Carnegie Mellon
School of Computer Science

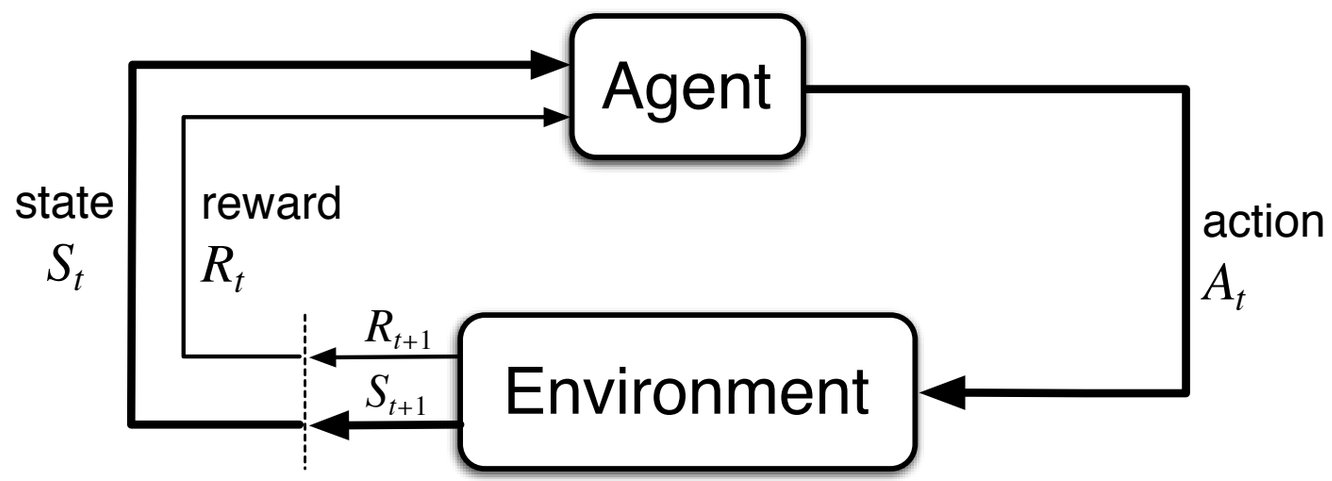
Deep Reinforcement Learning and Control

Model Based Reinforcement Learning

Katerina Fragkiadaki

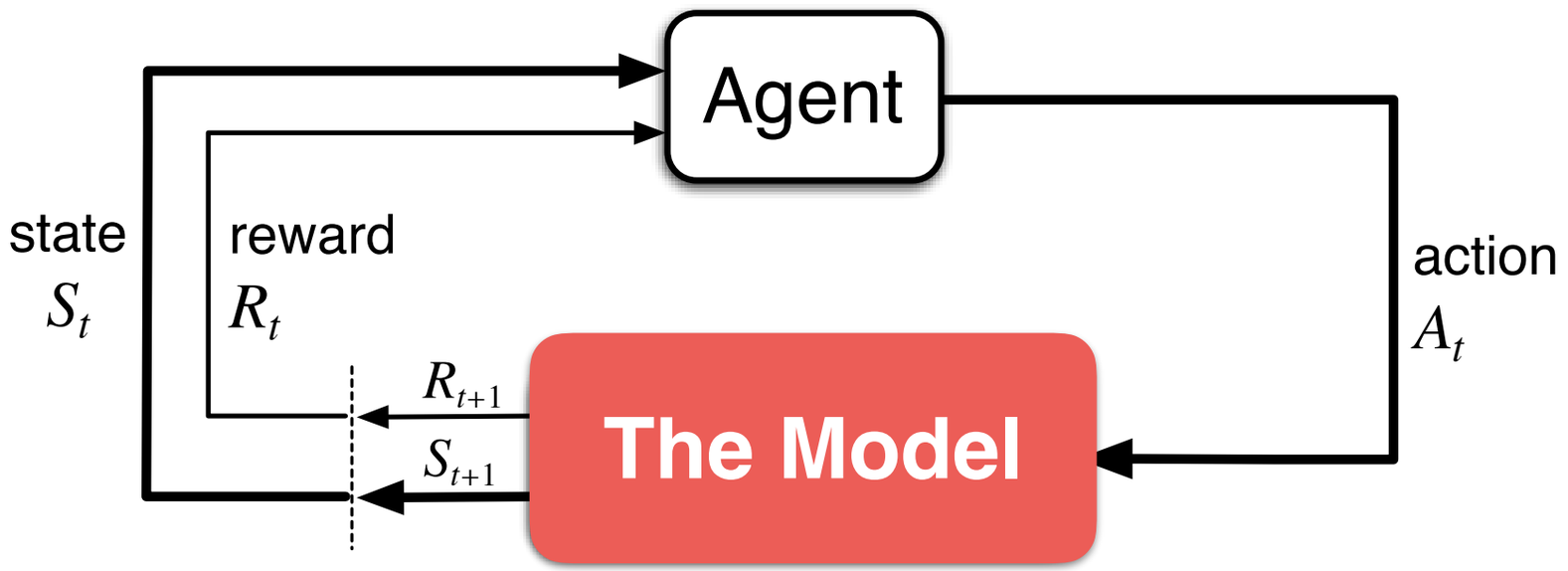
Planning

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



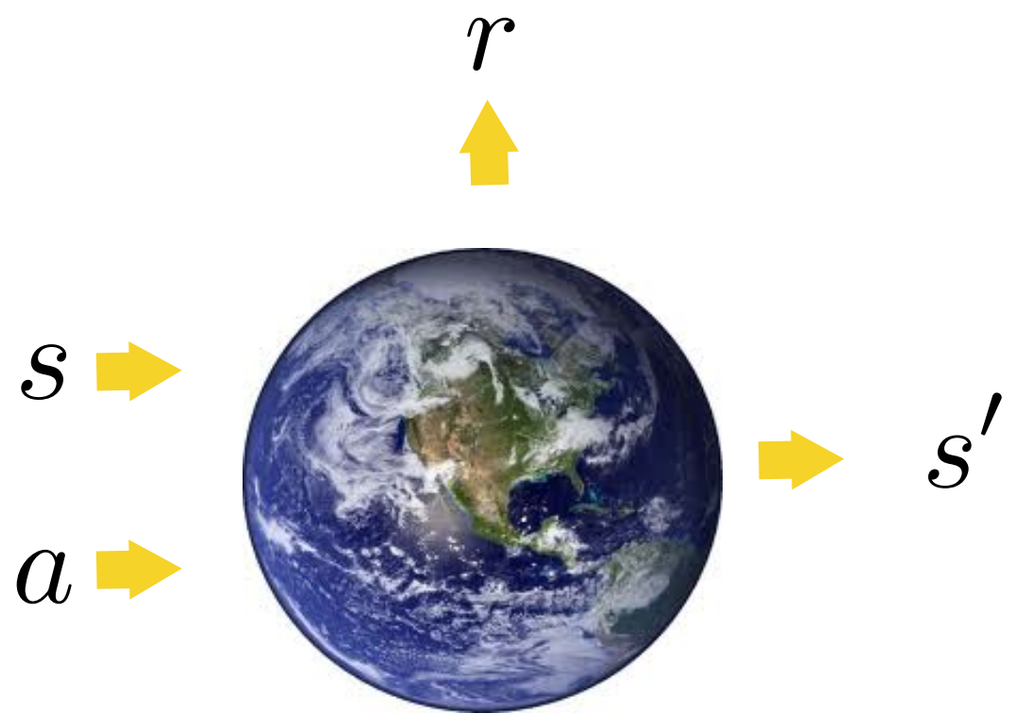
Planning

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



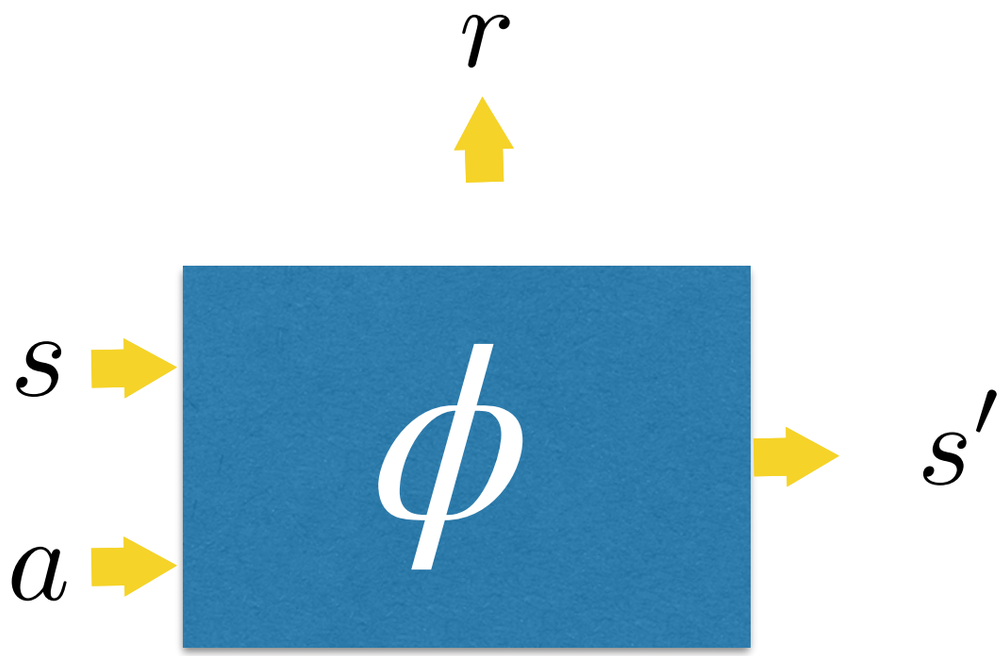
Model

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



Model learning

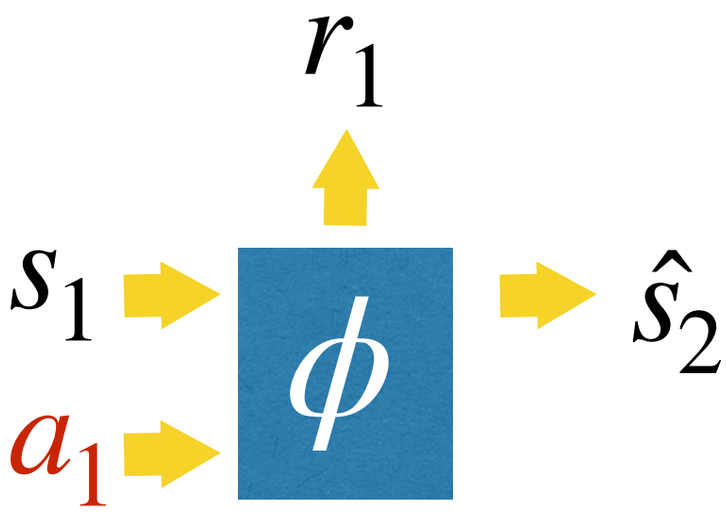
We will be learning the model using experience tuples. A supervised learning problem.



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

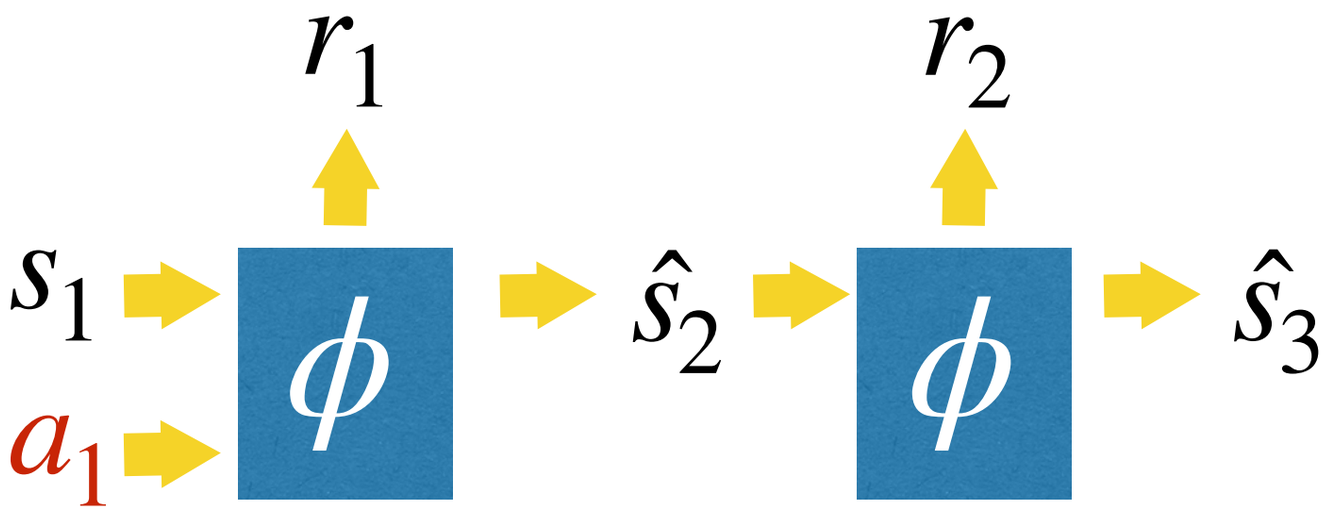
Model unrolling

Predict long term effects of actions



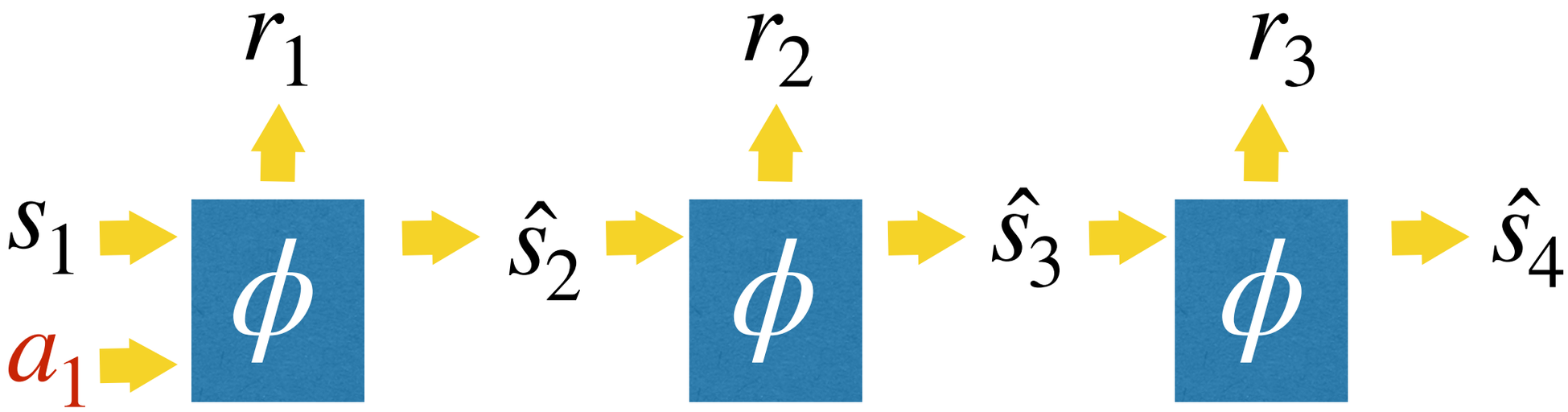
Model unrolling

Predict long term effects of actions



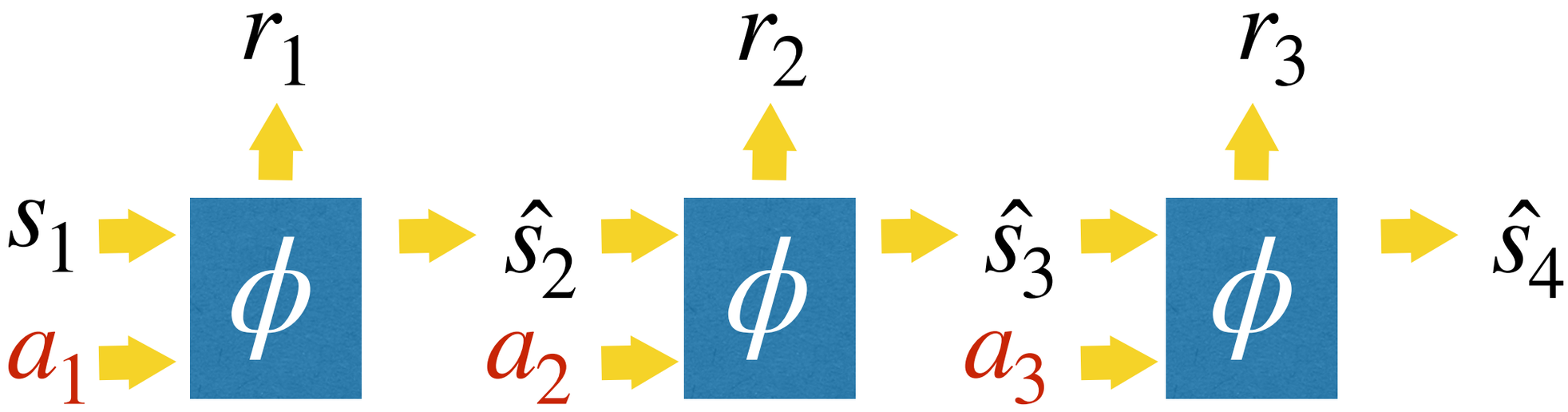
Model unrolling

Predict long term effects of actions



Model unrolling

Predict long term effects of sequences of actions



Model learning

Newtonian Physics equations

System identification: we assume the dynamics equations given and only have few unknown parameters

Much easier to learn but suffers from under-modeling

VERSUS

general parametric form (no prior from Physics knowledge)

Neural networks: lots of unknown parameters, generic structure

Very flexible, very hard to get it to generalize

Why model learning

- Model-based control: given an initial state s_0 estimate **action sequence** to reach a desired goal or maximize reward by unrolling the model forward in time
- Model-based RL: train **policies** using:
    1. a model-free RL method using simulated experience (experience sampled from the model)
    2. an imitation learning method by imitating the MPC planner
- Efficient Exploration guided by model uncertainty (exploration lecture)

Why model learning

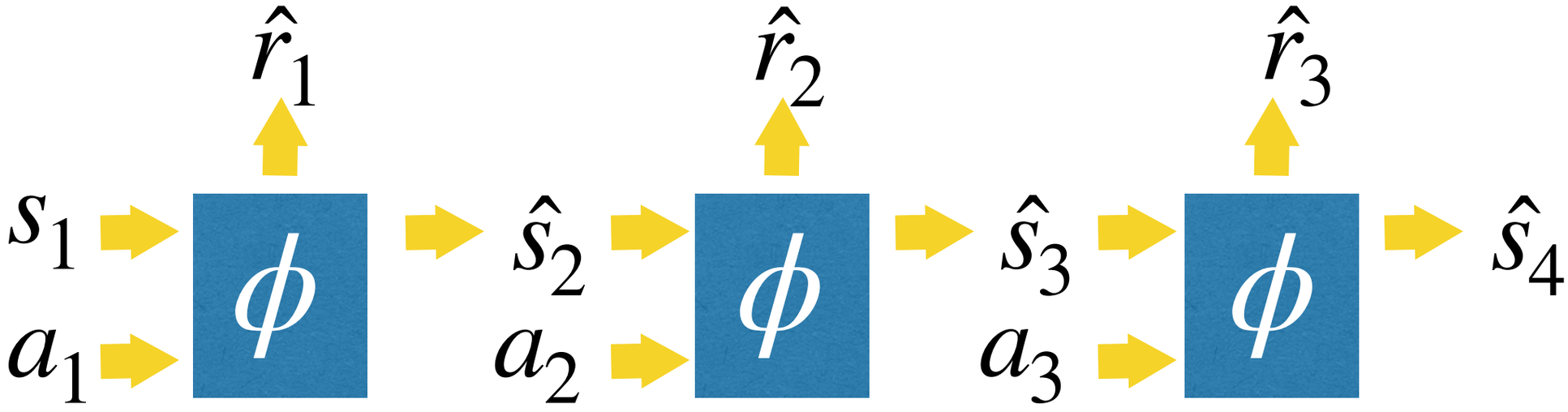
- Model-based control: given an initial state s_0 estimate **action sequence** to reach a desired goal or maximize reward by unrolling the model forward in time
- Model-based RL: train **policies** using:
    1. a model-free RL method using simulated experience (experience sampled from the model)
    2. an imitation learning method by imitating the MB planner
- Efficient Exploration guided by model uncertainty (later lecture)

Model-based control

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

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

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



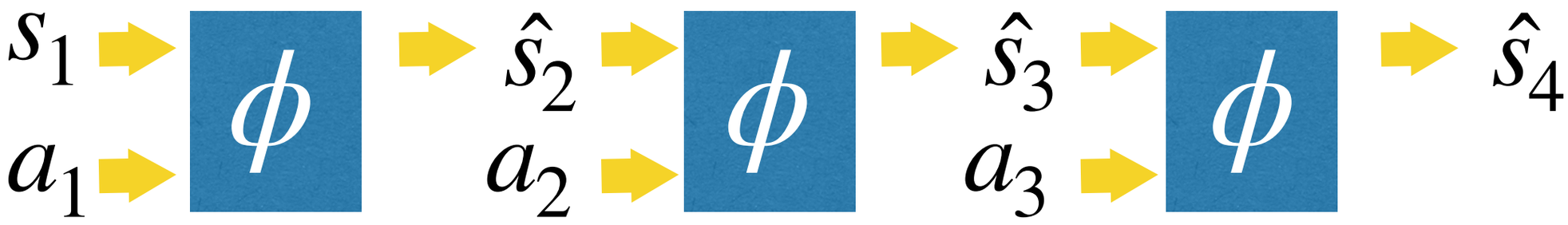
Model-based control - SGD

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

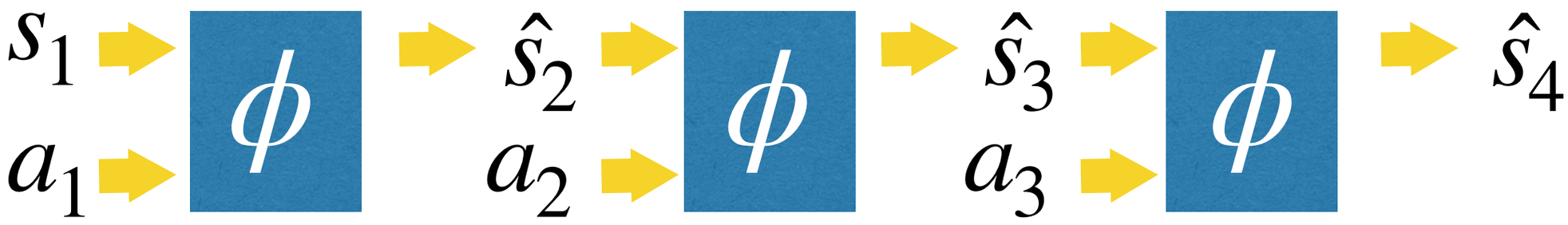
1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and compute error against a desired final state s_*
4. Backpropagate the error to the action sequence

$$\|\hat{s}_4 - s_*\|$$

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 s_*
4. Backpropagate the error to the action sequence (differentiable)

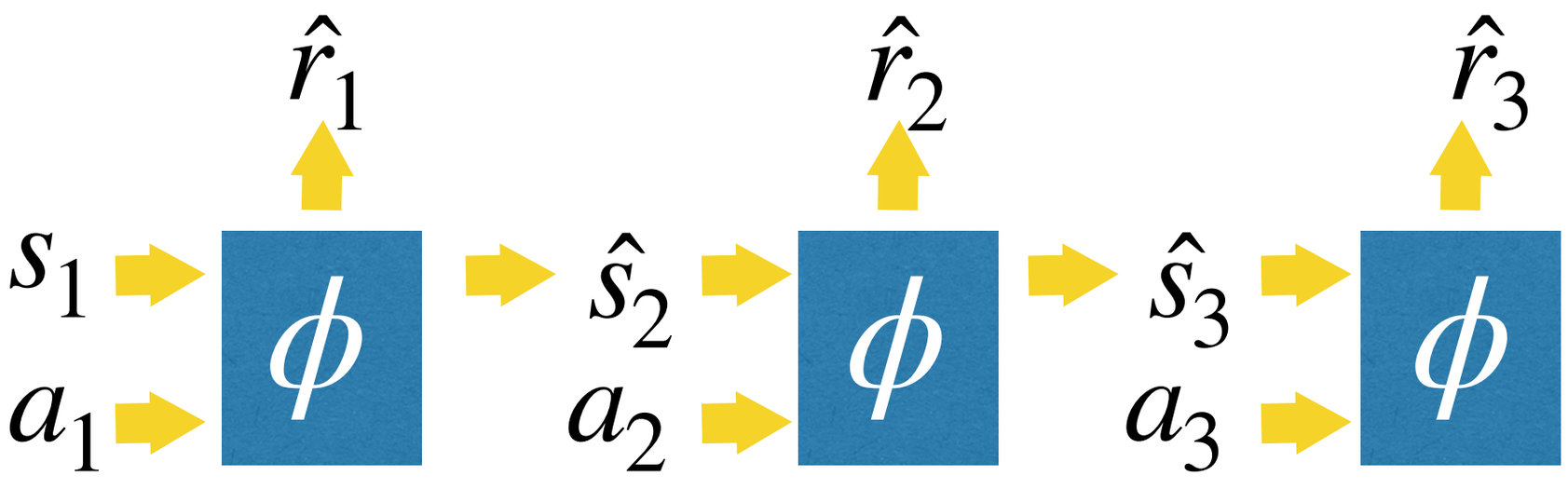
$$\min_{a_1 \cdots a_T} . \ \|s_T - s_*\|$$



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 gradient to the action sequence

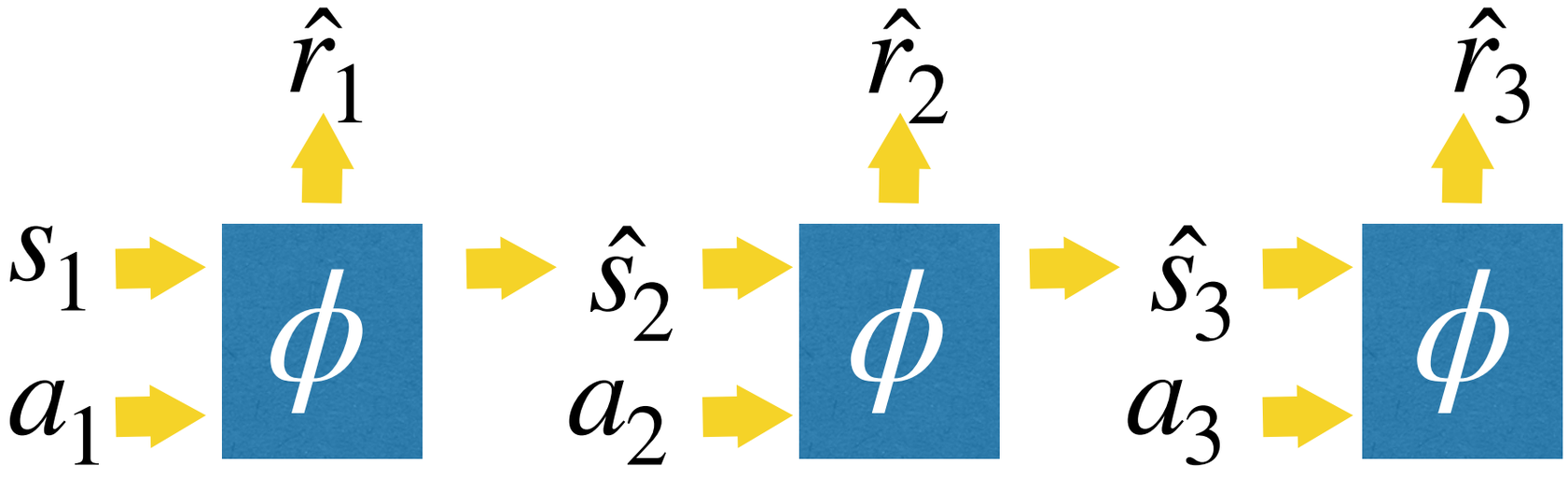
$$\max_{a_1 \cdots a_T} . \sum_{t=1}^{T} \hat{r}_t$$



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

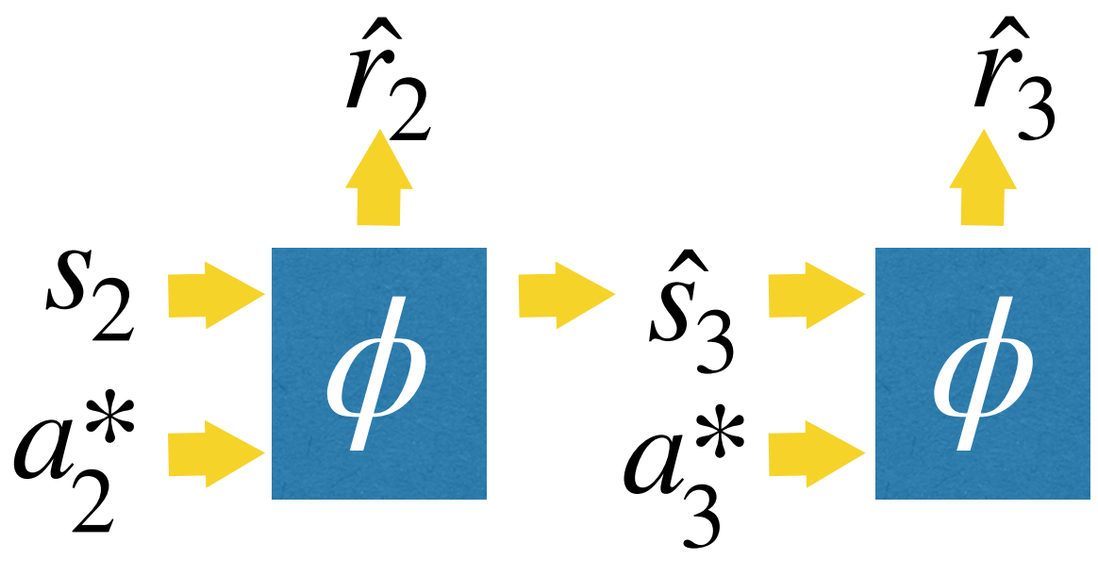
$$\max_{a_1 \cdots a_T} . \quad J(a_1 \cdots a_T | \phi) = \sum_{t=1}^{T} \hat{r}_t$$



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**

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

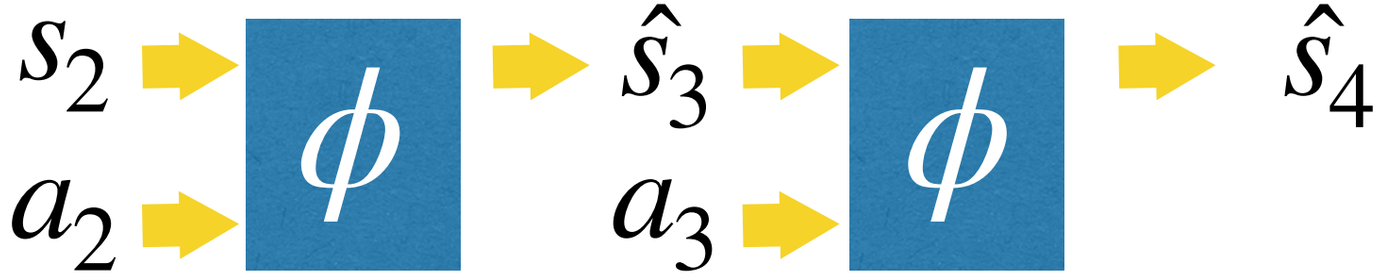


Q: Are the predicted rewards and future states the same as in the previous slide?

Model-predictive control

1. Given an initial action sequence
2. Unroll the model forward in time
3. Compare and compute error against a desired final state
4. Backpropagate the error to the action sequence
5. Execute **only the first action**
6. **GOTO 1**

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

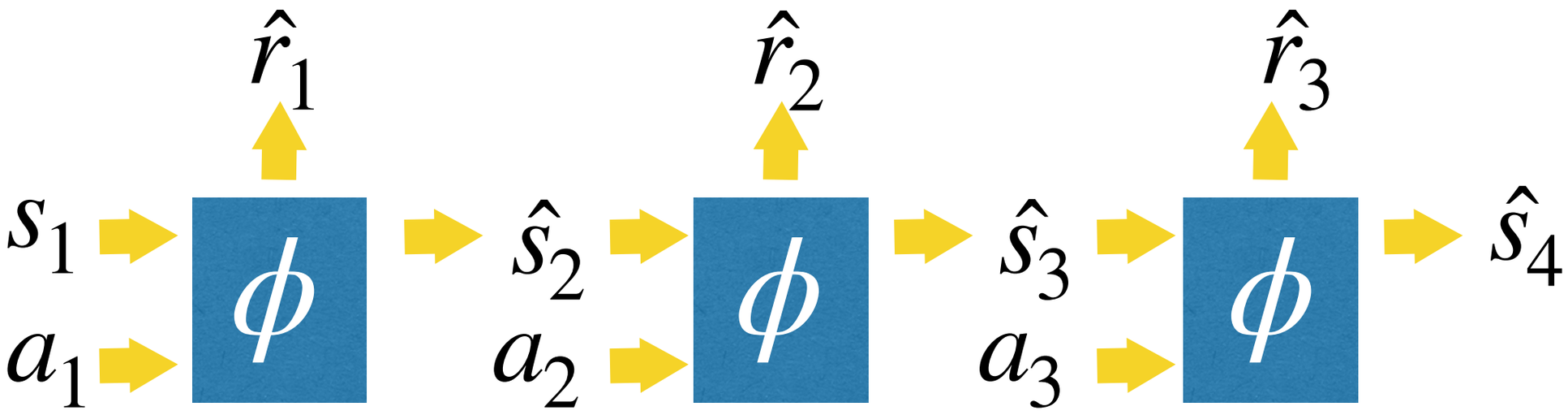


Model-based control - derivative-free

$$\min_{a_1 \cdots a_T} . \ \|\hat{s}_T - s_*\| \qquad \max_{a_1 \cdots a_T} . \ \sum_{t=1}^{T} \hat{r}_t$$

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

Optimize over action selection using CMA-ES or CEM (sample actions, unroll, compute error, survival of the fittest, repeat)



Q: why would we use evolution instead of SGD?

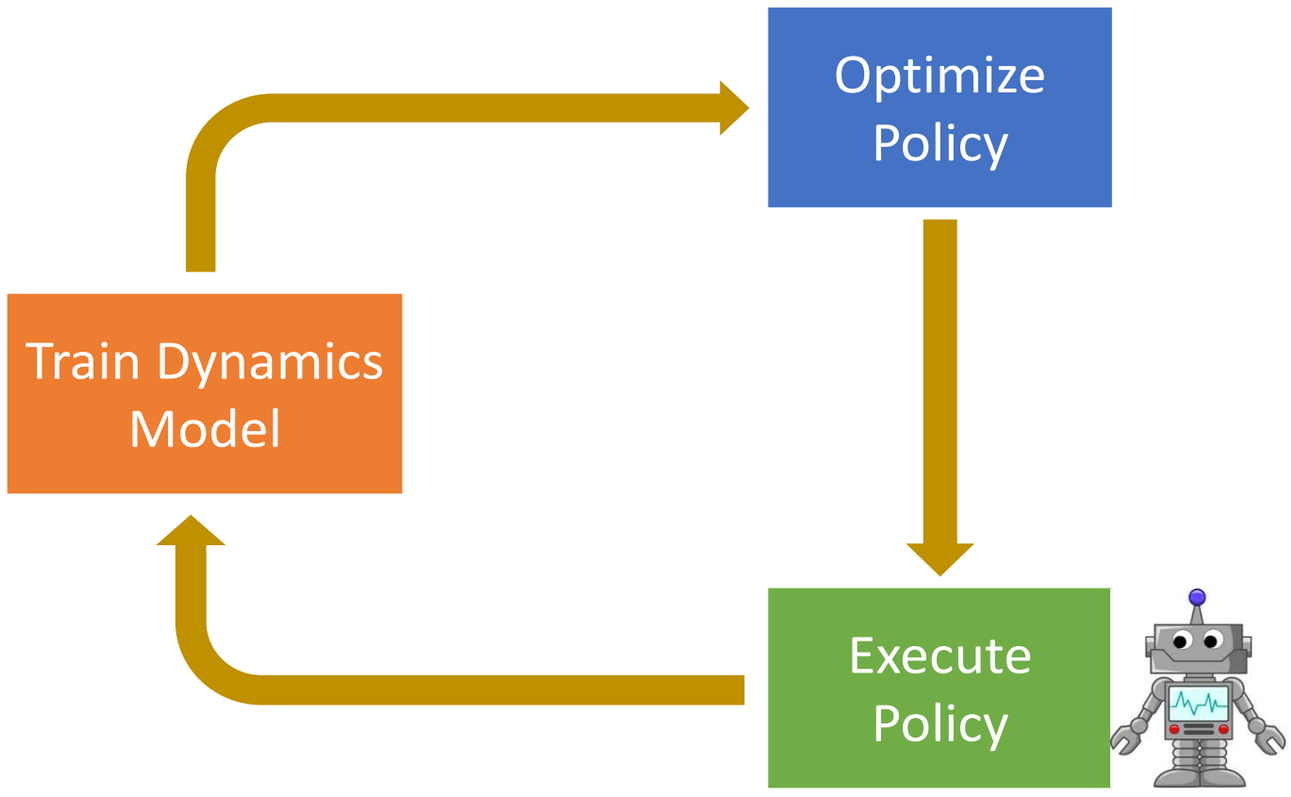
Why model learning

- Model-based control: given an initial state s_0 estimate **action sequence** to reach a desired goal or maximize reward by unrolling the model forward in time
- Model-based RL: train **policies** using:
    1. a model-free RL method using simulated experience (experience sampled from the model)
    2. an imitation learning method by imitating the MB planner
- Efficient Exploration guided by model uncertainty (later lecture)

Alternating between model and policy learning

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

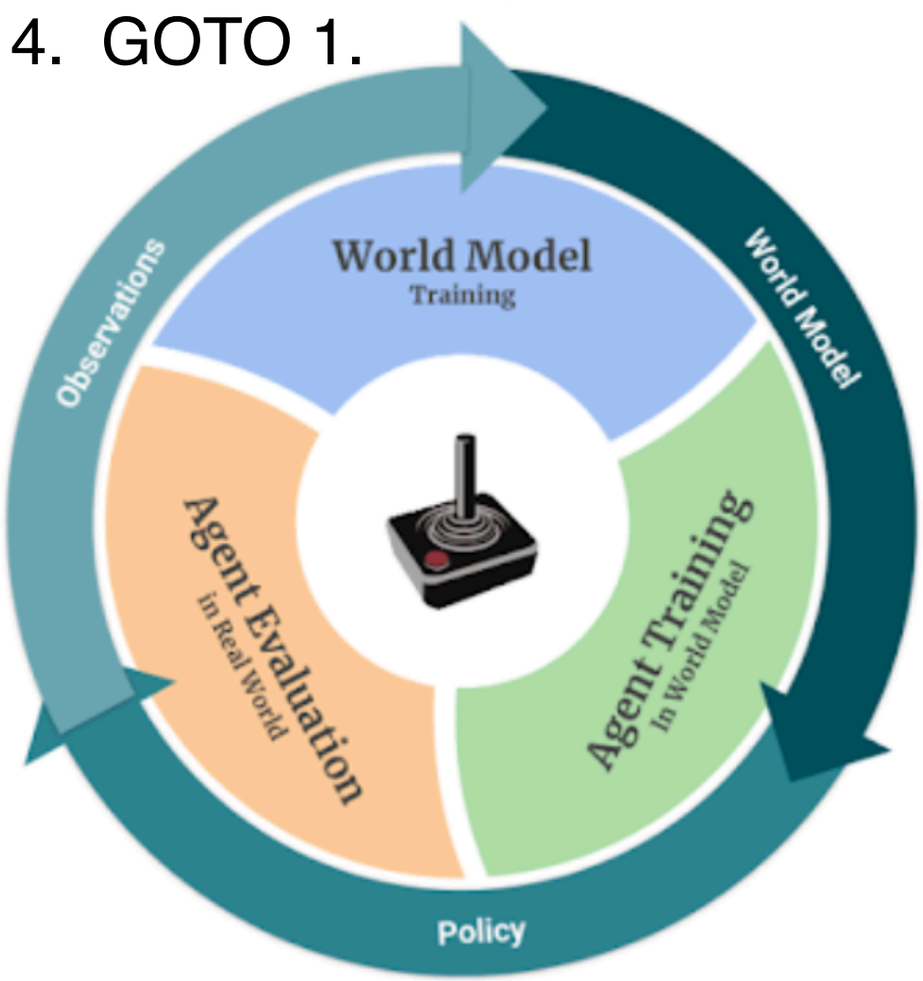
1. Run the policy and update experience tuples dataset D.
2. Train a dynamic model using D: $(s', r') = f(s, a; \phi)$
3. Update the policy using
    1. model-free RL method on simulated experience sampled from the model
    2. Immitating a model-based controller
4. GOTO 1.

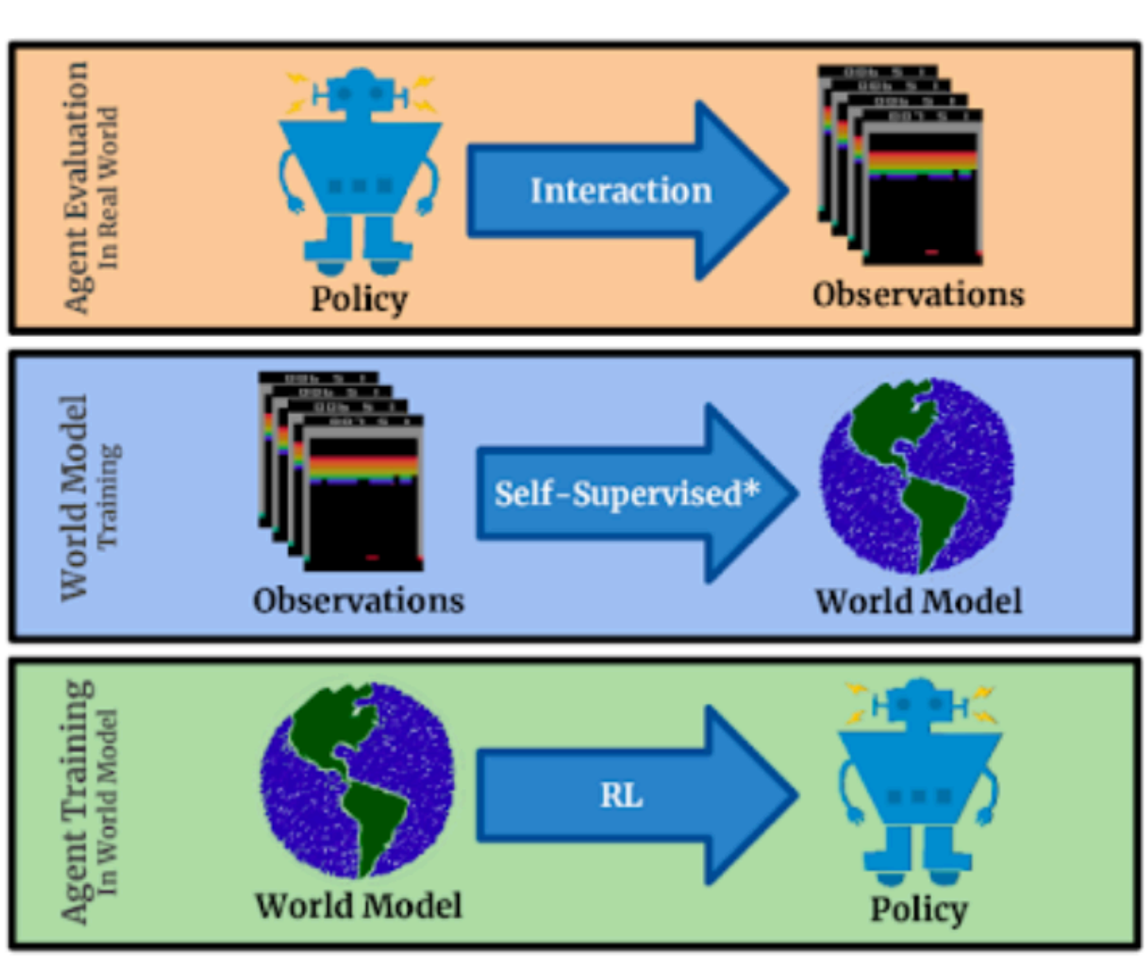


Alternating between model and policy learning

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 using
    1. model-free RL method on simulated experience sampled from the model
    2. Immitating a model-based controller
4. GOTO 1.

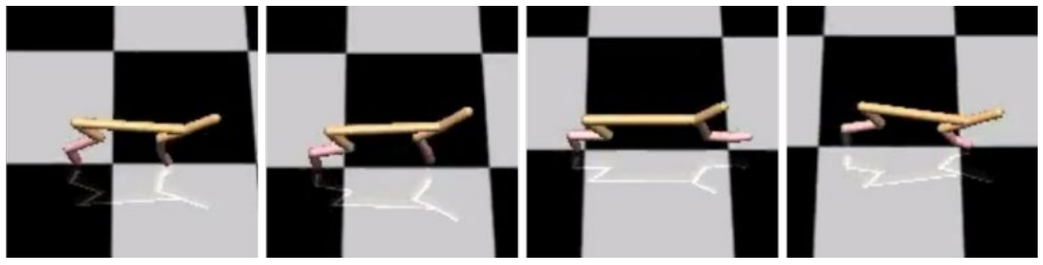


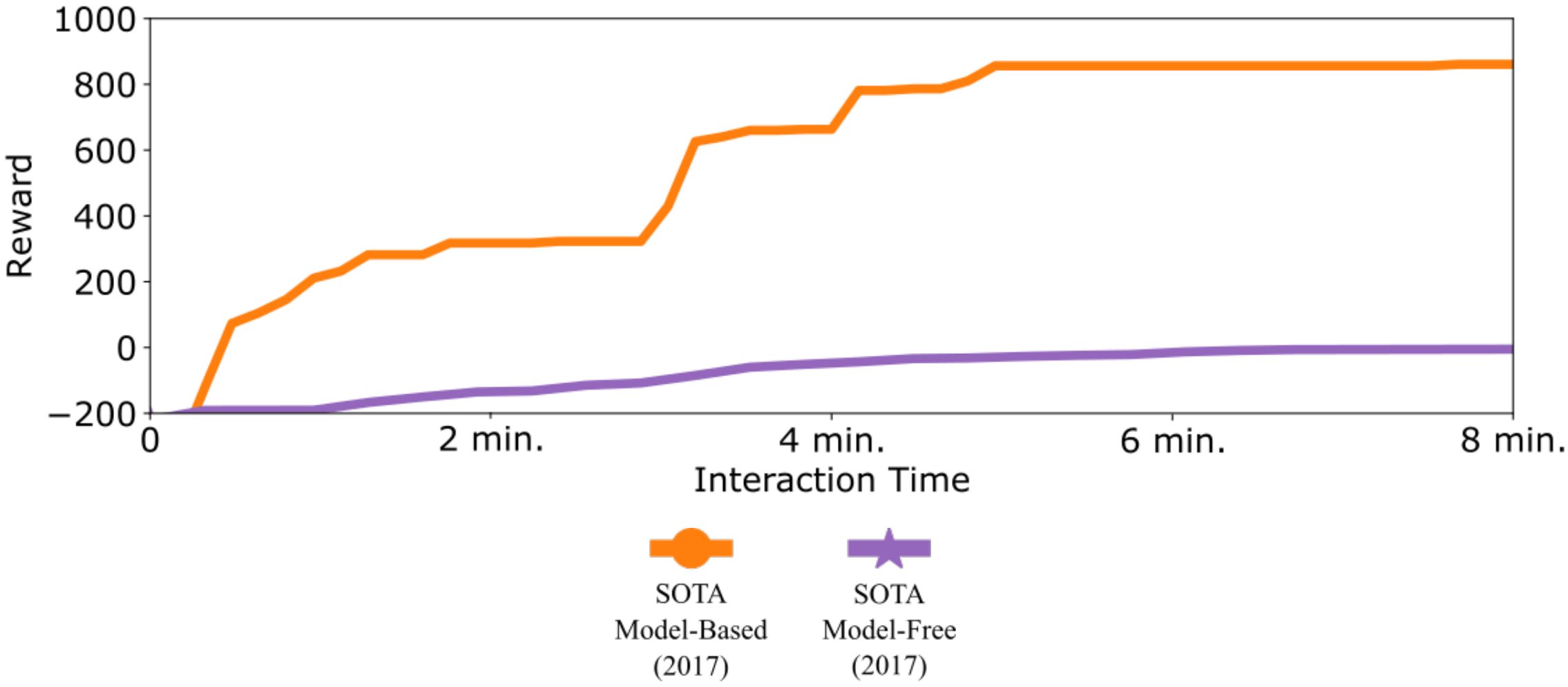


Challenges in model learning

- 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, if the model is mistakenly optimistic

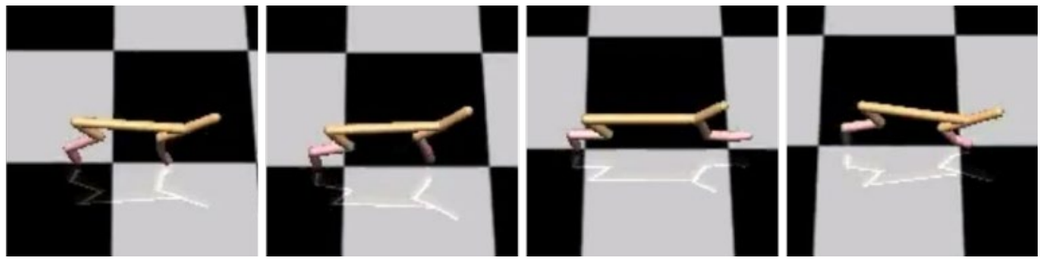
Comparative Performance on HalfCheetah

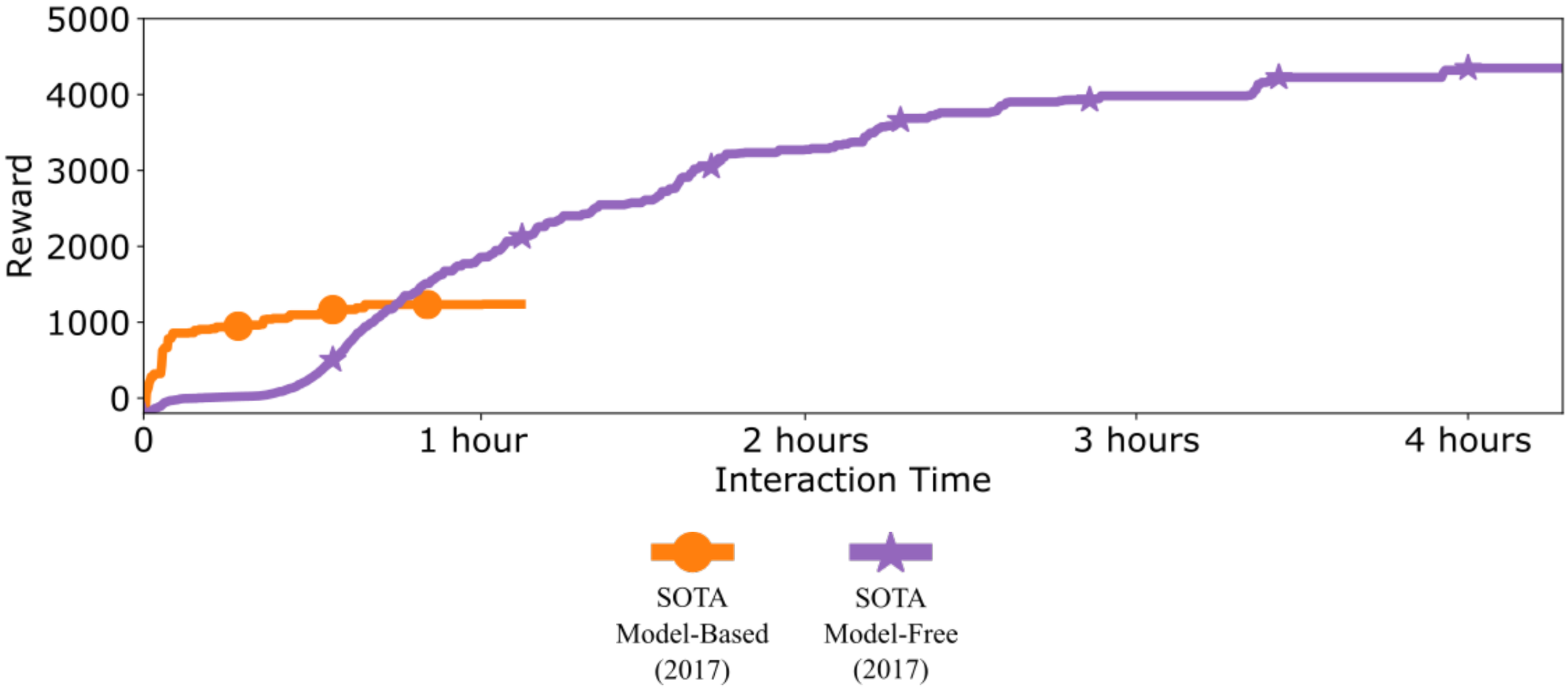




slide from Sergey Levine

Comparative Performance on HalfCheetah

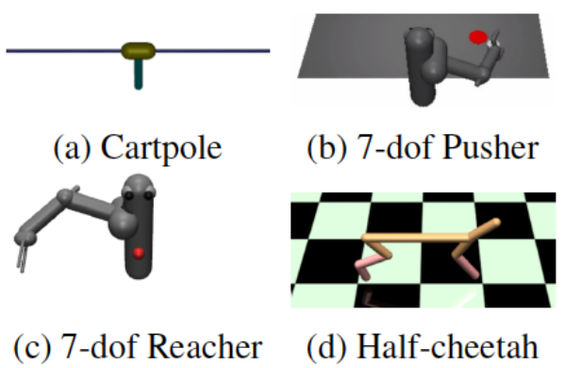




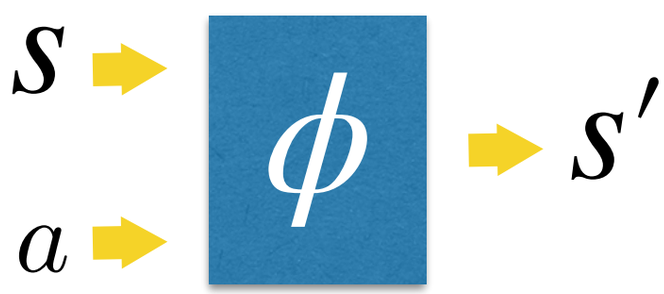
slide from Sergey Levine

Model Learning

- Where a low dimensional state is observed and given:

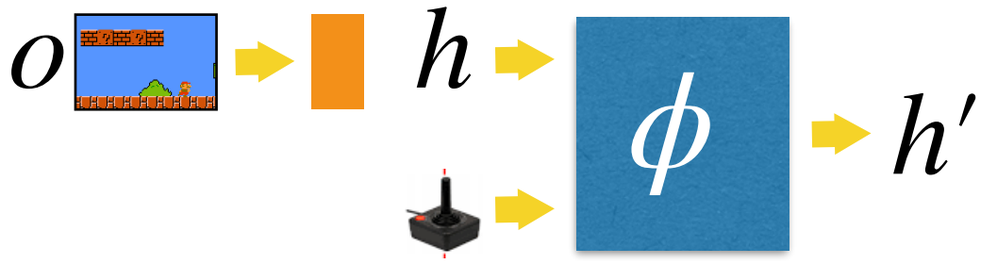


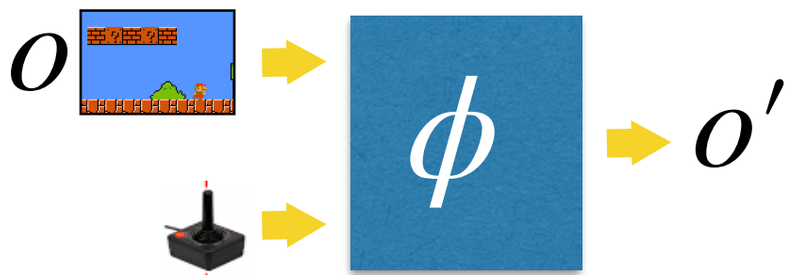
state can be 3D locations and 3D velocities of agent joints, actions can be torques



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

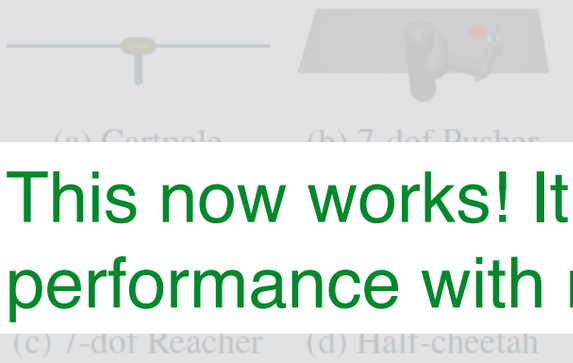
e.g., Atari game playing





Model Learning

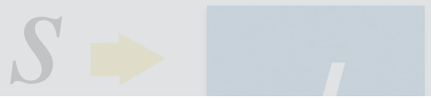
- Where a low dimensional state is observed and given:



(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

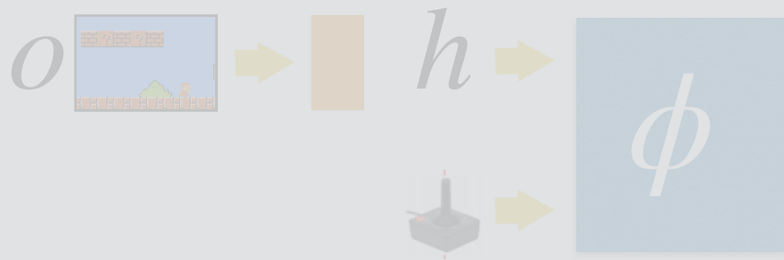


This now works! It outperforms model-free RL methods: reaches same final performance with much fewer samples! :-)

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

e.g., Atari game playing

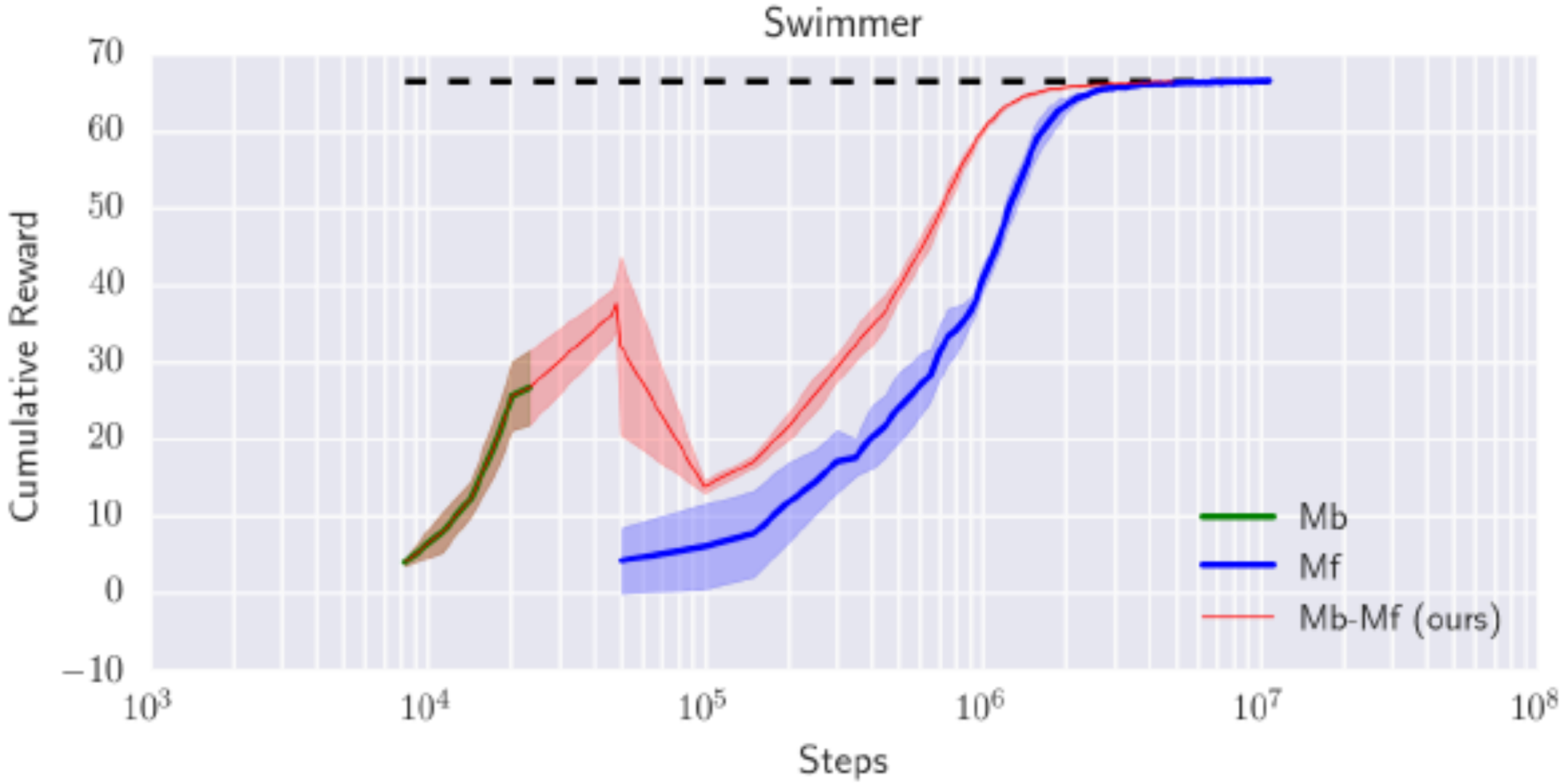
Still an open problem :-(



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

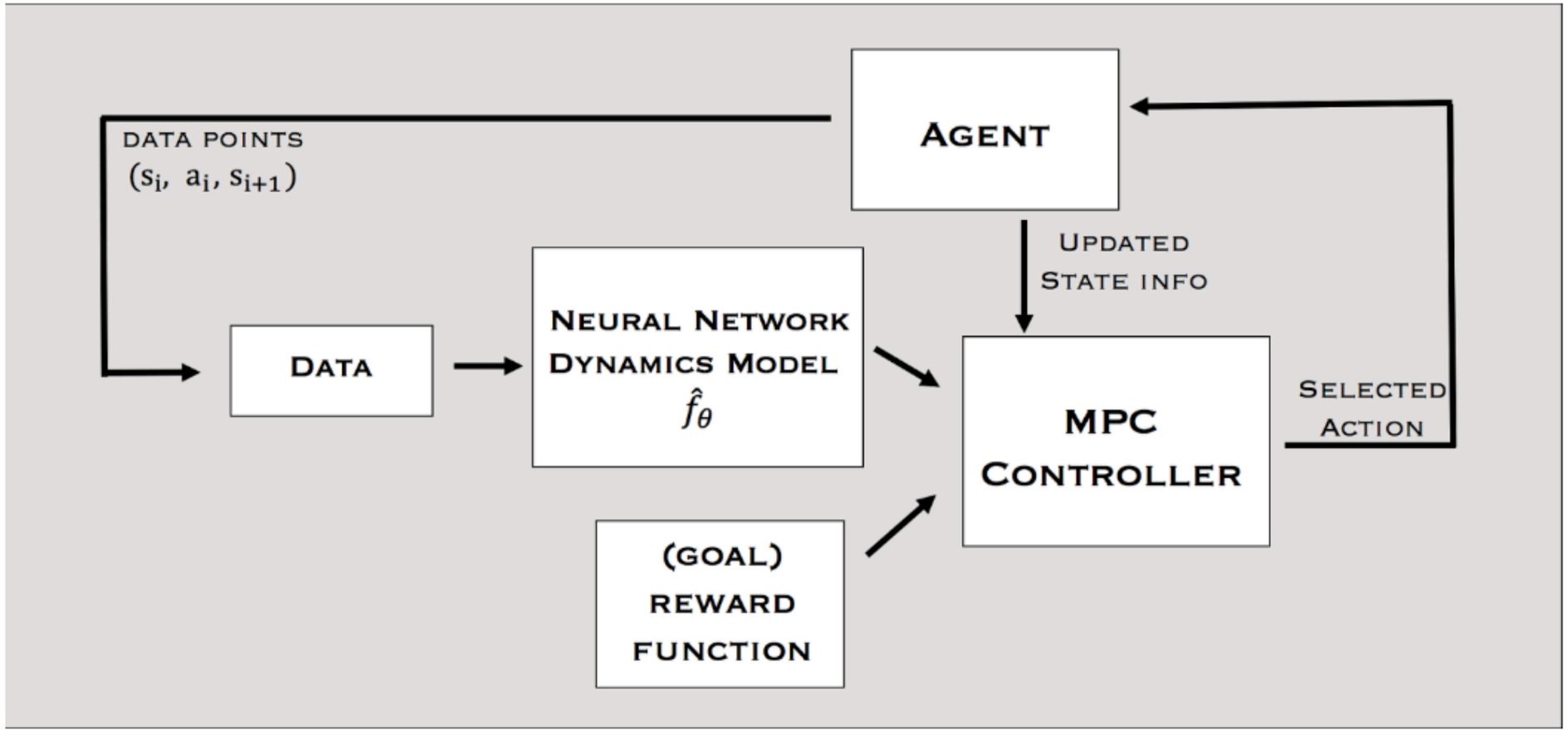


Neural network dynamics for model-based Deep RI with model-free finetuning, Nagabandi et al.

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

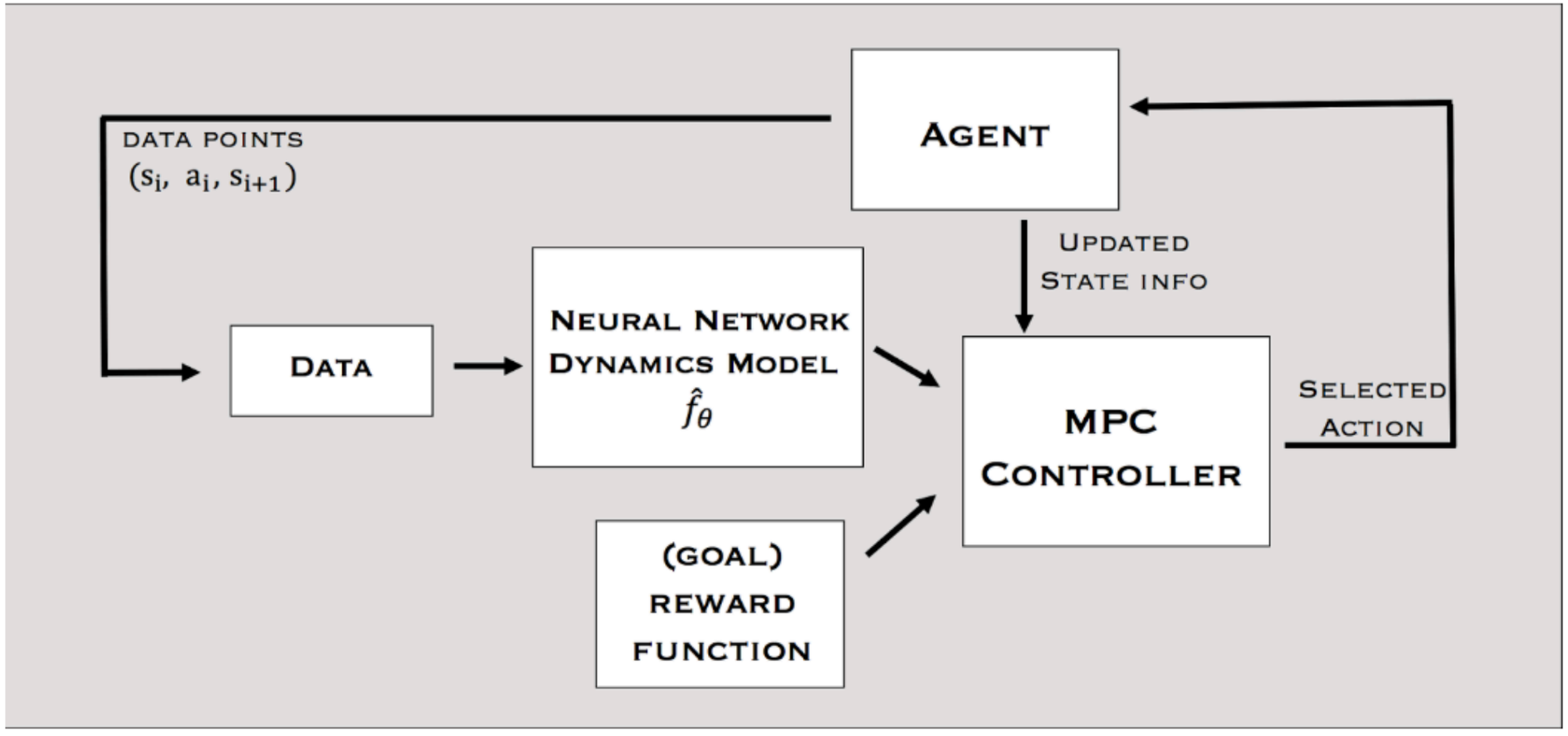


Neural network dynamics for model-based Deep RI with model-free finetuning, Nagabandi et al.

Model-based RL

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

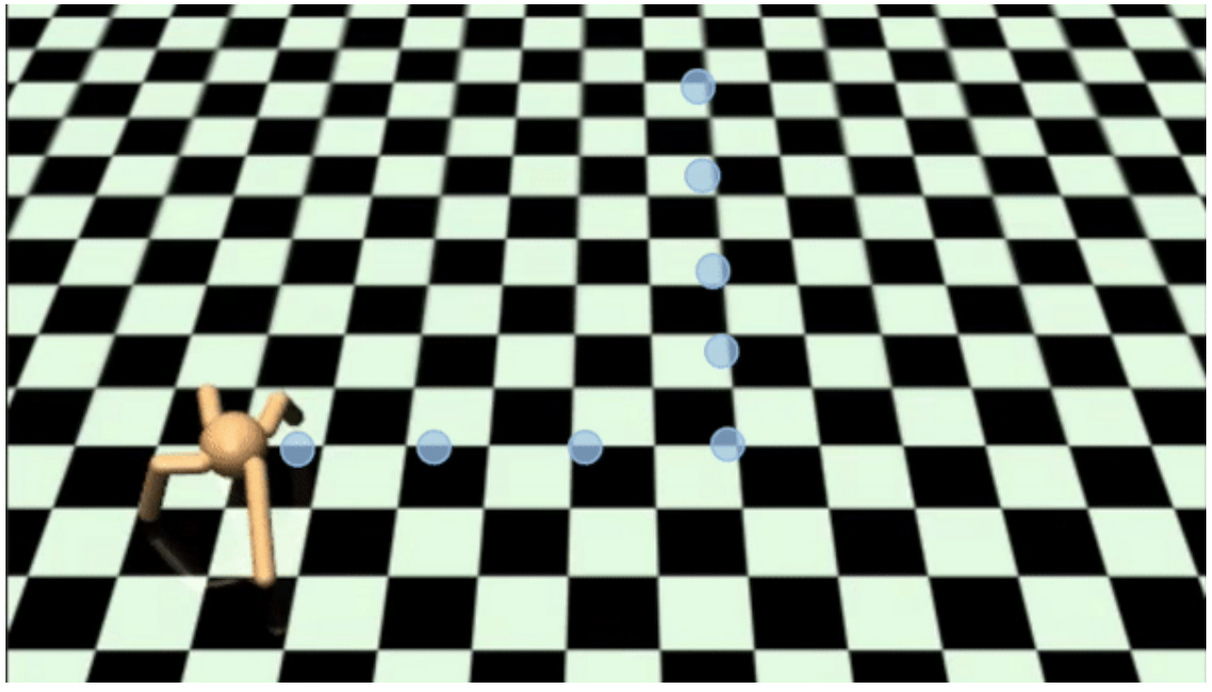
Collect a dataset D of random experience tuples (s,a,s')

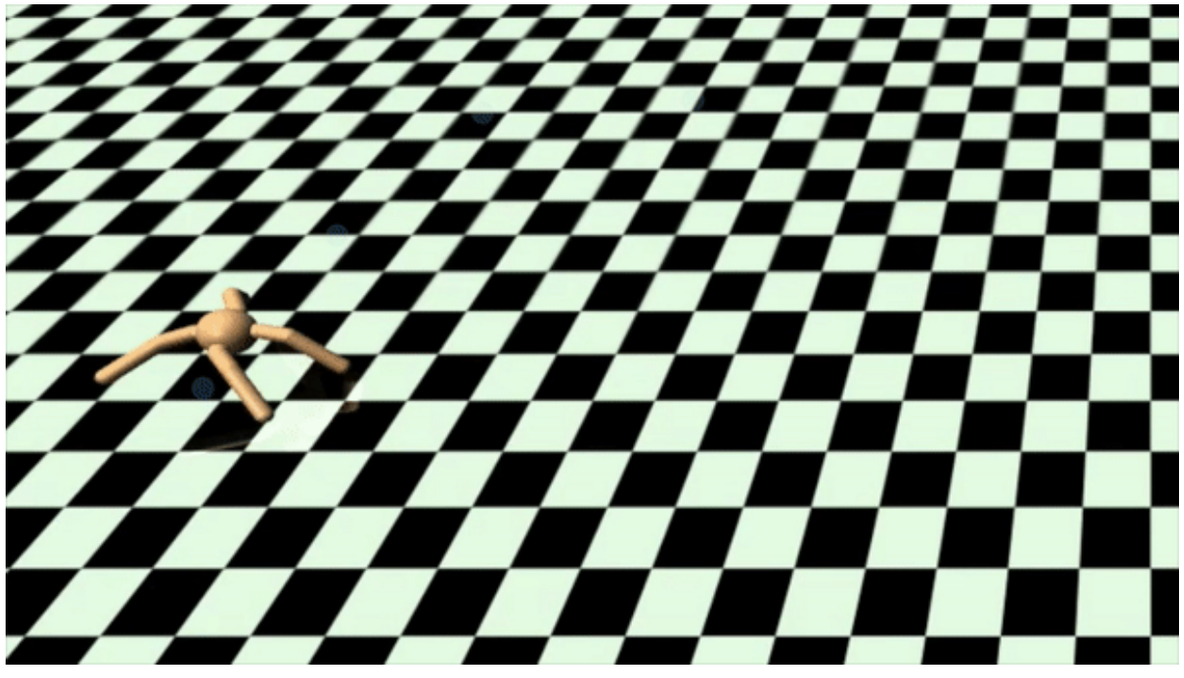
1. Train transition dynamics $s' = s + f(s, a; \phi)$
2. Optimize action sequences using MPC with random search
3. **Aggregate** experience dataset with the inferred (s,a) sequences
4. GOTO 1

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

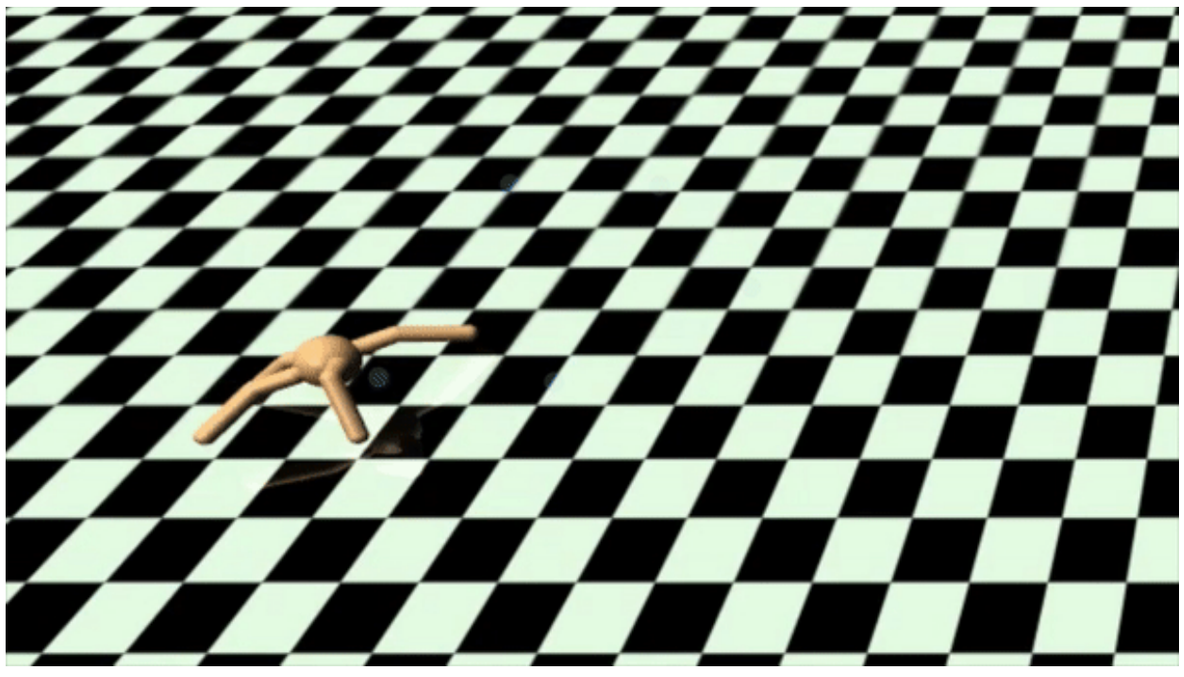
Finetune the policy using any model-free method, e.g., TRPO.

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

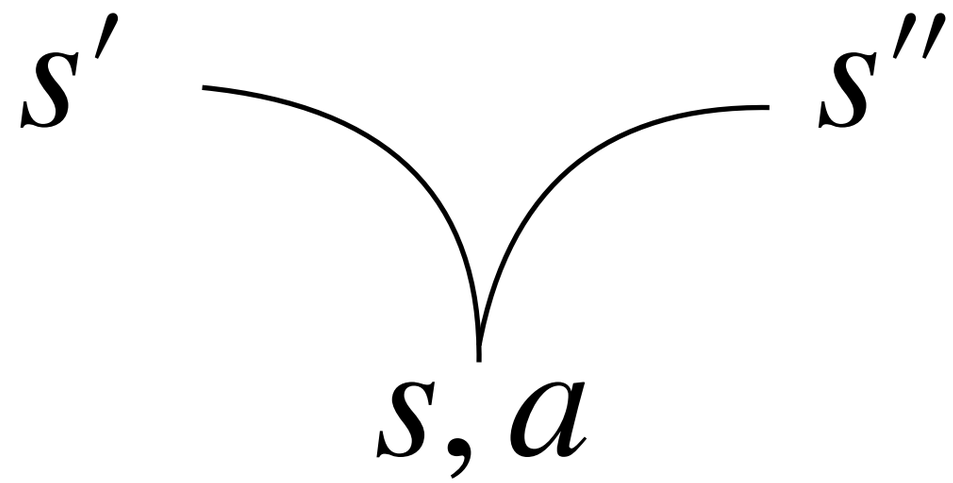
Kurtland Chua **Roberto Calandra** **Rowan McAllister** **Sergey Levine**
Berkeley Artificial Intelligence Research
University of California, Berkeley
{kchua, roberto.calandra, rmcallister, svlevine}@berkeley.edu

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

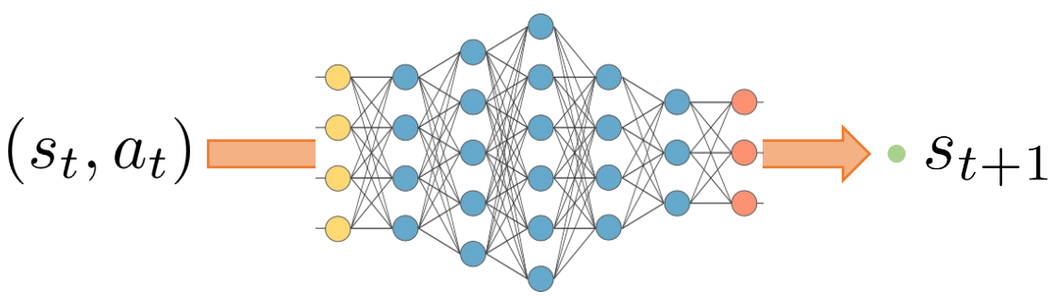


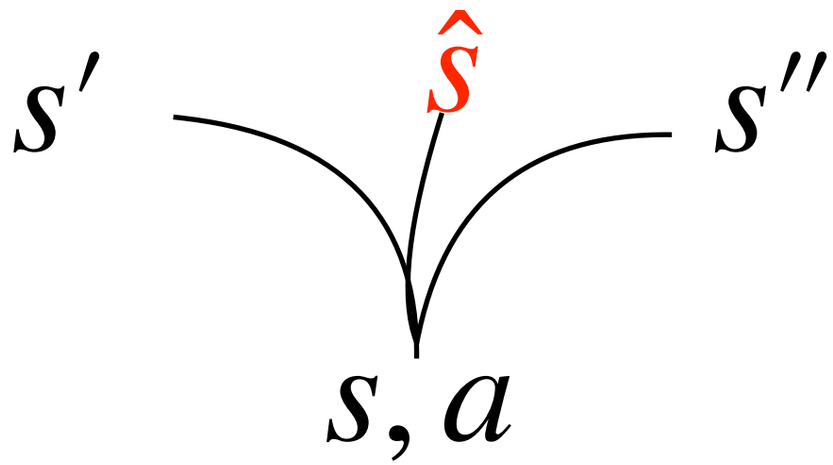
- This means our state does not capture enough information to help us dileneate the possible future outcomes.
- What is stochastic under one state representation, may not be stochastic under another. Is this true? Could we ever predict exactly what we will see in the TV when we switch the channel?
-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

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



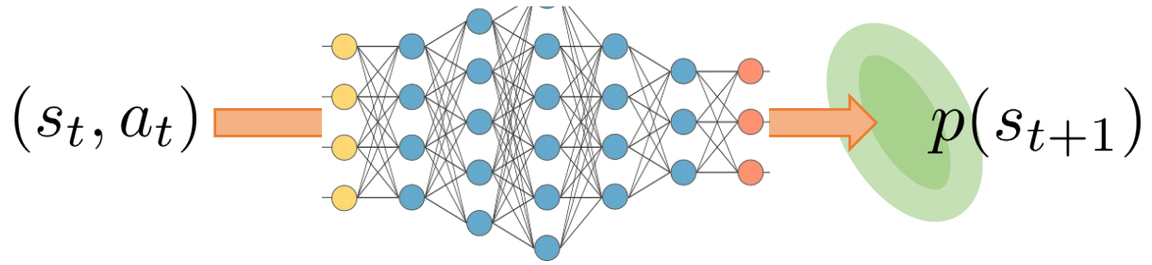


Failing means: not only we cannot capture the distribution, but we output a solution that does not agree with any of the modes

Aleatoric uncertainty in model learning

- Consider experience tupples $\mathrm{D} = \{(s_i, a_i, s_i'), i = 1 \cdots N\}$.
- 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

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



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

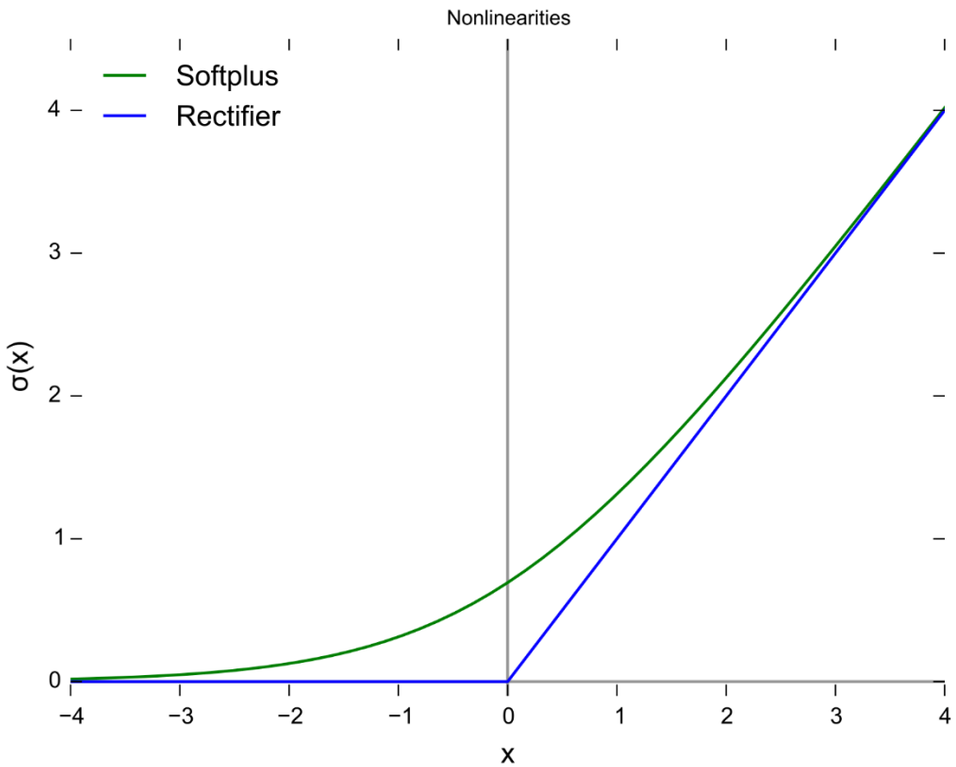
$$= \frac{1}{2}(s_i' - \mu(s_i, a_i; \phi))^T \boldsymbol{\Sigma}(s_i, a_i; \phi)^{-1}(s_i' - \mu(s_i, a_i; \phi)) + \frac{1}{2}\log(\det \boldsymbol{\Sigma}(s_i, a_i; \phi)) + \text{ const.}$$

Aleatoric uncertainty in model learning

Variance should be always positive, what do we do?

We output log(variance) and we exponentiate.

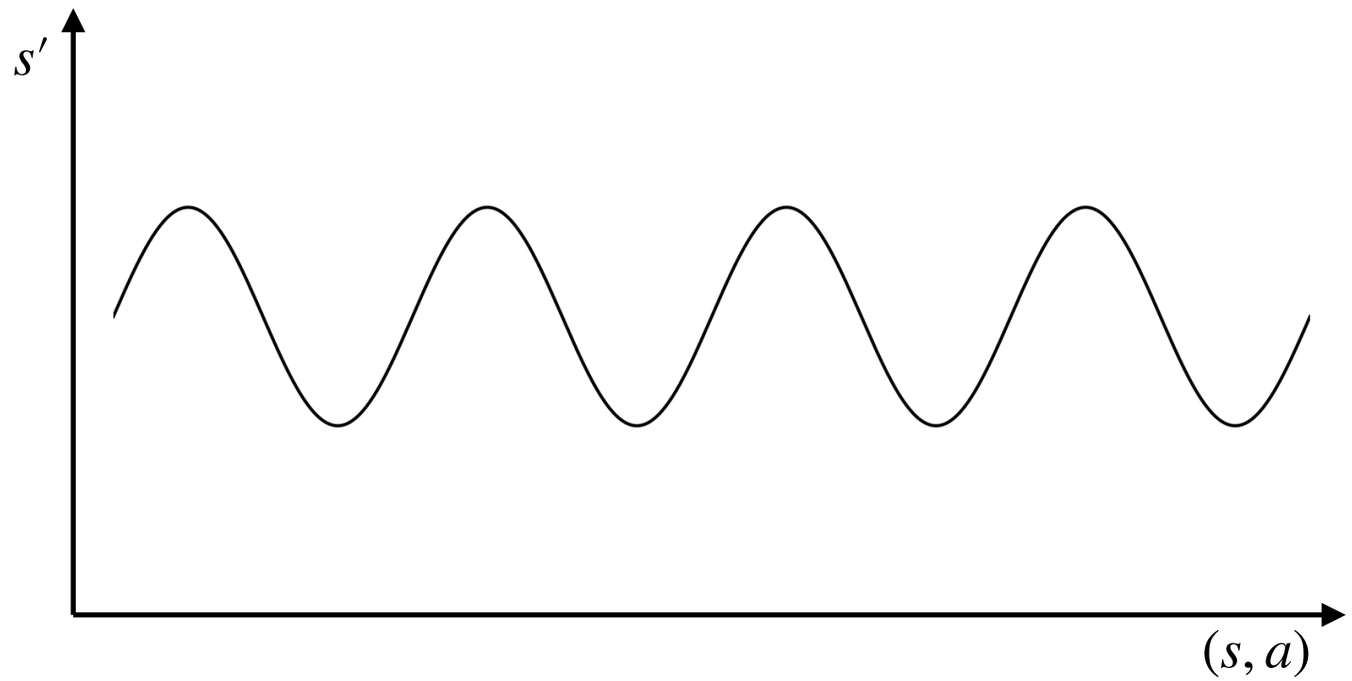
```
logvar = max_logvar - tf.nn.softplus(max_logvar - logvar)
logvar = min_logvar + tf.nn.softplus(logvar - min_logvar)
var = tf.exp(logvar)
```



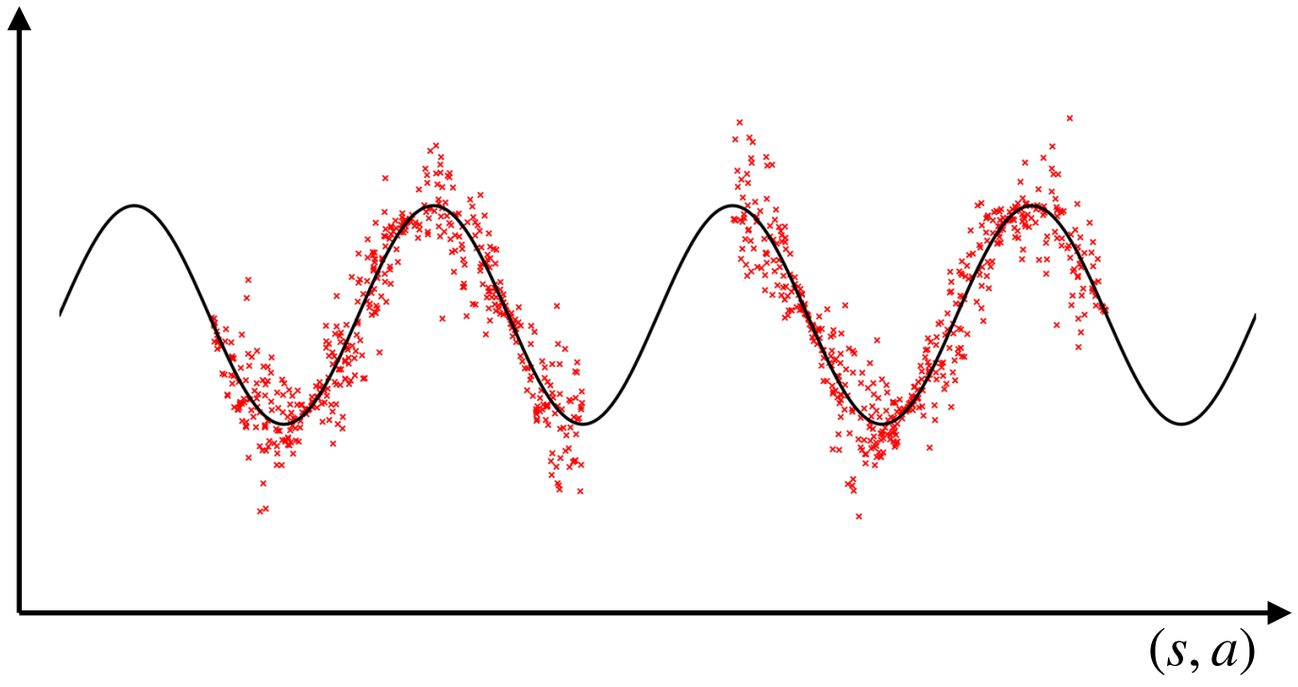
$\text{Rectifier}(x) = \max(0, x)$

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

Epistemic uncertainty in Model Learning

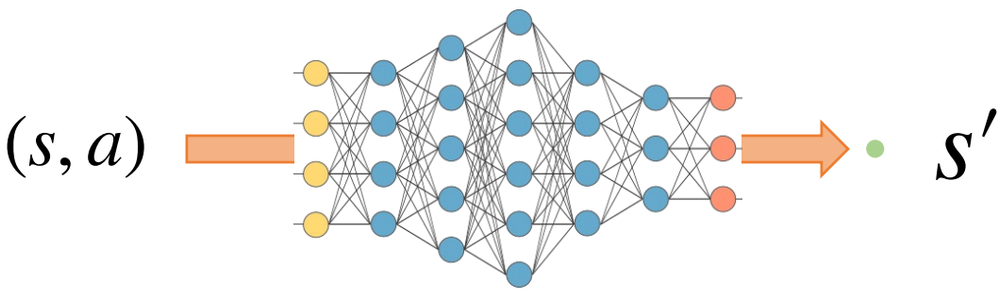


Epistemic uncertainty in Model Learning

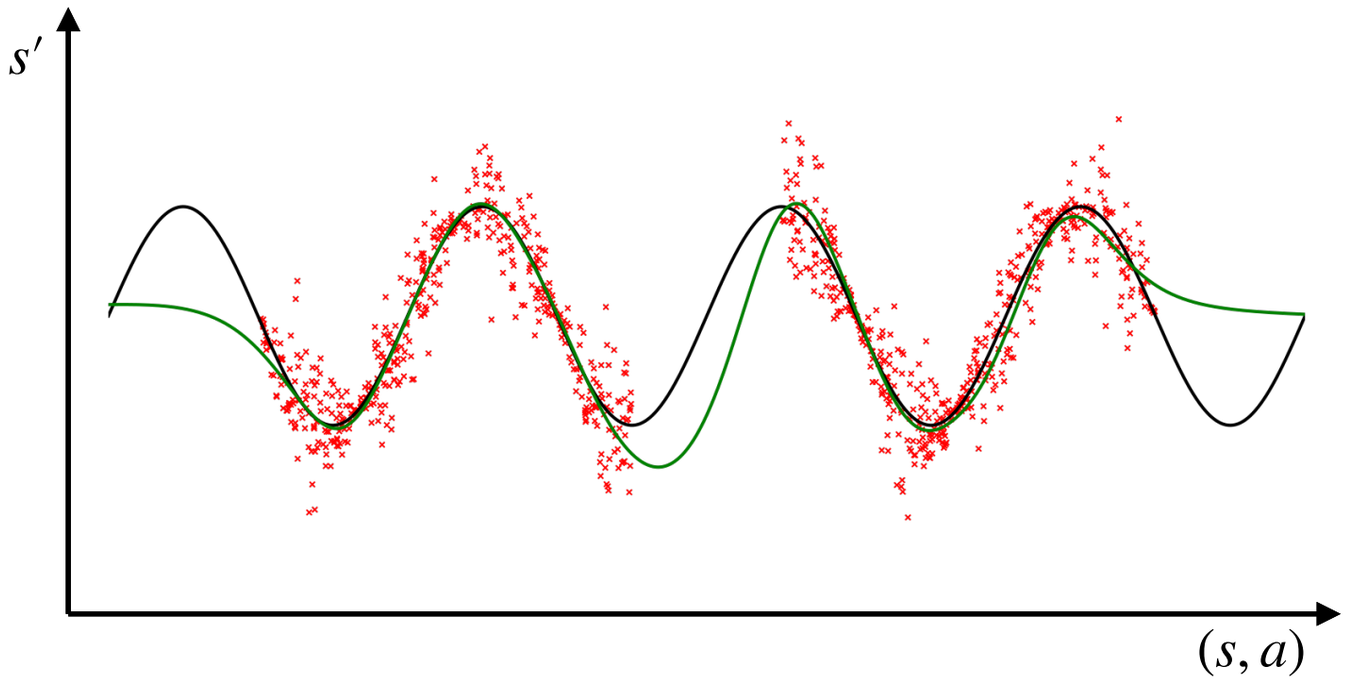


Red are observed data points (s,a,s')

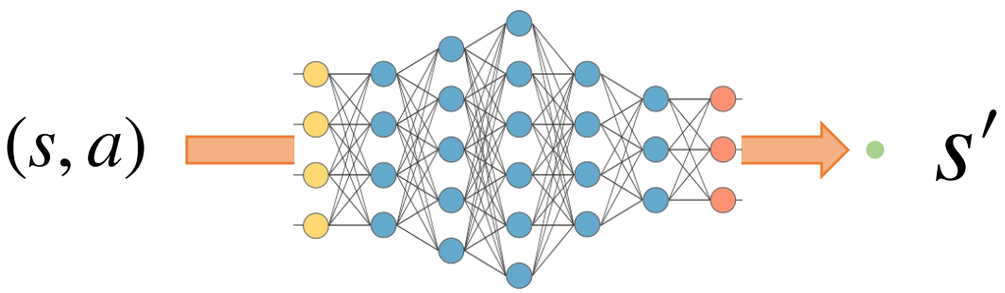
Epistemic uncertainty in Model Learning



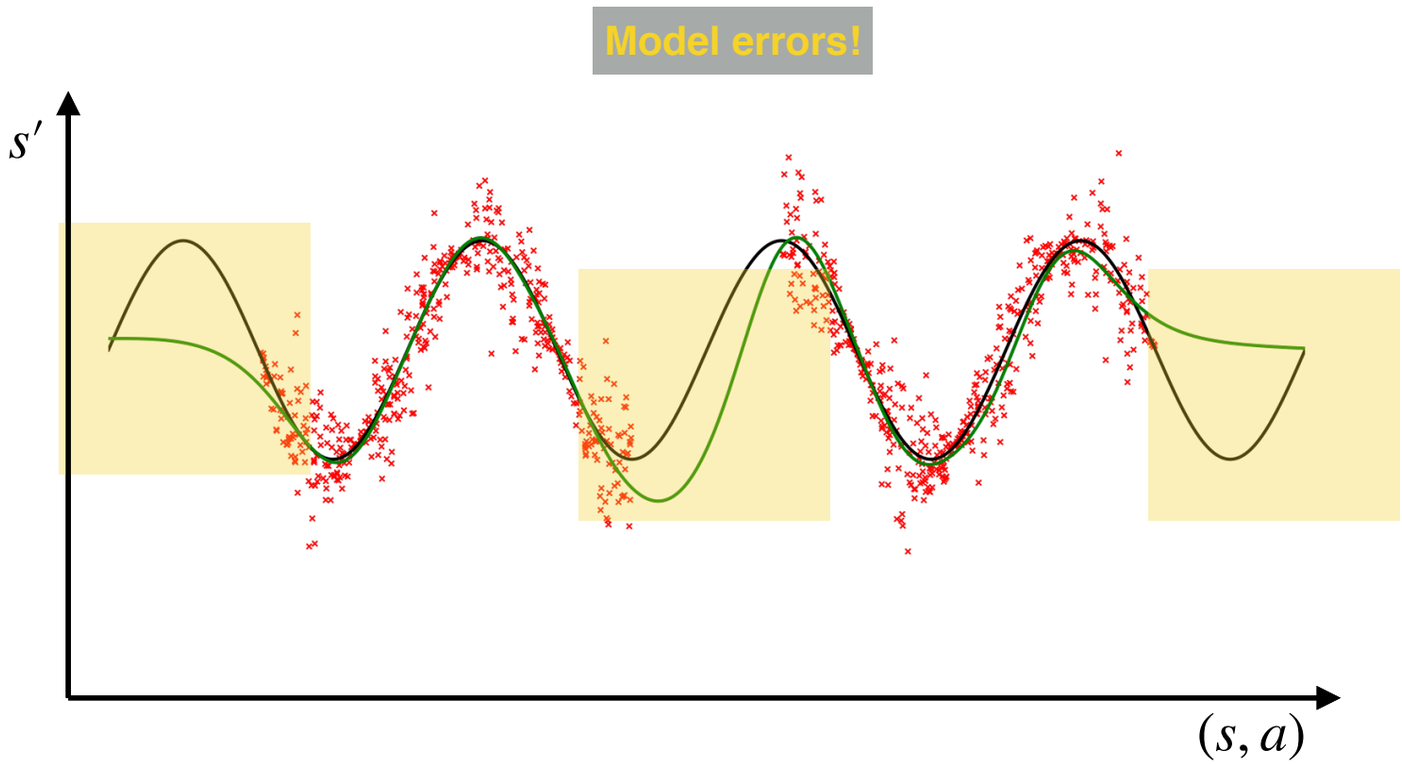
Fitting a deterministic neural network



Epistemic uncertainty in Model Learning

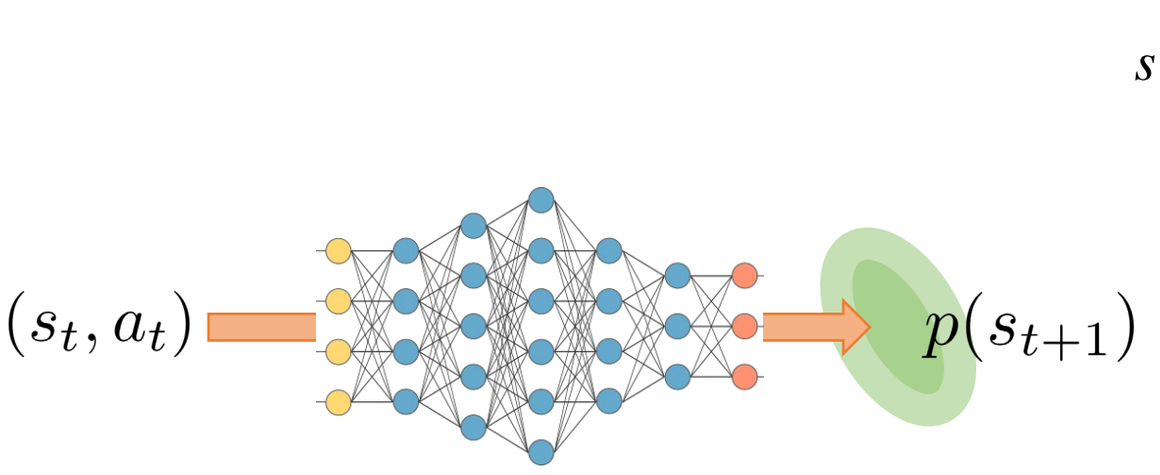


Fitting a deterministic neural network

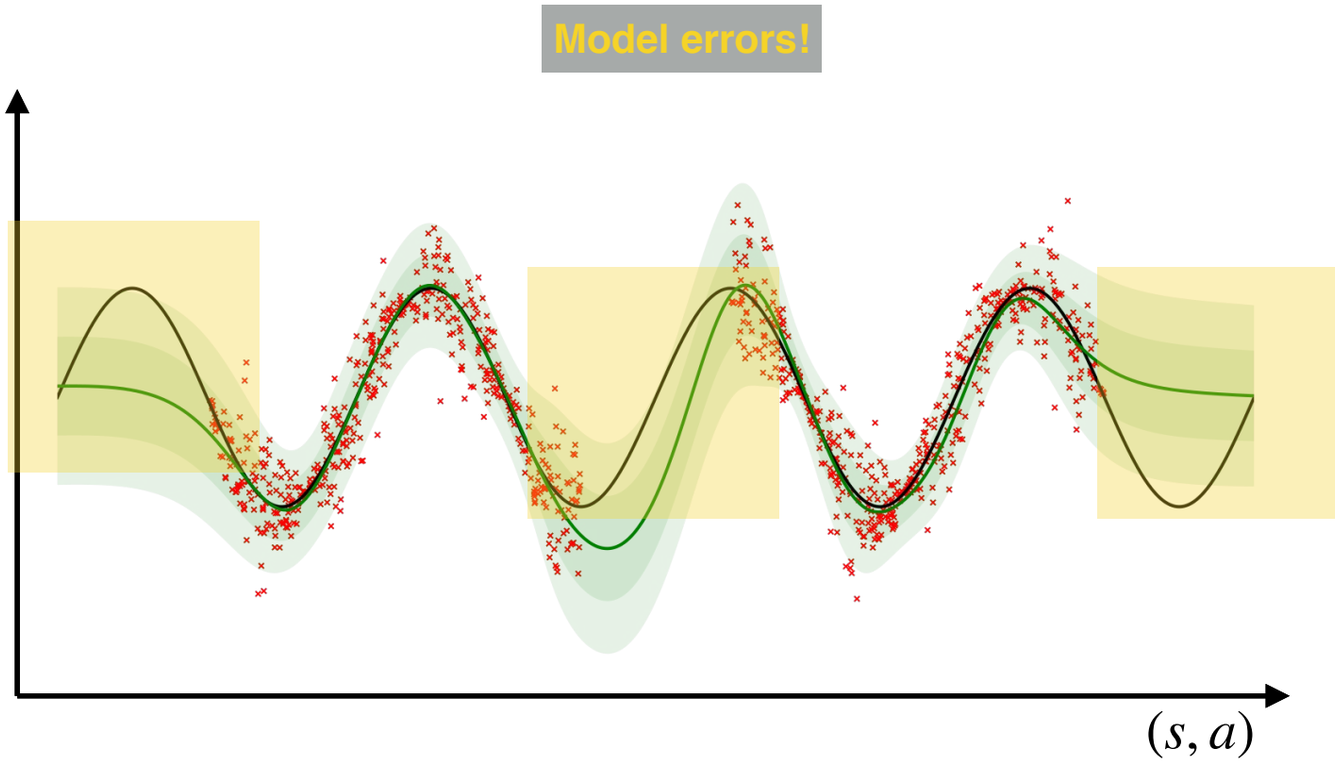


There is a unique answer for s’ (no stochasticity) but I do not know it due to lack of data!

Epistemic uncertainty in Model Learning

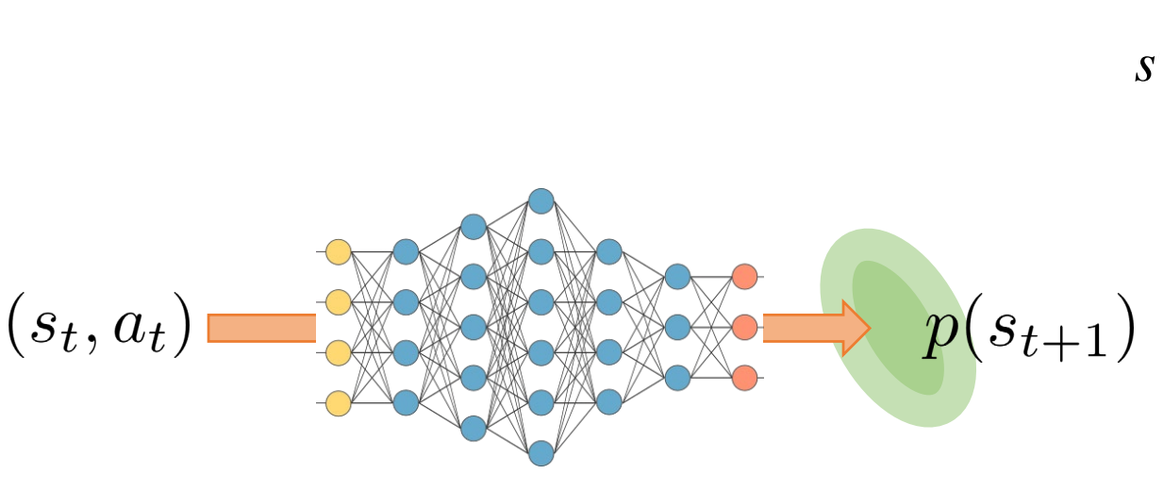


Fitting a neural network that parametrizes a distribution

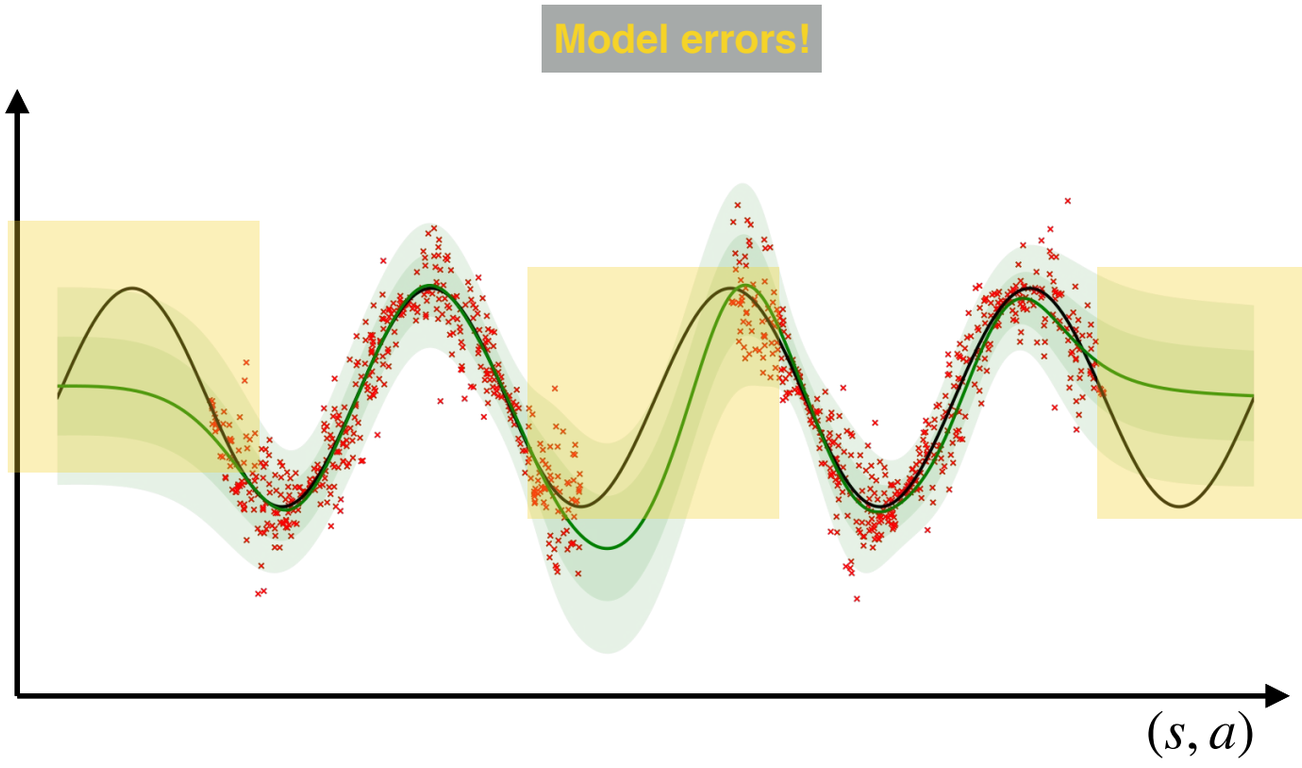


There is a unique answer for s’ (no stochasticity) but I do not know it due to lack of data!
Predicting a distribution won’t help! The predictions will suffer from lack of data and will be wrong.

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 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!

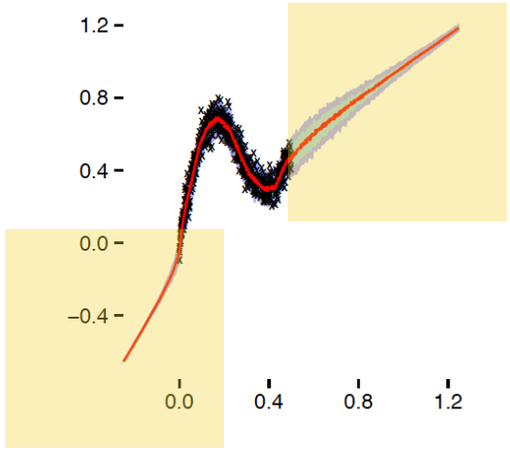
Bayes Rule

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

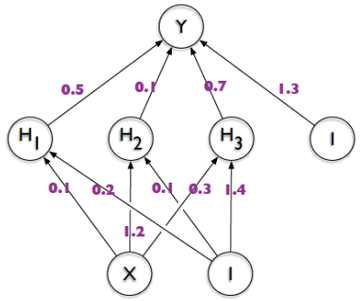
- Q: What are the hypotheses here?
- A: Hypotheses here are weights for our learning model, i.e., weights of our neural networks that learns the transition dynamics

- Q: Is this still usefull 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.

Reverend Thomas Bayes (1702-1761)



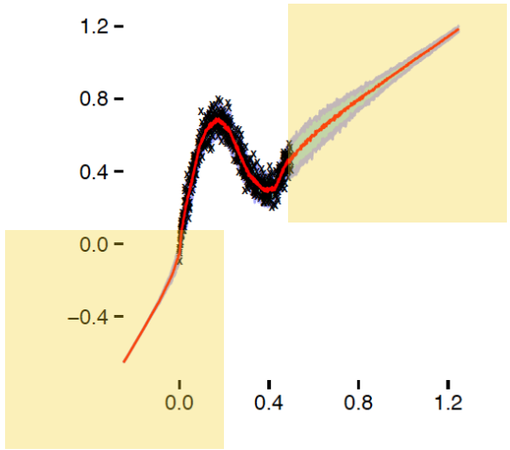
regression network



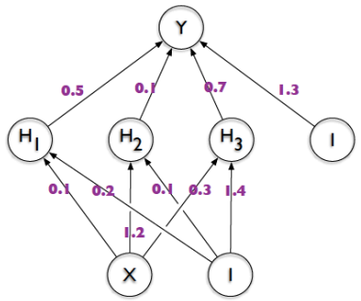
$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left(P(D \mid \phi) + \log P(\phi) \right)$$

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

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

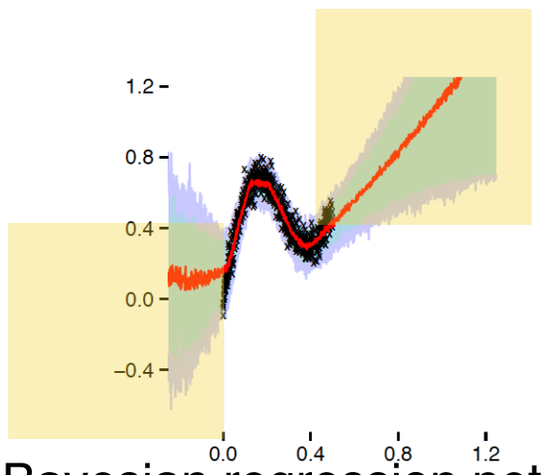


regression network



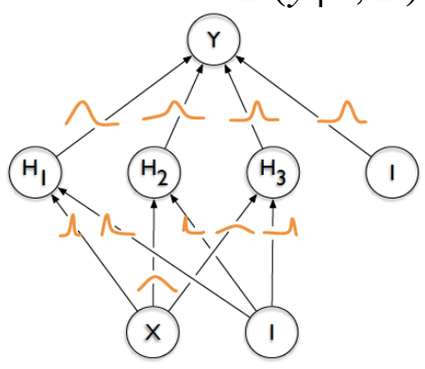
$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left(P(D \mid \phi) + \log P(\phi) \right)$$

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



Bayesian regression network

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

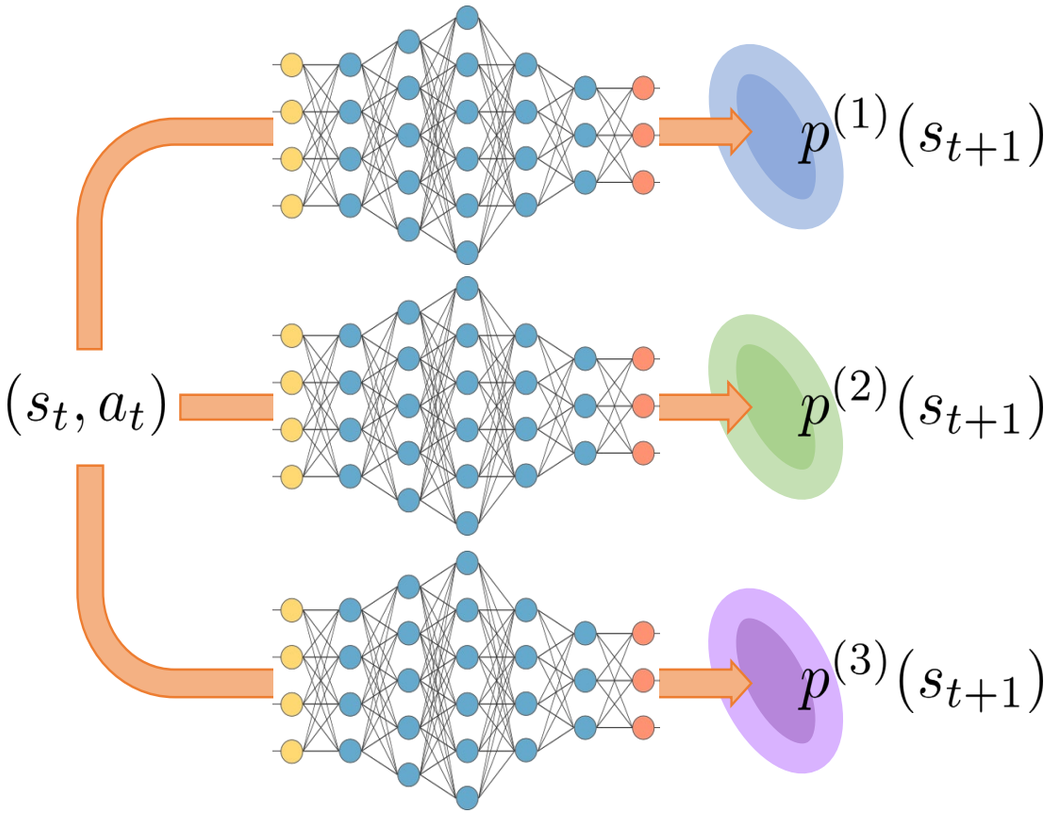


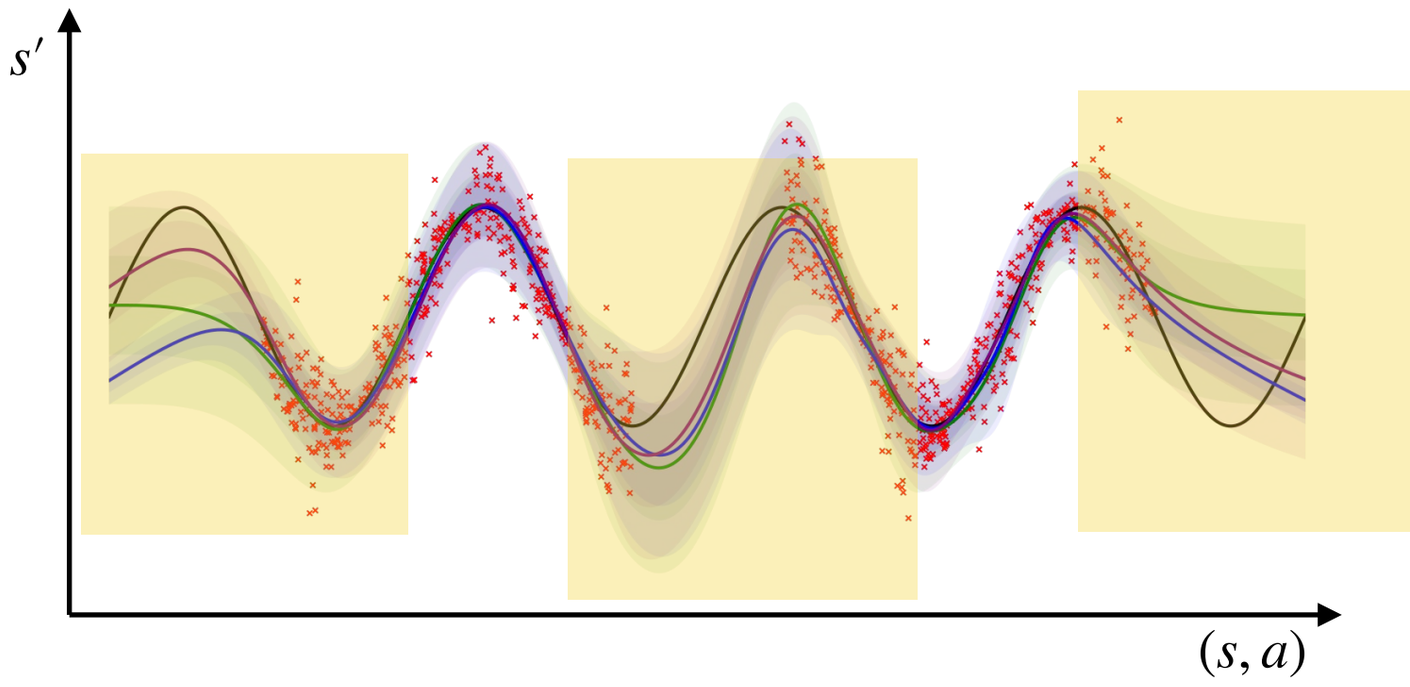
$P(\phi \mid D)$

- 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

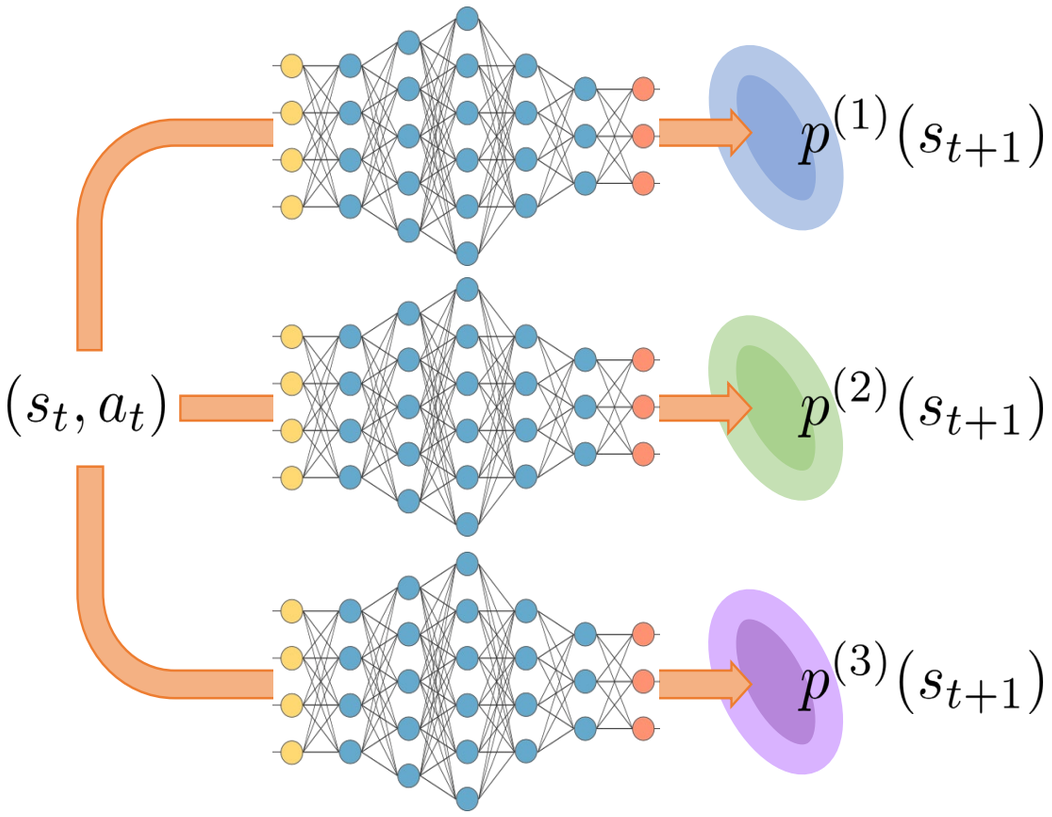
- Neural network Ensembles are a good approximation to Bayesian Nets.
- Instead of having explicit posteriors distributions for each neural net parameter, you just have a small set of neural nets, *each trained on separate data*.
  - On the data they have seen, they all agree (low entropy of predictions)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

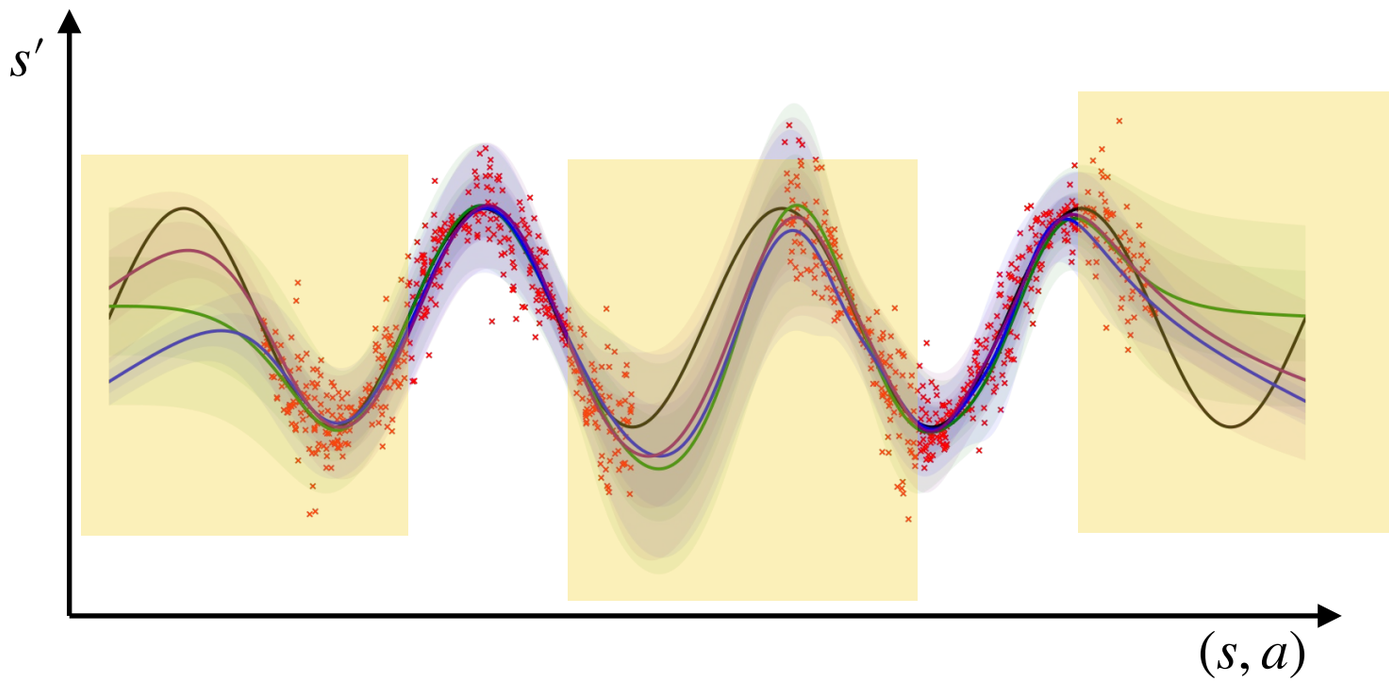




NN Ensembles for representing Epistemic uncertainty

- 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





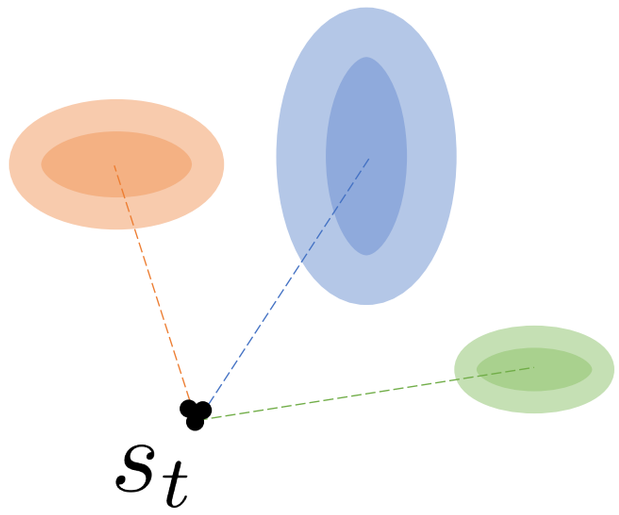
Model Unrolling

$s_t^{\bullet}$

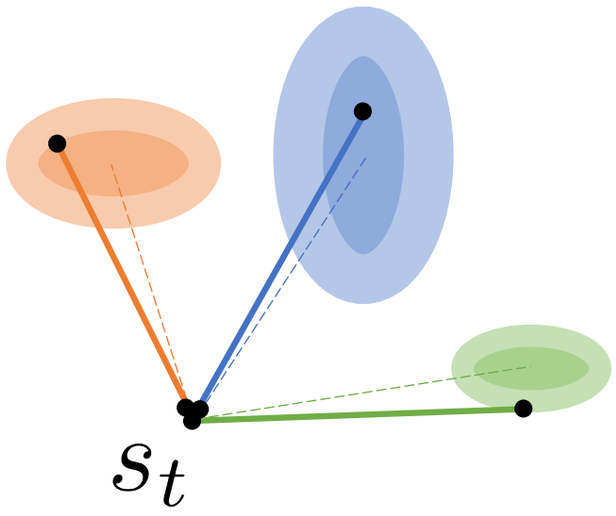
Model Unrolling

s_t

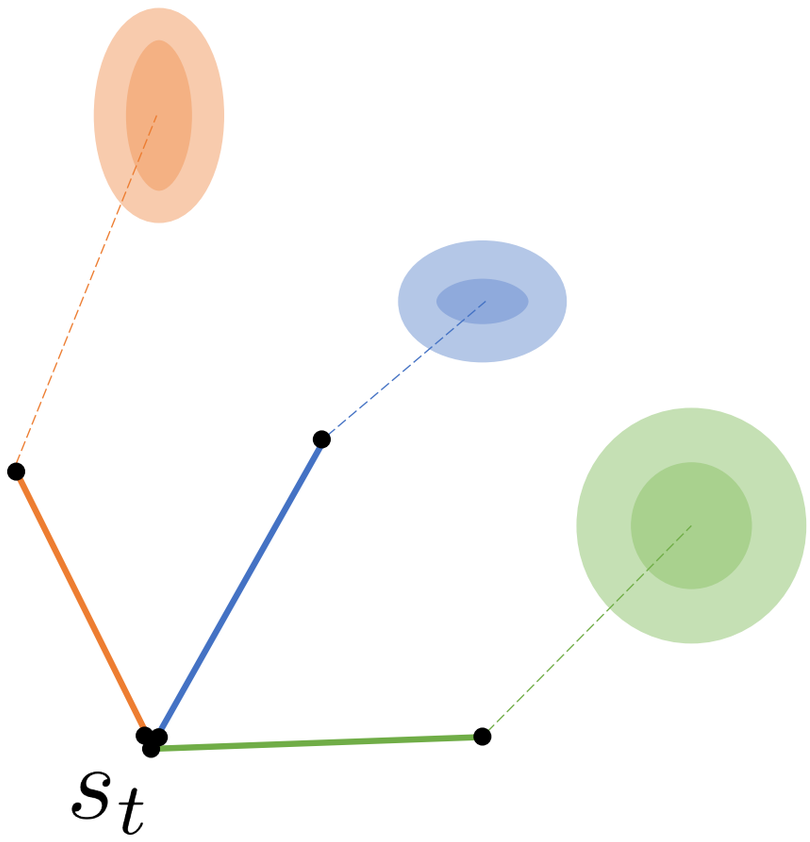
Model Unrolling



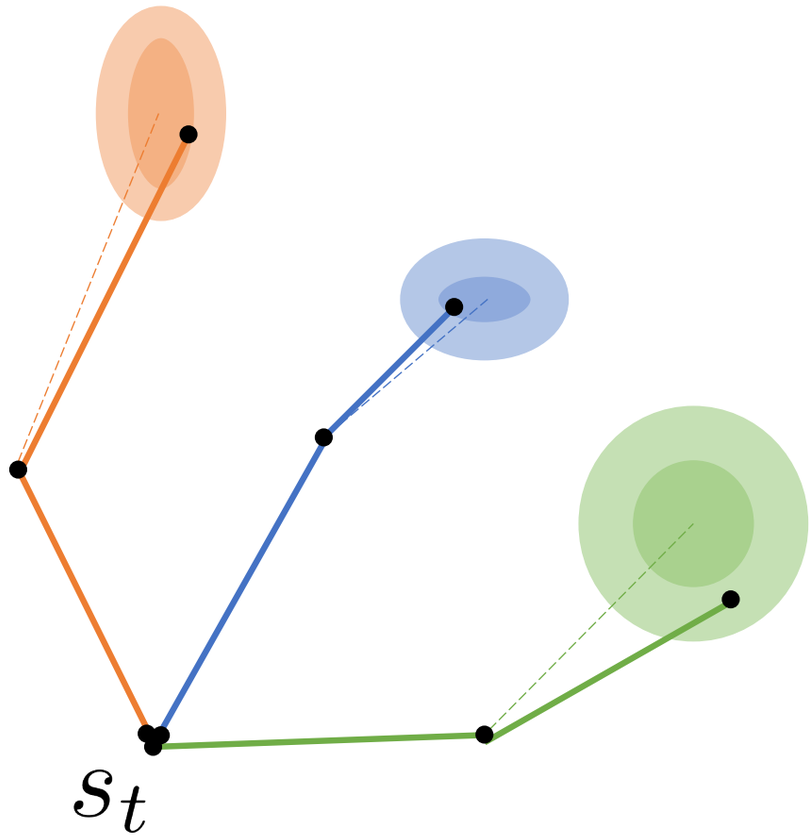
Model Unrolling



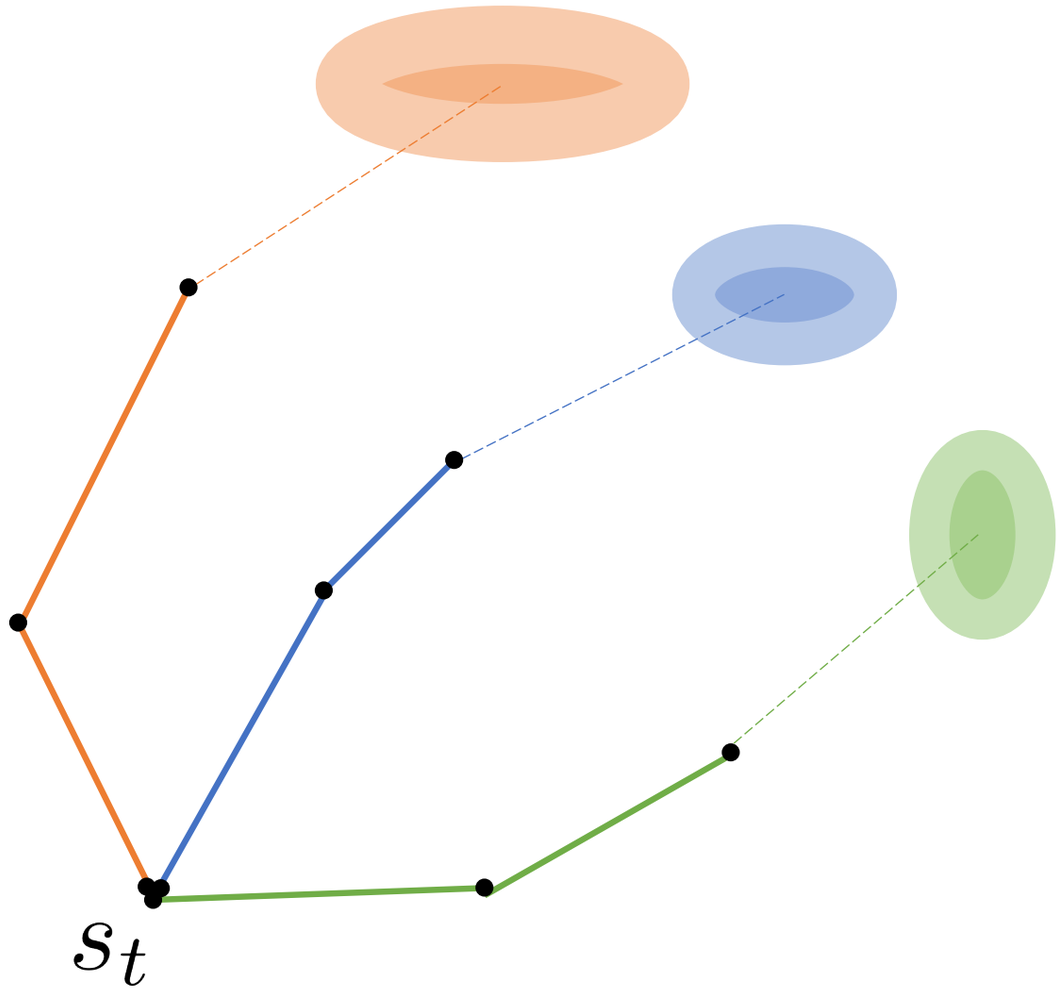
Model Unrolling



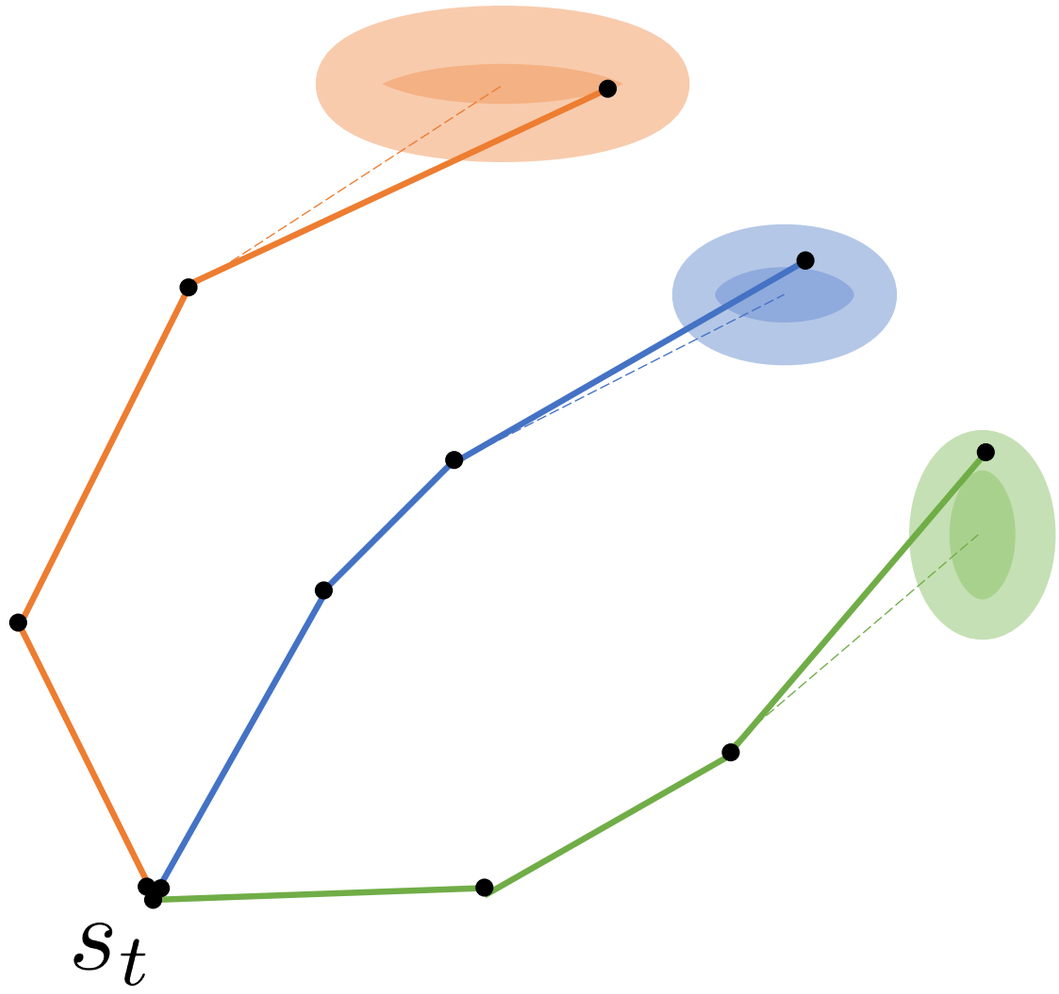
Model Unrolling



Model Unrolling

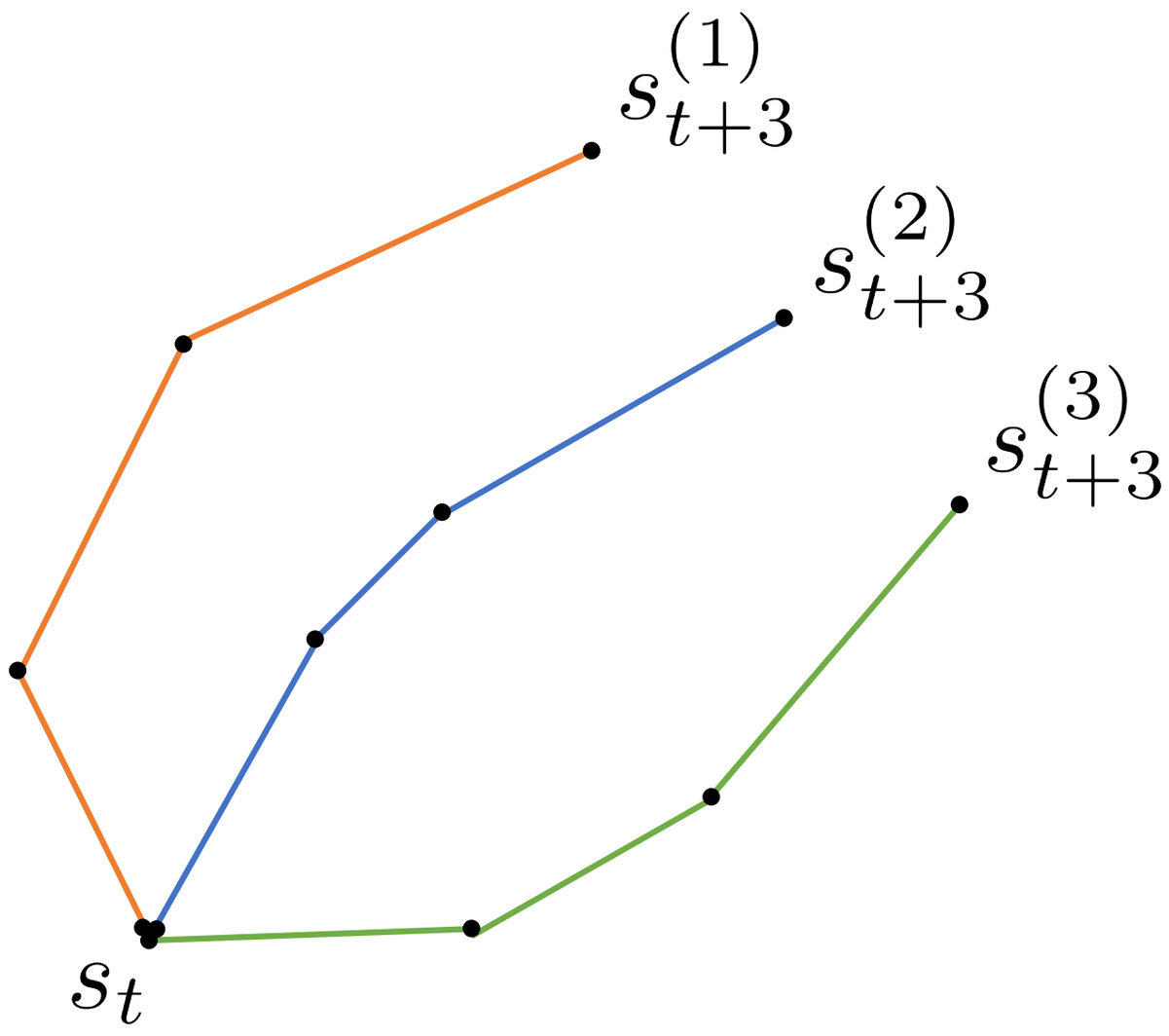


Model Unrolling



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

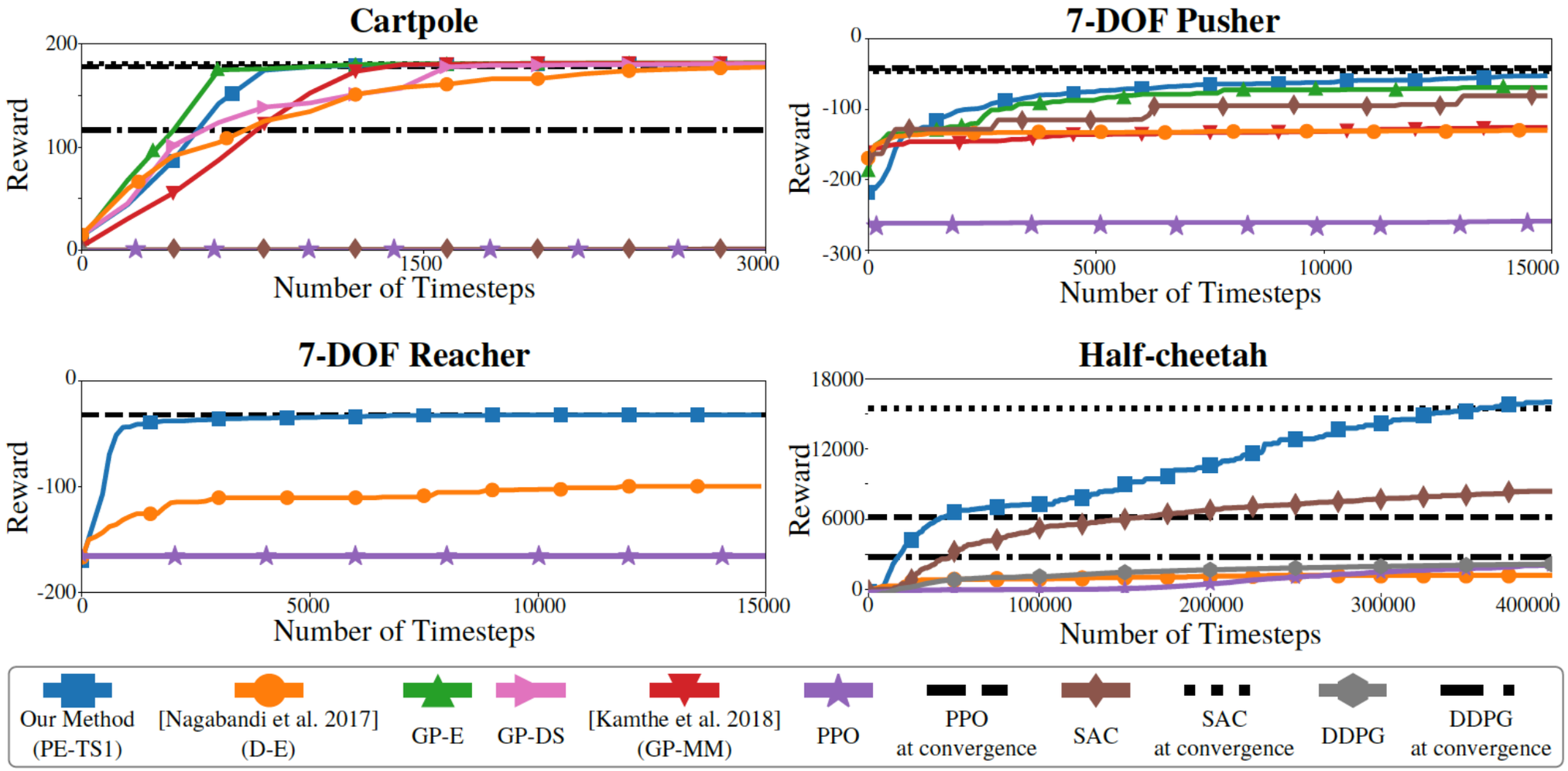
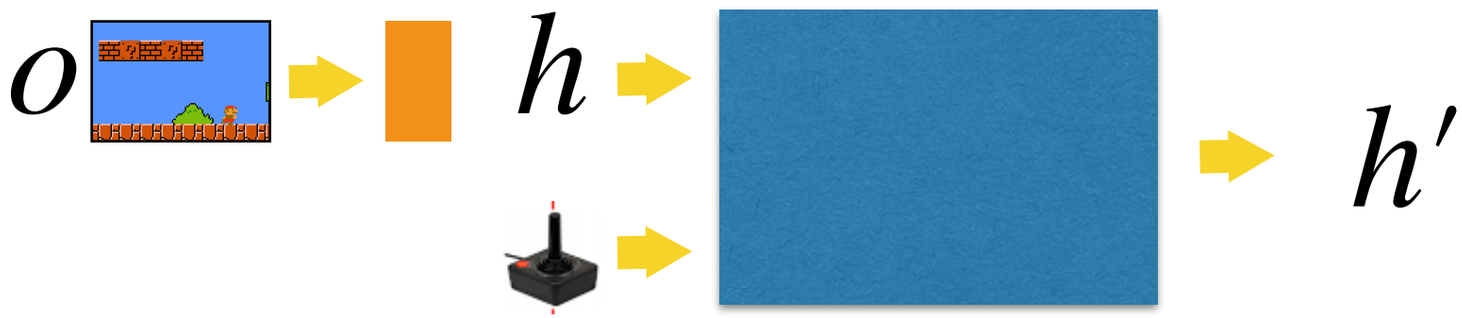


Figure 3: Learning curves for different tasks and algorithm. For all tasks, our algorithm learns in under 100K time steps or 100 trials. With the exception of Cartpole, which is sufficiently low-dimensional to efficiently learn a GP model, our proposed algorithm significantly outperform all other baselines.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

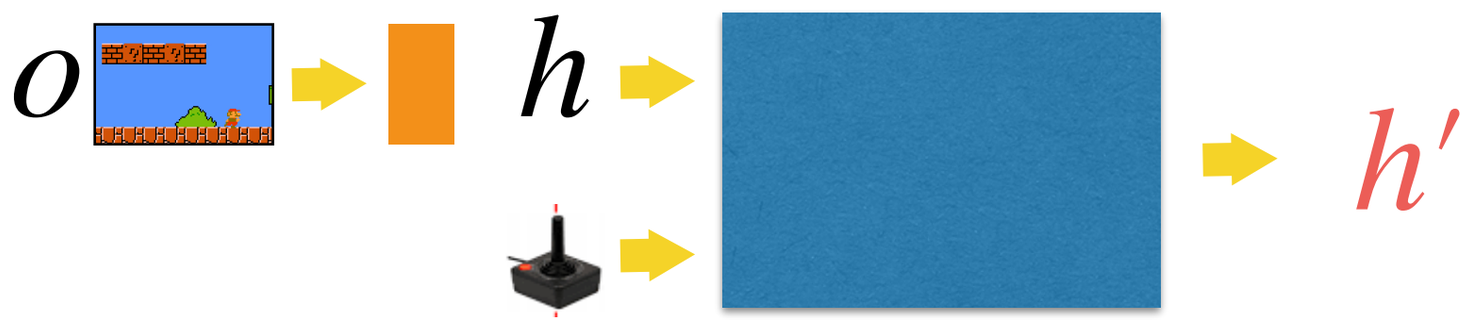
Model-based RL in sensory space

Model Learning - 3 Qs always in mind

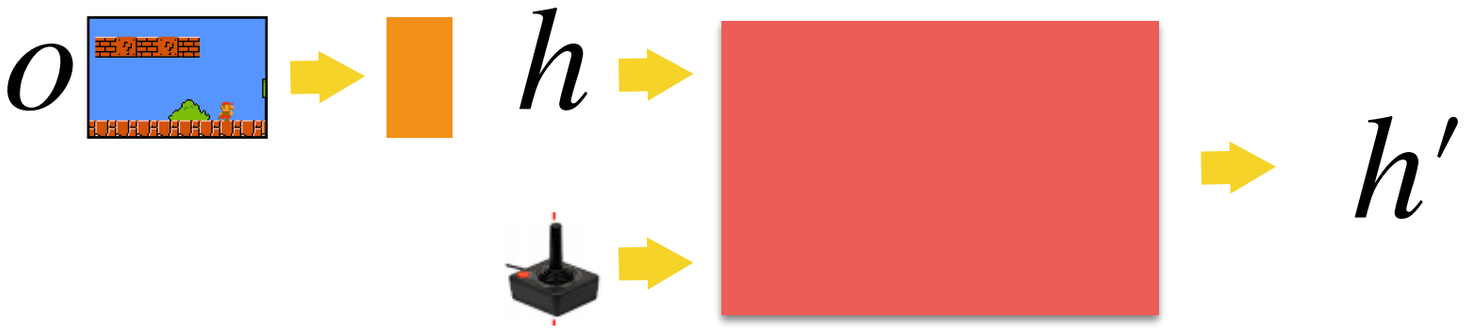


Model Learning - 3 Qs always in mind

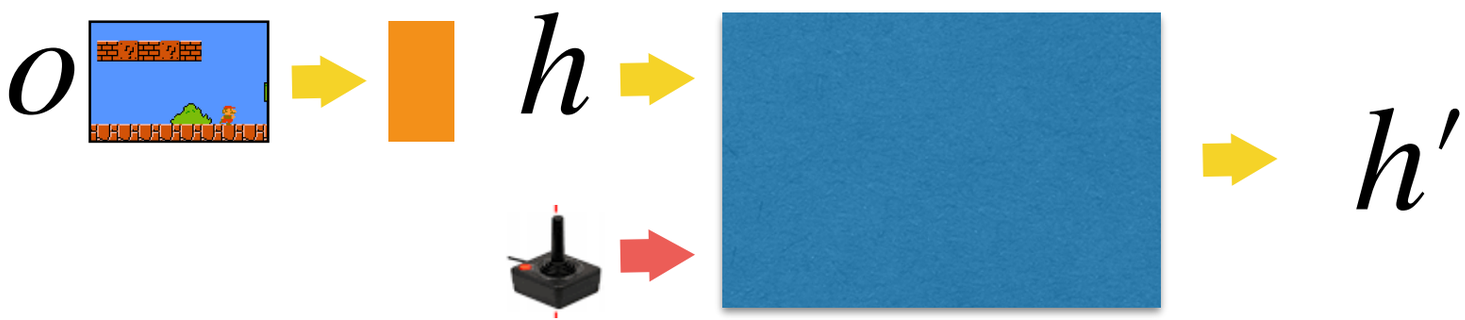
- What shall we be predicting?



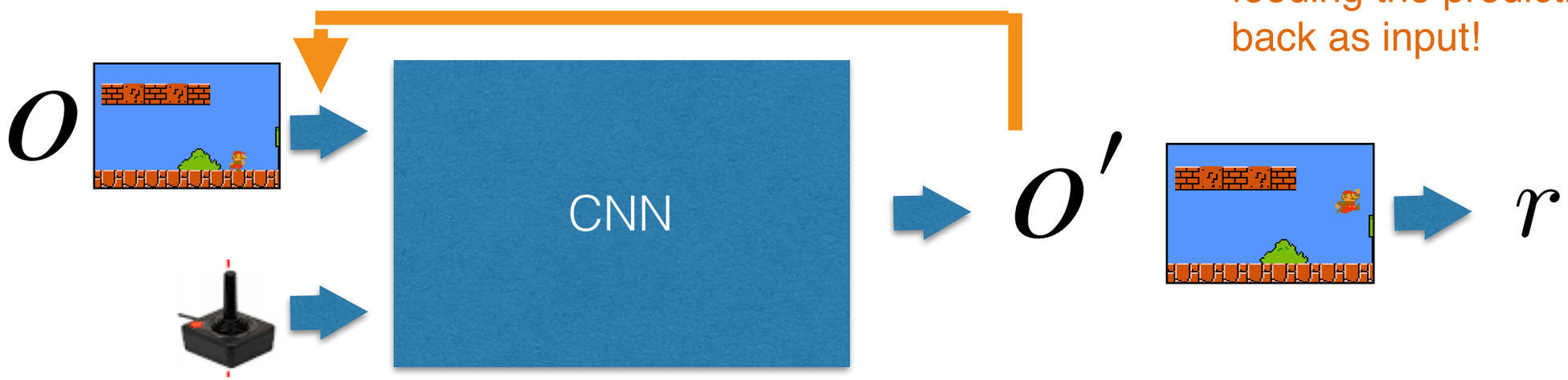
- What is the architecture of the model, what structural biases should we add to get it to generalize?



- What is the action representation?



Model learning in image space



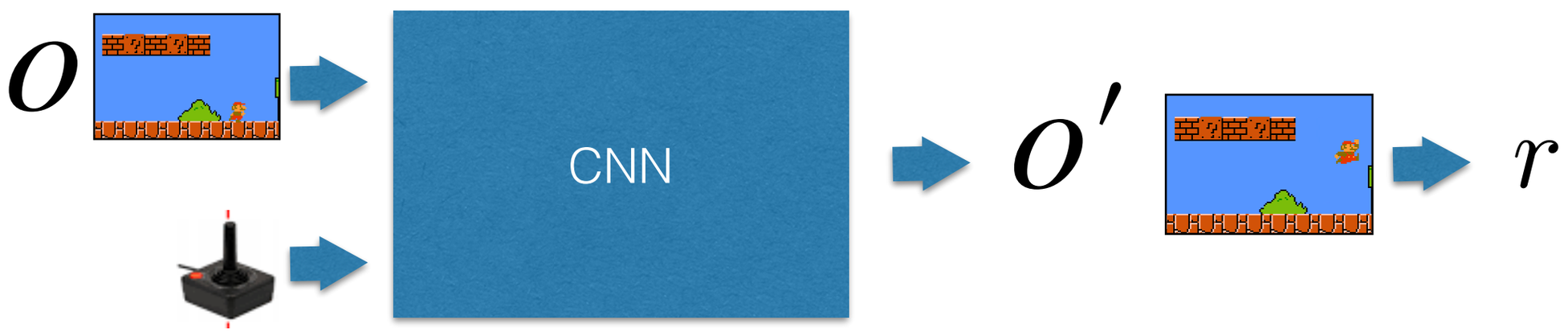
Unroll the model by feeding the prediction back as input!

MANY different rewards can be computed from the future visual observation, e.g., make Mario jump, make Mario move to the right, to the left, lie down, make Mario jump on the well and then jump back down again etc..

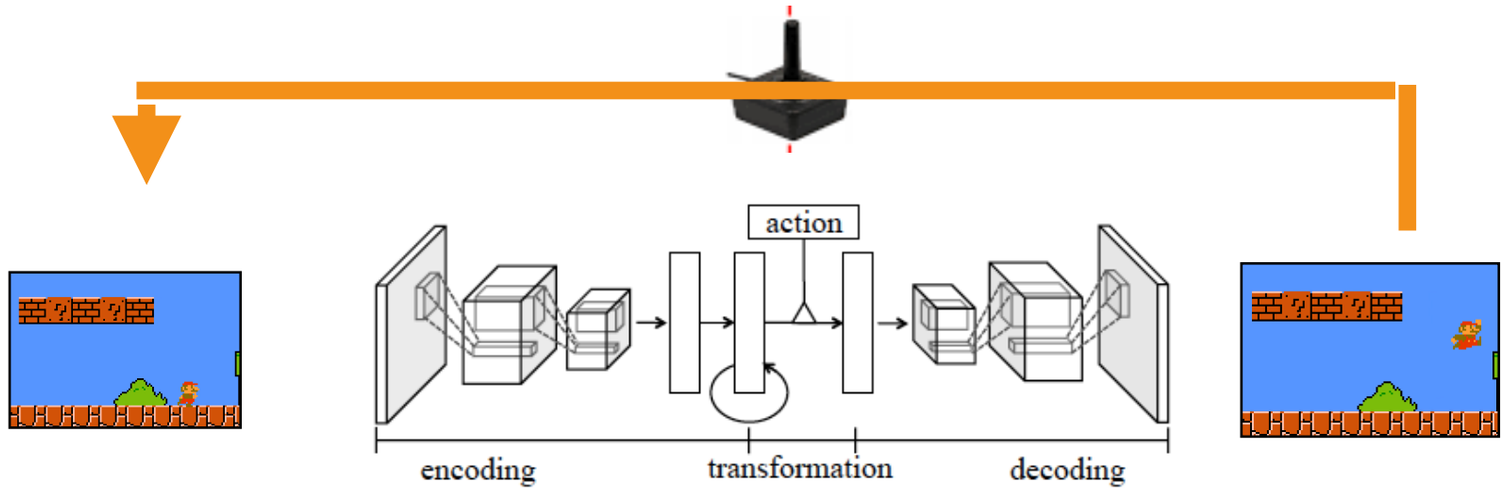
Action-Conditional Video Prediction using Deep Networks in Atari Games

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



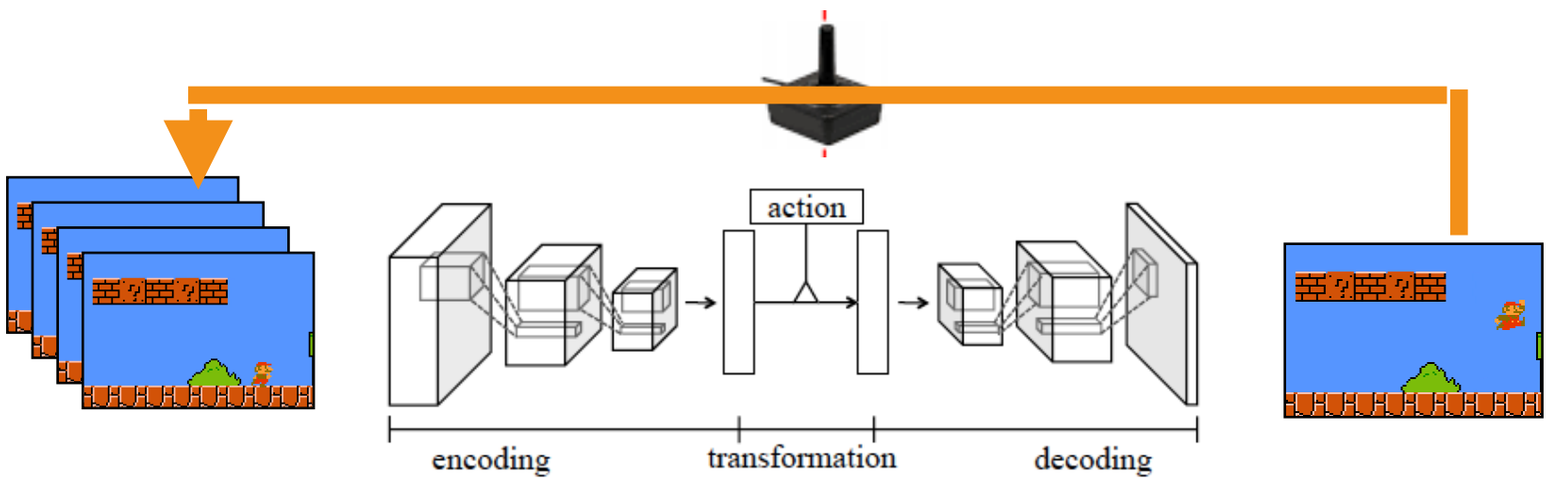
Unroll the model by feeding the prediction back as input!

Progressively increase k (the unrolling horizon) so that we do not feed garbage predictions as input to the predictive model:

$$\mathcal{L}_K(\theta) = \frac{1}{2K} \sum_i \sum_t \sum_{k=1}^{K} \left\| \hat{\mathbf{x}}_{t+k}^{(i)} - \mathbf{x}_{t+k}^{(i)} \right\|^2$$

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

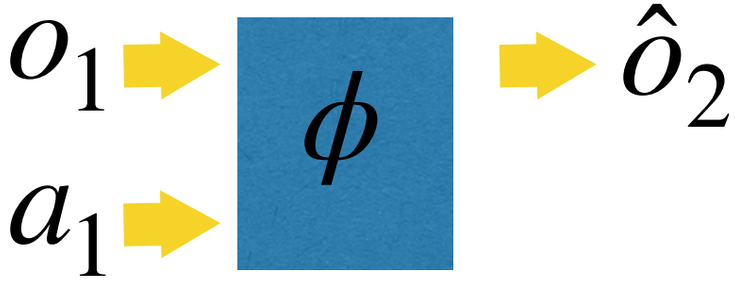
Minimizing error accumulation during unrolling



Q: Can I train my model using tupples (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

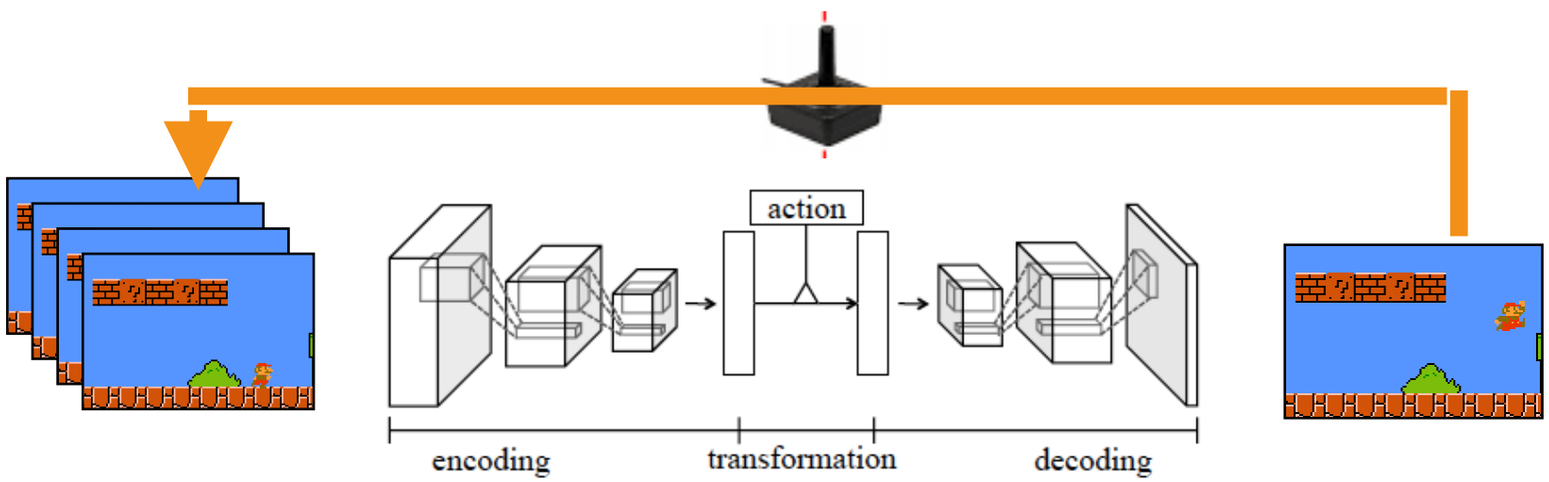
Solution: Progressively increase the unrolling horizon k at training time so that the model learns to handle its mistakes:

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



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

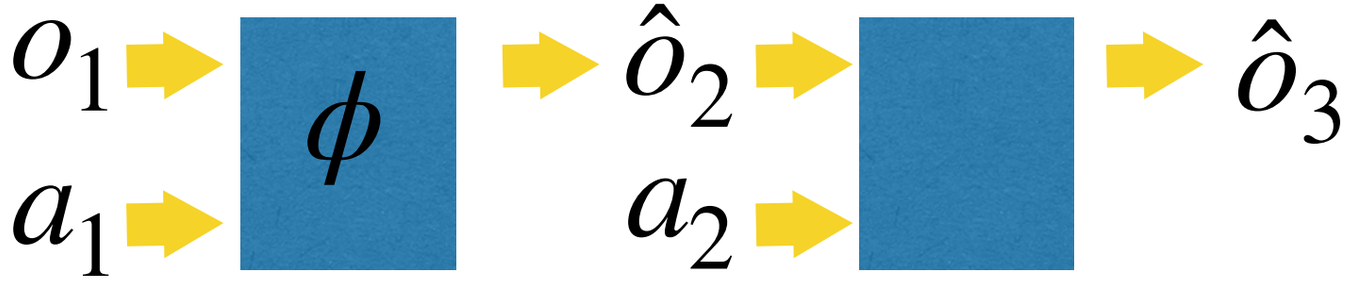
How to train our model so that unrolling works



Q: Can I train my model using tupples (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

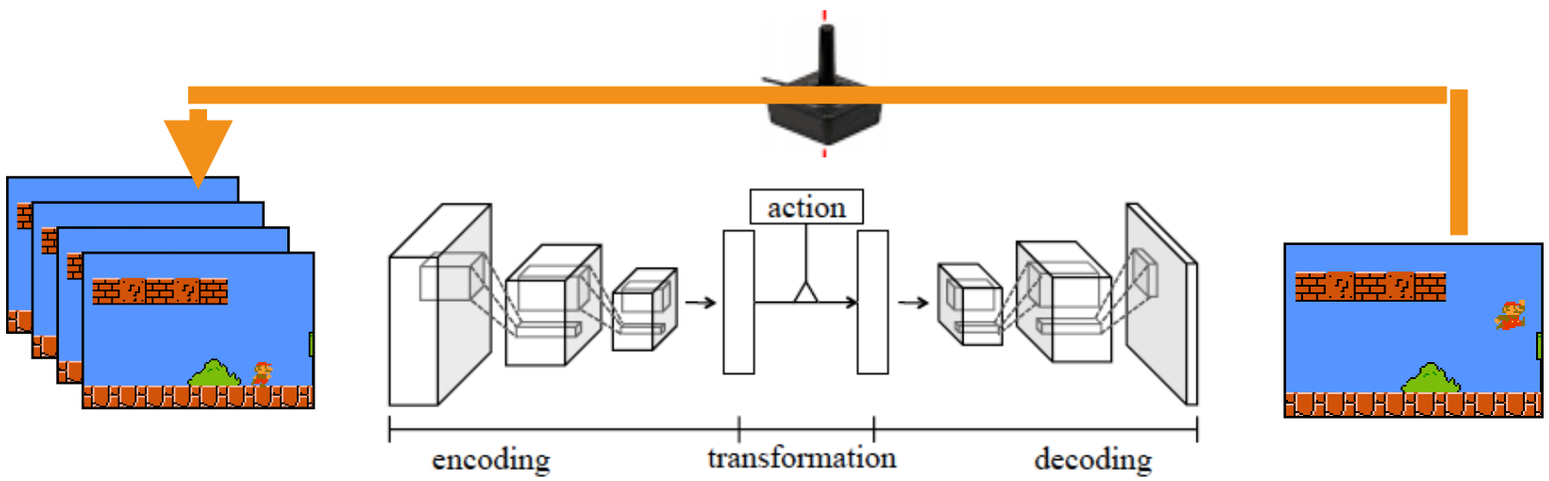
Solution: Progressively increase the unrolling horizon k at training time so that the model learns to handle its mistakes:

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



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

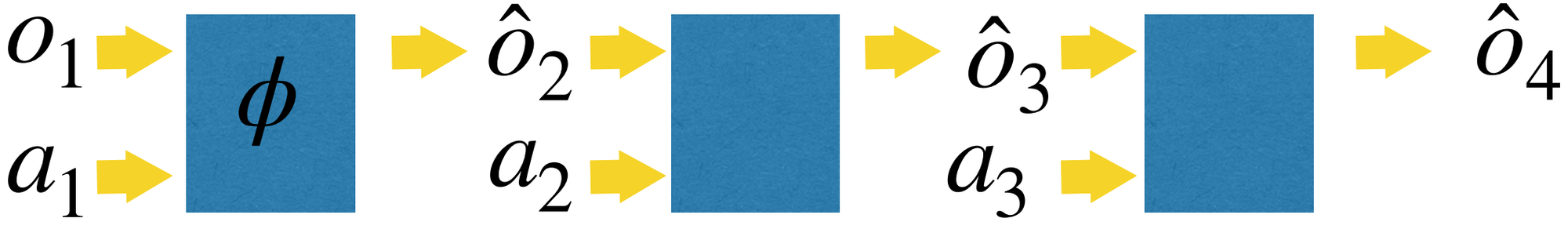
How to train our model so that unrolling works



Q: Can I train my model using tupples (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?

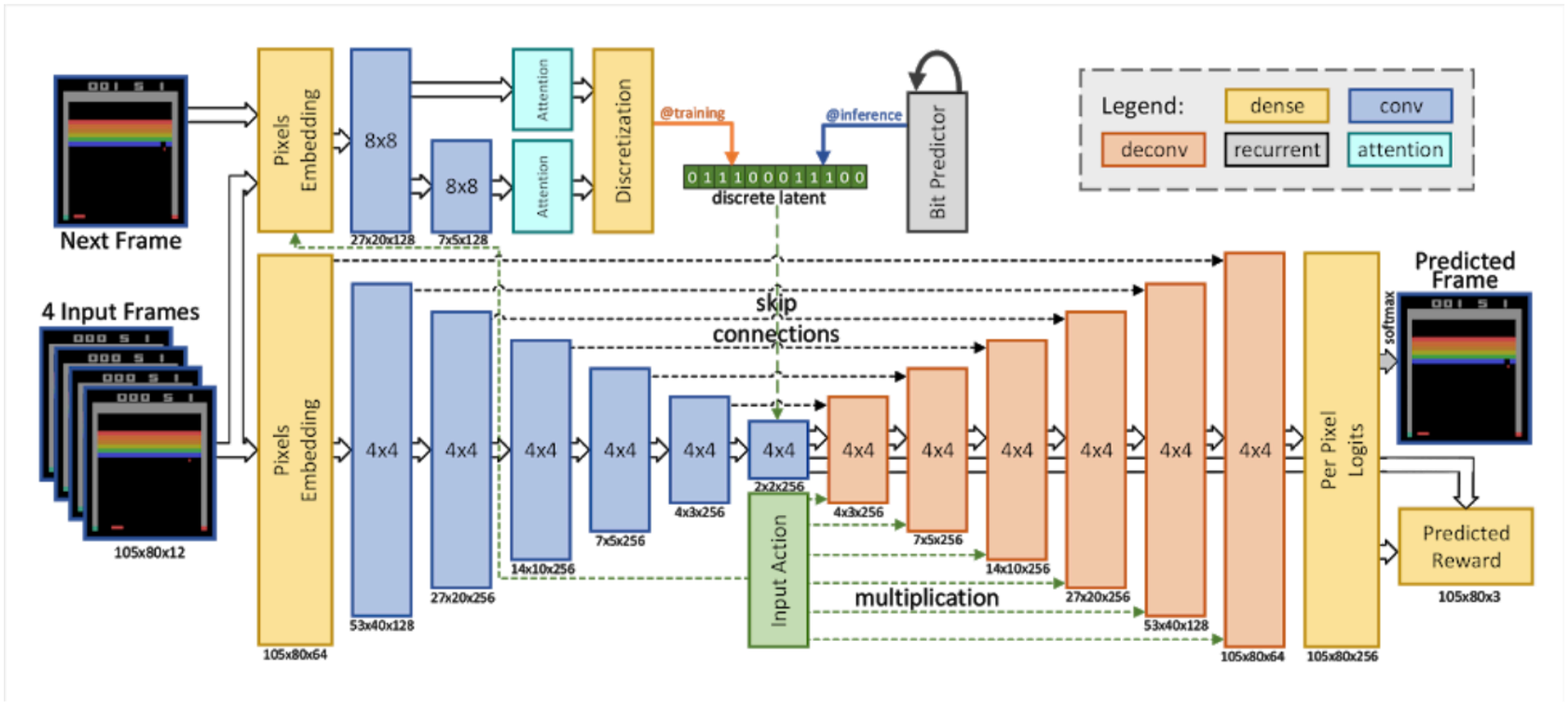
A:They induce a tiny mean pixel prediction loss (despite the fact they may be very task-relevant!)

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

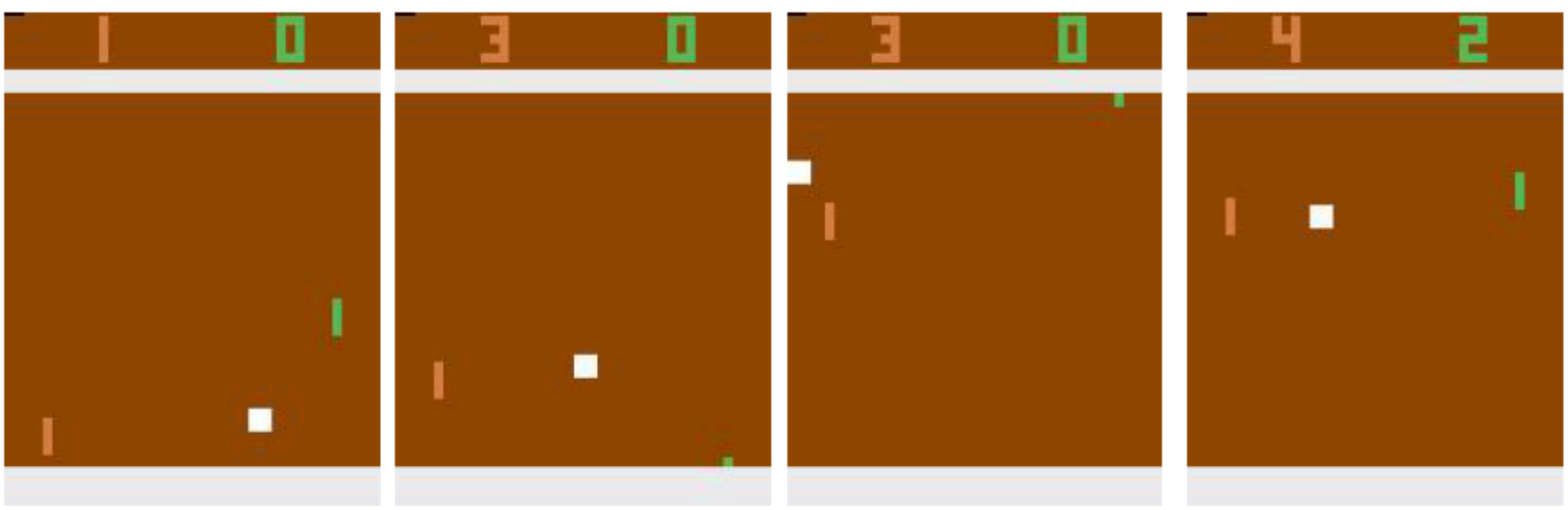
Łukasz Kaiser **Ryan Sepassi** Google Brain
Henryk Michalewski **Piotr Miłoś** University of Warsaw
Błażej Osiński deepsense.ai

Similar architecture as before but..



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

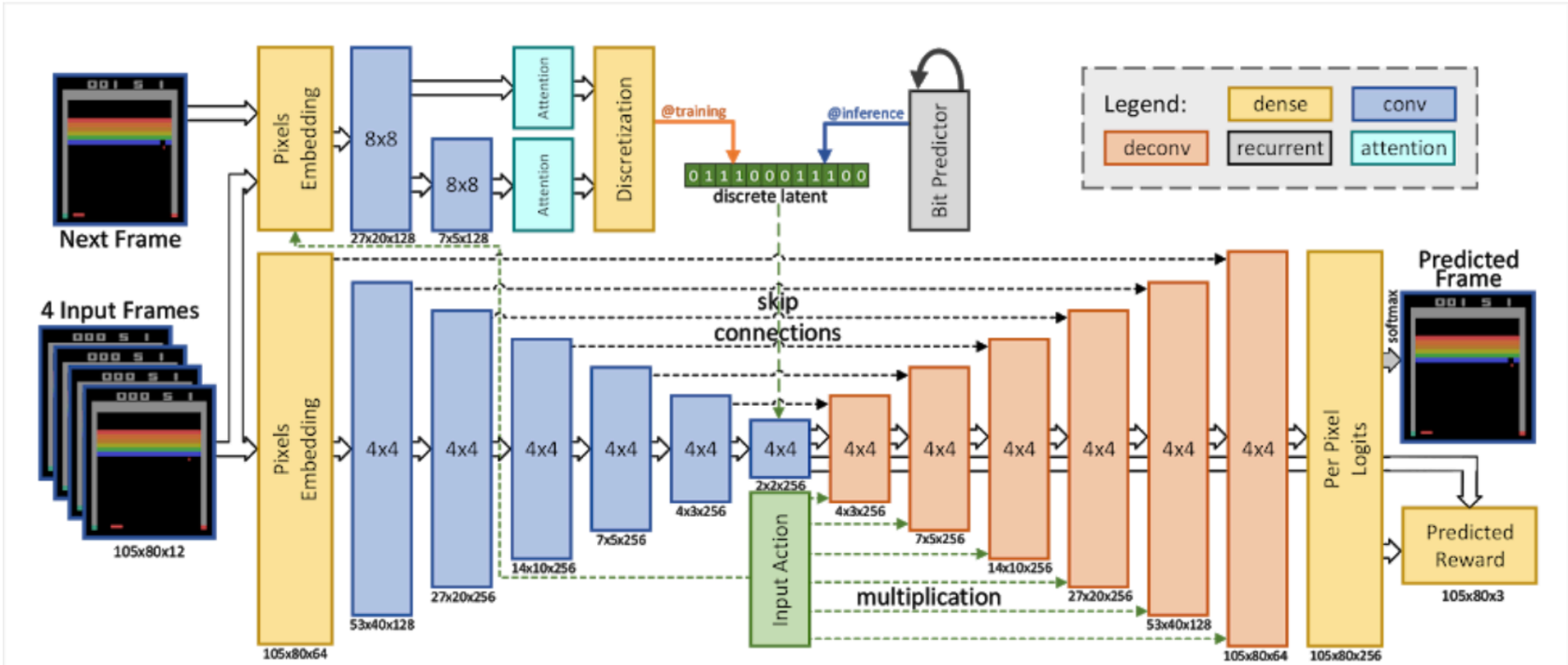
Results

- Number of frames required to reach human performance

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

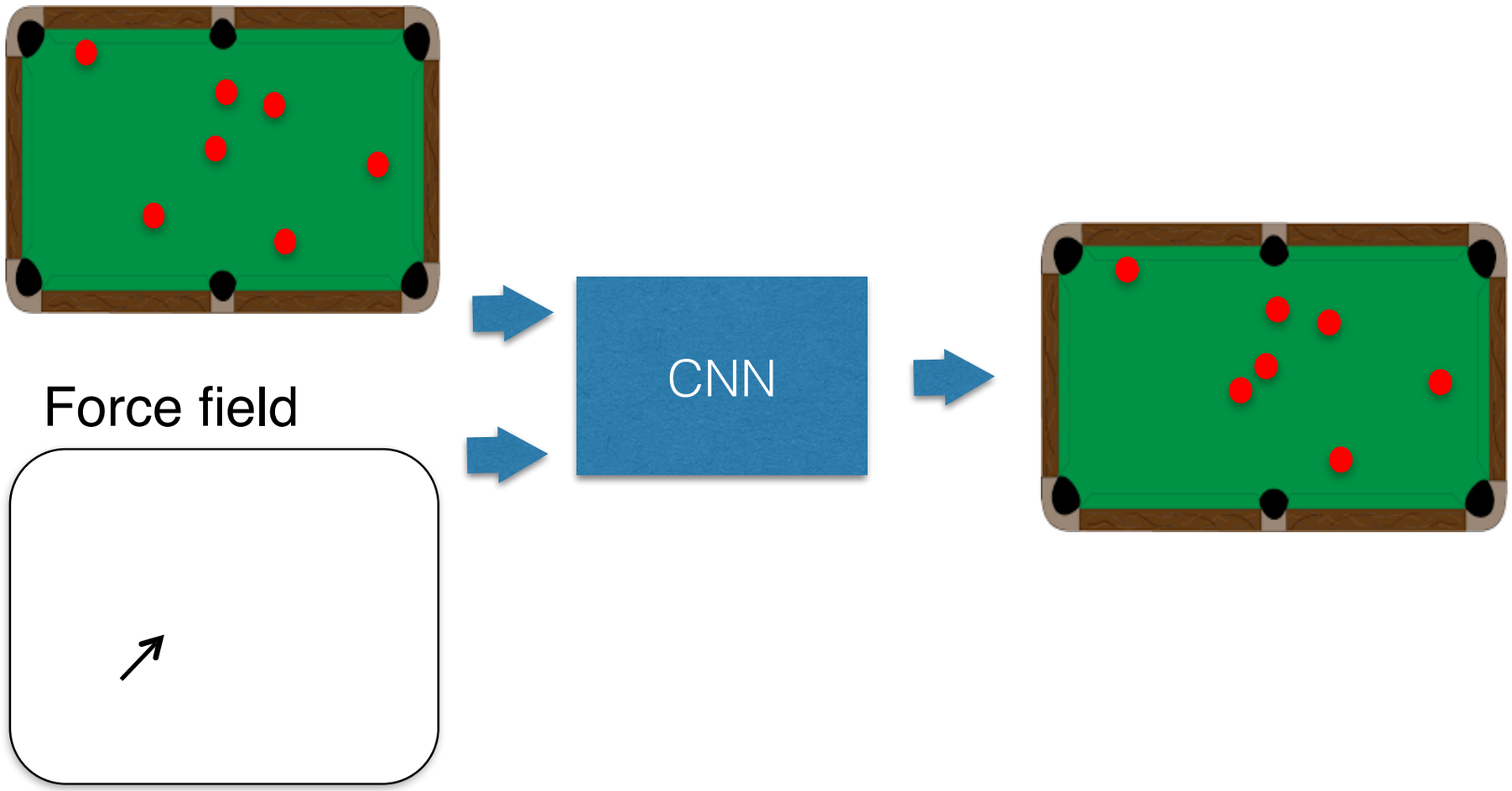
results

Generalization in convolutional frame prediction



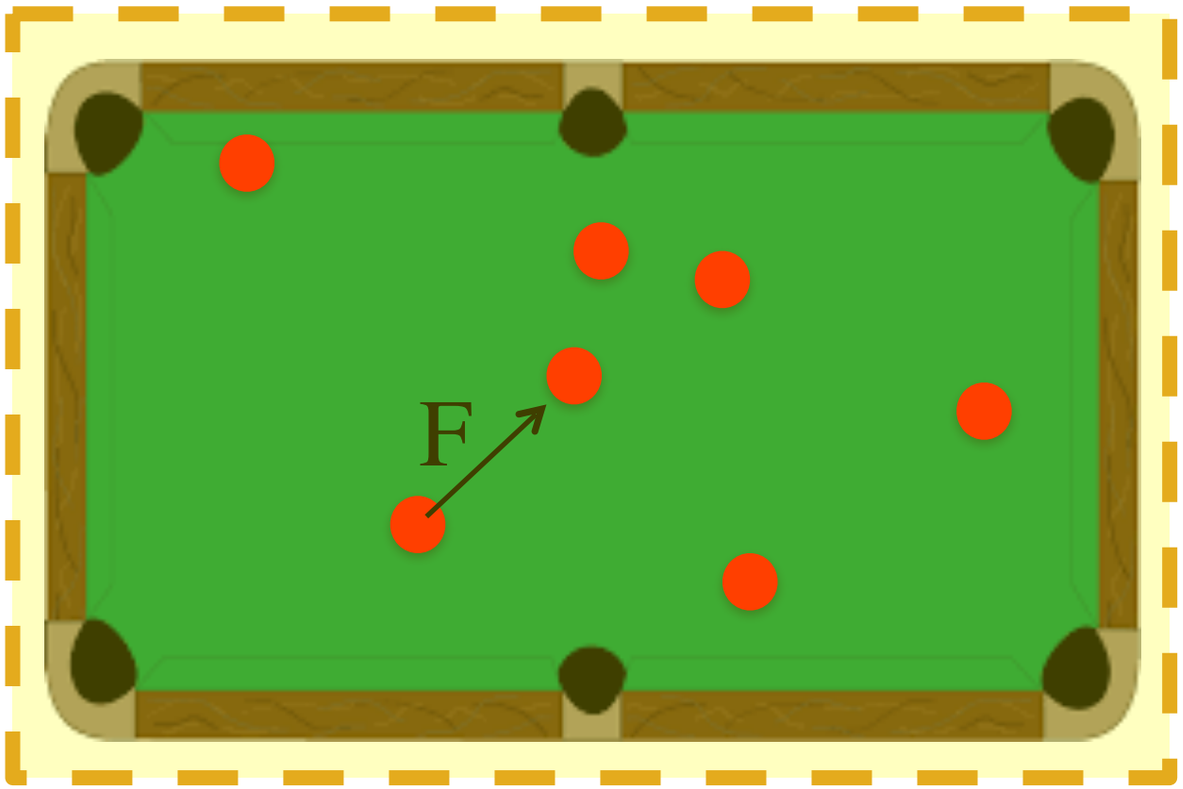
- Can the model generalize to scenes with different number of objects?

Learning Action-Conditioned Billiard Dynamics

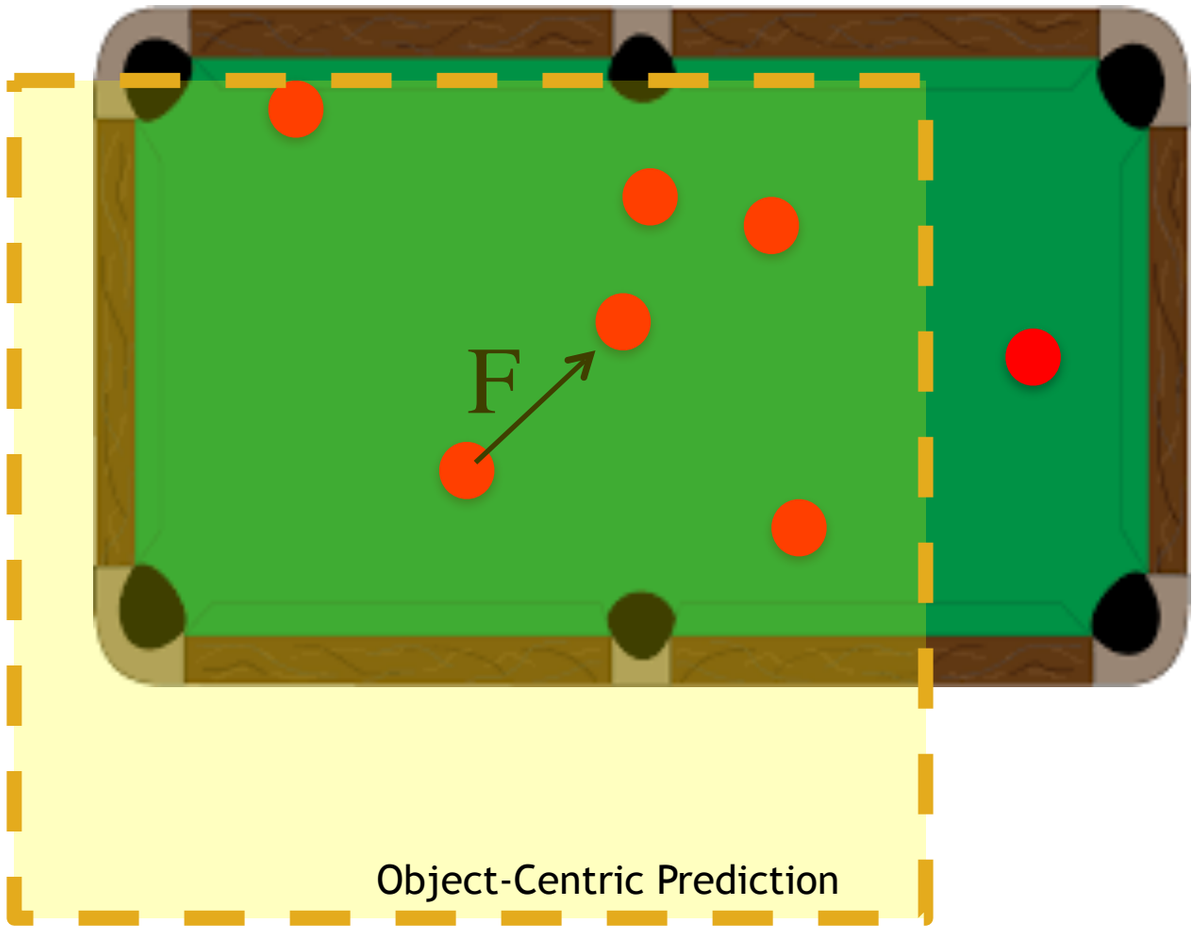


Q: Will our model be able to generalize across different number of balls present?

Object-centric dynamics

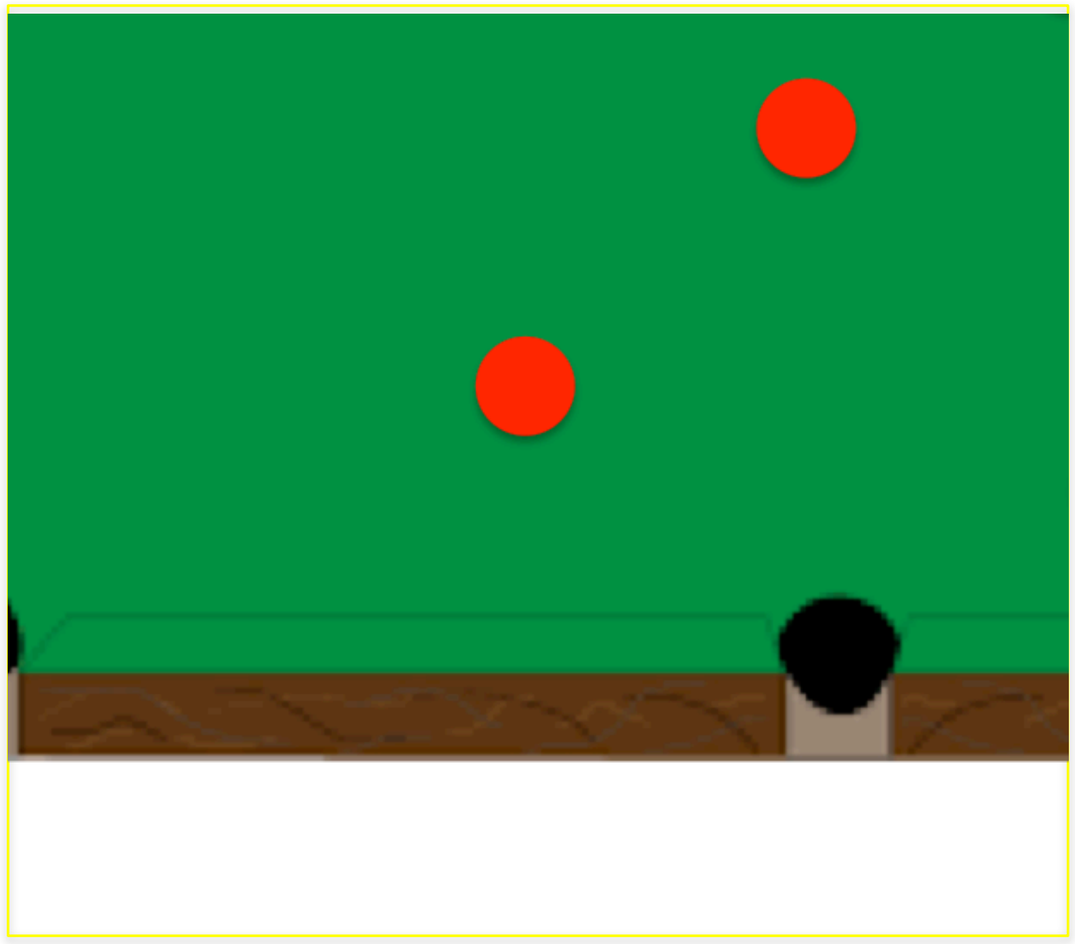


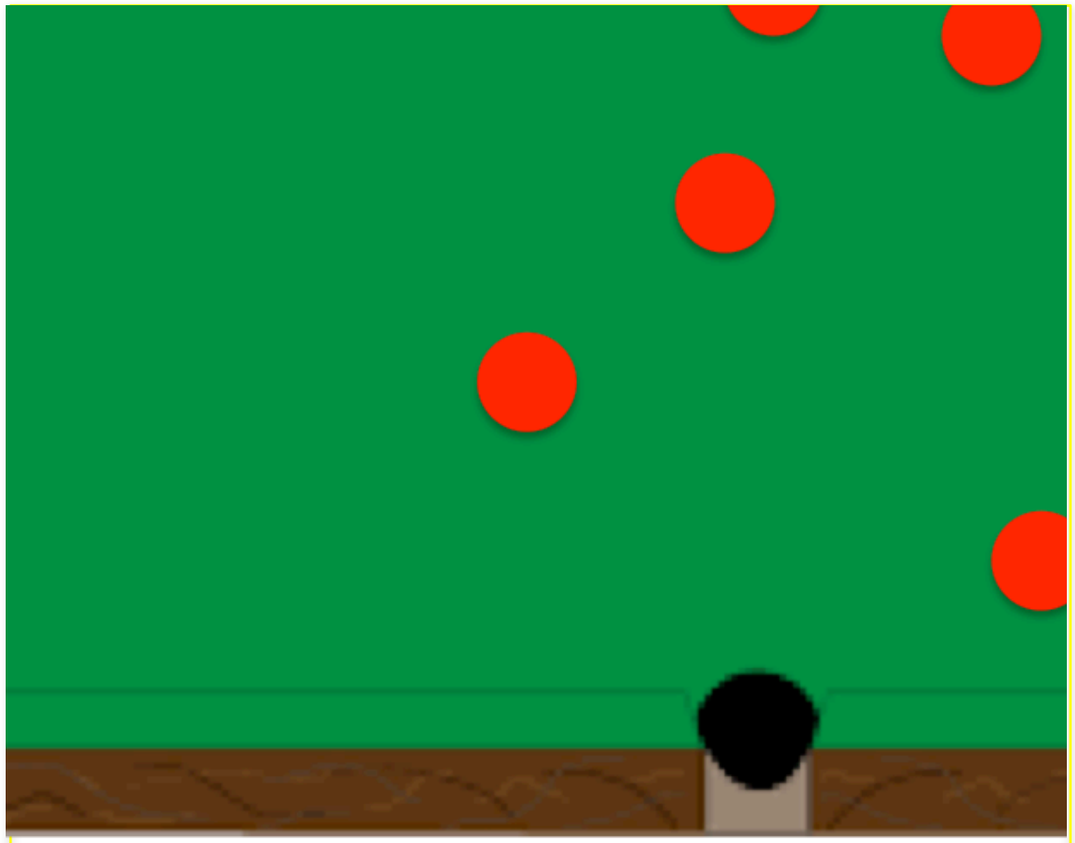
World-Centric Prediction

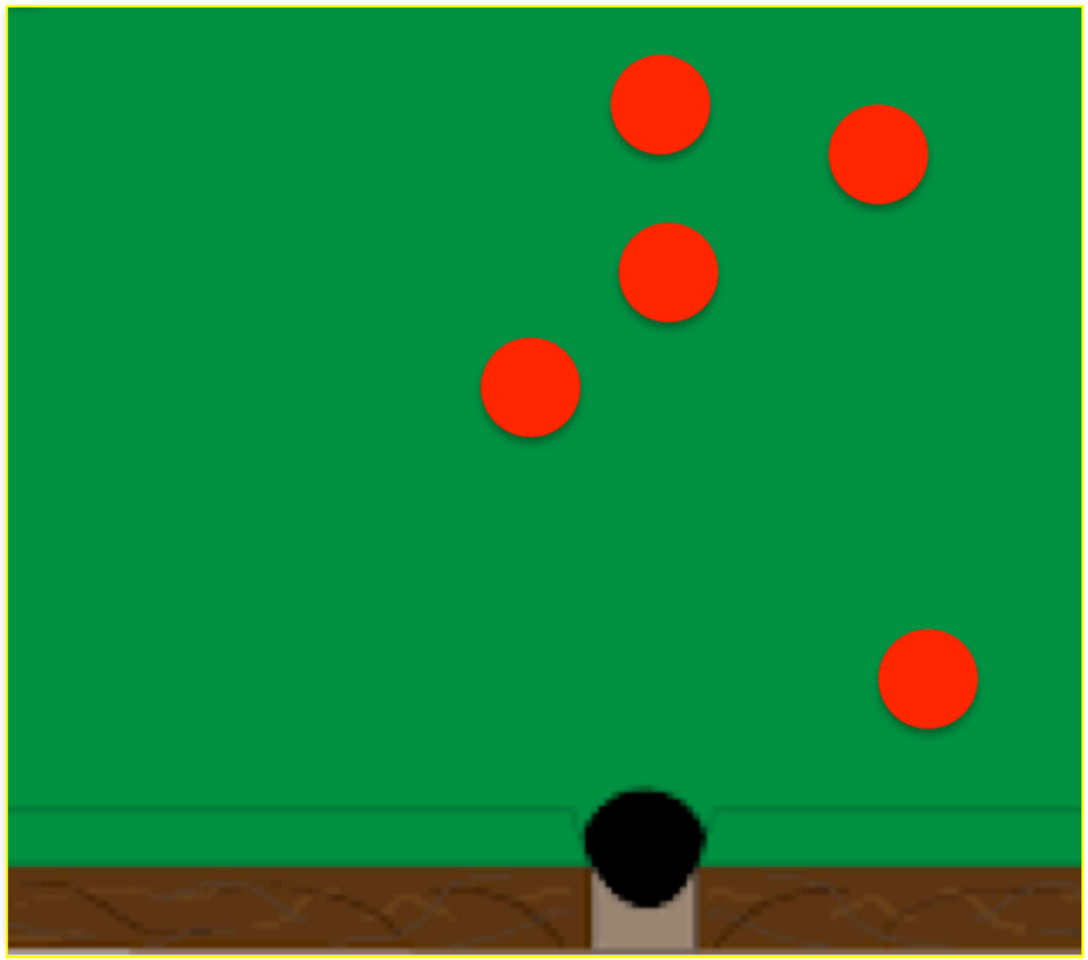


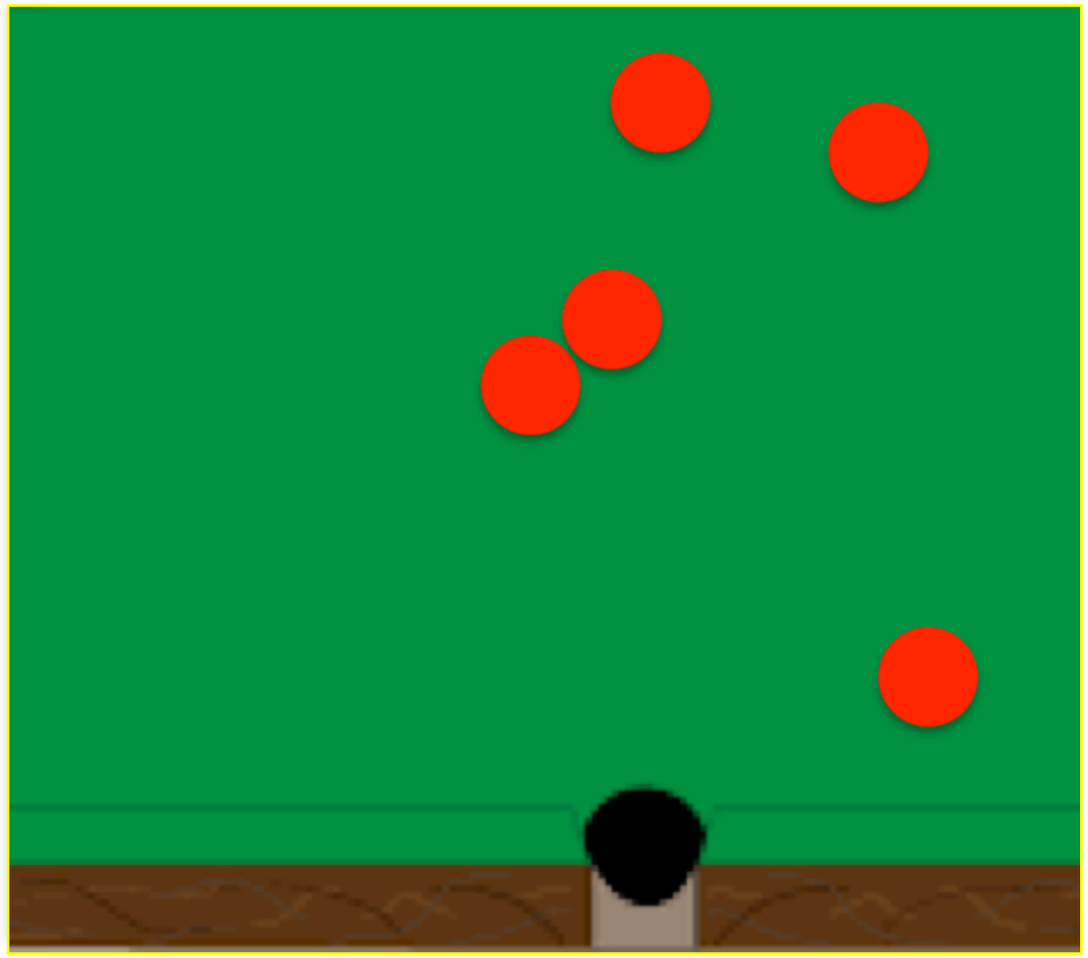
Object-Centric Prediction

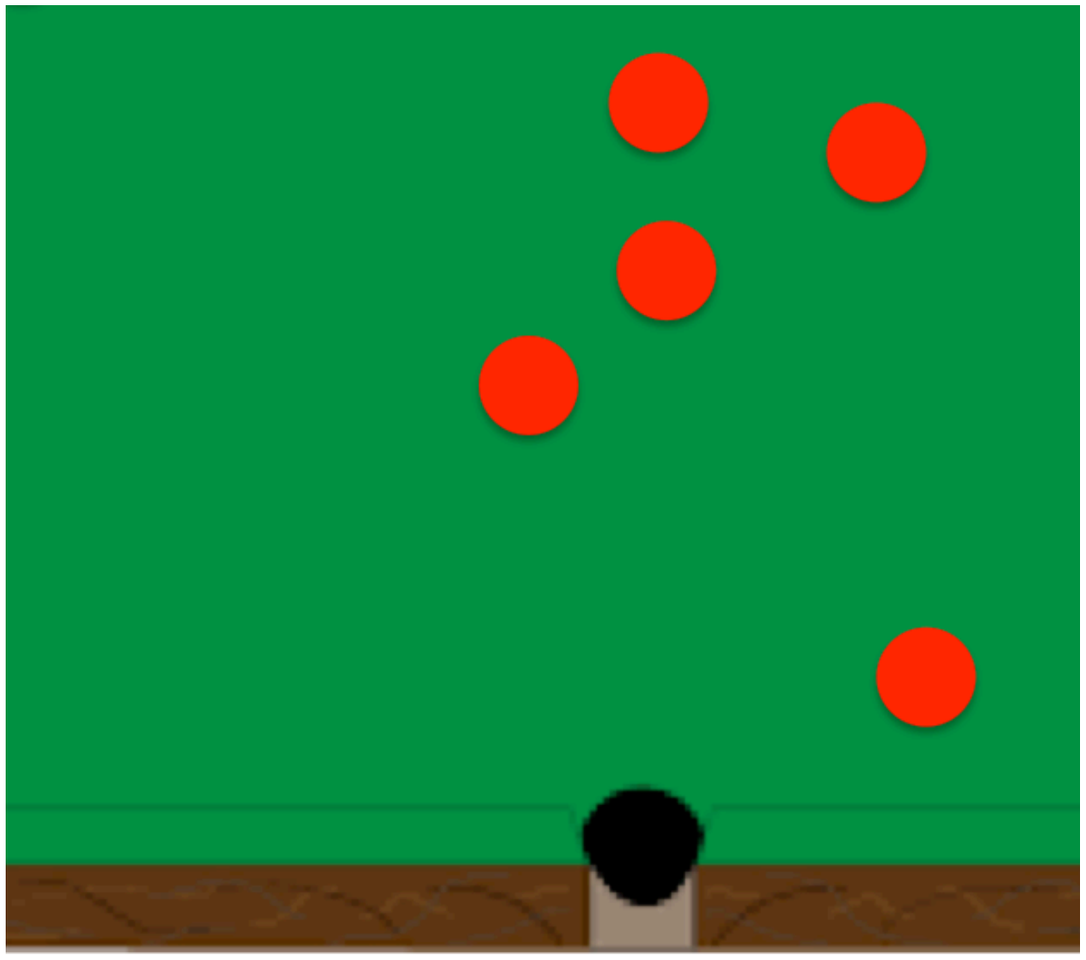
Q: Will our model be able to generalize across different number of balls present?

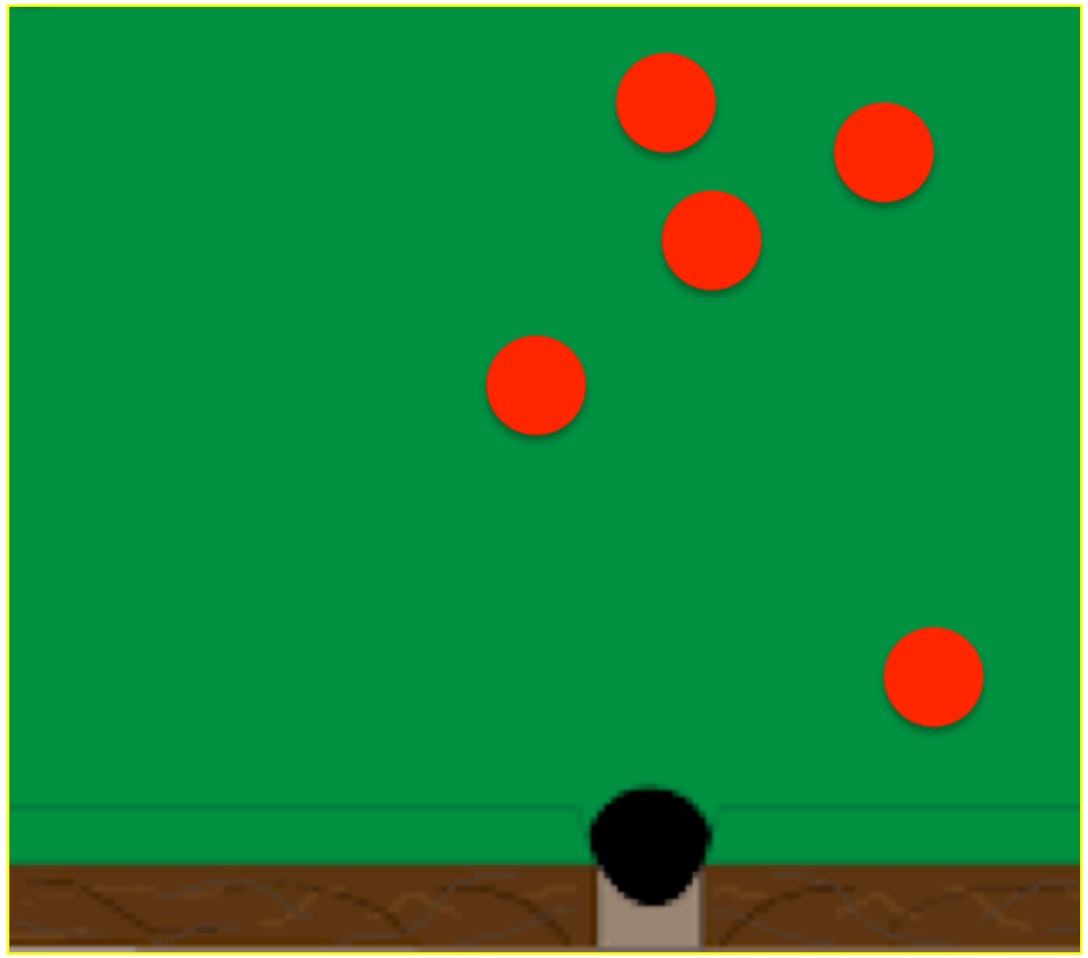




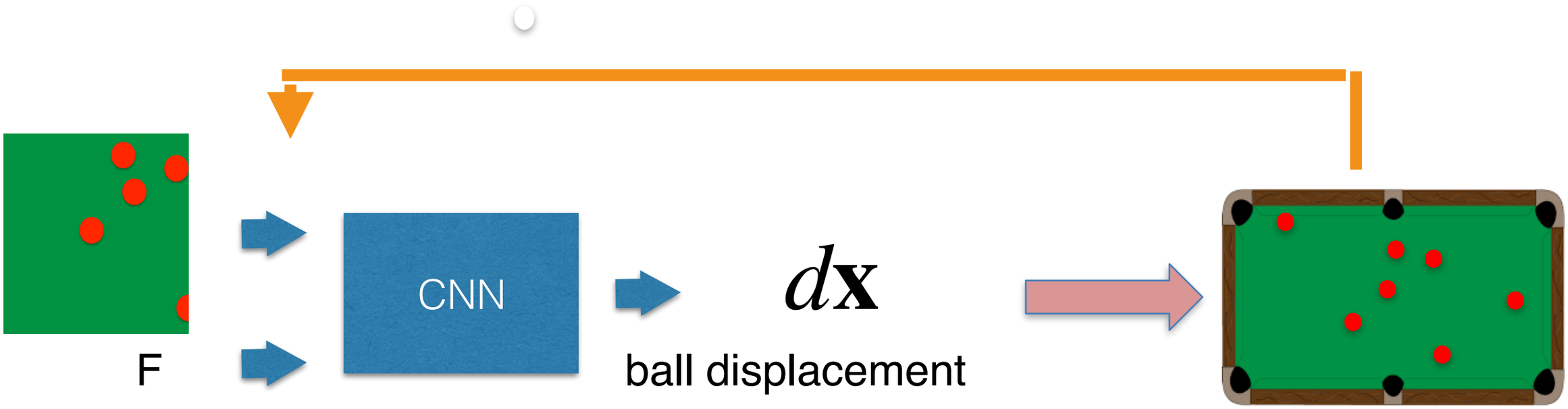




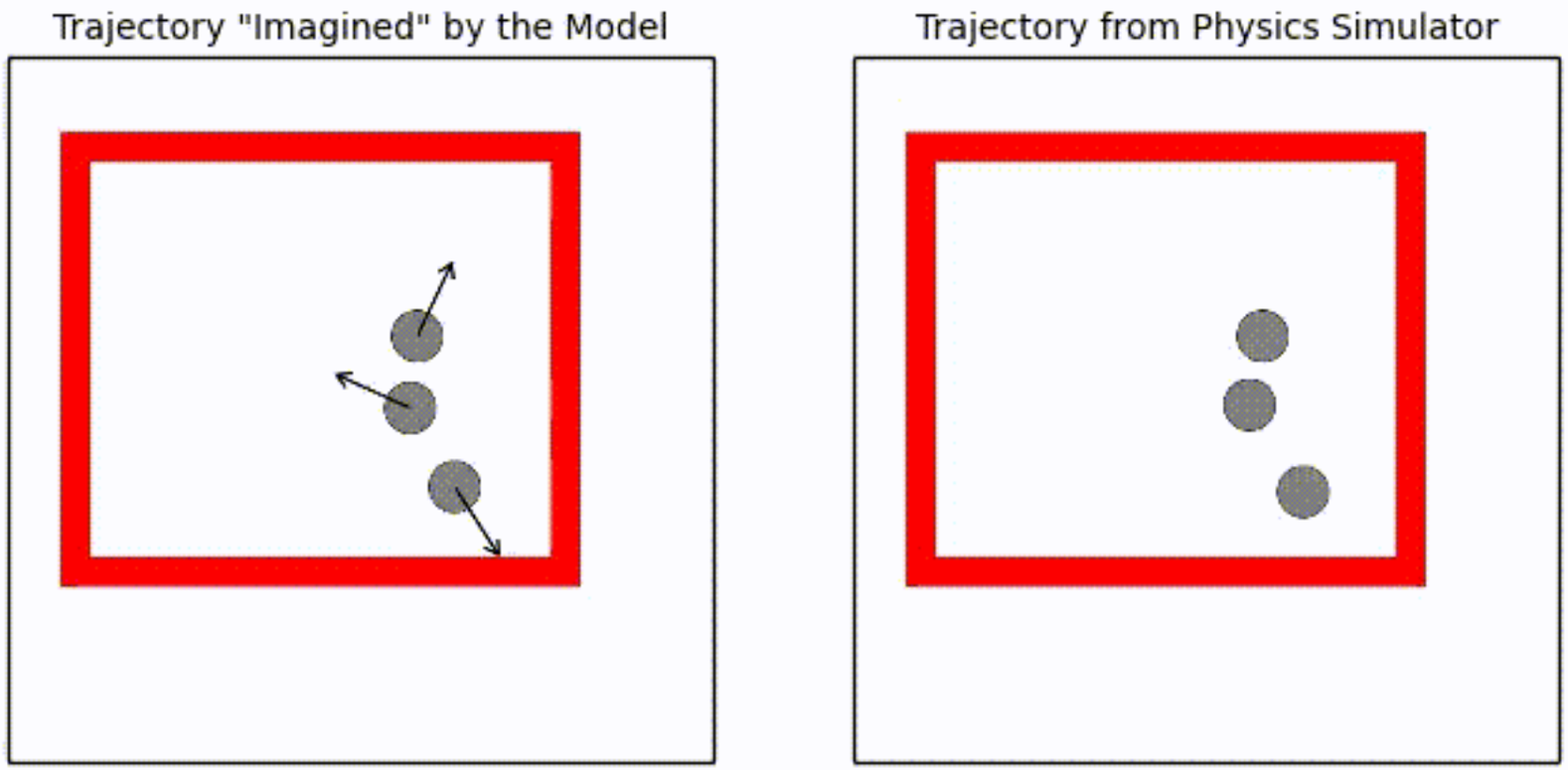


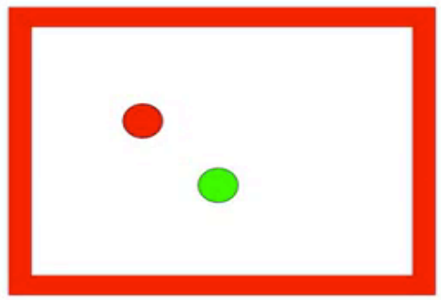


Object-centric Billiard Dynamics



- The object-centric CNN is shared across all objects in the scene.
- We apply it one object at a time to predict the object's future displacement.
- We then copy paste the ball at the predicted location, and feed back as input.


Trajectory "Imagined" by the Model
Trajectory from Physics Simulator



How should I push the red ball so that it collides with the green on?
CEM for searching across forces

Learning Dynamics

Two useful ideas:

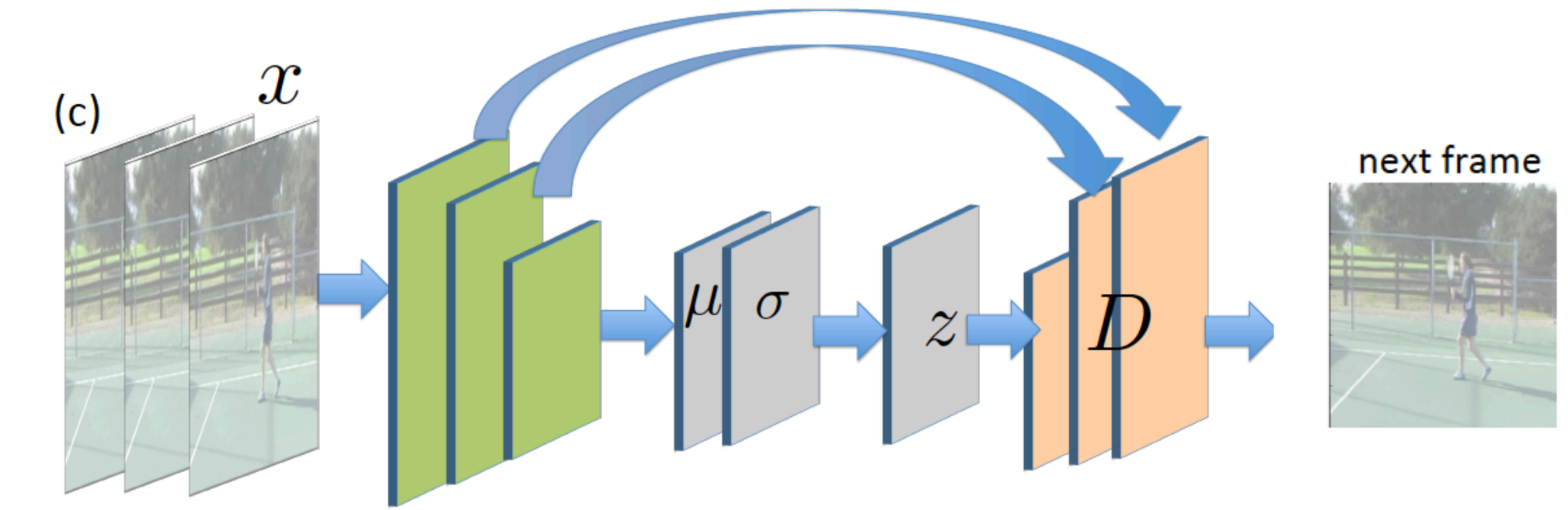
1) Object-centric observations. Such encoding allows to generalize across different number of entities in the scene.
2) Predict motion instead of appearance. Since appearance does not change, predicting motion suffices. Let's predict only the dynamic properties and keep the static ones fixed.

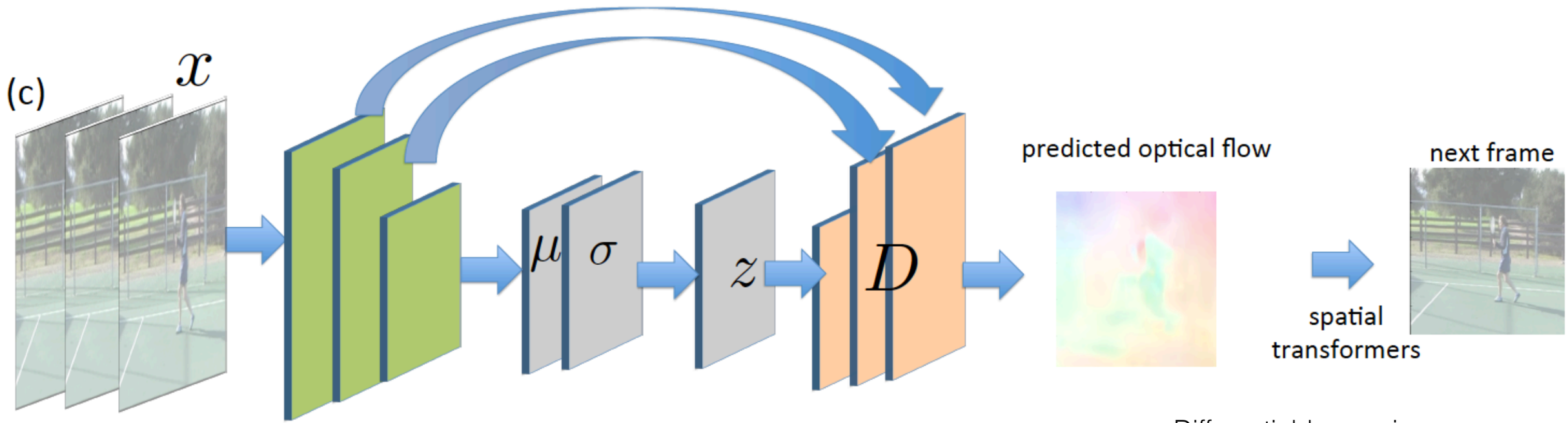
Learning Dynamics

Two good ideas so far:

1) object graphs instead of images. Such encoding allows to generalize across different number of entities in the scene.
2) predict motion instead of appearance. Since appearance does not change, predicting motion suffices. Let's predict only the dynamic properties and keep the static one fixed.

Visual dynamics using motion transformation





Differentiable warping

Visual dynamics using motion transformation

green: input, red: sampled future motion field and corresponding frame completion

What should we be predicting?

Do we really need to be predicting observations?

What if we knew what are the quantities that matter for the goals i care about?
For example, I care to predict where the object will end up during pushing but I do not care exactly where it will end up when it falls off the table, or I do not care about its intensity changes due to lighting.

Let’s assume we knew this set of important to predict features. Would we do better?
Yes! we would win the competition in Doom the minimum.

LEARNING TO ACT BY PREDICTING THE FUTURE

Alexey Dosovitskiy
Intel Labs

Vladlen Koltun
Intel Labs

Main idea: You are provided with a set of measurements m paired with input visual (and other sensory) observations.
Measurements can be health, ammunition levels, enemies killed.
Your goal can be expressed as a combination of those measurements.

measurement offsets for the next τ_n frames are the prediction targets: $\mathbf{f} = (\mathbf{m}_{t+\tau_1} - \mathbf{m}_t, \cdots, \mathbf{m}_{t+\tau_n} - \mathbf{m}_t)$

(multi) goal representation: $u(\mathbf{f}, \mathbf{g}) = \mathbf{g}^\top \mathbf{f}$

LEARNING TO ACT BY PREDICTING THE FUTURE

Alexey Dosovitskiy
Intel Labs

Vladlen Koltun
Intel Labs

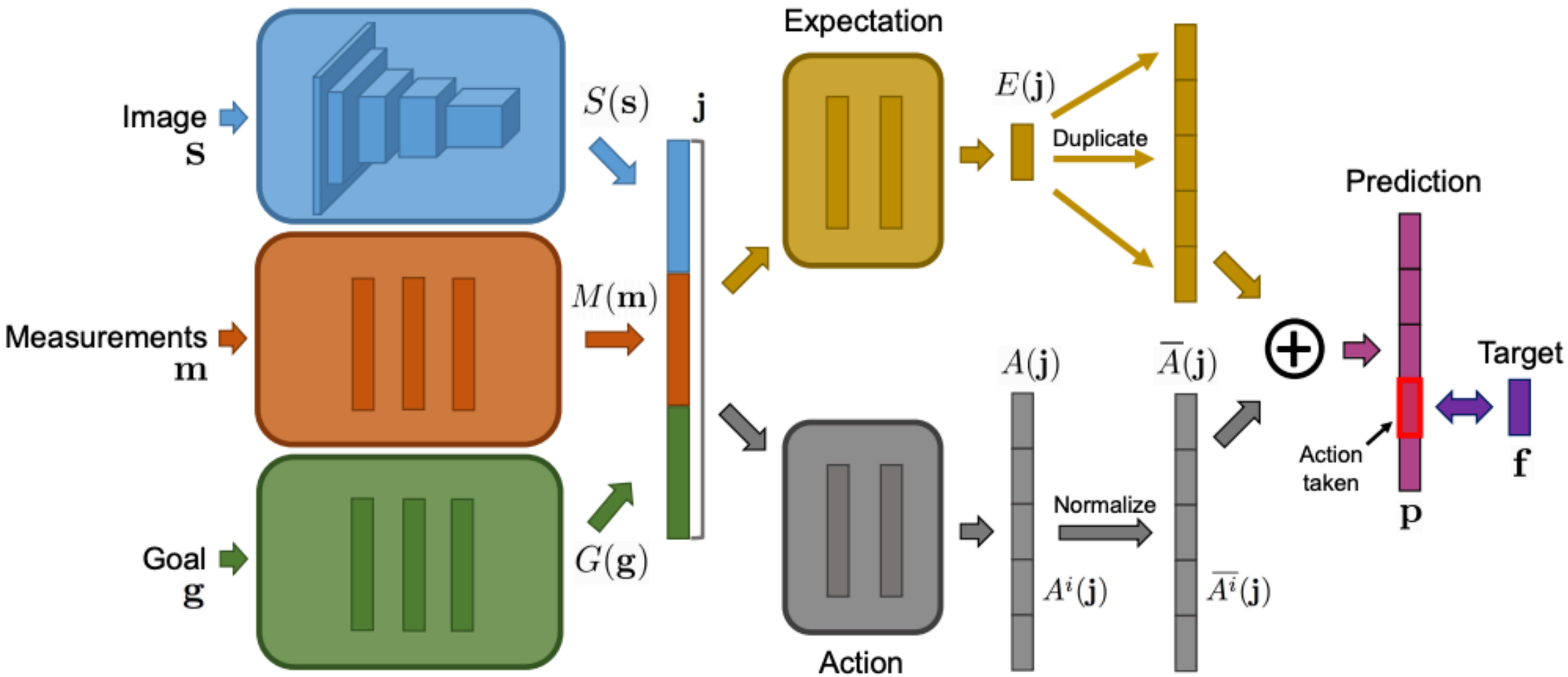
Train a deep predictor. No unrolling! One shot prediction of future values:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \left\| F(\mathbf{o}_i, a_i, \mathbf{g}_i; \boldsymbol{\theta}) - \mathbf{f}_i \right\|^2$$

No policy, direct action selection:

$$a_t = \arg\max_{a \in \mathcal{A}} \mathbf{g}^\top F(\mathbf{o}_t, a, \mathbf{g}; \boldsymbol{\theta})$$

Learning dynamics of goal-related measurements

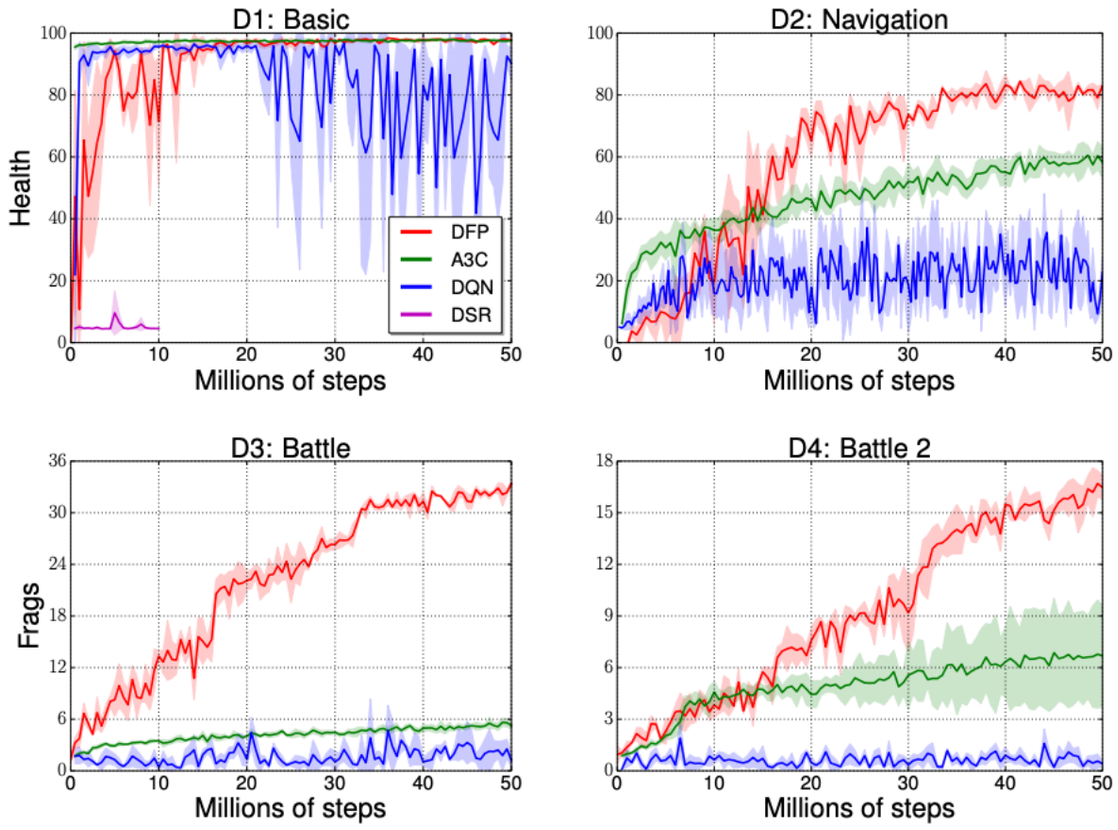


Action selection:

$$a_t = \arg\max_{a \in \mathcal{A}} \mathbf{g}^\top F(\mathbf{o}_t, a, \mathbf{g}; \boldsymbol{\theta})$$

Training: we learn the model using \epsilon-greedy exploration policy over the current best chosen actions.

Learning dynamics of goal-related measurements



	D1 (health)	D2 (health)	D3 (frags)	D4 (frags)	steps/day
DQN	89.1 ± 6.4	25.4 ± 7.8	1.2 ± 0.8	0.4 ± 0.2	7M
A3C	$\mathbf{97.5 \pm 0.1}$	59.3 ± 2.0	5.6 ± 0.2	6.7 ± 2.9	80M
DSR	4.6 ± 0.1	–	–	–	1M
DFP	$\mathbf{97.7 \pm 0.4}$	$\mathbf{84.1 \pm 0.6}$	$\mathbf{33.5 \pm 0.4}$	$\mathbf{16.5 \pm 1.1}$	70M

Table 1: Comparison to prior work. We report average health at the end of an episode for scenarios D1 and D2, and average frags at the end of an episode for scenarios D3 and D4.

Learning dynamics of goal-related measurements

Learning to Act by Predicting the Future

Alexey Dosovitskiy Vladlen Koltun