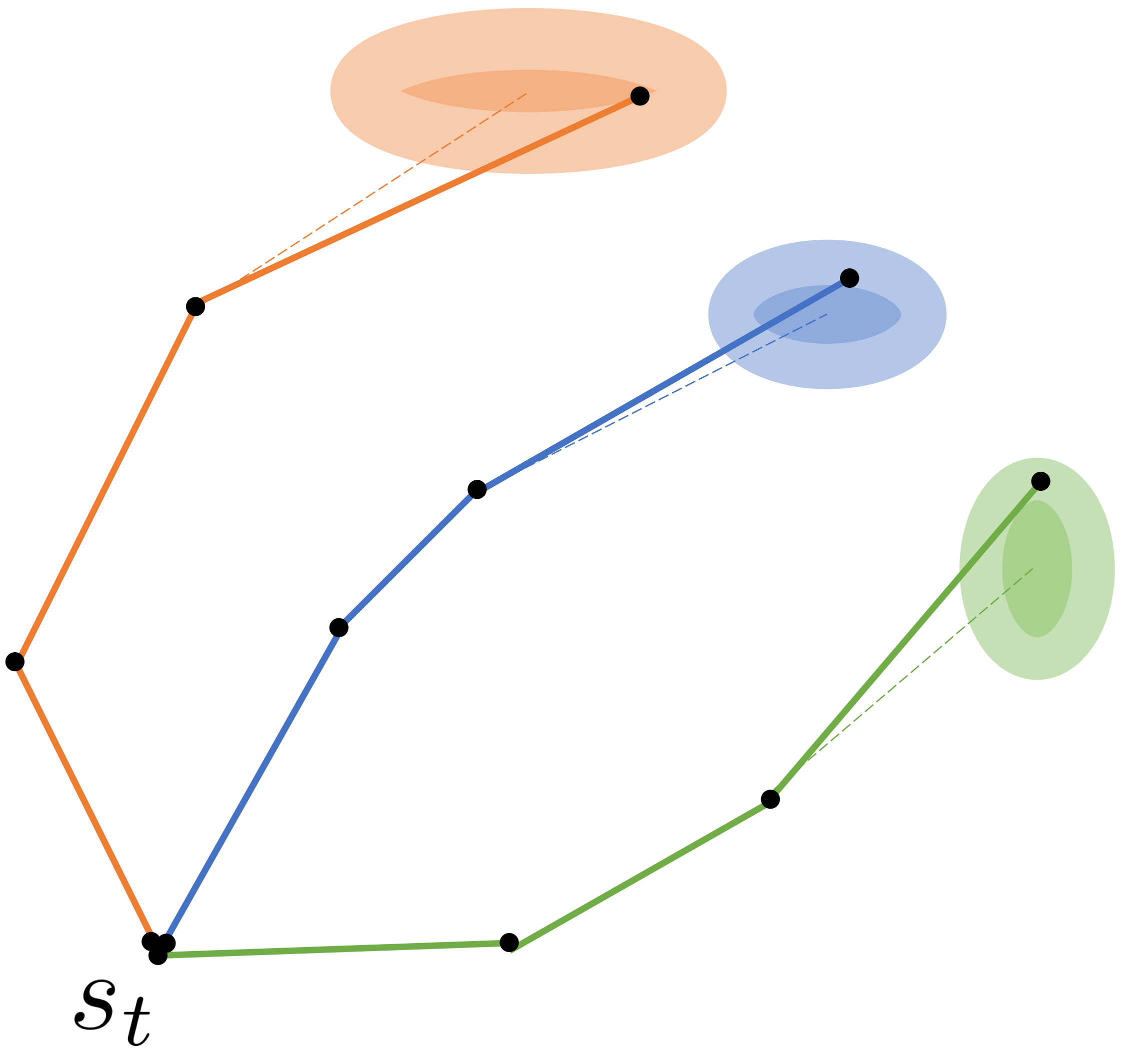
# Model Unrolling



# Model Unrolling

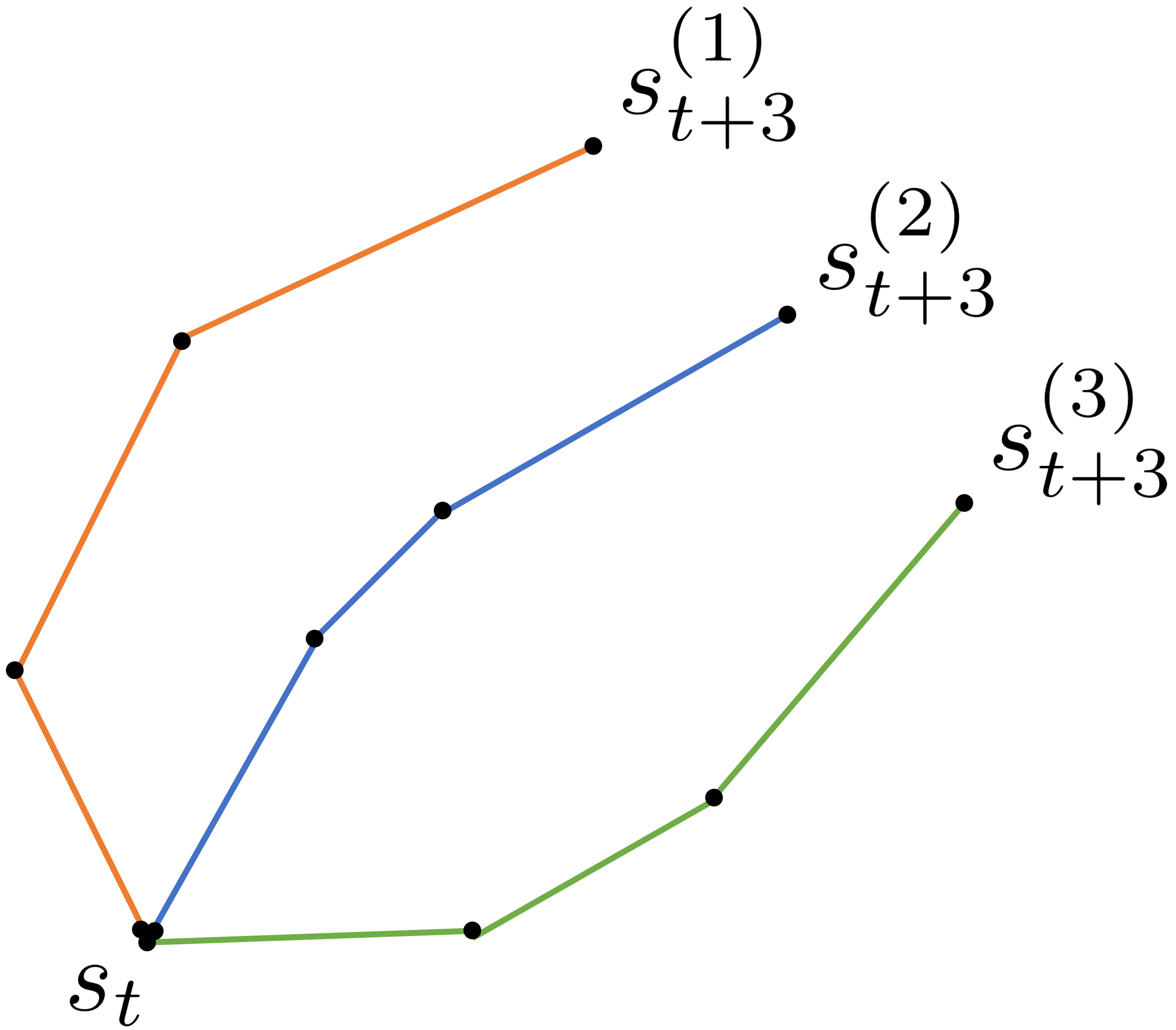


# Model Unrolling



# 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 Trajectory 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 $a^*_{t\ldots T}$ 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

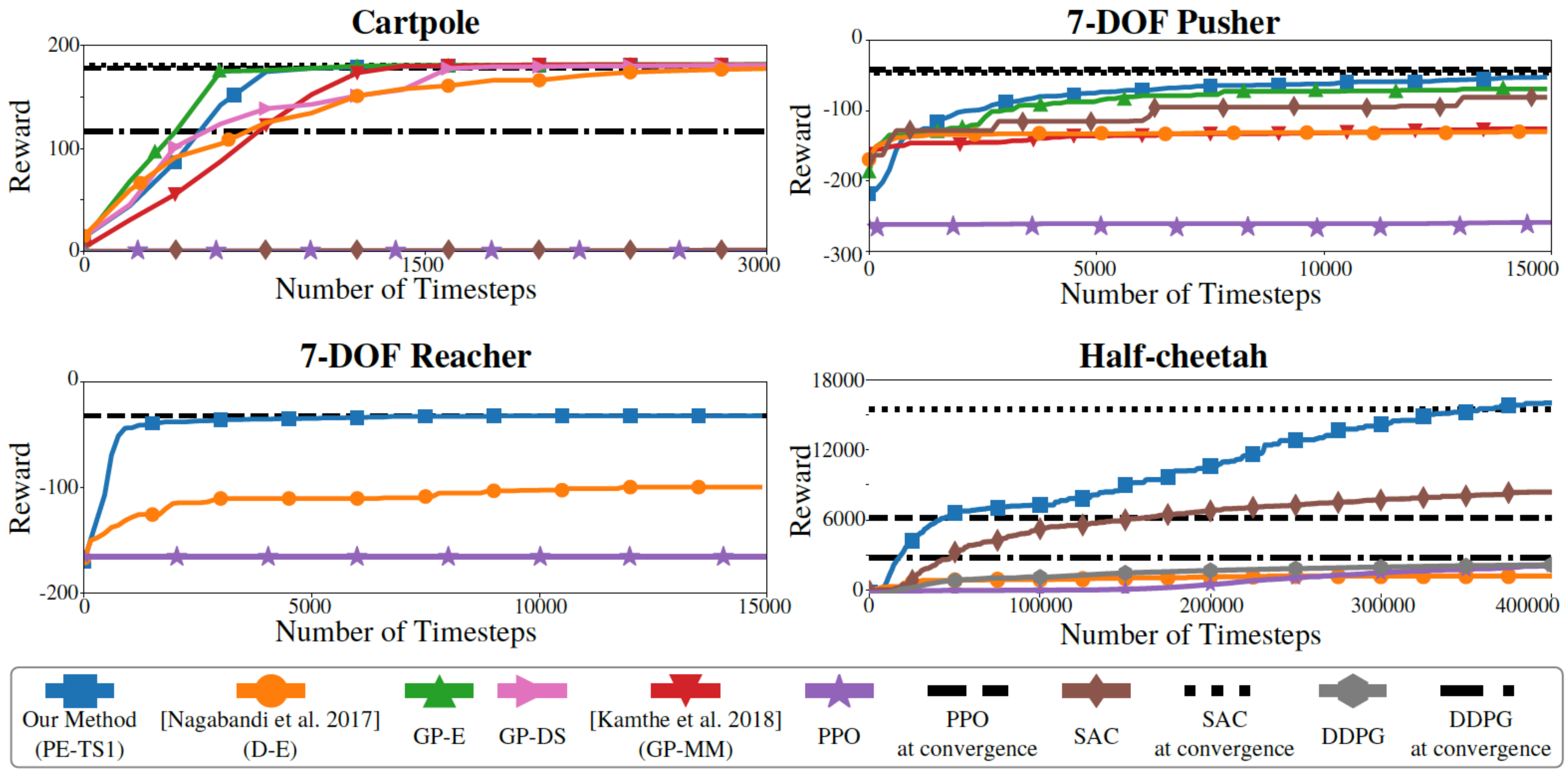



Figure 3: Learning curves for different tasks and algorithm. For all tasks, our algorithm learns in under 100K time steps or 100 trials. With the exception of Cartpole, which is sufficiently low-dimensional to efficiently learn a GP model, our proposed algorithm significantly outperform all other baselines.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

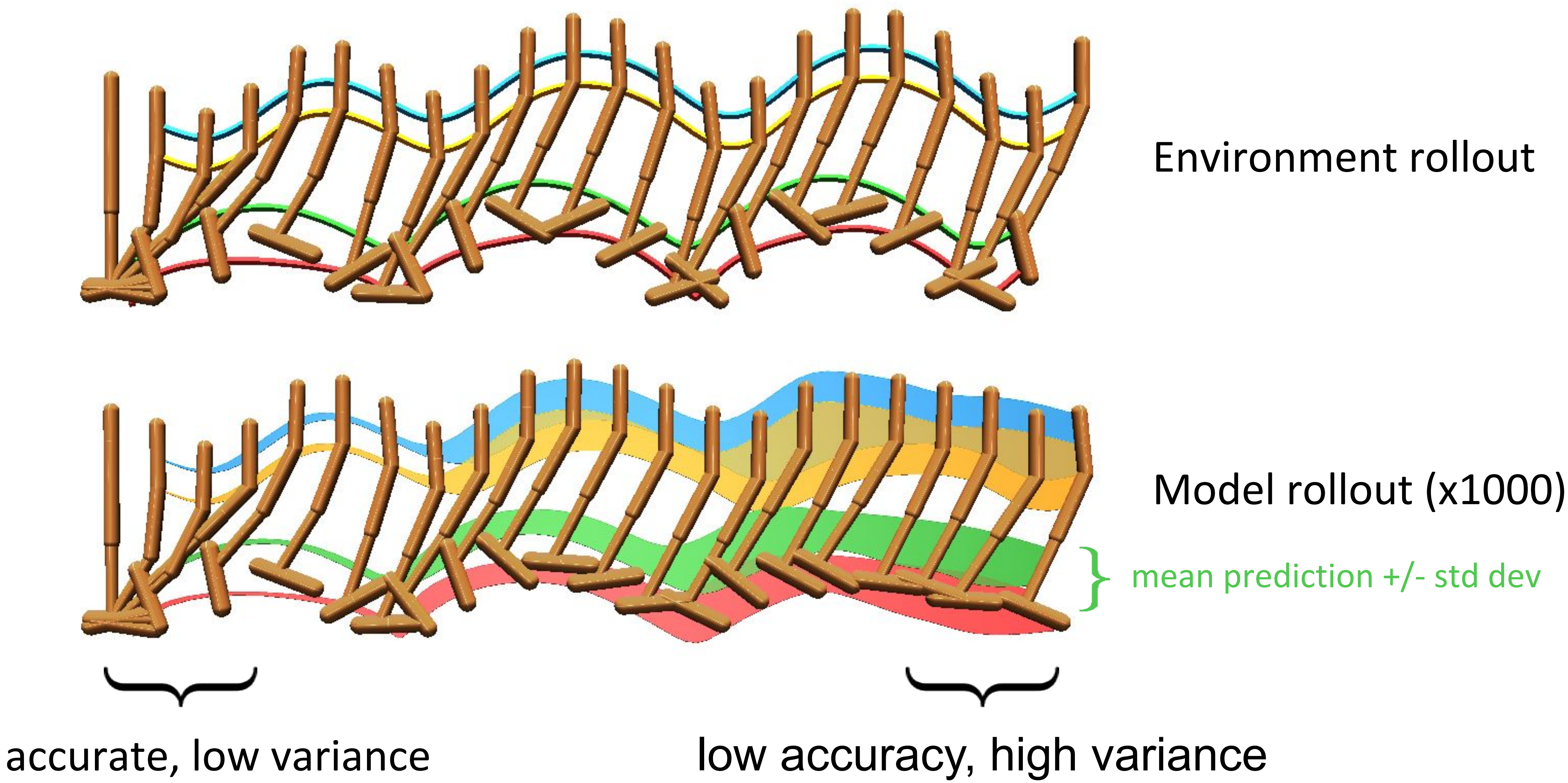
# Model-ensemble trust region policy optimization

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 $D_{env}$
   $(s', r') = f^i(s, a; \theta), i = 1..N$
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

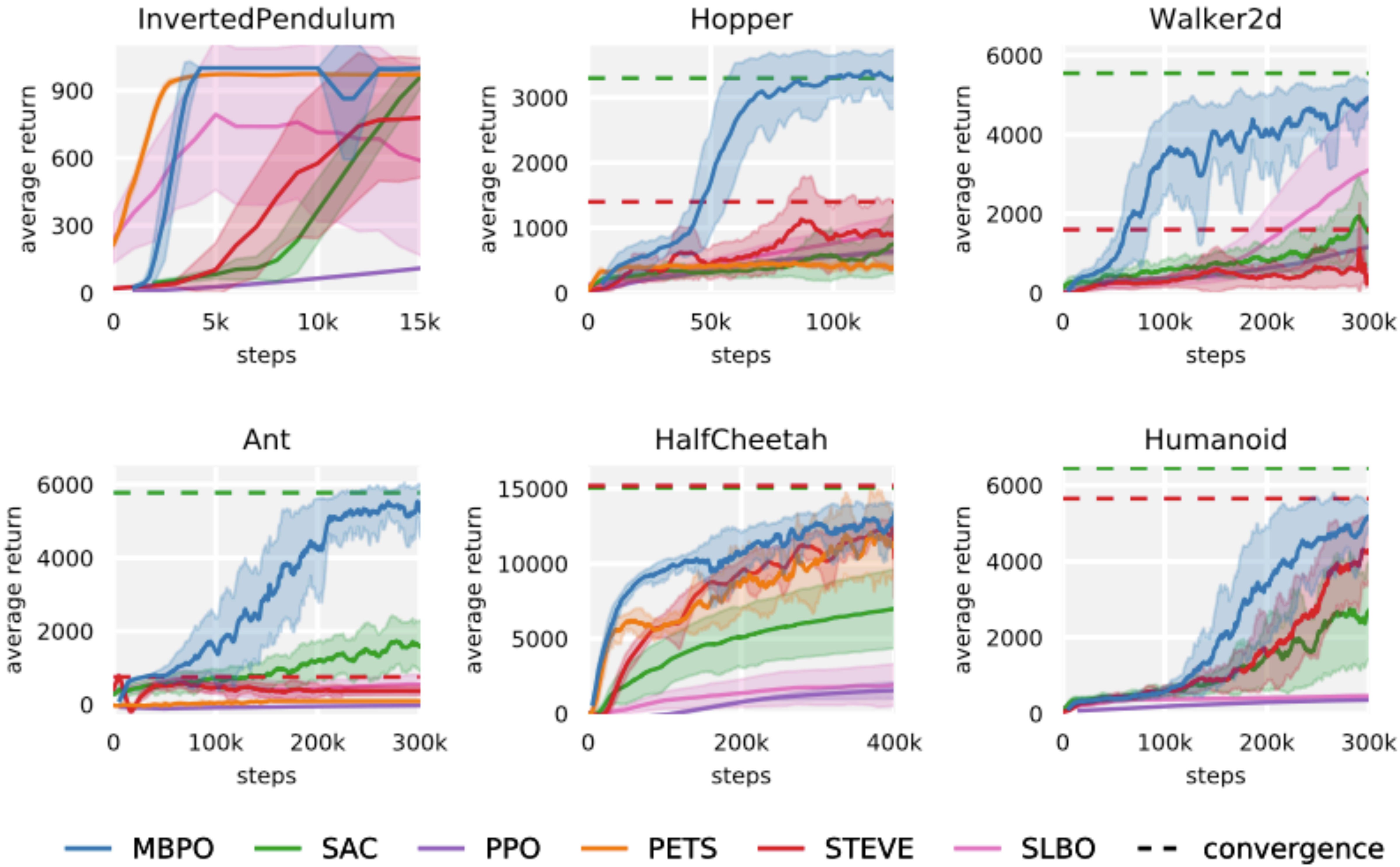


# When to Trust Your Model: Model-Based Policy Optimization

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 $D_{env}$
   $(s', r') = f^i(s, a; \theta), i = 1..N$
3. For M model rollouts
   1. **Sample** $s_t$ from the experience buffer and then collect simulated experience $D_{model}$ by sampling from the model for k time steps using the policy $\pi_\theta$ 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
Hopper
Walker2d
Ant
HalfCheetah
Humanoid
average return
steps
0
300
600
900
5k
10k
15k
1000
2000
3000
50k
100k
2000
4000
6000
100k
200k
300k
5000
10000
15000
400k
MBPO
SAC
PPO
PETS
STEVE
SLBO
convergence

Learning models from videos as opposed to low-dim states

# MBRL from sensory inputs

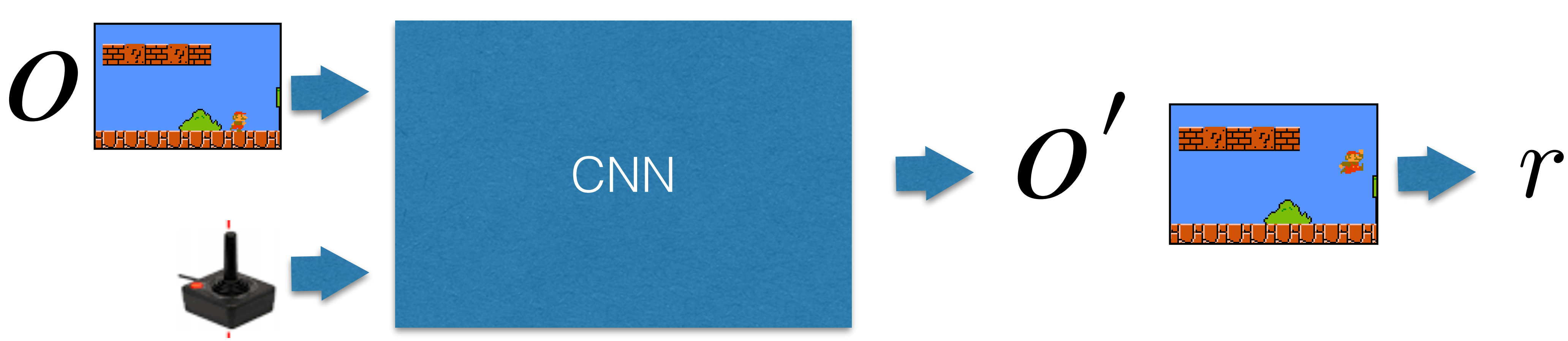
An open research problem:

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?

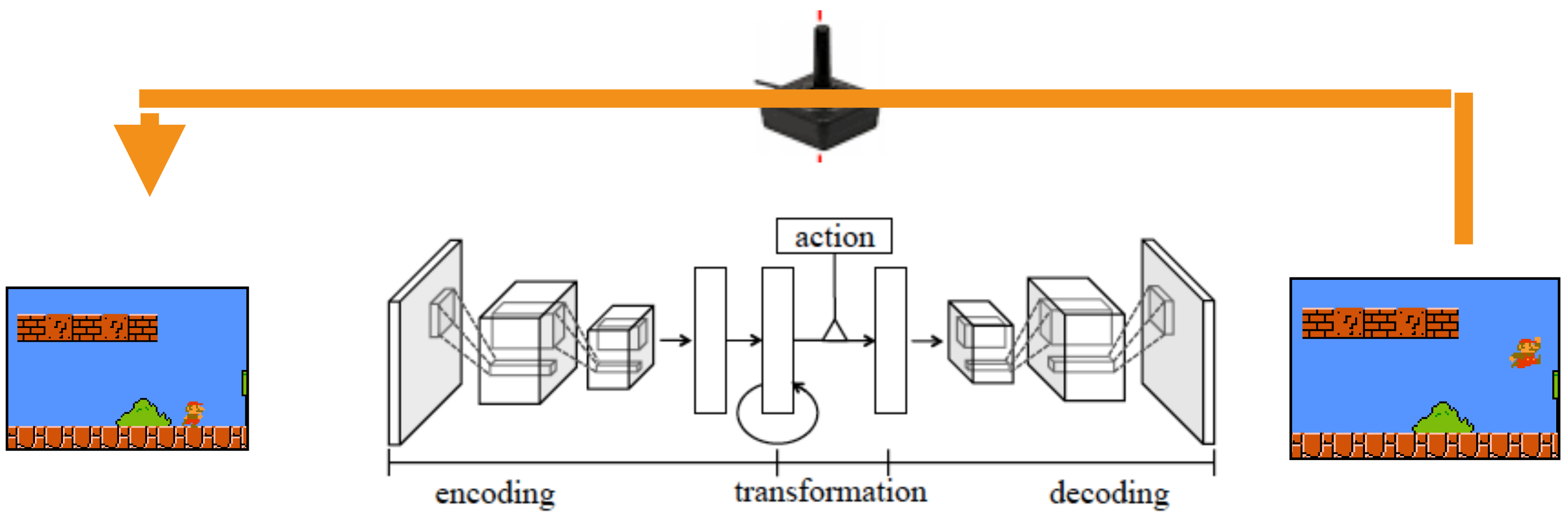
# Action-Conditional Video Prediction using Deep Networks in Atari Games

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



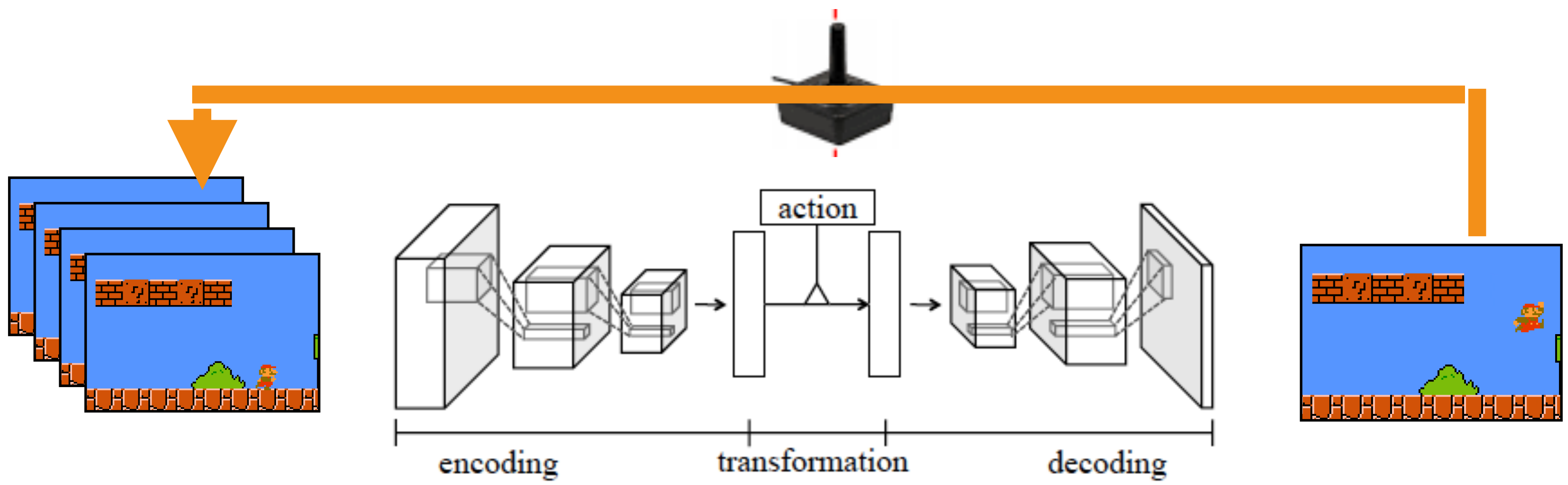
# Minimizing error accumulation during unrolling




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



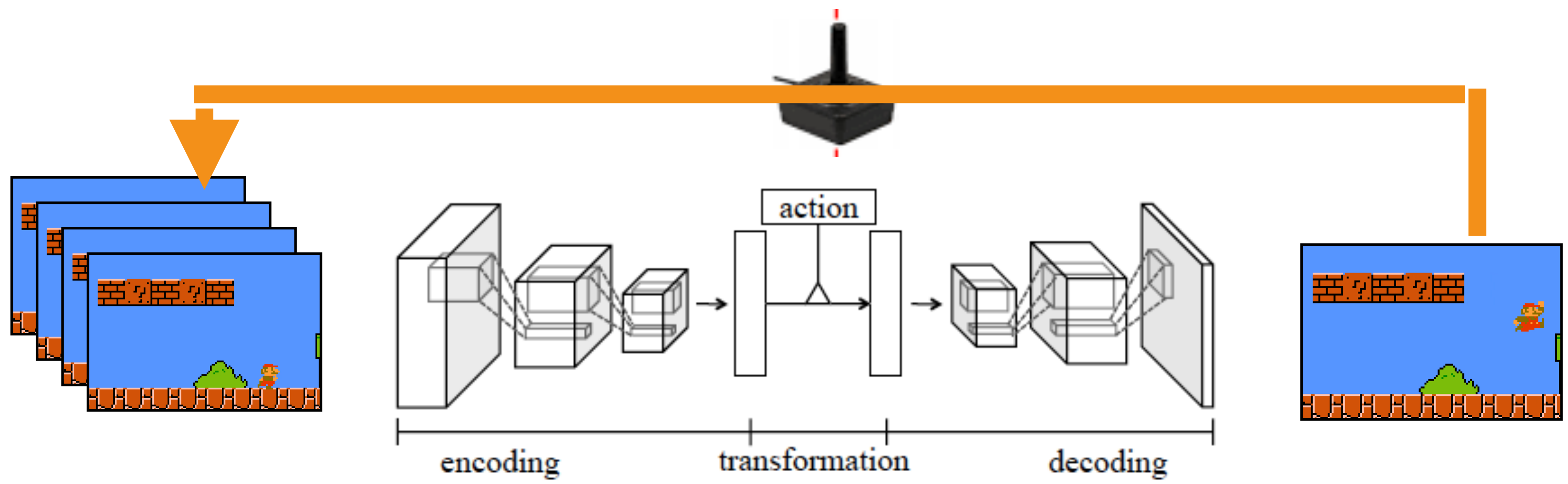

Q: Can I train my model using tuples $(o, a, o')$ and at test time unroll it over time?
A: No, we will have distribution shift, same as in imitation learning: tiny mistakes will soon cause the model to drift

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

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


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

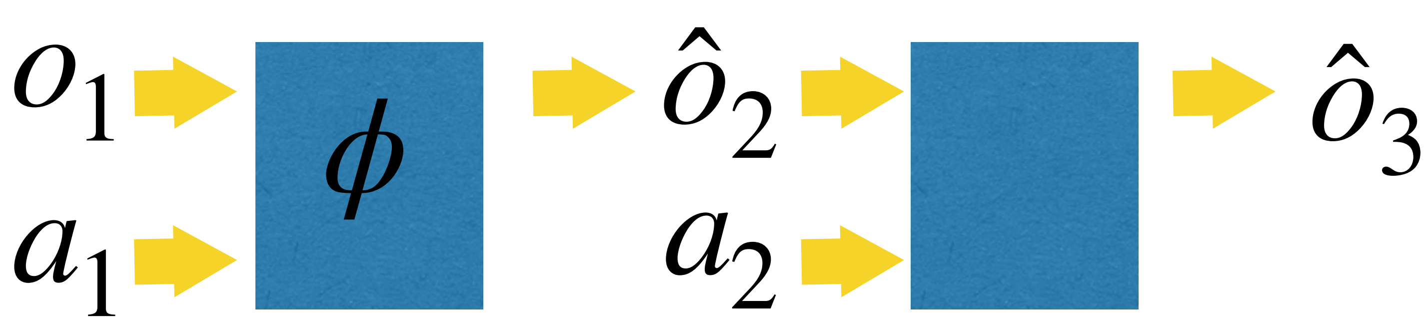
# How to train our model so that unrolling works




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

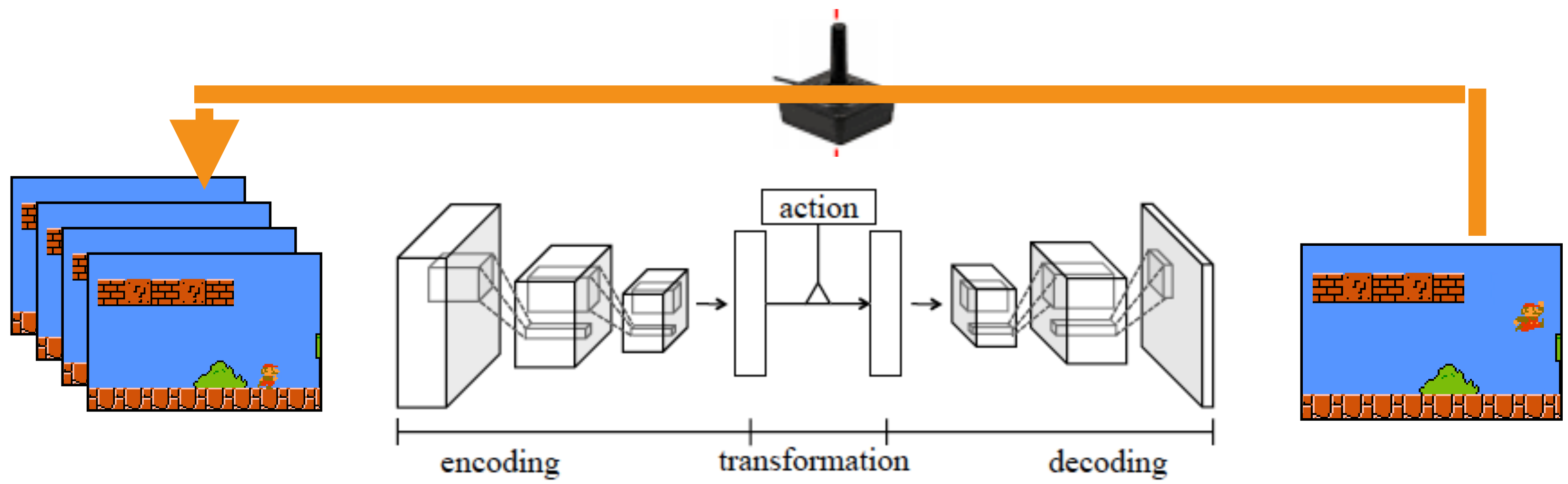
Solution: Progressively increase the unrolling horizon k at training time so that the model learns to handle its mistakes:

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




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

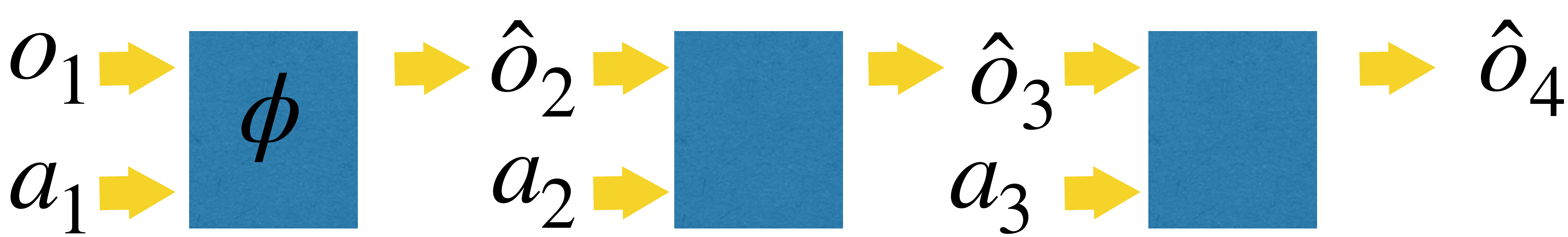
# How to train our model so that unrolling works




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?

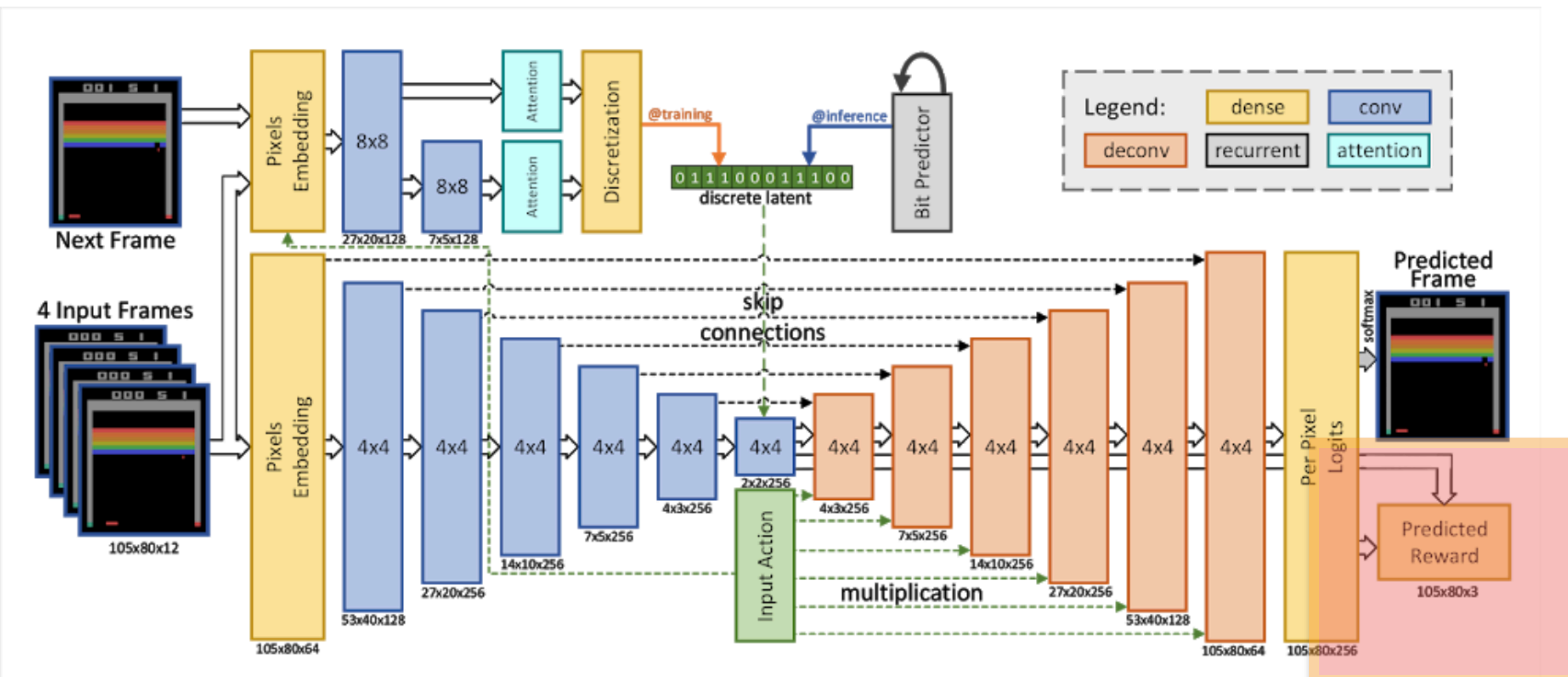
# Model-Based Reinforcement Learning for Atari

**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

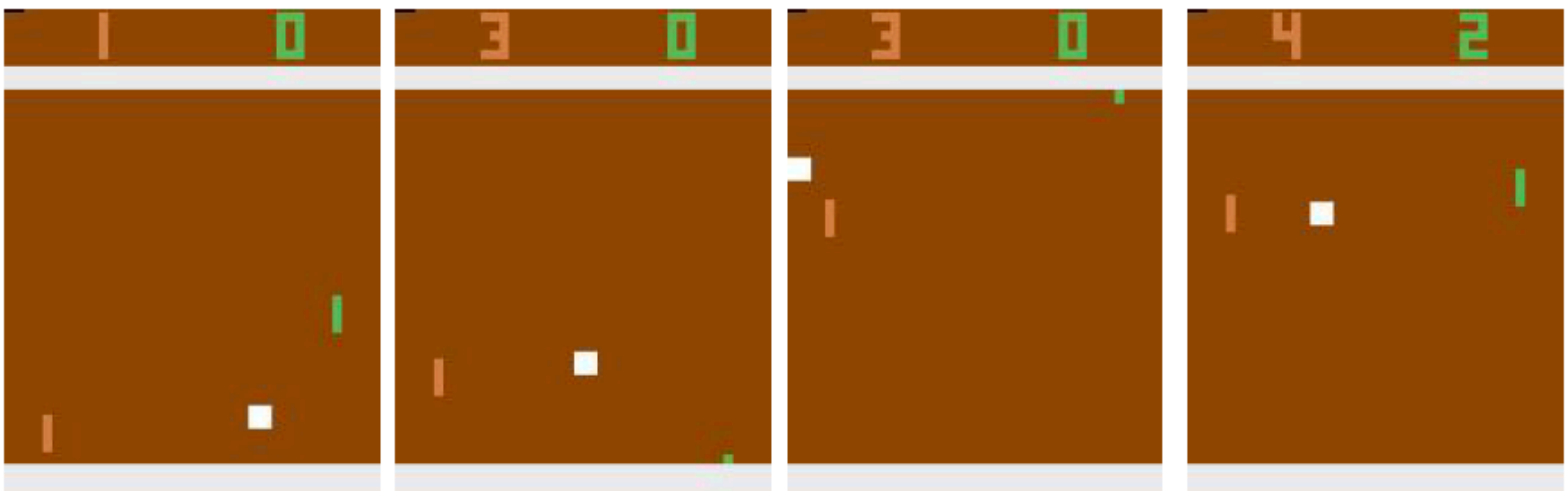
Initialize policy $\pi(s; \theta)$ and $D_{env} = \{\}$.

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 $\pi_\theta$ 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
- $D_{model} >> D_{env}$
- 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

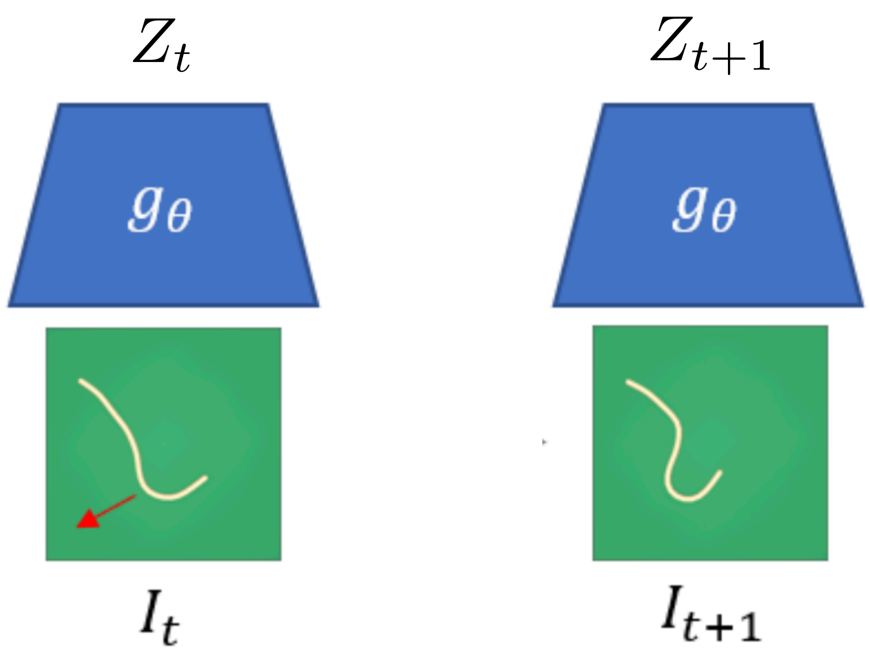
# Results

- Number of frames required to reach human performance

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

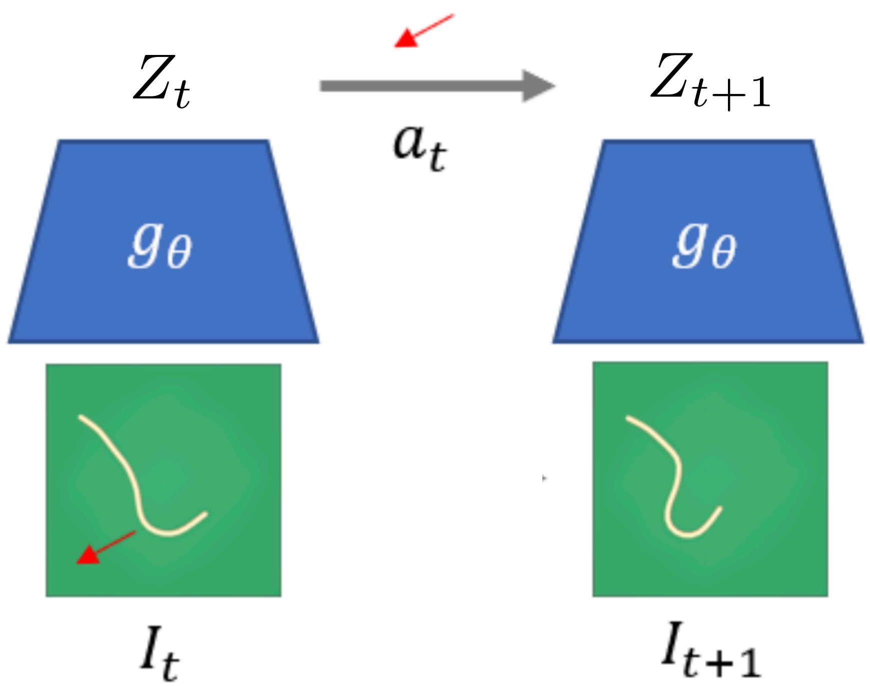
results

# Contrastive forward models for deformable object manipulation




*Learning Predictive Representations for Deformable Objects using Contrastive Estimation*

# Contrastive forward models for deformable object manipulation




*Learning Predictive Representations for Deformable Objects using Contrastive Estimation*

# Contrastive forward models for deformable object manipulation

Contrastive loss:

$$\mathcal{L} = -\mathbb{E}_{\mathcal{D}}\left[\log \frac{h(\hat{z}, z_{\text{pos}})}{\sum_{i=1}^{k} h(\hat{z}, z_{\text{neg}})}\right]$$

*Learning Predictive Representations for Deformable Objects using Contrastive Estimation*

# Contrastive forward models for deformable object manipulation

Contrastive loss:

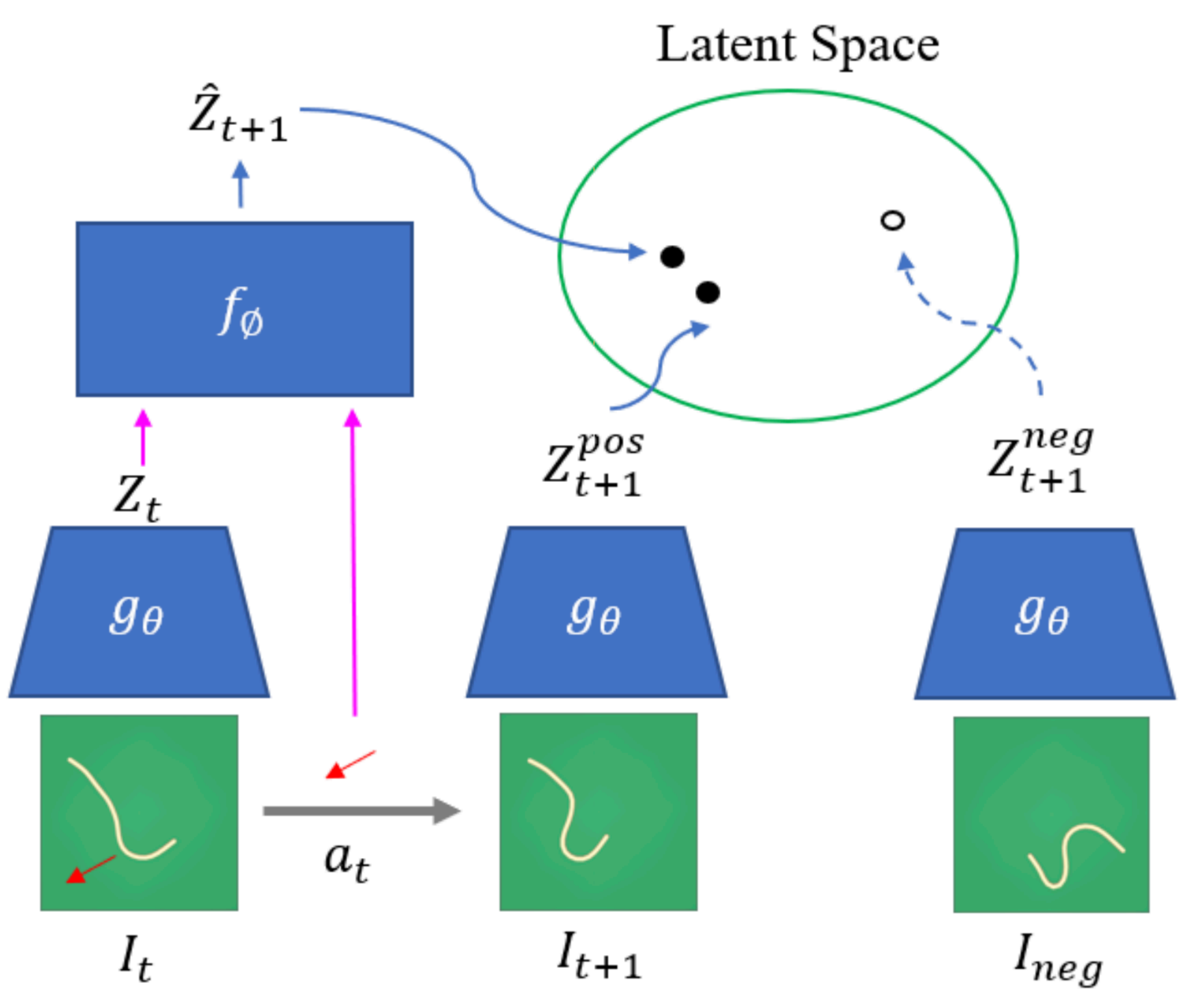
$$\mathcal{L} = -\mathbb{E}_{\mathcal{D}}\left[\log \frac{h(\hat{z}, z_{\text{pos}})}{\sum_{i=1}^{k} h(\hat{z}, z_{\text{neg}})}\right]$$

Similarity function:

$$h(z_1, z_2) = \exp(z_1^T z_2)$$

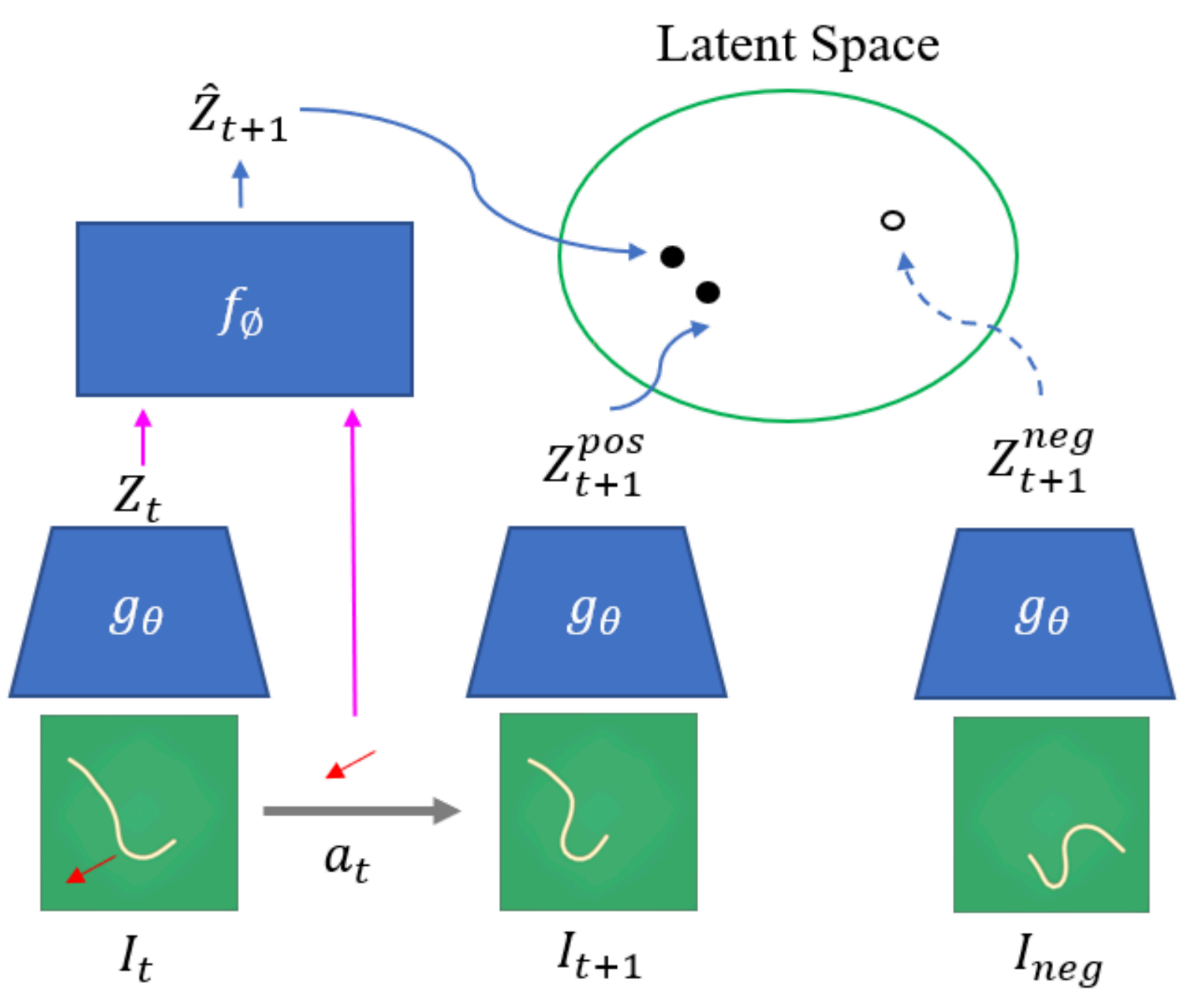
*Learning Predictive Representations for Deformable Objects using Contrastive Estimation*

# Contrastive forward models for deformable object manipulation




*Learning Predictive Representations for Deformable Objects using Contrastive Estimation*

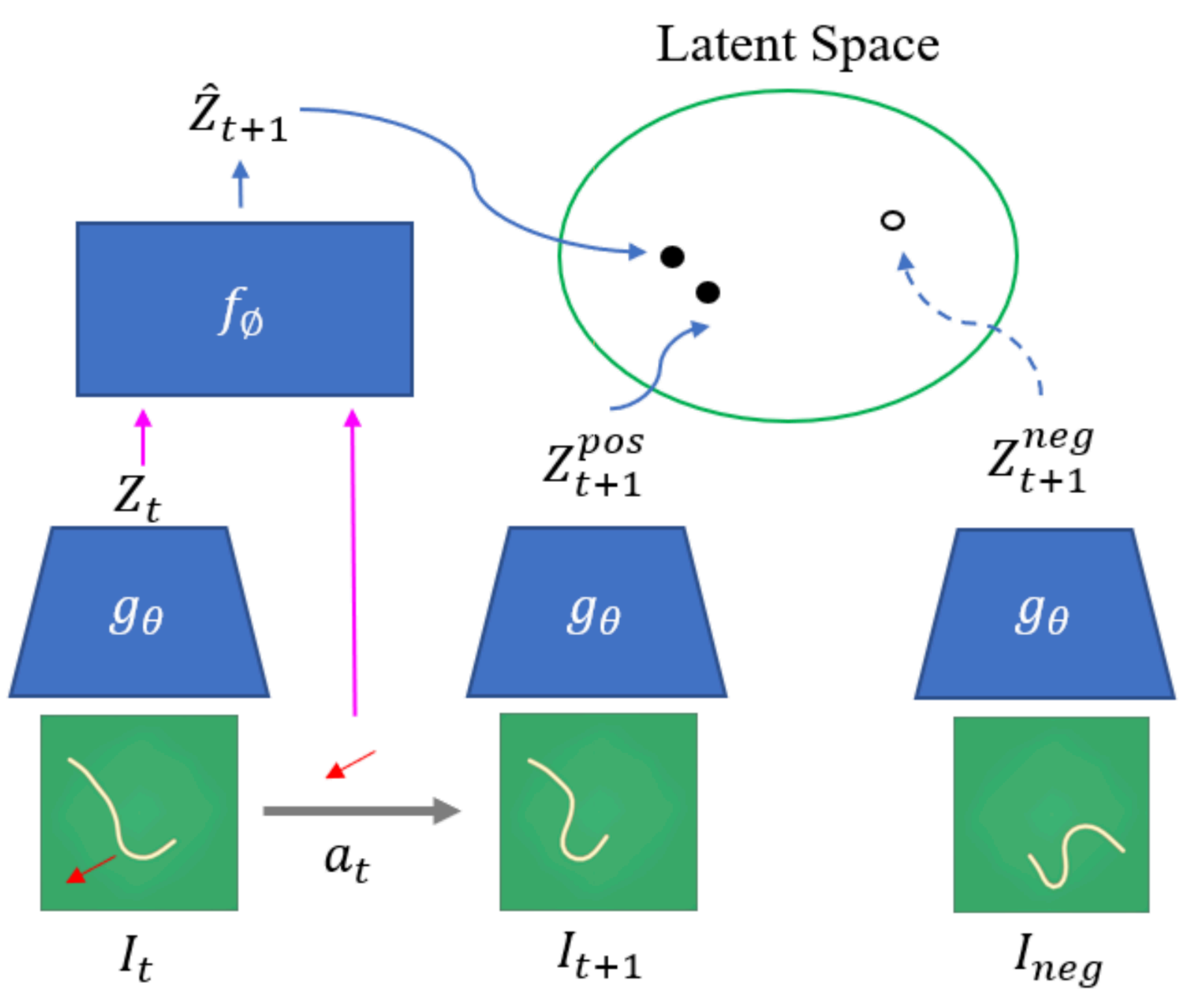
# Contrastive forward models for deformable object manipulation




$$\mathcal{L} = -\mathbb{E}_{\mathcal{D}}\left[\log \frac{h(\hat{z}_{t+1}, z_{t+1})}{\sum_{i=1}^{k} h(\hat{z}_{t+1}, \tilde{z}_i)}\right]$$

*Learning Predictive Representations for Deformable Objects using Contrastive Estimation*

# Contrastive forward models for deformable object manipulation

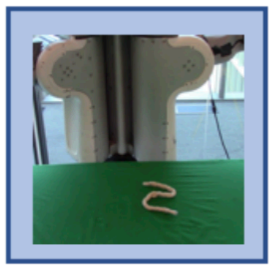



$$\mathcal{L} = -\mathbb{E}_{\mathcal{D}}\left[\log \frac{h(\hat{z}_{t+1}, z_{t+1})}{\sum_{i=1}^{k} h(\hat{z}_{t+1}, \tilde{z}_i)}\right]$$

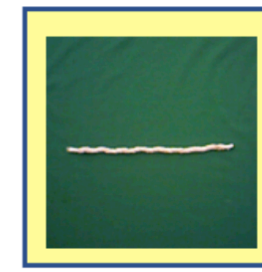
$$h(z_1, z_2) = \exp(-\|z_1 - z_2\|^2)$$

# One step Model-Predictive Control

Start

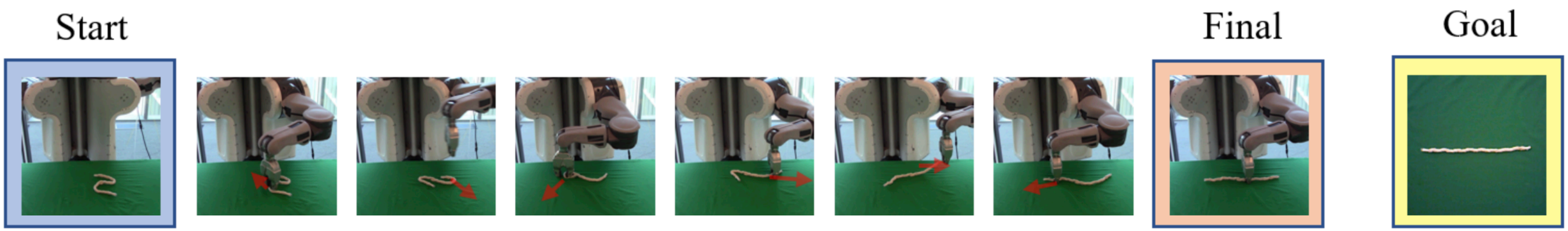


Goal



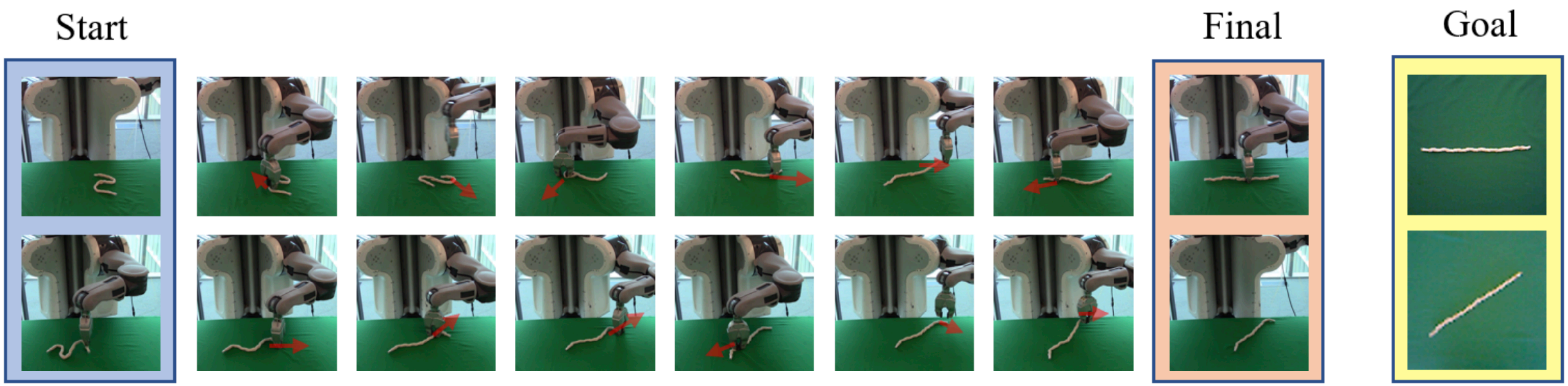
$$a_t = \max h(f_\phi(z_t, a), z_g)$$

# One step Model-Predictive Control




$$a_t = \max\ h(f_\phi(z_t, a), z_g)$$

# One step Model-Predictive Control



# One step Model-Predictive Control

# Qualitative evaluation