Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

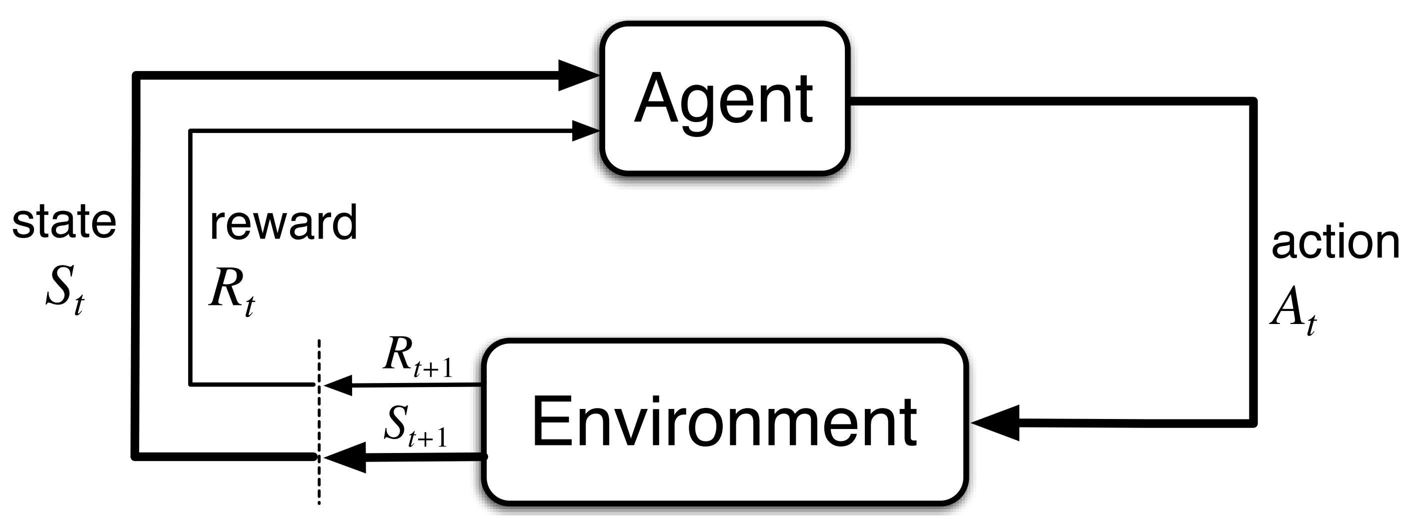
Model Based Reinforcement Learning I

Katerina Fragkiadaki

Planning

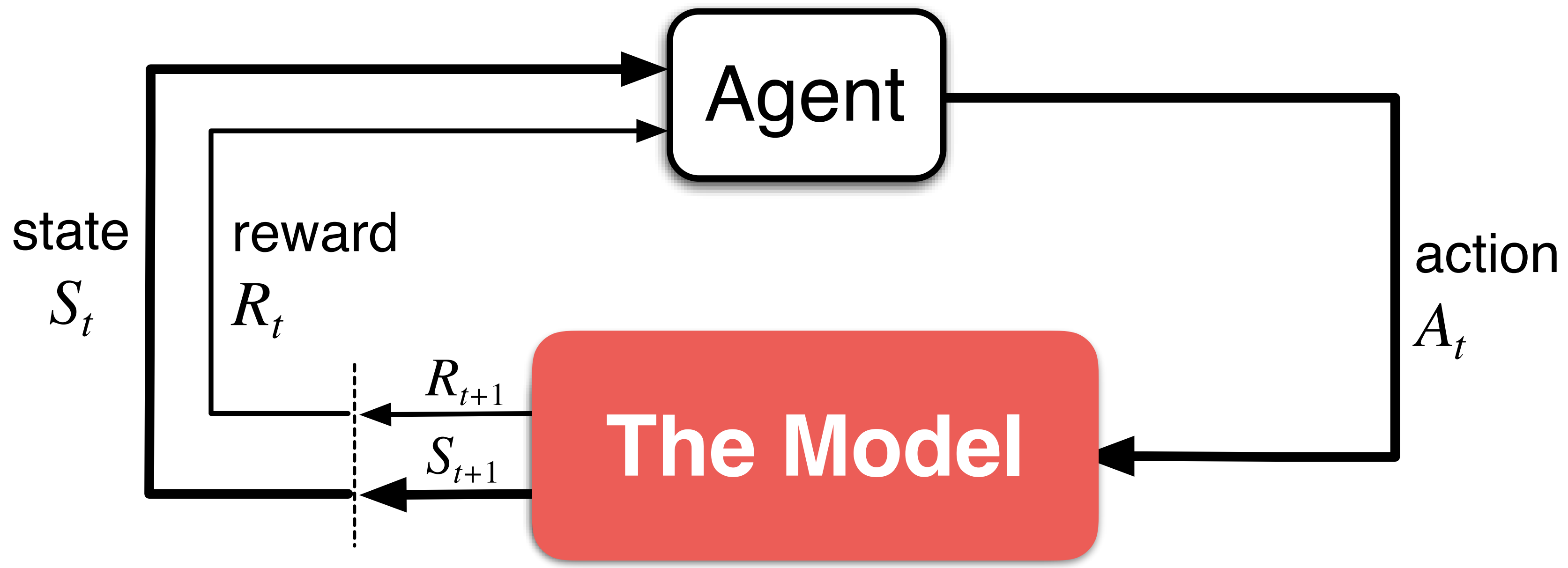
Planning: any computational process that uses a model to create or improve a policy

$$\text{Model} \xrightarrow{\text{Planning}} \text{Policy}$$



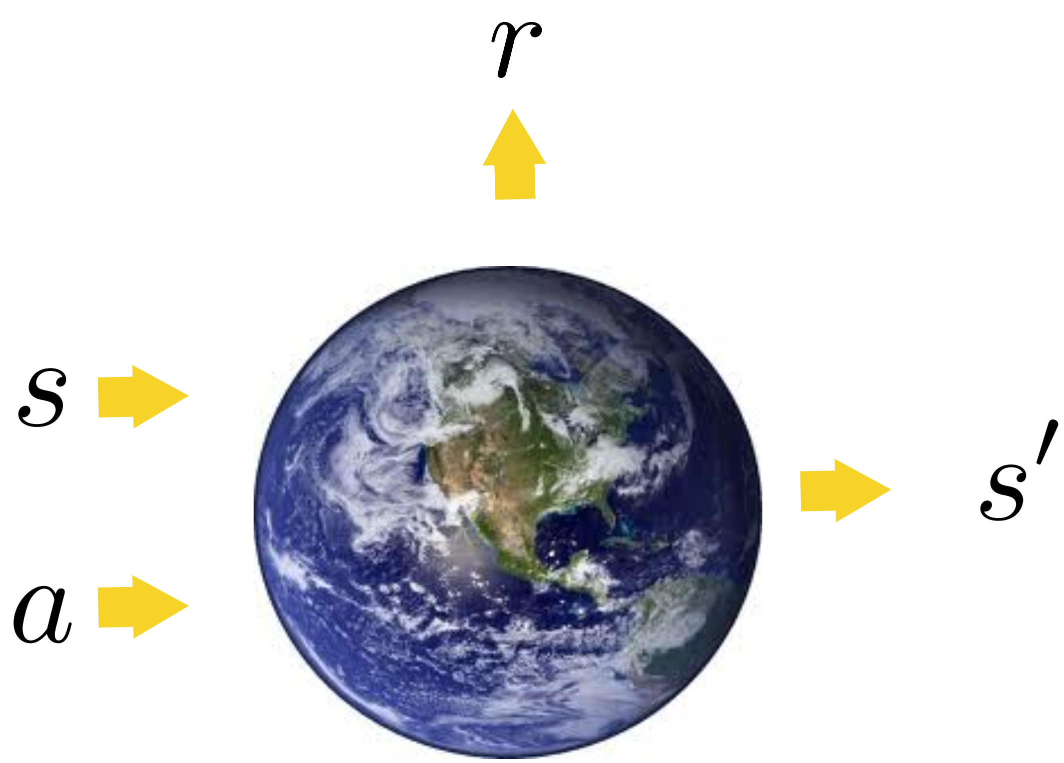
Planning

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



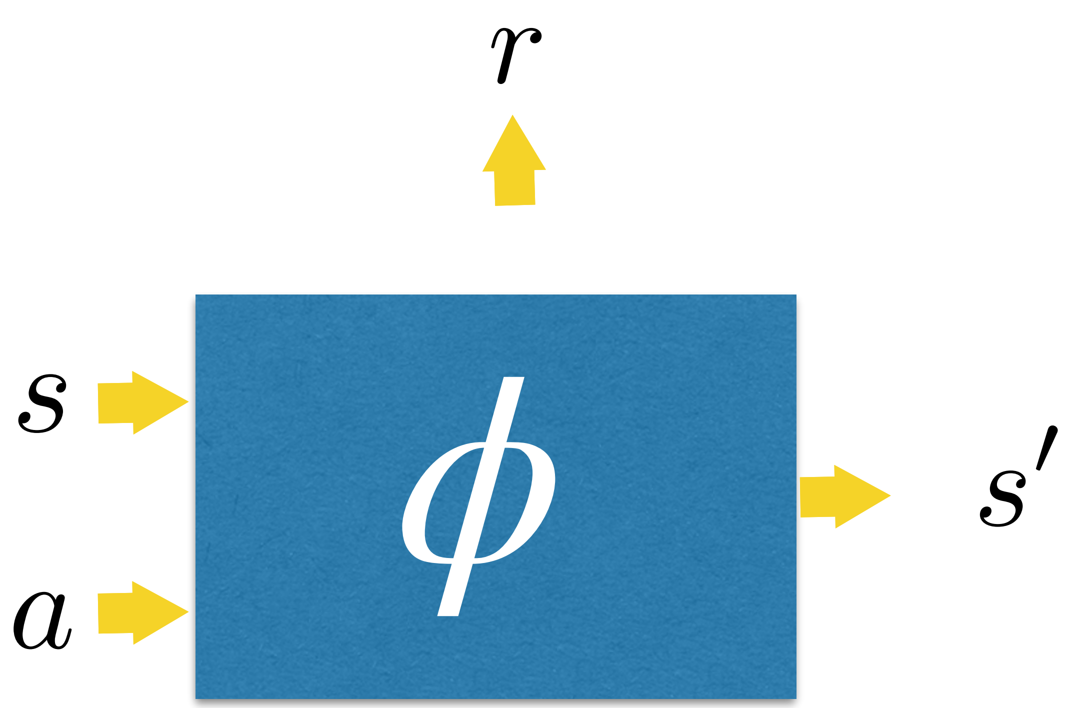
Model

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



Model learning

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



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

Model learning

Newtonian Physics equations

VERSUS

general parametric form (no prior from Physics knowledge)

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

Much easier to learn but suffers from under-modeling

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 (later 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

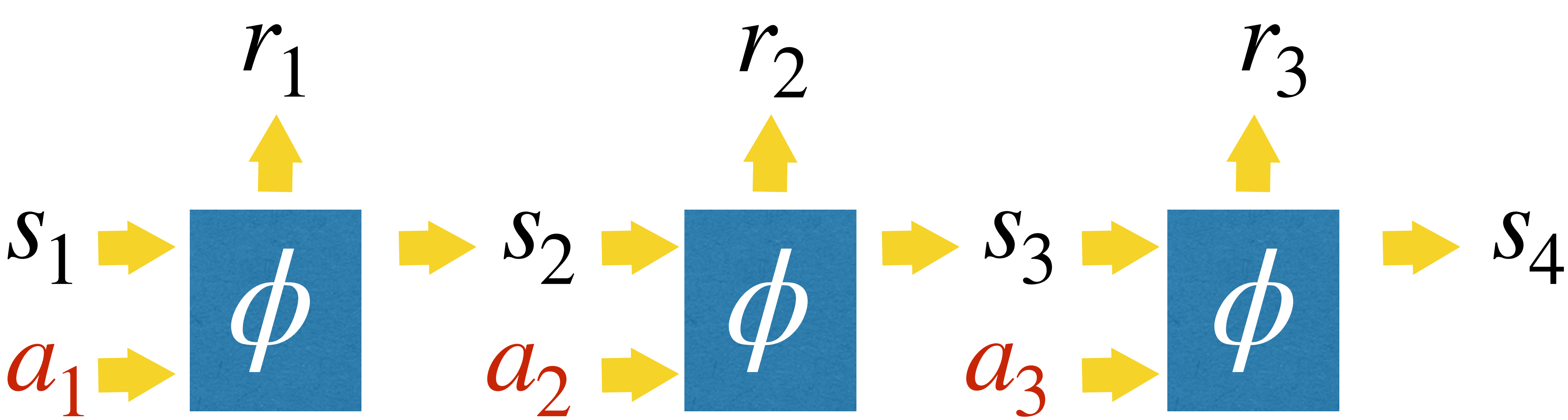
$$\min_{a_1 \cdots a_T} . \ \|s_T - s_*\|$$

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

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

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.



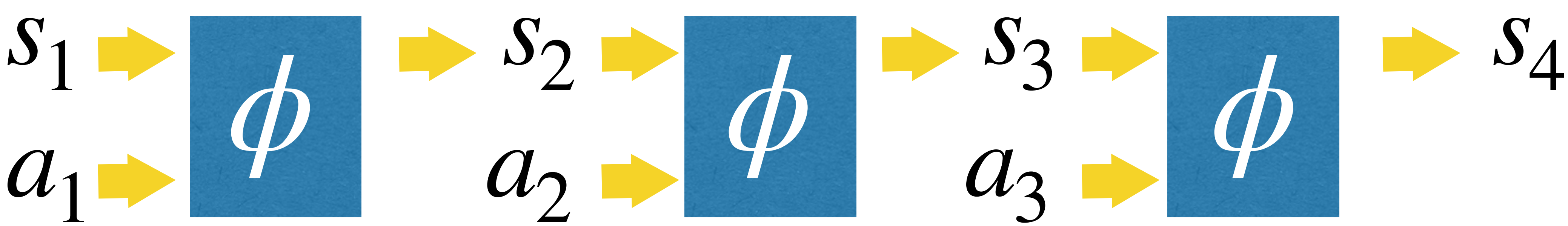
Model-based control- SGD

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

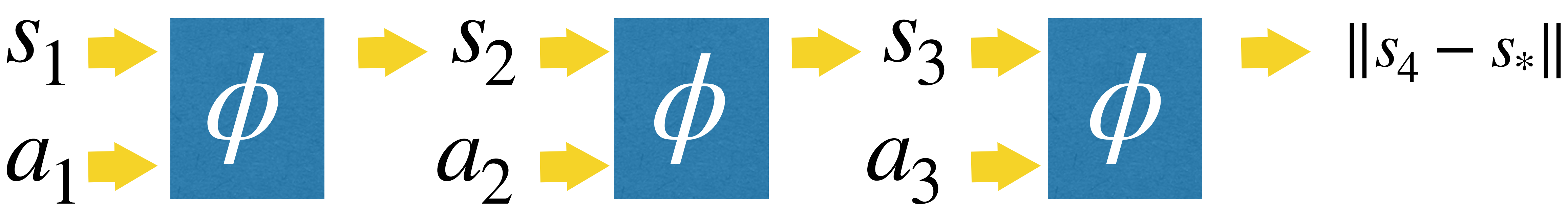
Model-based control

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



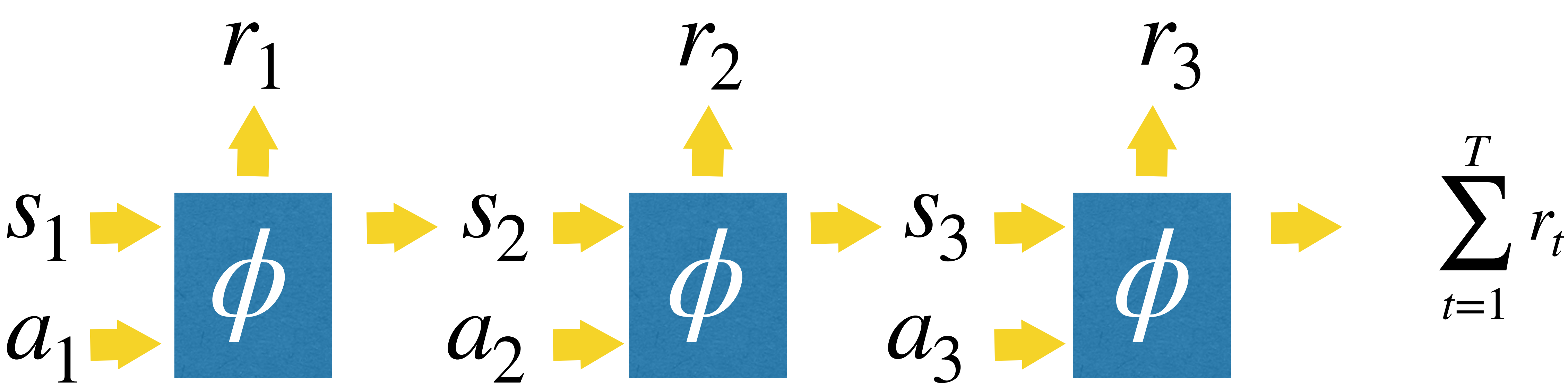
Model-based control

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



Model-based control

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



Model-based control

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

We execute only the first action and then GOTO 1, to avoid error accumulation. (Model Predictive Control)

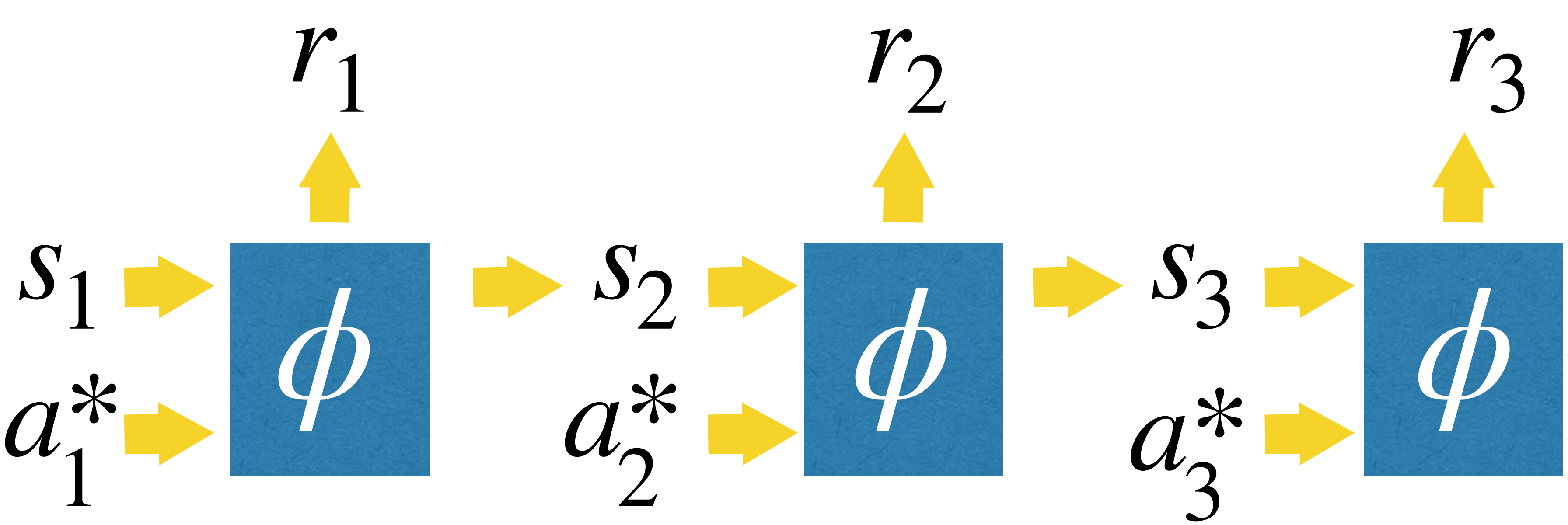
$$\min_{a_1 \cdots a_T} . \ \|s_4 - s_*\|$$

Model-based control

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

We execute only the first action and then GOTO 1, to avoid error accumulation. (Model Predictive Control)

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



Model-based control - derivative-free

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

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

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

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

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

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)

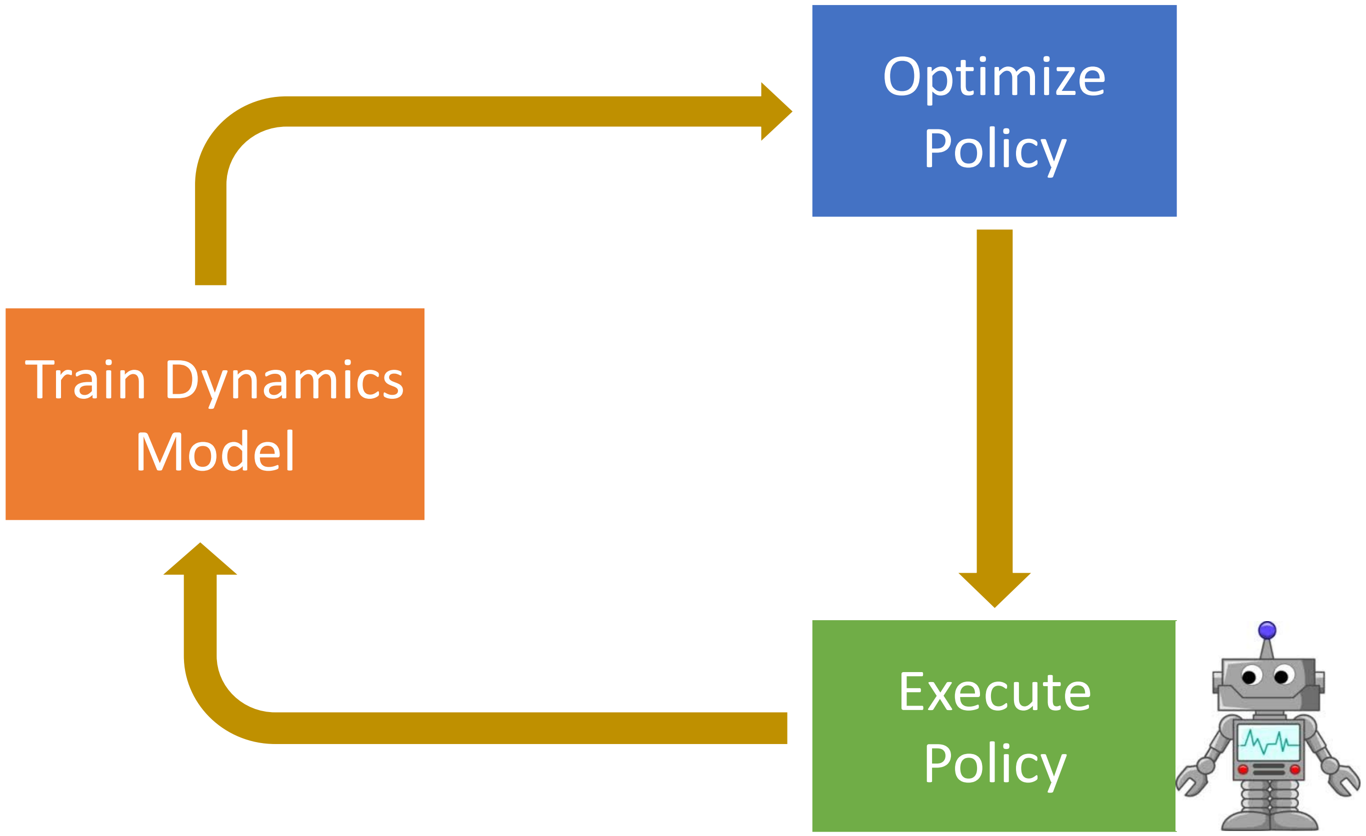
    2. an imitation learning method by imitating the MB planner

- Efficient Exploration guided by model uncertainty (later lecture)

Alternating between model and policy learning

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

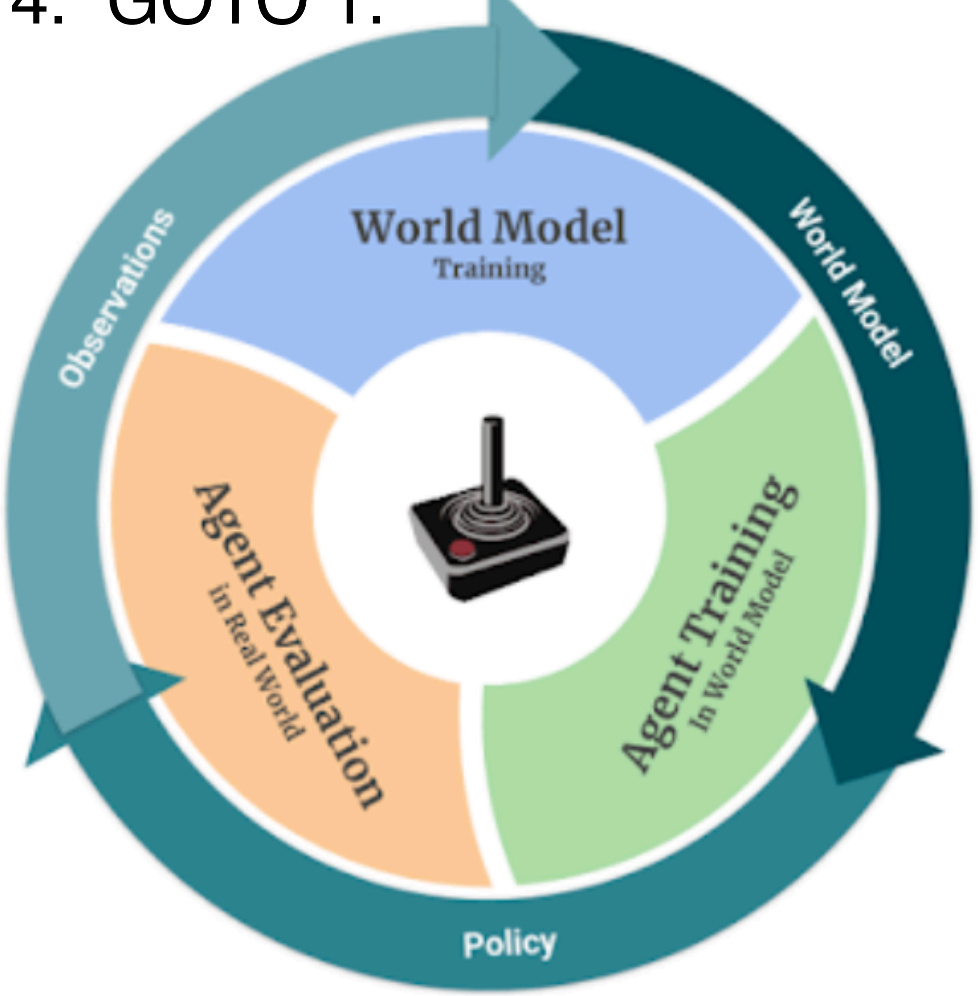
1. Run the policy and update experience tuples dataset D.
2. Train a dynamic model using D: $(s', r') = f(s, a; \phi)$
3. Update the policy using
    1. model-free RL method on simulated experience sampled from the model
    2. Immitating a model-based controller
4. GOTO 1.

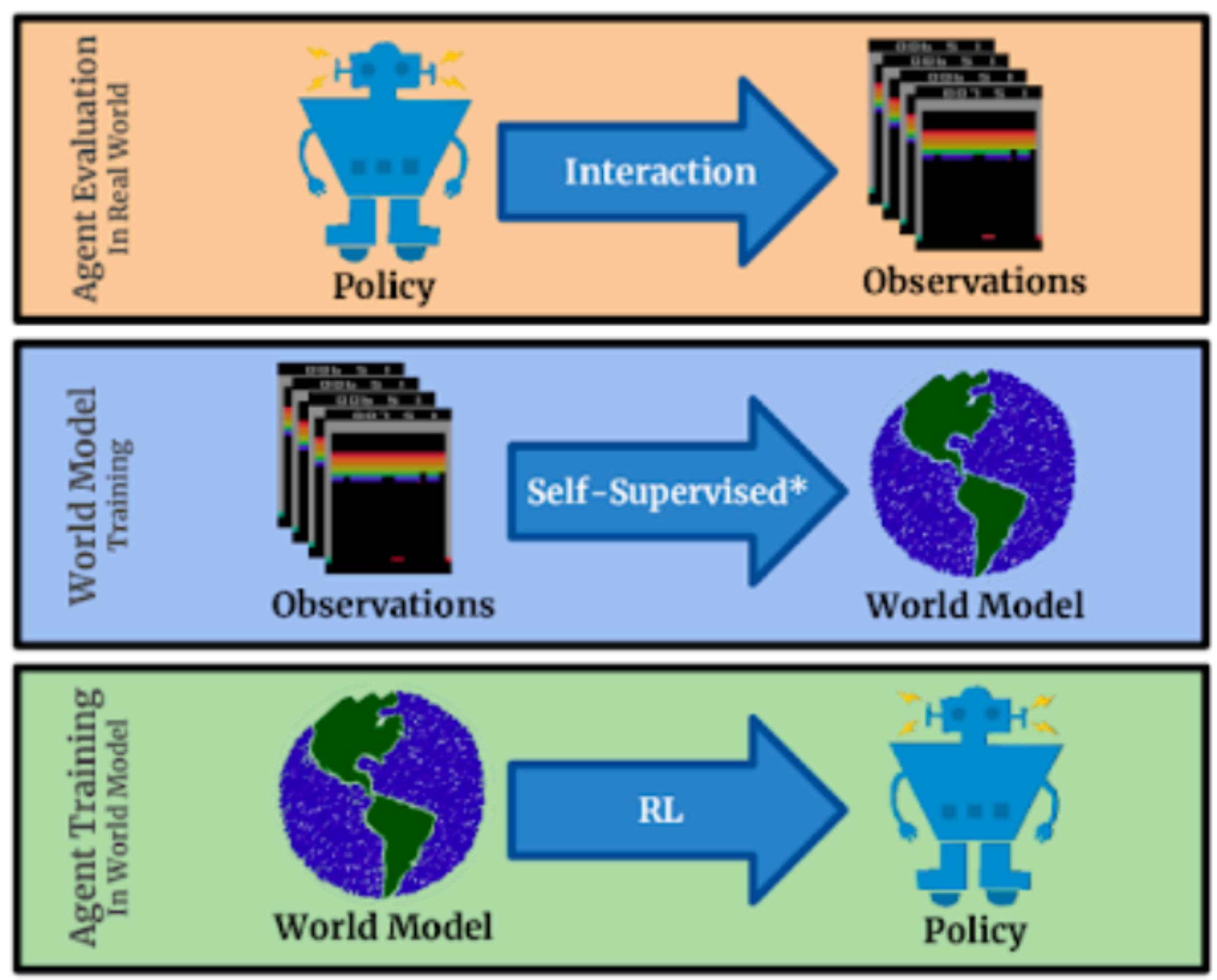


Model-based RL

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.



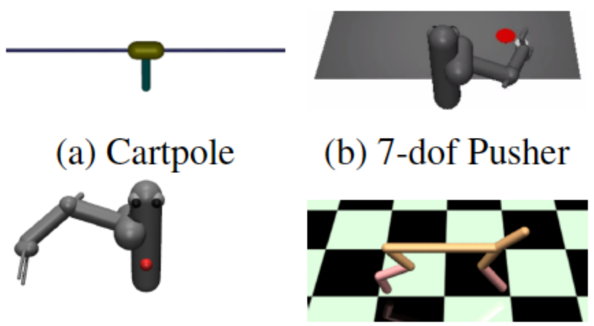


Challenges in model learning

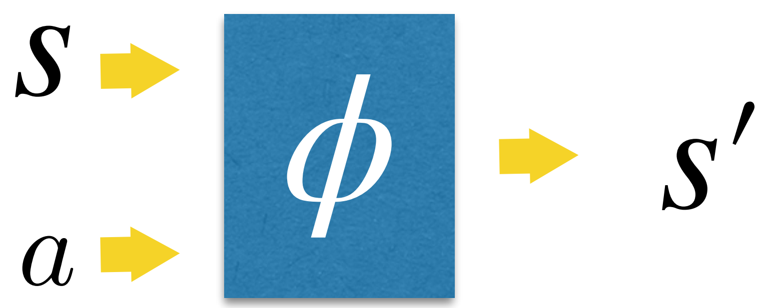
- Under-modelling: If the model class is restricted (e.g., linear function or gaussian process) we have under-modeling: we cannot represent complex dynamics, e.g., contact dynamics that are not smooth. As a result, though we learn faster than model free in the beginning, MBRL ends up having worse asymptotic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

Model Learning

- Where a low dimensional state is observed and given:


(a) Cartpole (b) 7-dof Pusher

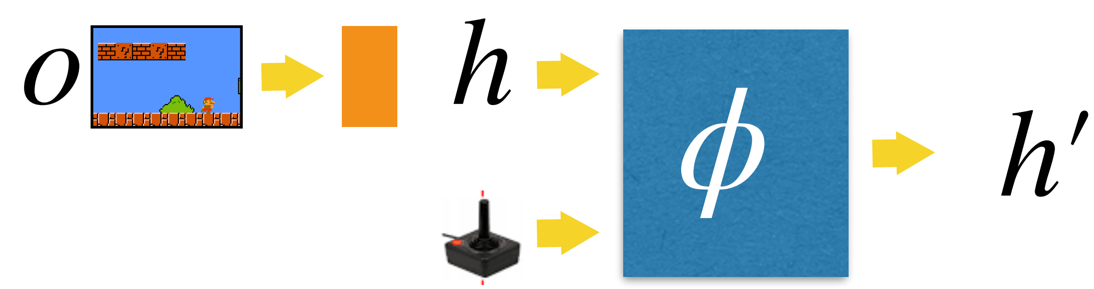
(c) 7-dof Reacher (d) Half-cheetah

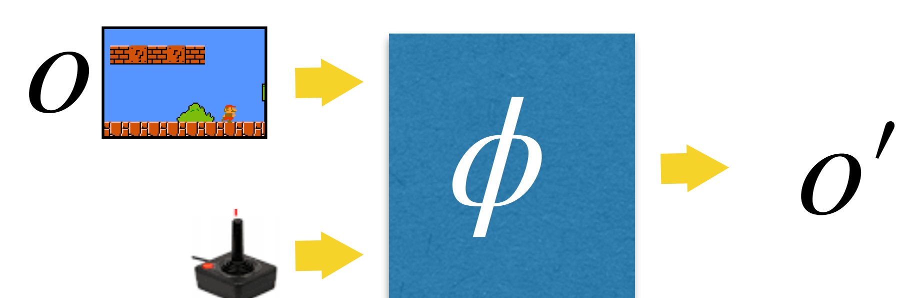


state can be 3D locations and 3D velocities of agent joints, actions can be torques

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

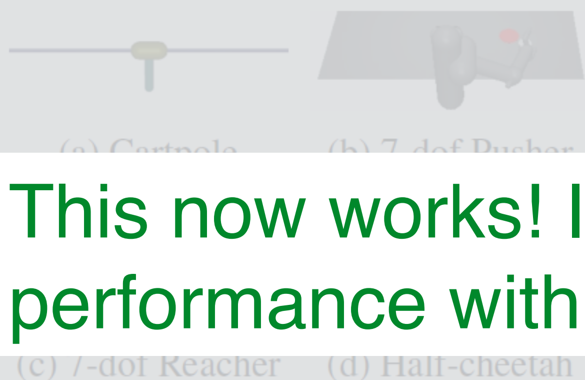
e.g., Atari game playing





Model Learning

* Where a low dimensional state is observed and given:



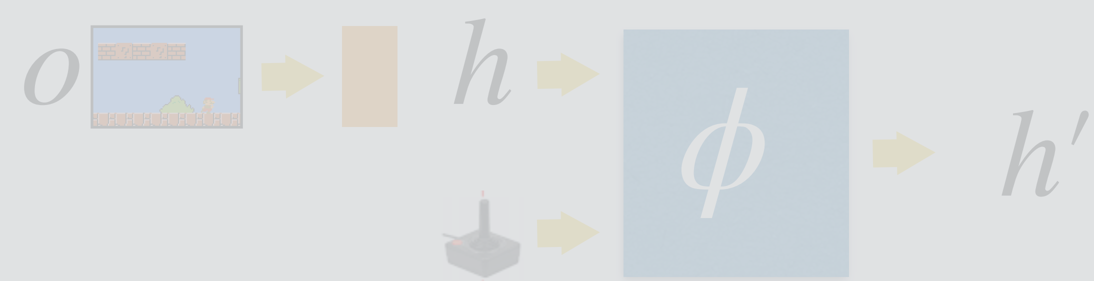
This now works! It outperforms model-free RL methods: reaches same final performance with much fewer samples! :-)

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., image or touch:

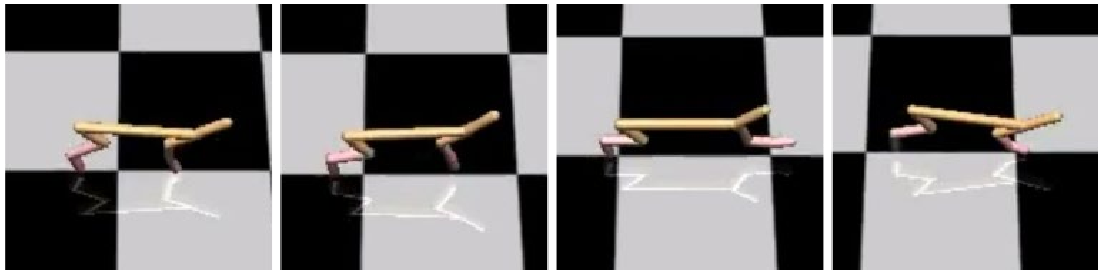
e.g., Atari game playing

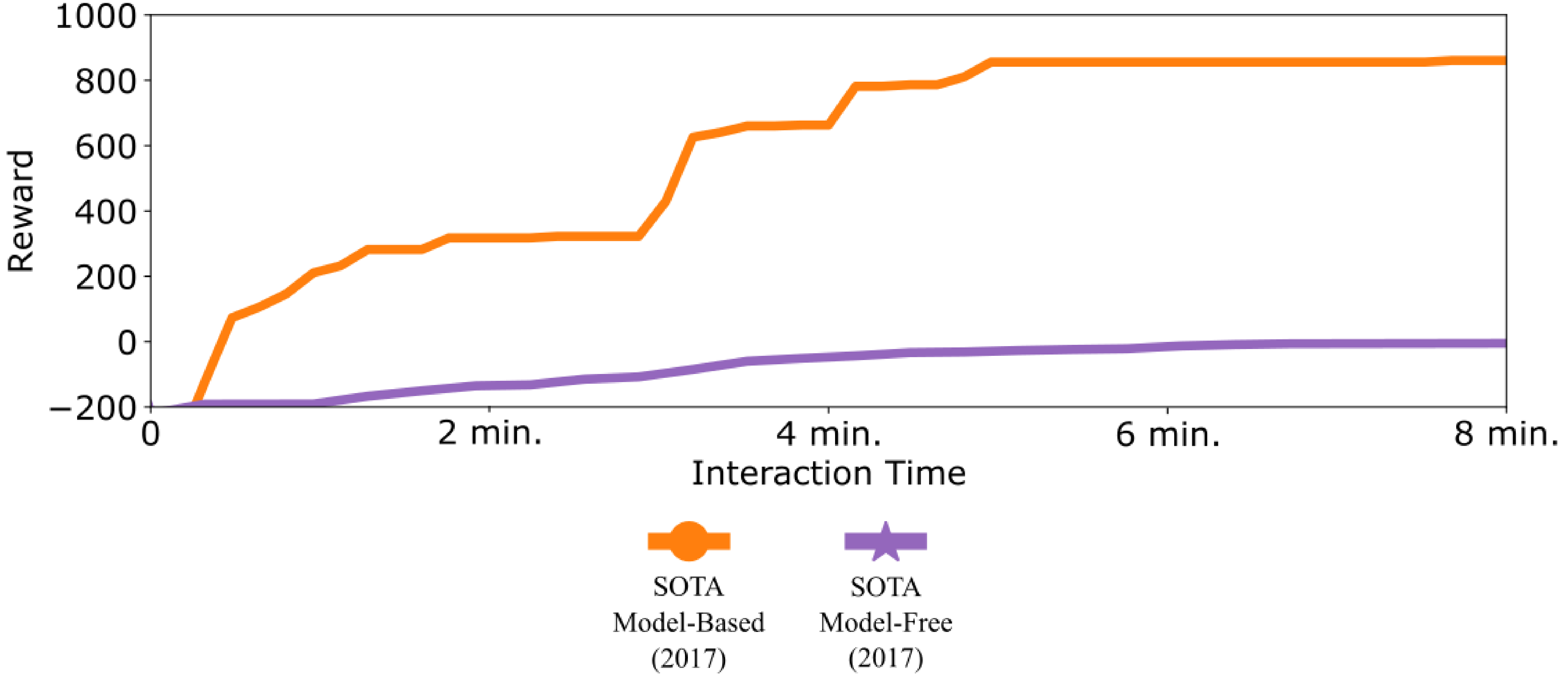
Still an open problem :-(



Model-based RL in a low-dim state space

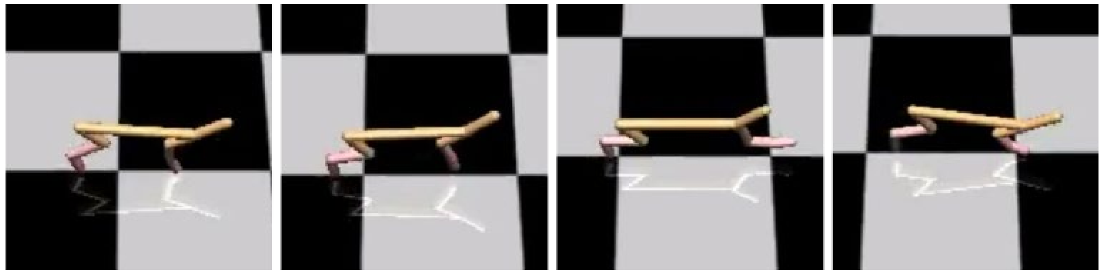
Comparative Performance on HalfCheetah

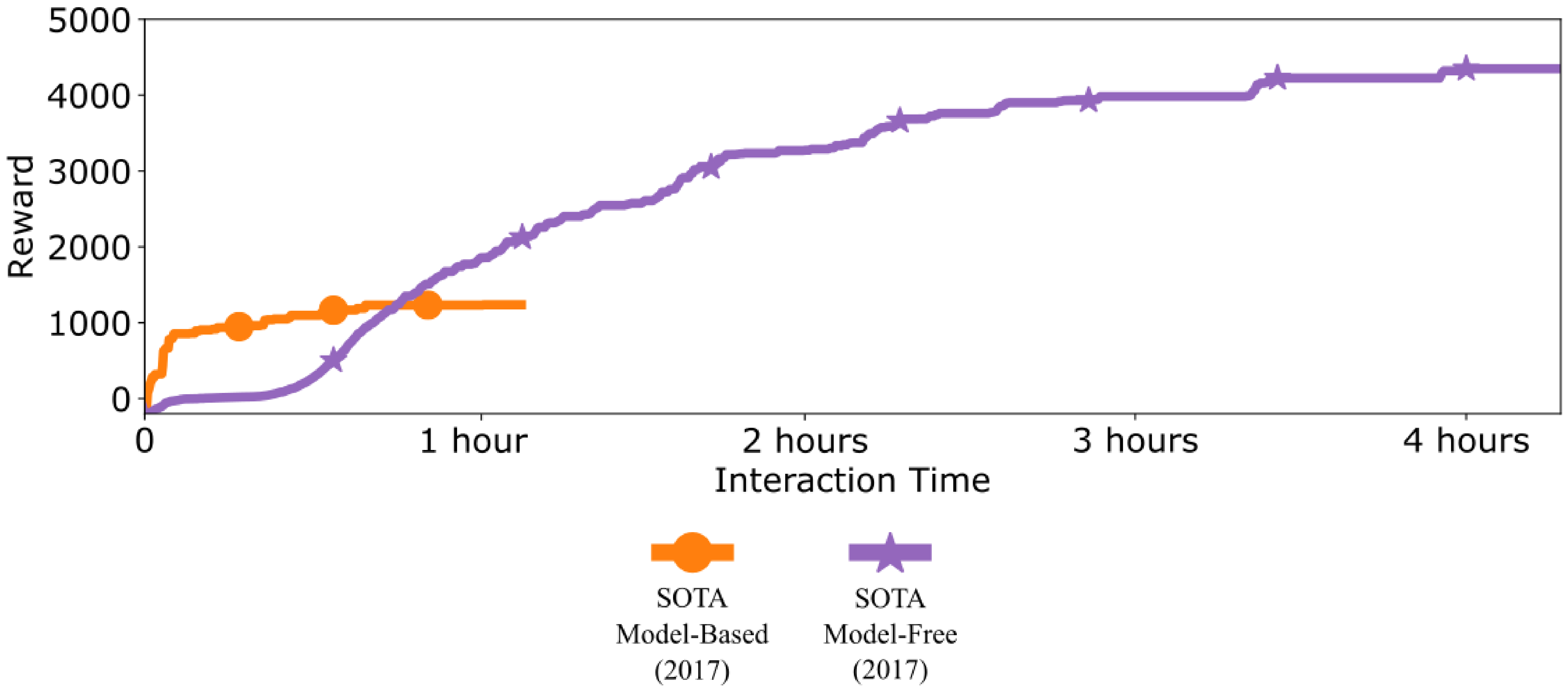




slide from Sergey Levine

Comparative Performance on HalfCheetah

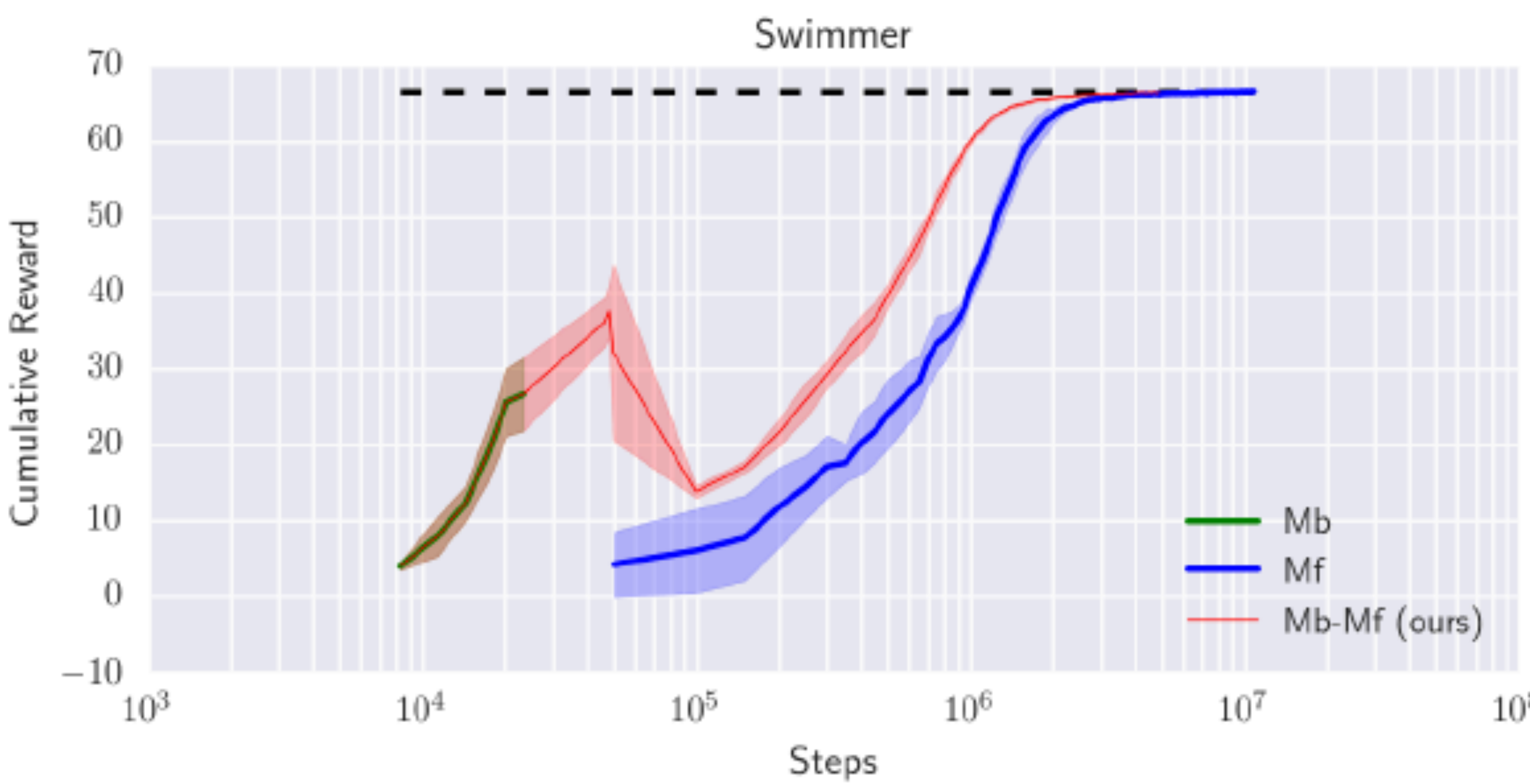




slide from Sergey Levine

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

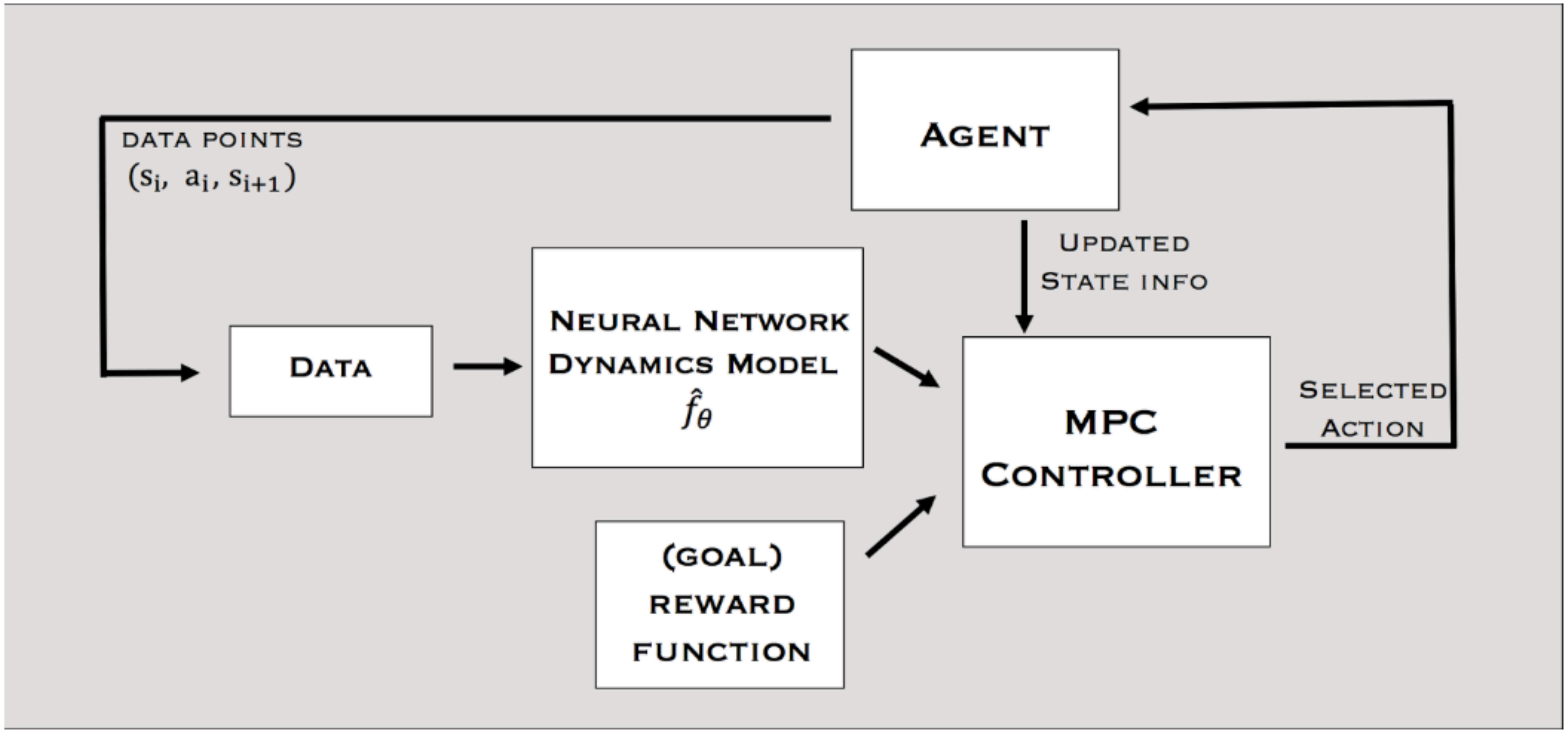


Neural network dynamics for model-based Deep Rl with model-free finetuning, Nagabandi et al.

Model-based RL

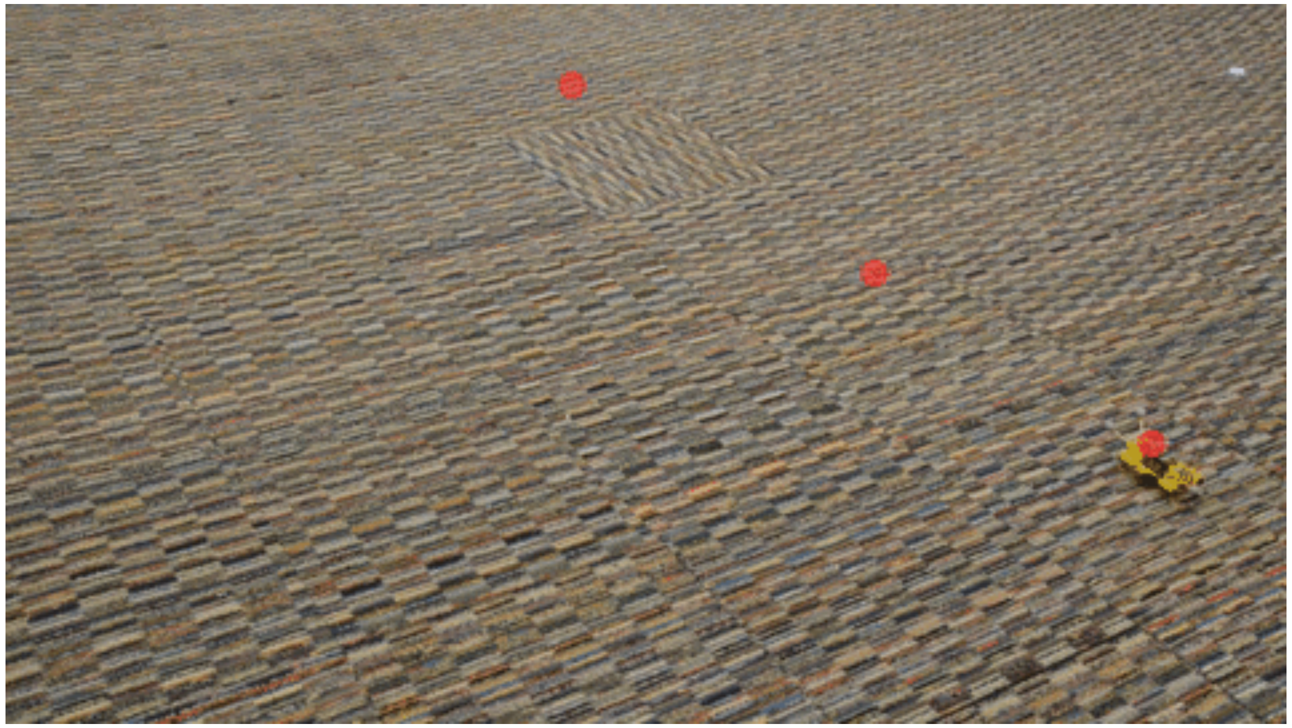
Collect a dataset D of random experience tuples (s,a,s')

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

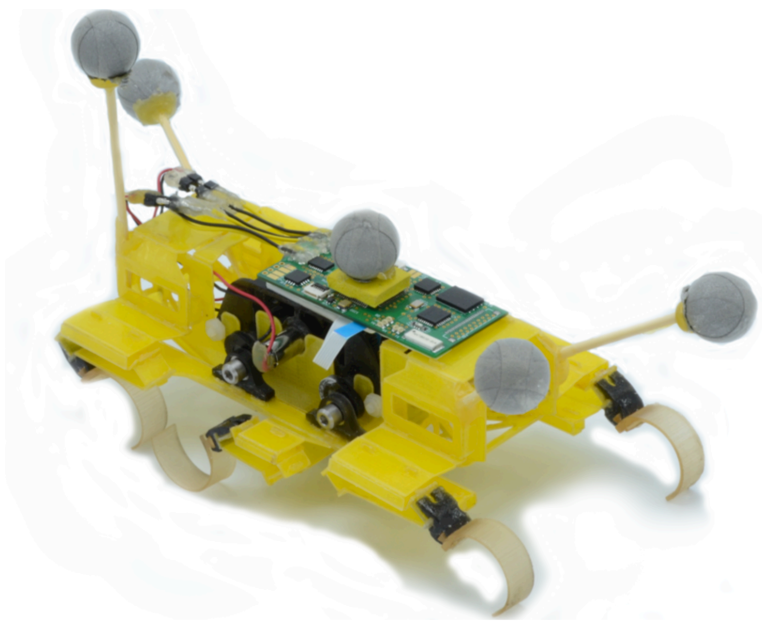


Neural network dynamics for model-based Deep Rl with model-free finetuning, Nagabandi et al.

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.



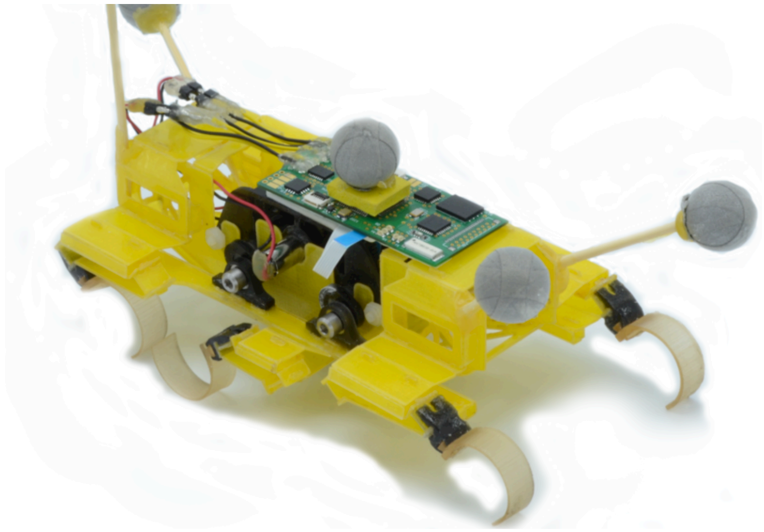
Neural network dynamics for model-based Deep RI with model-free finetuning, Nagabandi et al.

Model-based RL

	Carpet	Styrofoam
MB trained on carpet	5.69	18.62
MB trained on styrofoam	22.25	8.15
MB trained on both	7.52	15.76

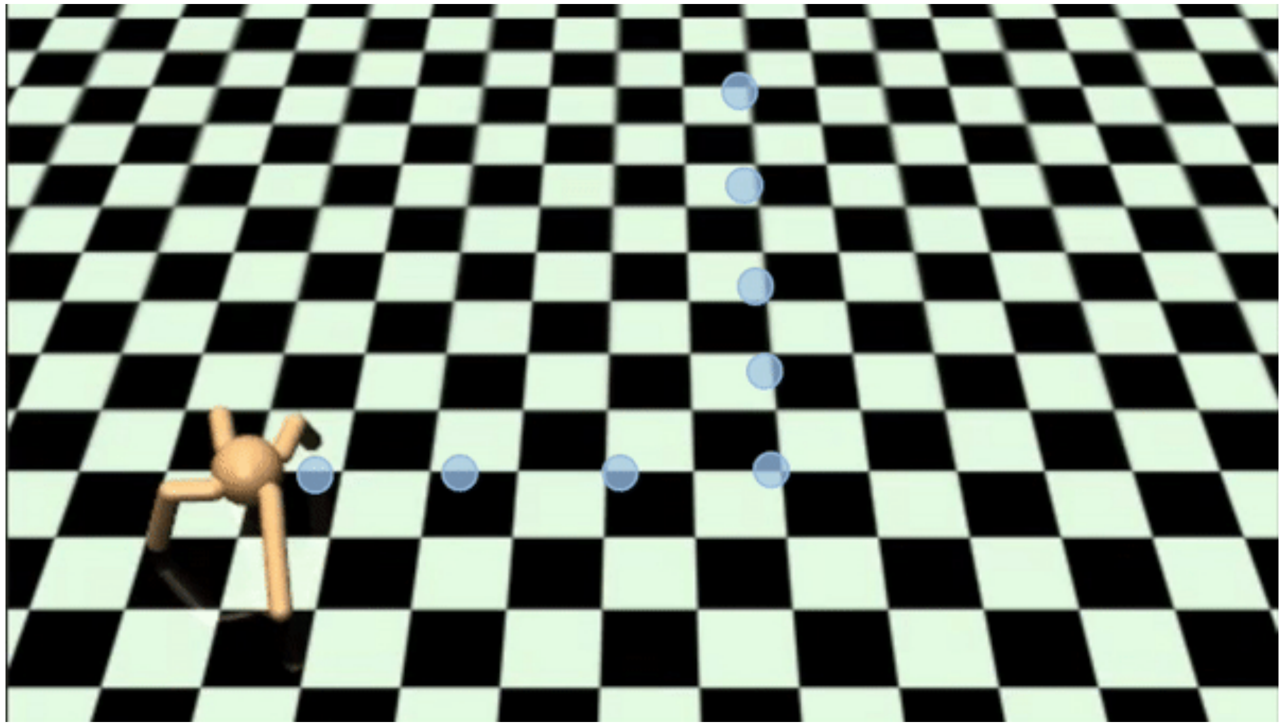
Table 1: Trajectory following costs incurred for models trained with different types of data and for trajectories executed on different surfaces.

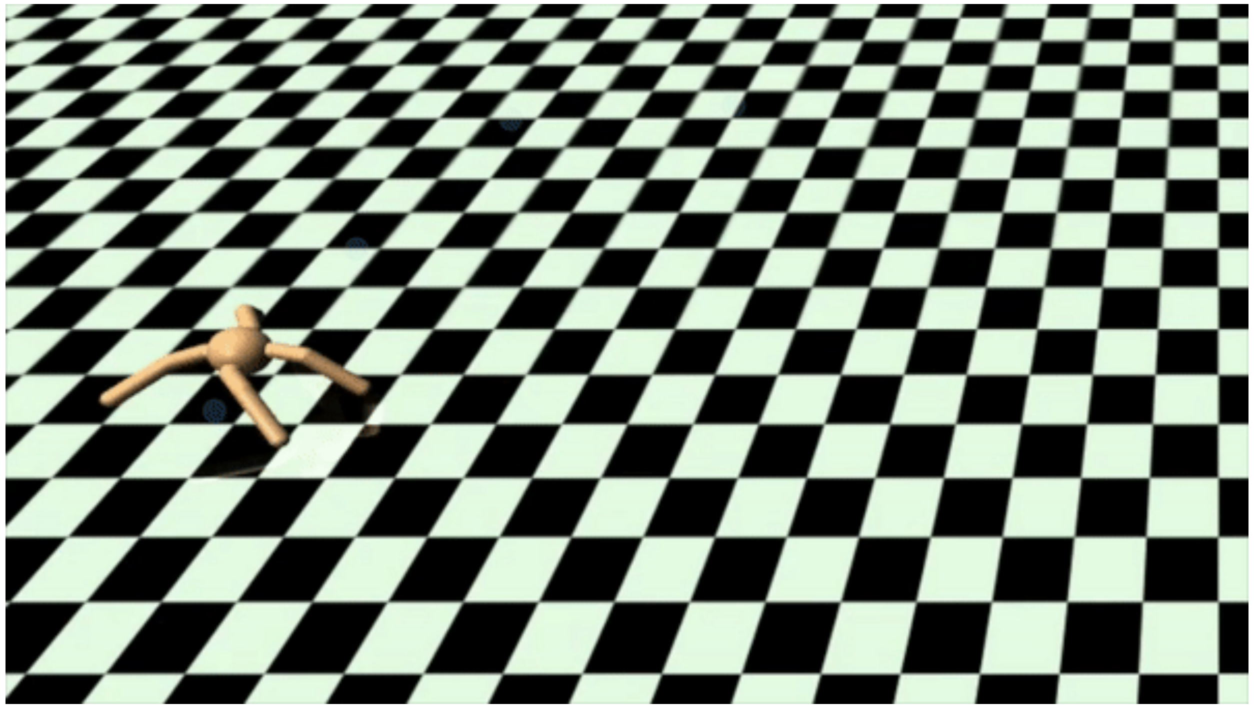
Training a model based controller allows to follow arbitrary trajectories at test time: the model allows you to optimize different reward function for different tasks, without any retraining.



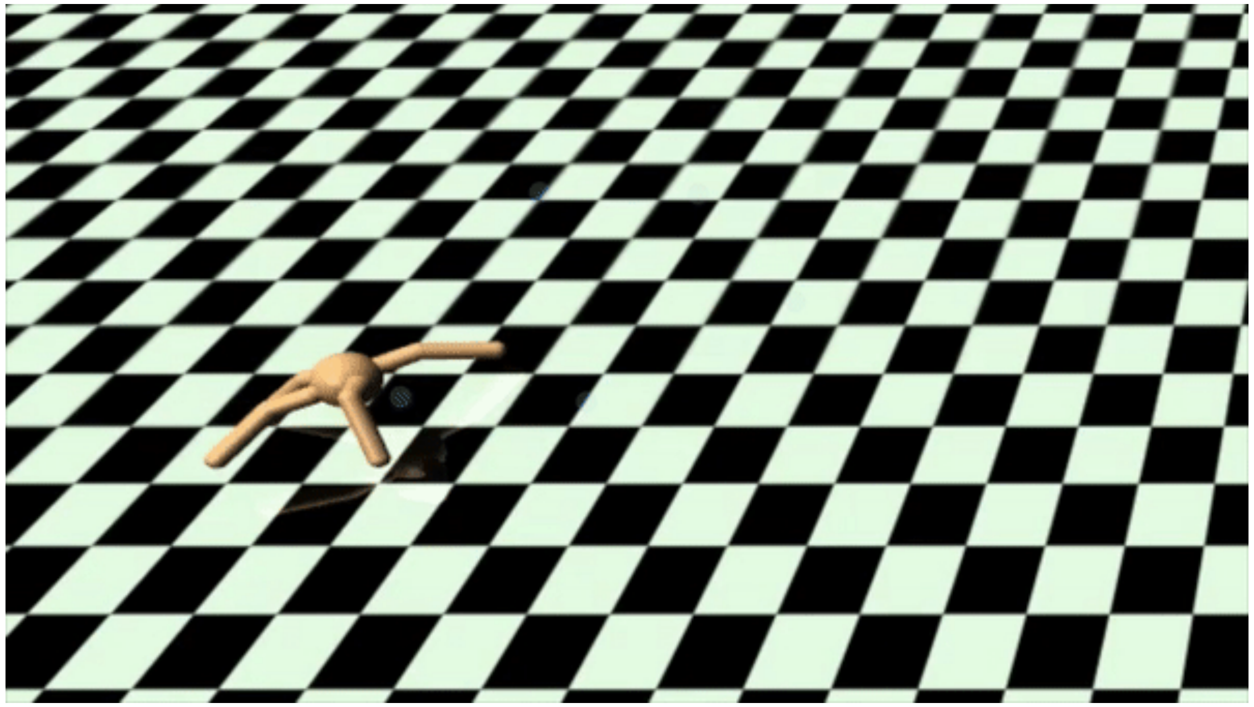
Neural network dynamics for model-based Deep Rl with model-free finetuning, Nagabandi et al.

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.

Model-based RL

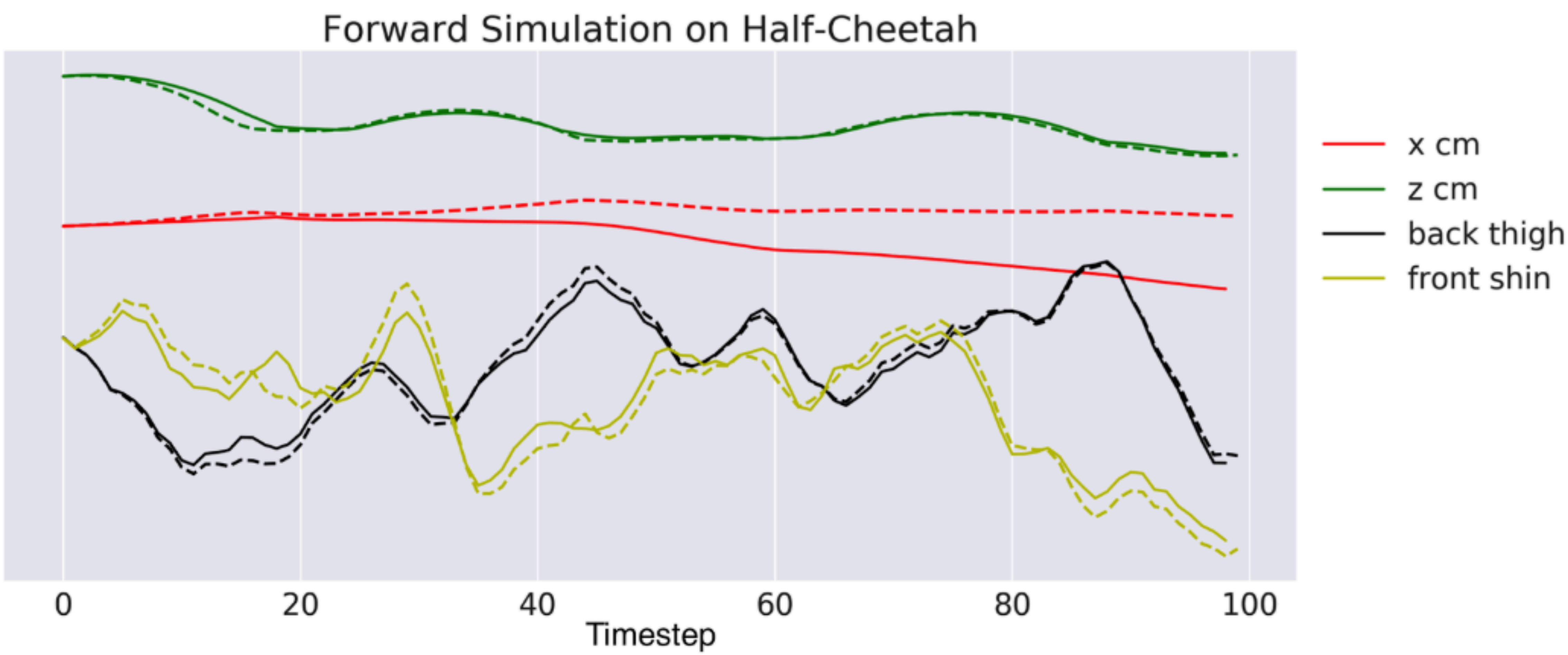


Fig 6: A 100-step forward simulation (open-loop) of the dynamics model, showing that open-loop predictions for certain state elements eventually diverge from the ground truth.

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

Model-based RL

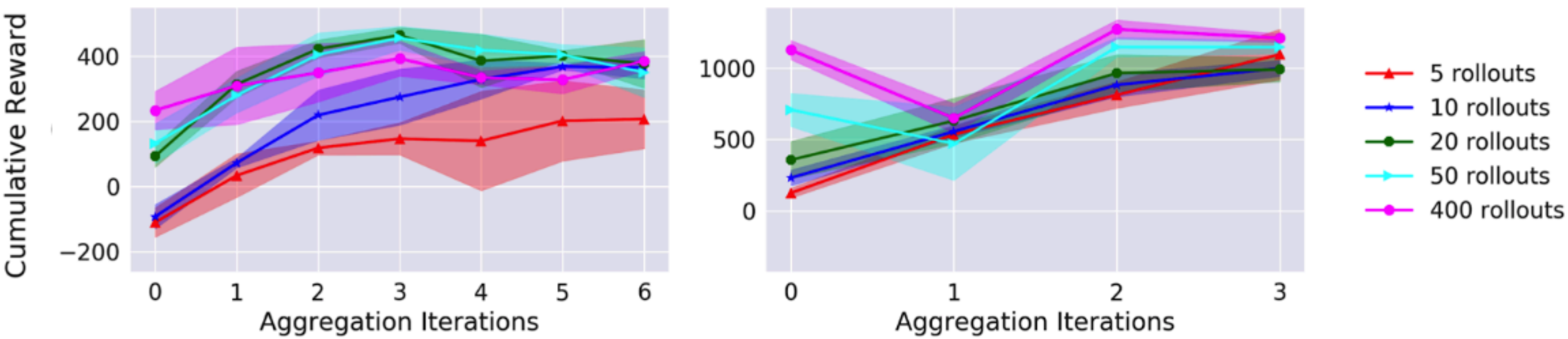


Fig 7: Plot of task performance achieved by dynamics models that were trained using differing amounts of initial random data.

Large random initial dataset helps
Aggregation helps

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

Model-based RL with model-free finetuning

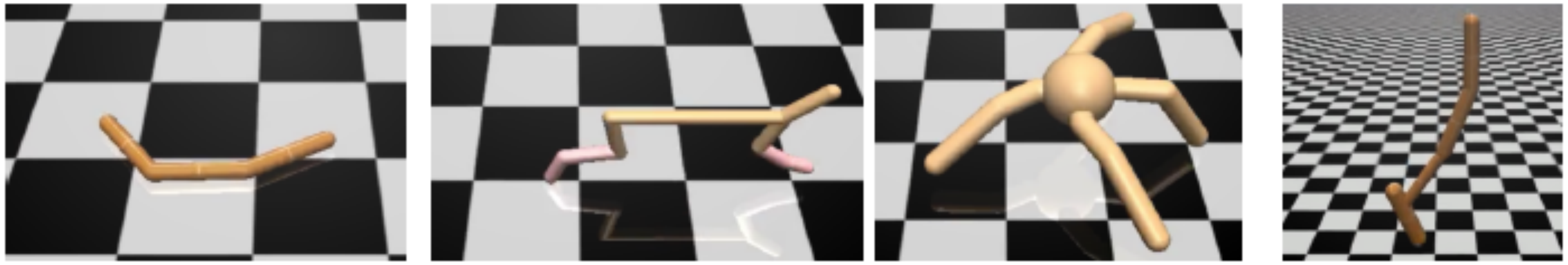
Collect a dataset D of random experience tuples (s,a,s')

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

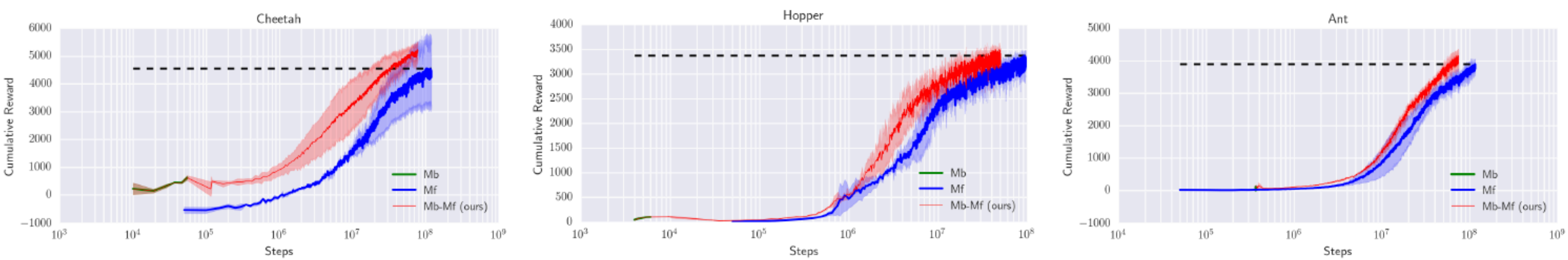
Initialize a policy $\pi(s; \theta)$ by imitating the MPC planner using DAGGER

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

Experiments



(a) Swimmer (b) Cheetah (c) Ant (d) Hopper



Note: Model-based RL alone looks hopeless!

Q: Can model-based RL outperform model-free in terms of both sample complexity and asymptotic performance?

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:

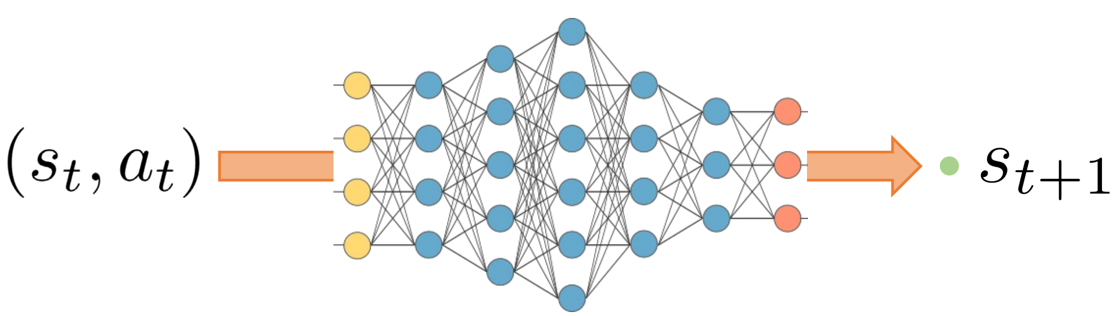
1. Epistemic uncertainty: uncertainty due to lack of data (that 'd permit to uniquely determine the underline system exactly)
2. Aleatoric uncertainty: uncertainty due to inherit stochasticity of the system

Aleatoric uncertainty in model learning

$$\mathrm{D} = \{(s_i, a_i, s_i'), i = 1 \cdots N\}$$

- We will use a neural network that outputs a distribution over the next state s_{t+1}.
- Specifically, a Gaussian distribution, where the NN predicts the mean and covariance matrix

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



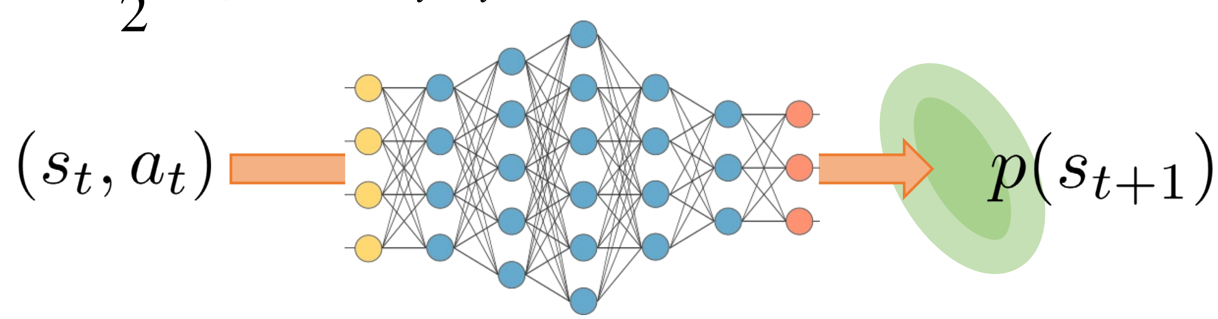
deterministic neural network

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

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

$$= \left(\frac{1}{2}(s_i' - \mu(s_i, a_i; \phi))^T \boldsymbol{\Sigma}(s_i, a_i; \phi)^{-1} (s_i' - \mu(s_i, a_i; \phi)\right)$$

$$+ \frac{1}{2} \log(\det \boldsymbol{\Sigma}(s_i, a_i; \phi)) + \text{ const.}$$

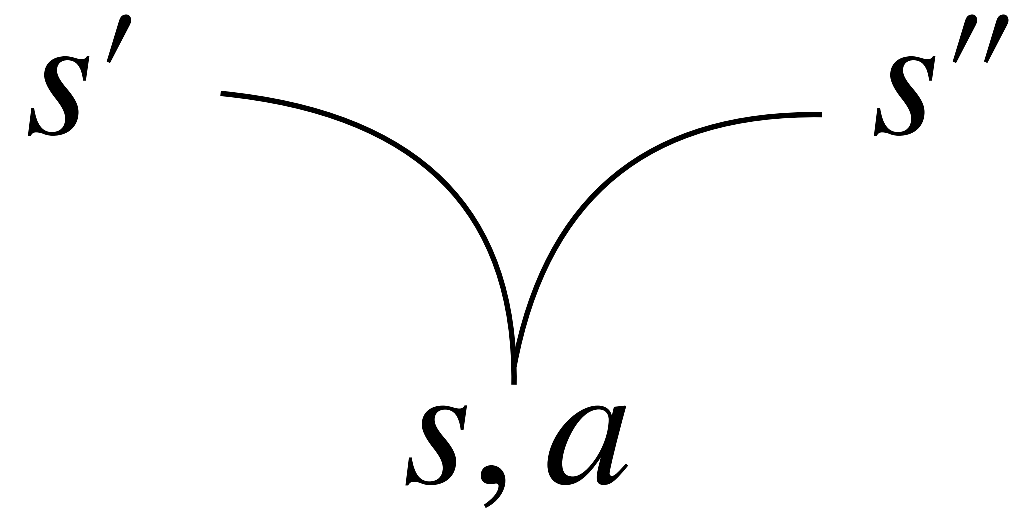


Aleatoric uncertainty in model learning

Assume we collected a dataset of experience tuples $D = \{(s_i, a_i, s_i'), i = 1 \cdots N\}$

We want to train a model, i.e., the state transition function (let's forget the reward for now). What can I do?

The environment can be stochastic



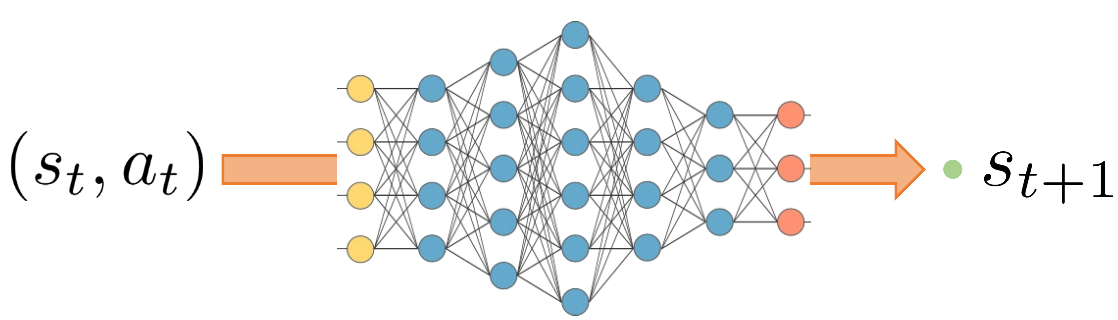
- This means our state does not capture enough information to help us dileneate the possible future outcomes.
- What is stochastic under one state representation, may not be stochastic under another.
- We will always have part of the information hidden, so stochasticity will always be there

Aleatoric uncertainty in model learning

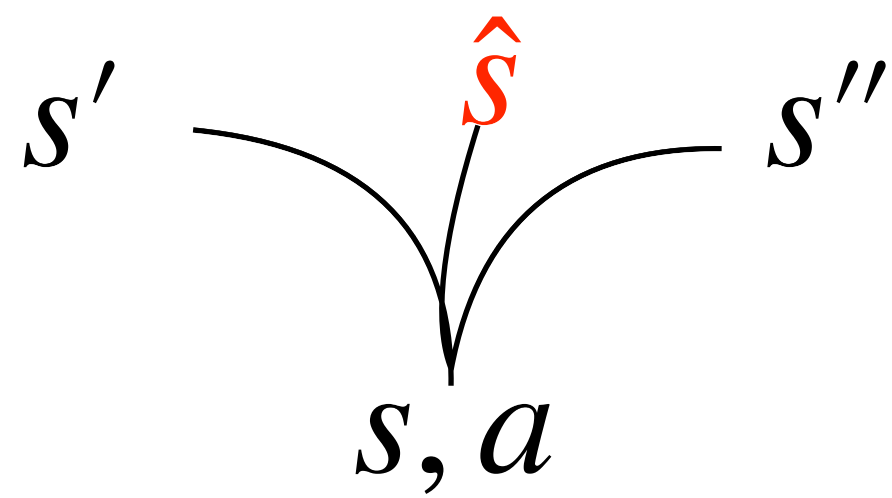
Assume we collected a dataset of experience tuples $\mathrm{D} = \{(s_i, a_i, s_i'), i = 1 \cdots N\}$

Training a deterministic regressor!

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



If the environment is stochastic, regression fails



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

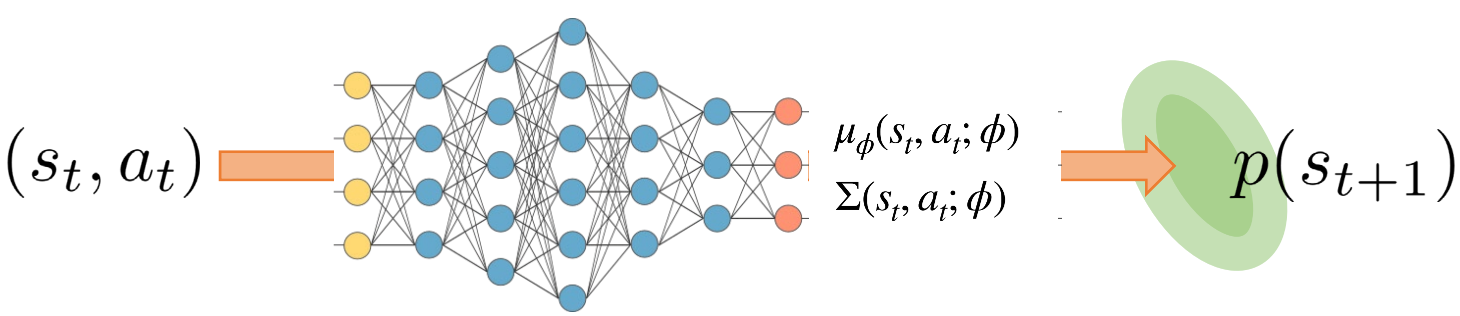
Aleatoric uncertainty in model learning

Assume we collected a dataset of experience tuples $\mathrm{D} = \{(s_i, a_i, s_i'), i = 1 \cdots N\}$

Training a probabilistic NN! Given a (s,a) as input, the NN outputs a mean vector and a set of variances, one for each dimension of the state vector. We train by maximizing log likelihood of our training set.

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

$$\begin{aligned}
\mathscr{L}_\phi &= -\frac{1}{N}\sum_{i=1}^{N} \log p(s_i' \mid s_i, a_i; \phi) \\
&= \left(\frac{1}{2}(s_i' - \mu(s_i, a_i; \phi))^T \boldsymbol{\Sigma}(s_i, a_i; \phi)^{-1}(s_i' - \mu(s_i, a_i; \phi))\right) \\
&+ \frac{1}{2}\log(\det \boldsymbol{\Sigma}(s_i, a_i; \phi)) + \text{ const.}
\end{aligned}$$



Aleatoric uncertainty in model learning

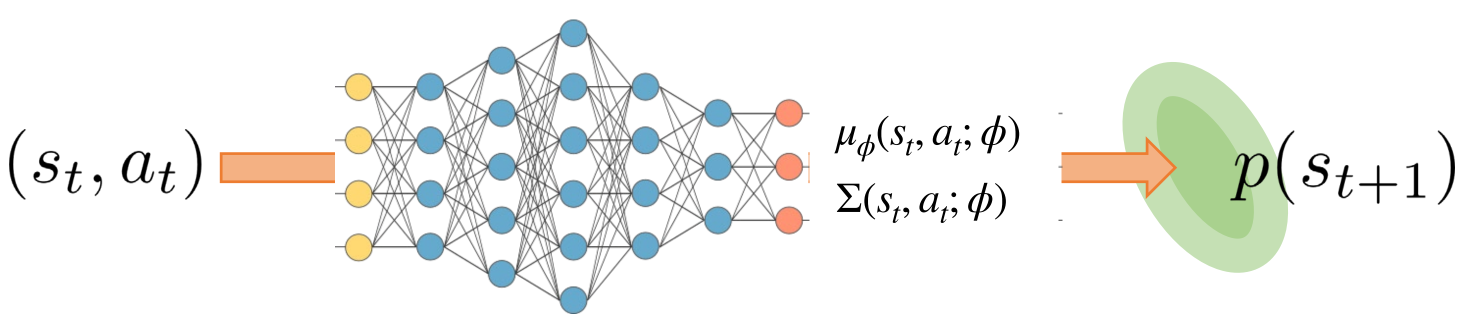
Assume we collected a dataset of experience tuples $\mathrm{D} = \{(s_i, a_i, s_i'), i = 1 \cdots N\}$

Training a probabilistic NN! Given a (s,a) as input, the NN outputs a mean vector and a set of variances, one for each dimension of the state vector. We train by maximizing log likelihood of our training set.

Q: variance should be always positive, what do we do?

A: we output logvar and we exponentiate

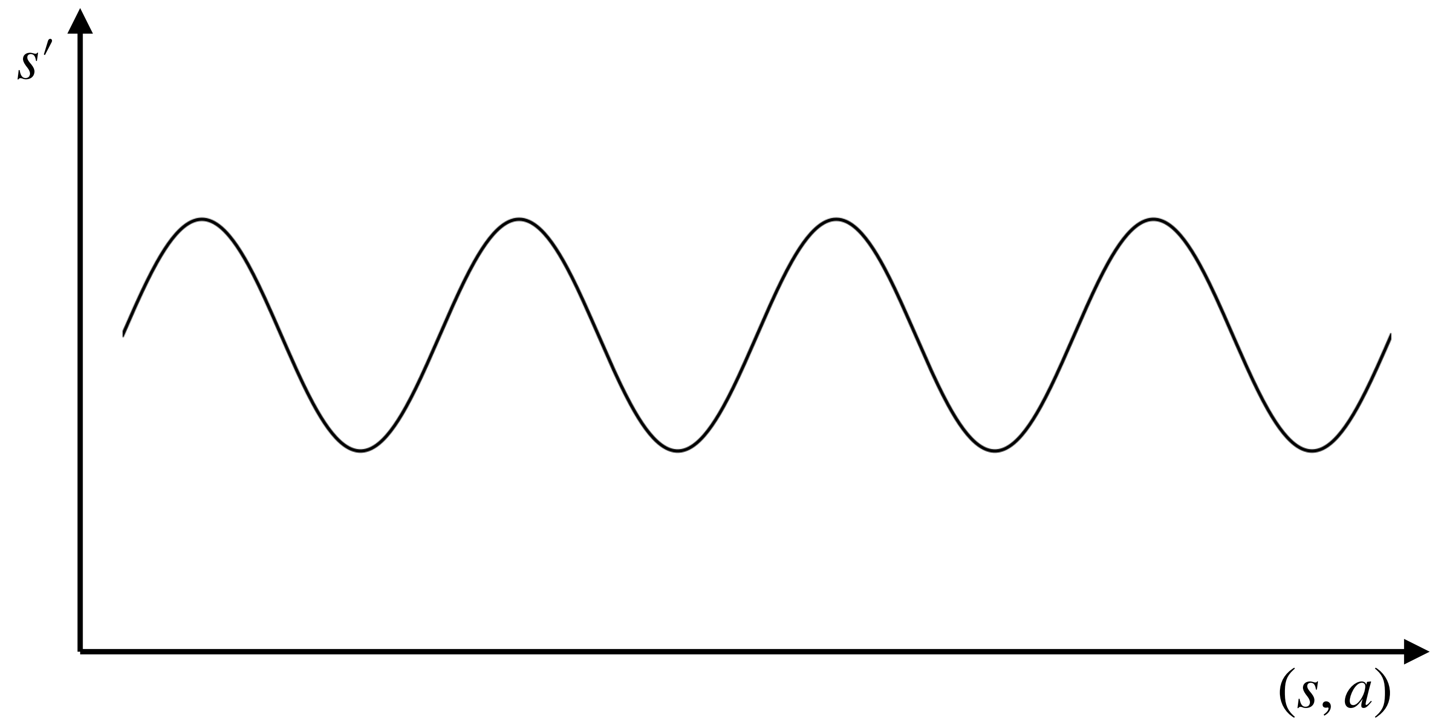
```
logvar = max_logvar - tf.nn.softplus(max_logvar - logvar)
logvar = min_logvar + tf.nn.softplus(logvar - min_logvar)
var = tf.exp(logvar)
```



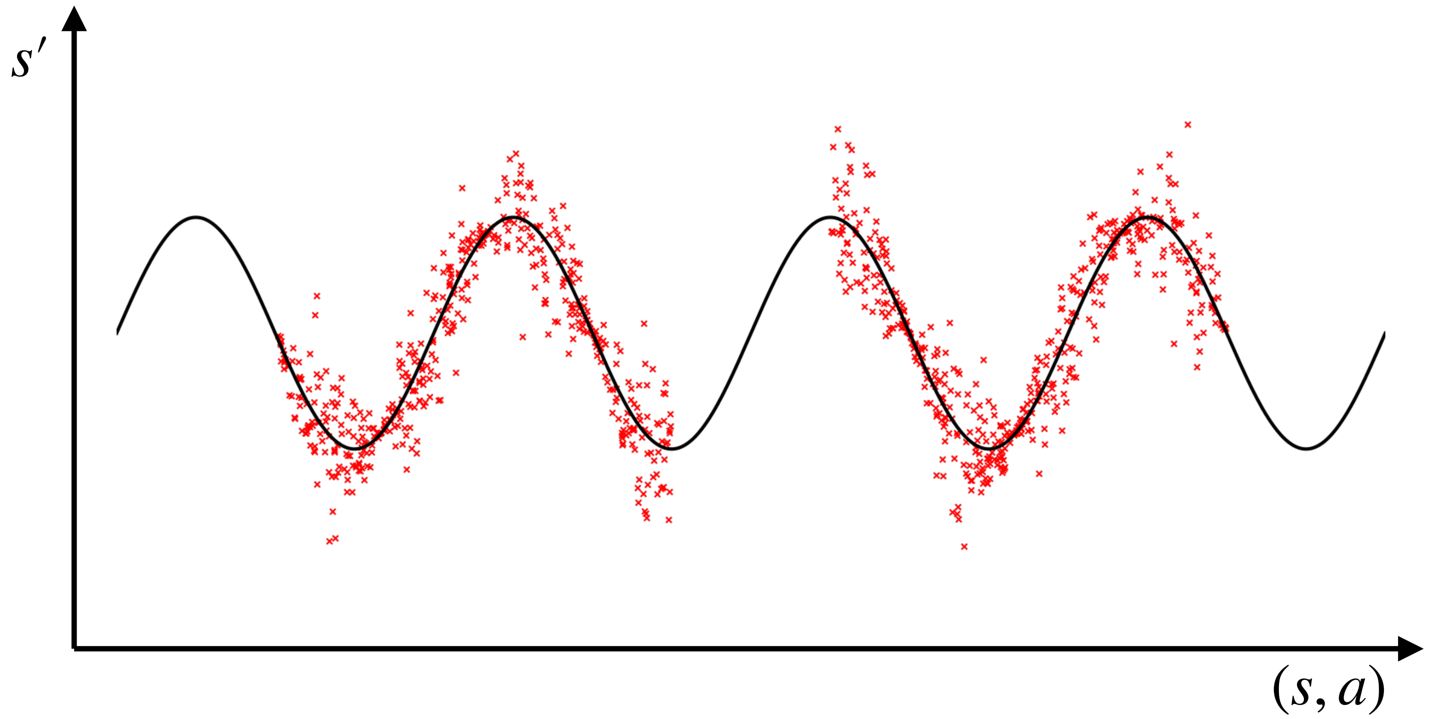
Epistemic uncertainty

- The principled way to handle such uncertainty is with Bayesian models, e.g., Gaussian processes, or Bayesian neural networks
- We will use neural network ensembles. It turns out they are a very good and efficient approximation to Bayesian neural networks.

Epistemic uncertainty in Model Learning

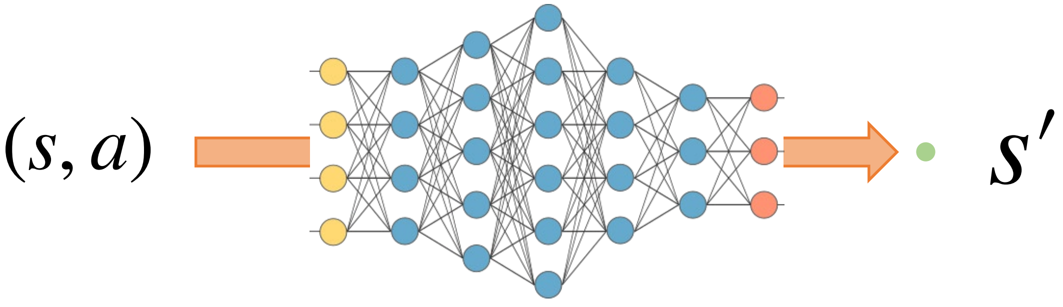


Epistemic uncertainty in Model Learning

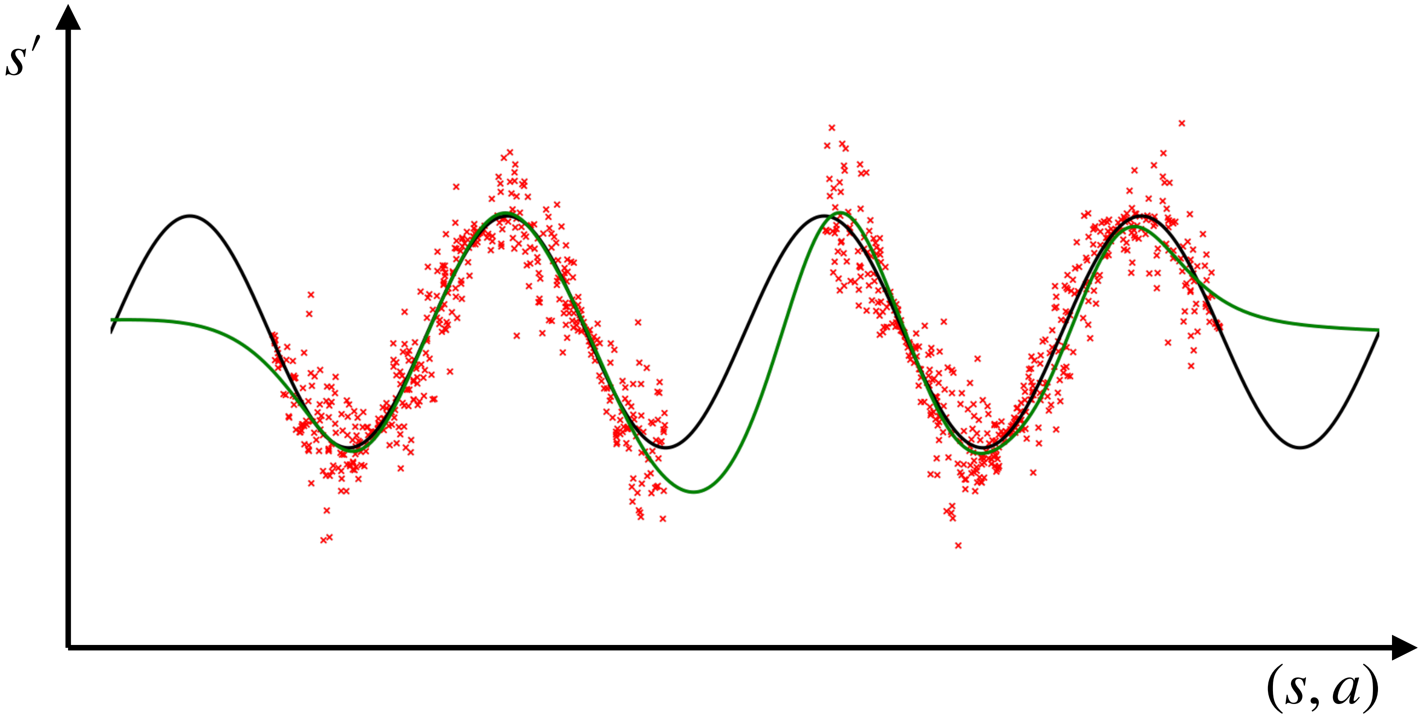


Red are observed data points (s,a,s')

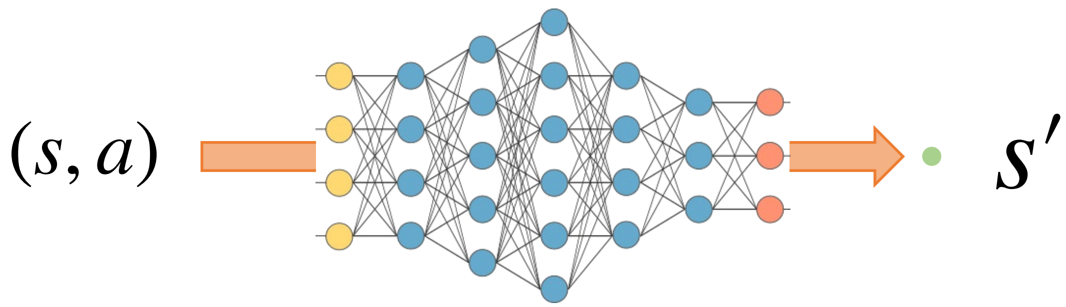
Epistemic uncertainty in Model Learning



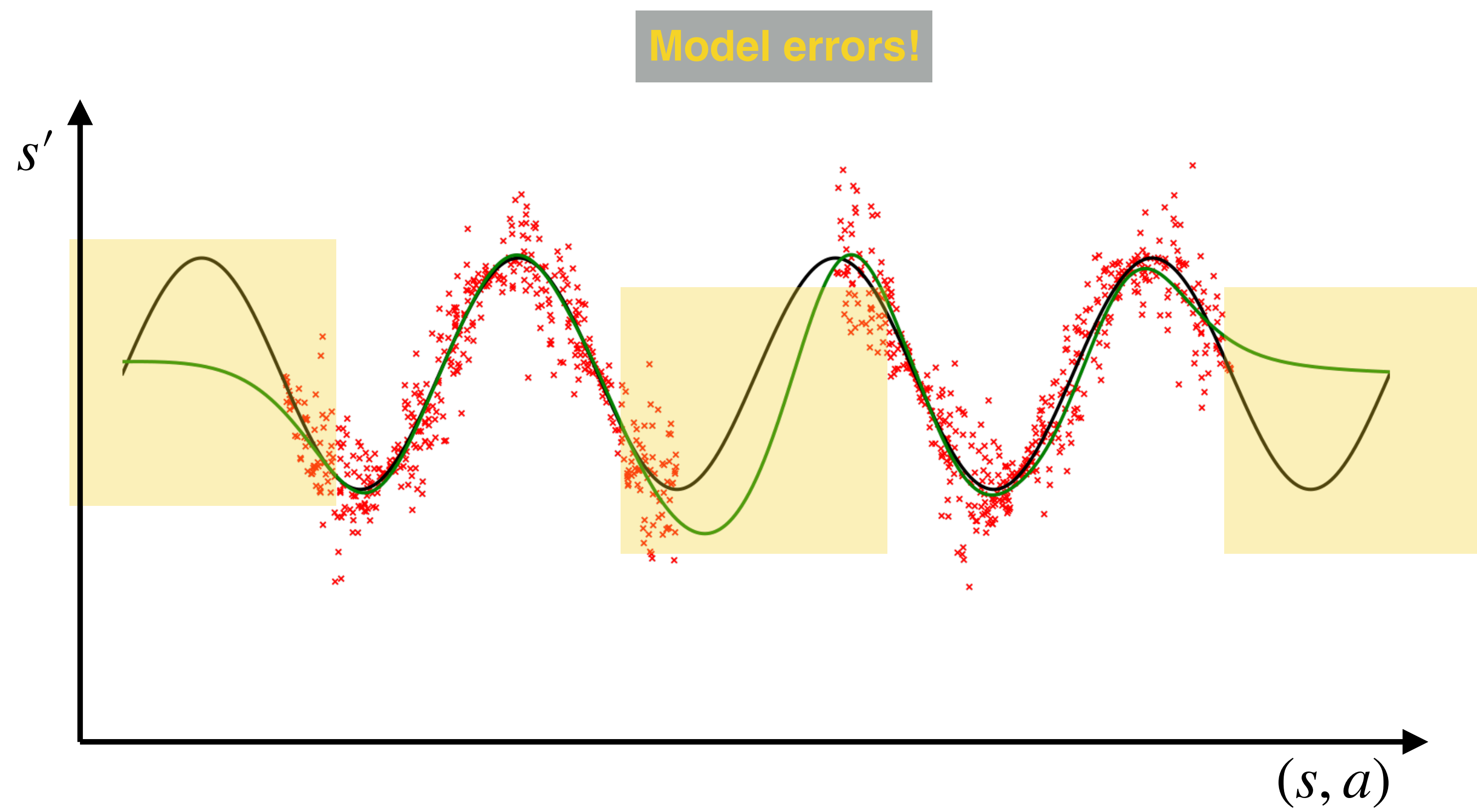
Fitting a deterministic neural network



Epistemic uncertainty in Model Learning

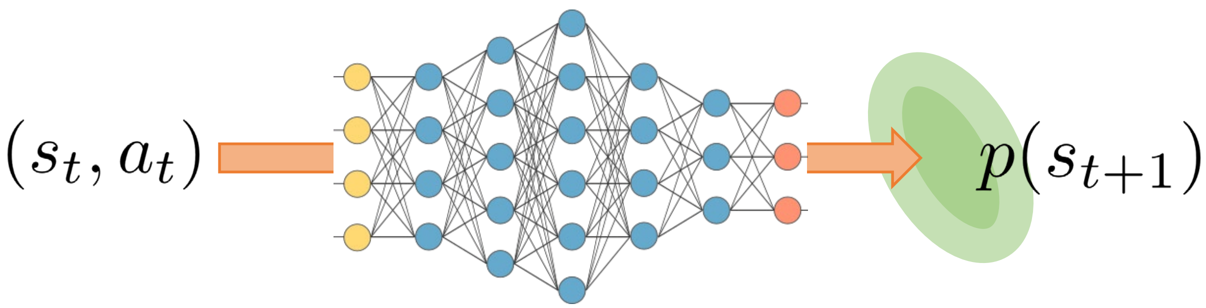


Fitting a deterministic neural network

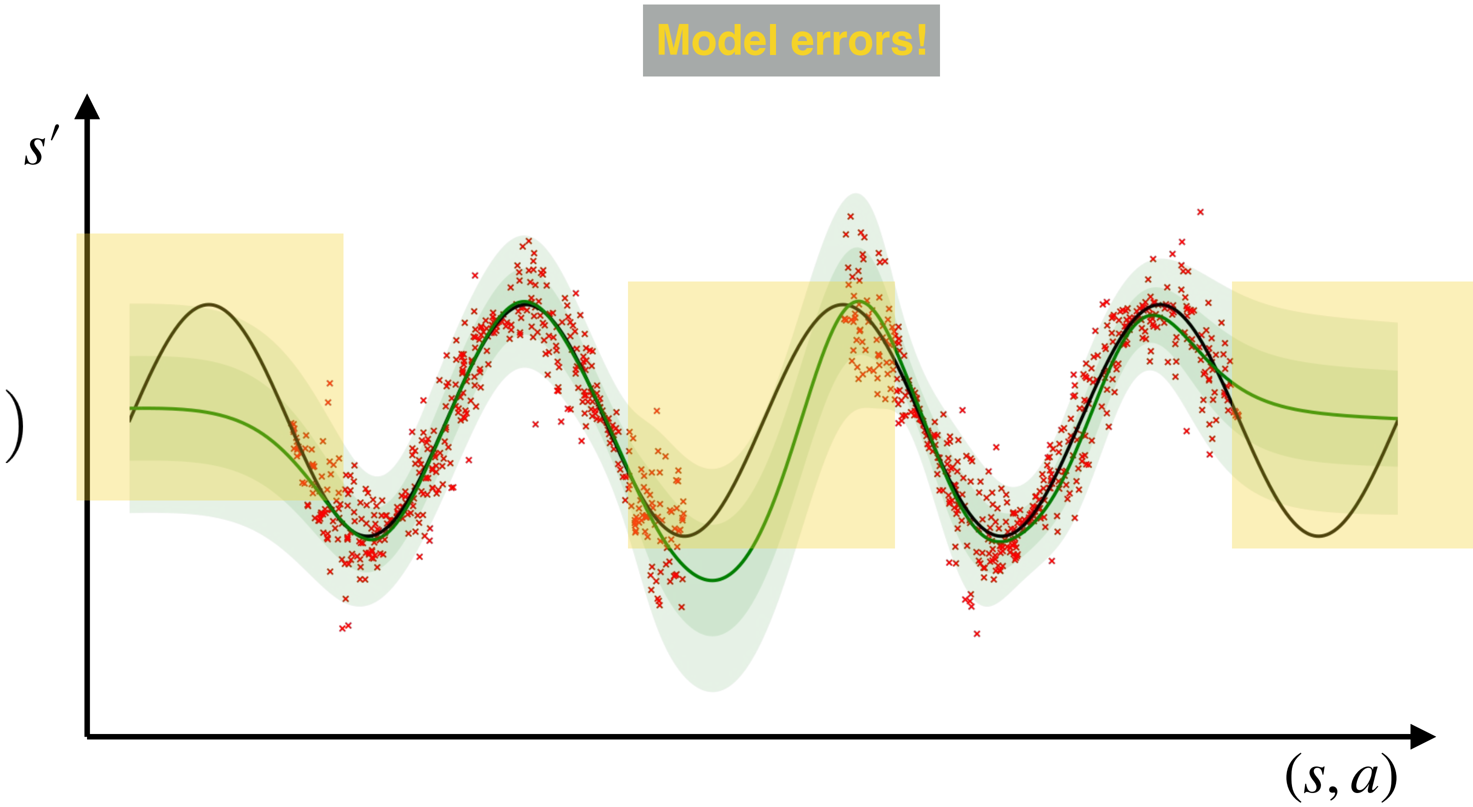


There is a unique answer for s' (no stochasticity) but I do not know it due to lack of data!

Epistemic uncertainty in Model Learning

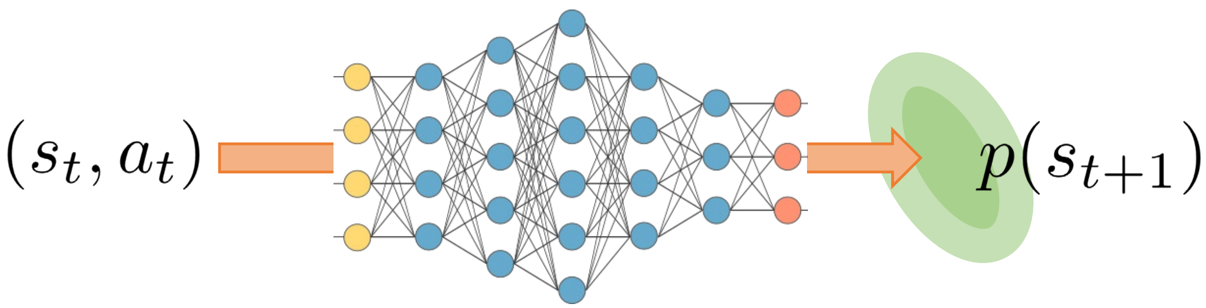


Fitting a neural network that parametrizes a distribution

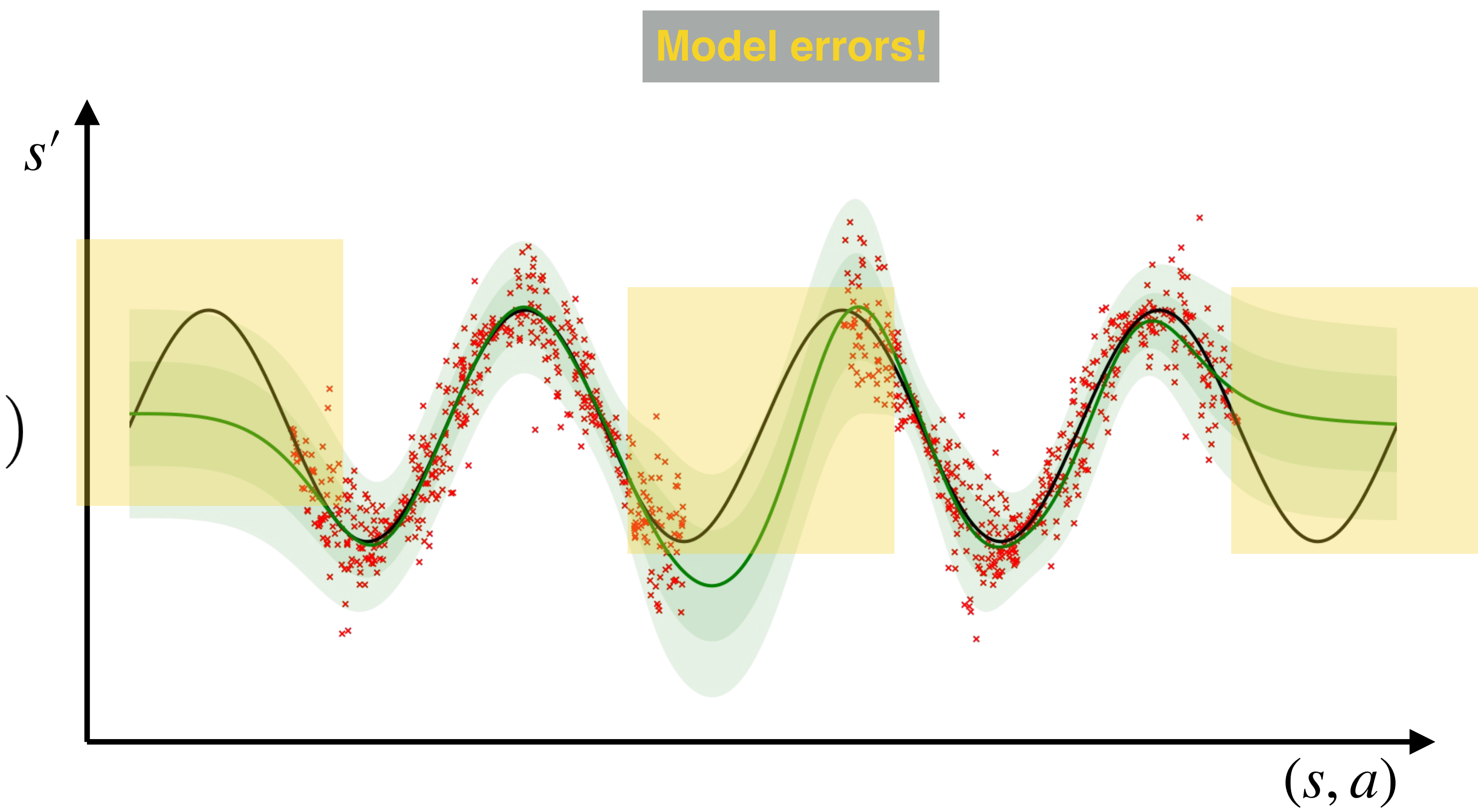


There is a unique answer for s’ (no stochasticity) but I do not know it due to lack of data!
Predicting a distribution won’t help! The predictions will suffer from lack of data and will be wrong.

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!

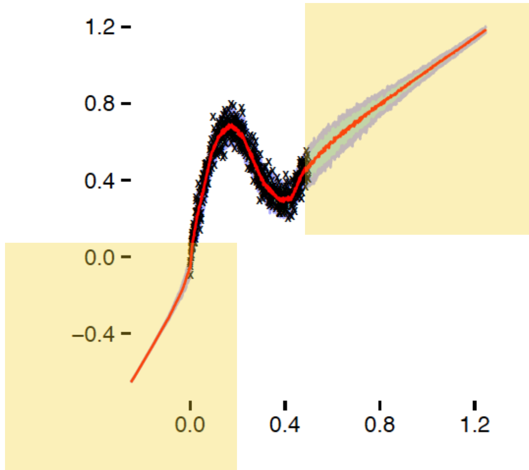
Bayes Rule

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

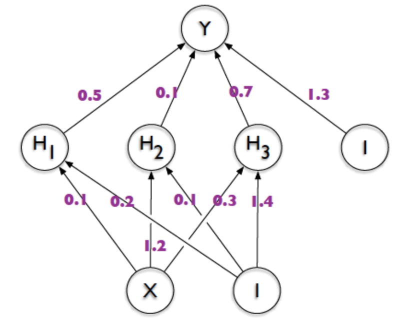
Hypotheses here are weights for our learning model, i.e., weights of our neural networks that learns the transition dynamics

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)



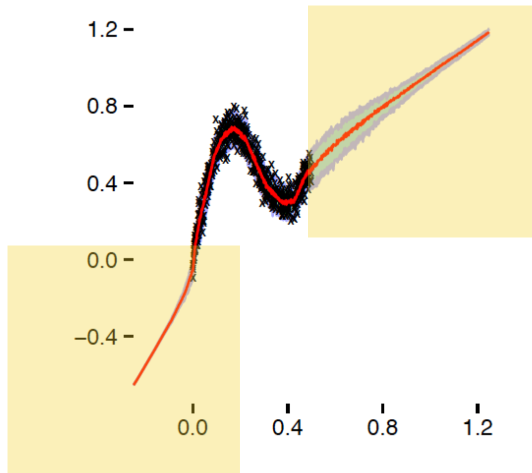
regression network



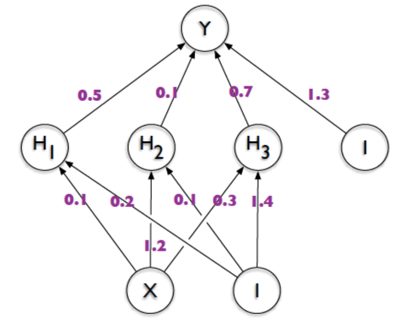
$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \big(P(D \mid \phi) + \log P(\phi)\big)$$

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

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

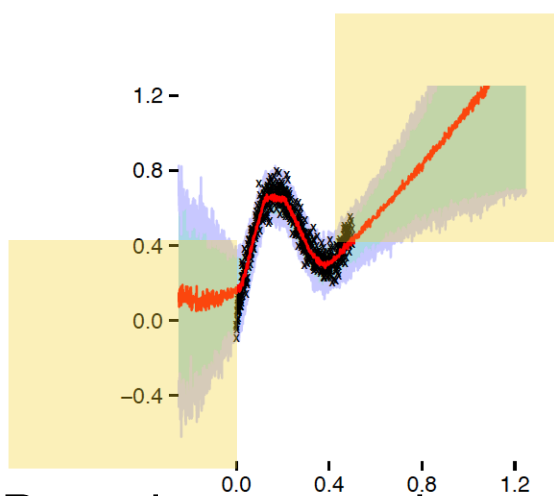


regression network



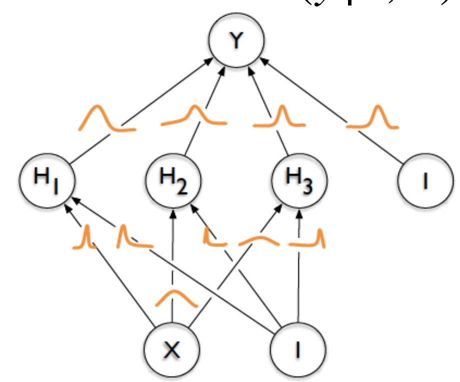
$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left(P(D \mid \phi) + \log P(\phi)\right)$$

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



Bayesian regression network

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

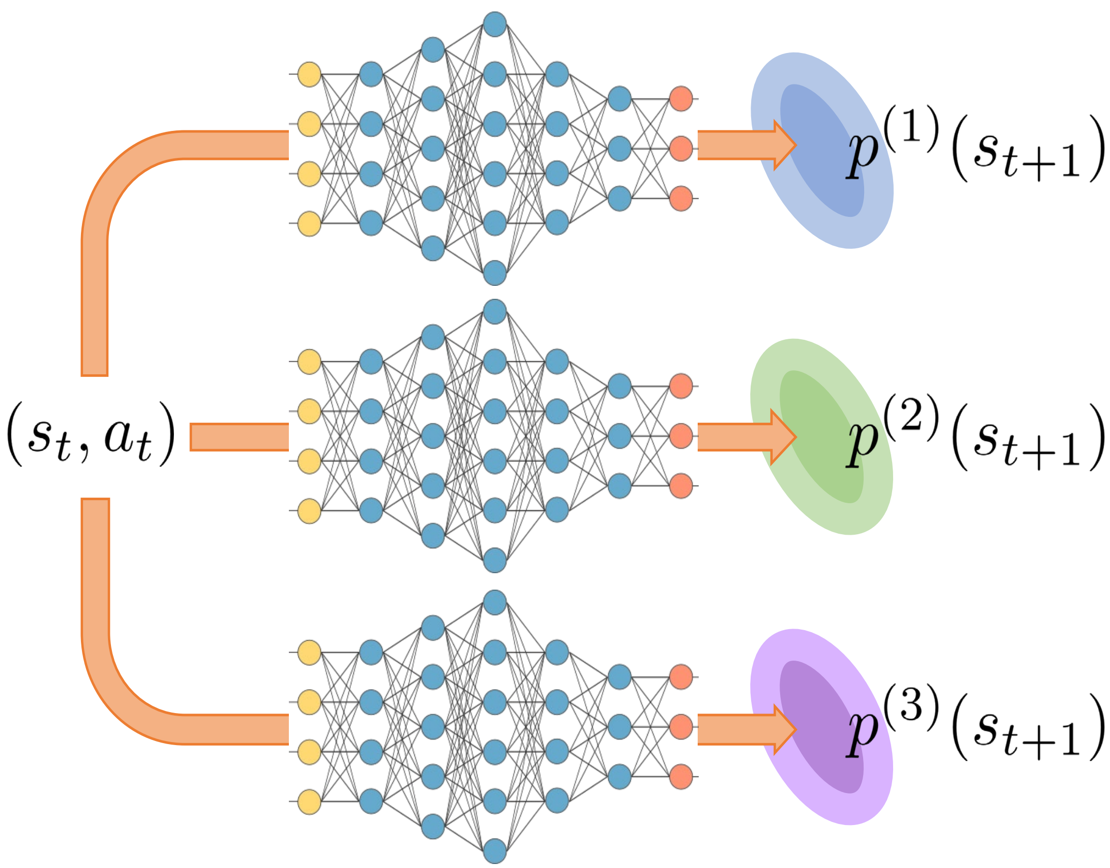


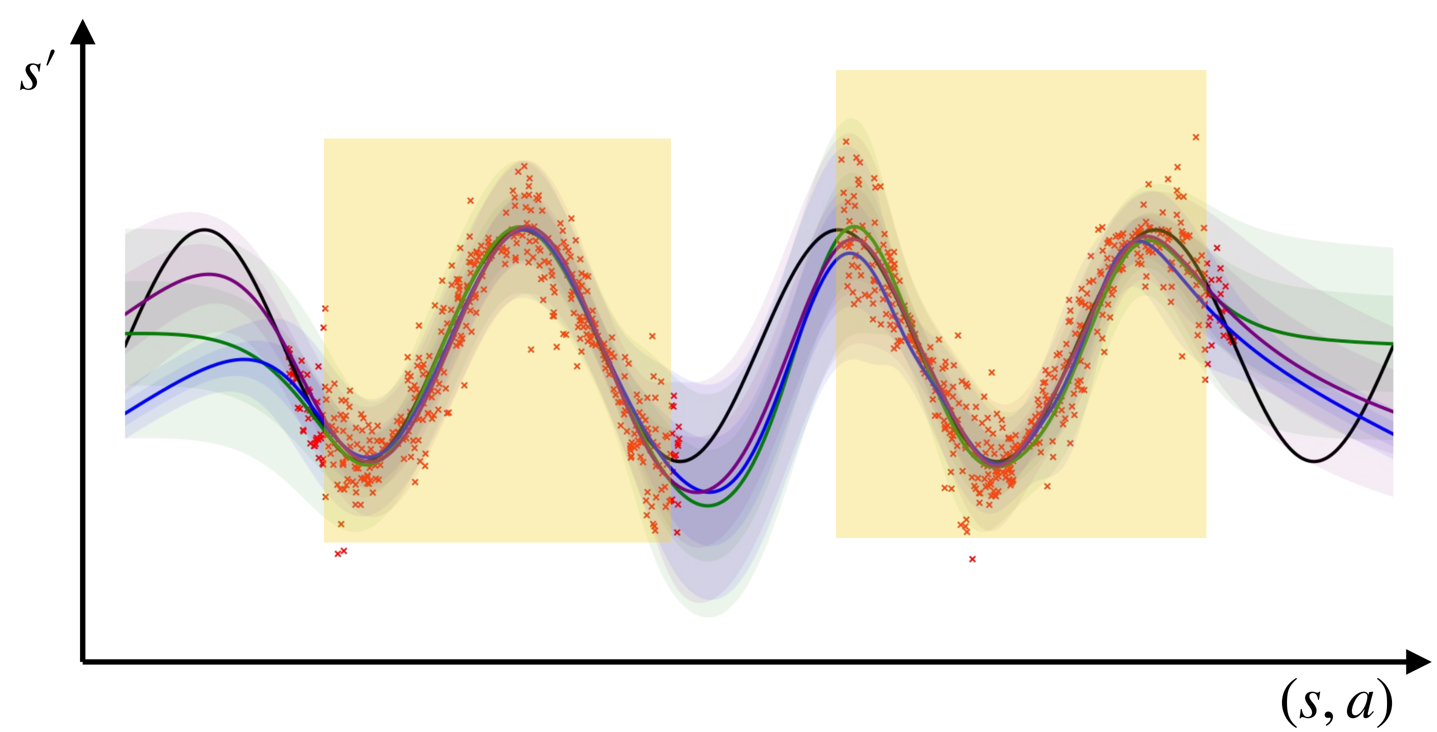
$P(\phi \mid D)$

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 (later lecture)

NN Ensembles for representing Epistemic uncertainty

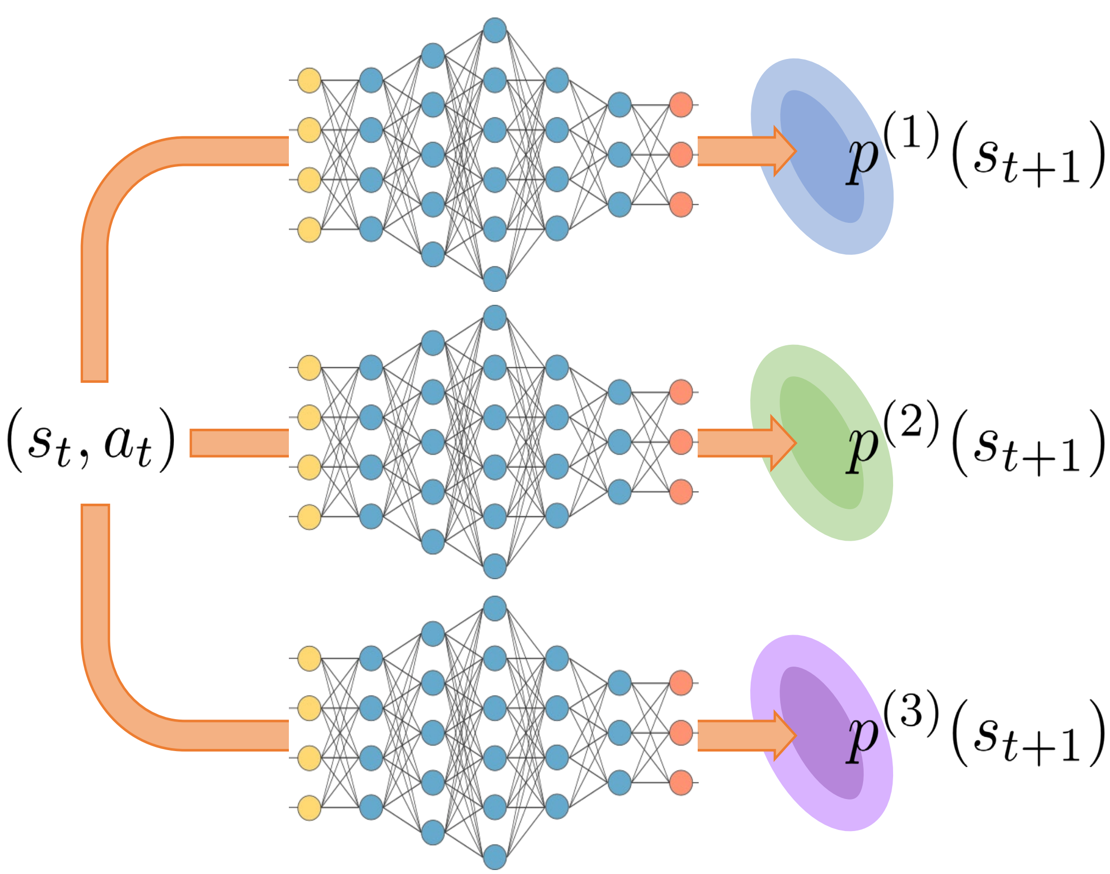
- Neural network Ensembles are a good approximation to Bayesian Nets.
- Instead of having explicit posteriors distributions for each neural net parameter, you just have a small set of neural nets, *each trained on separate data*.
    - On the data they have seen, they all agree (low entropy of predictions)

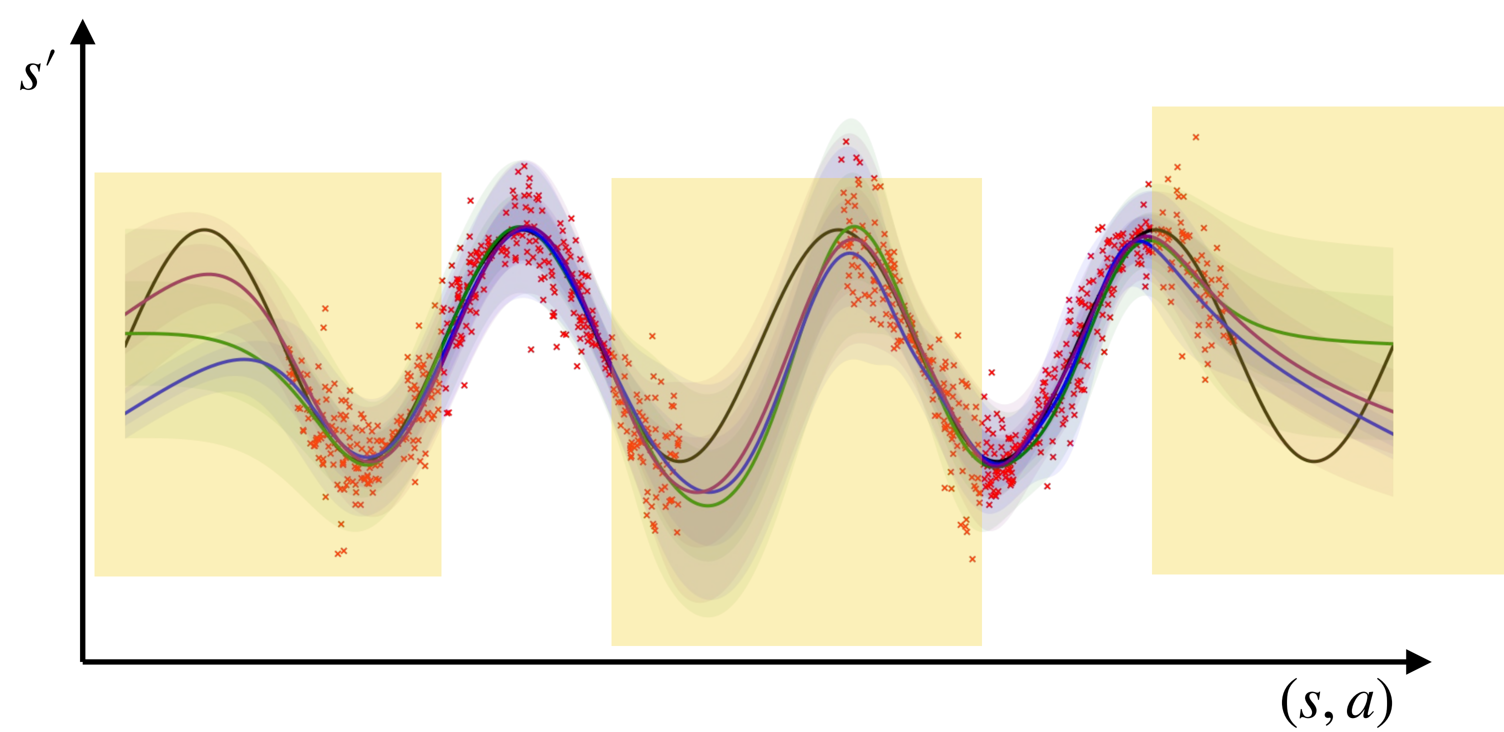




NN Ensembles for representing Epistemic uncertainty

- 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)





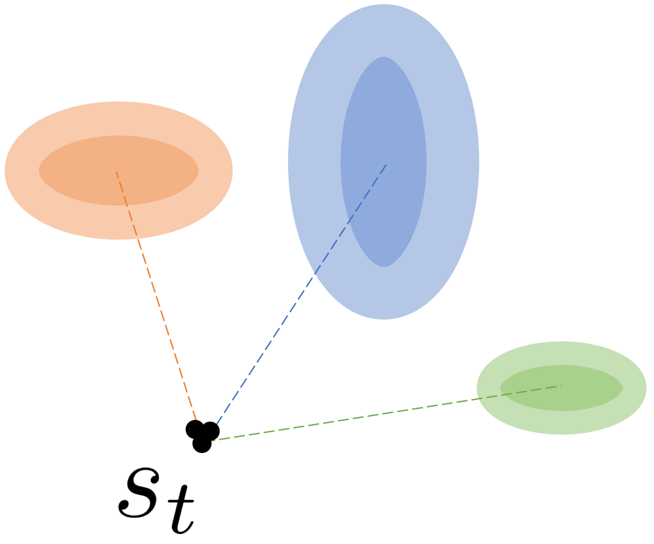
Propagating Particles

s_t

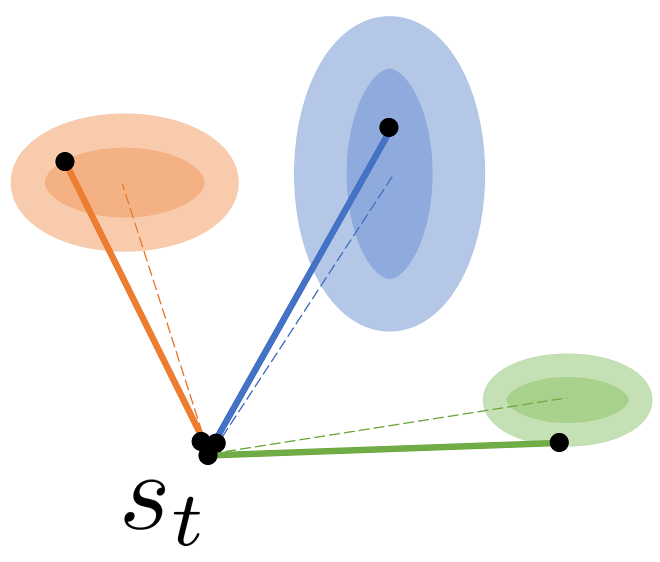
Propagating Particles

s_t

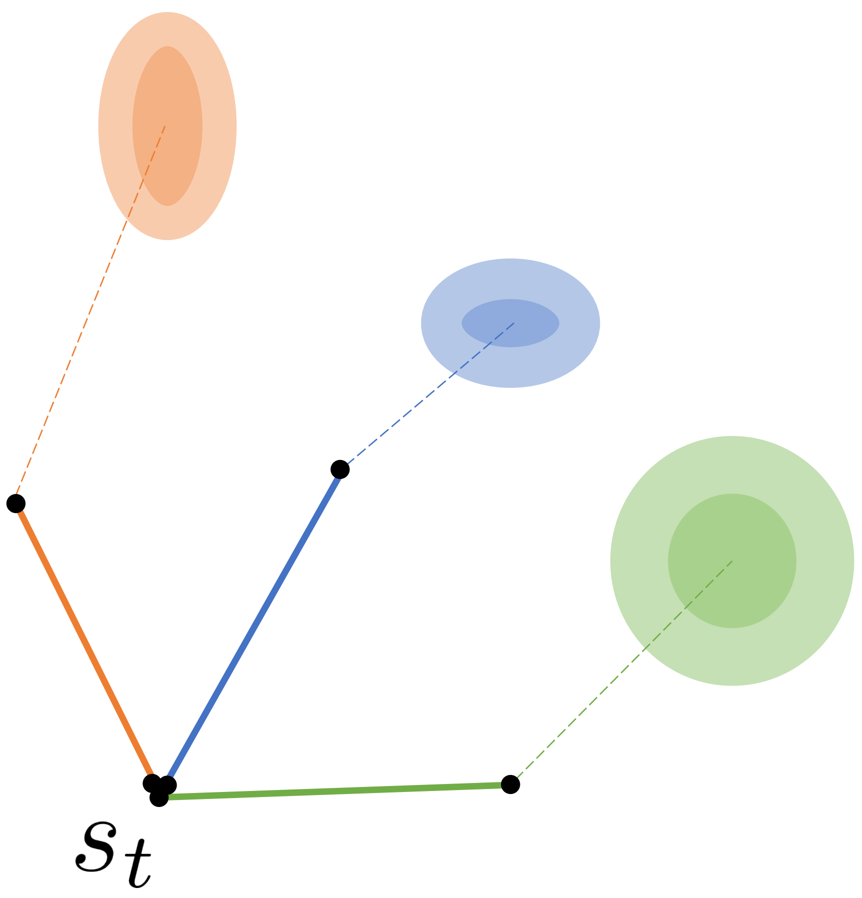
Propagating Particles



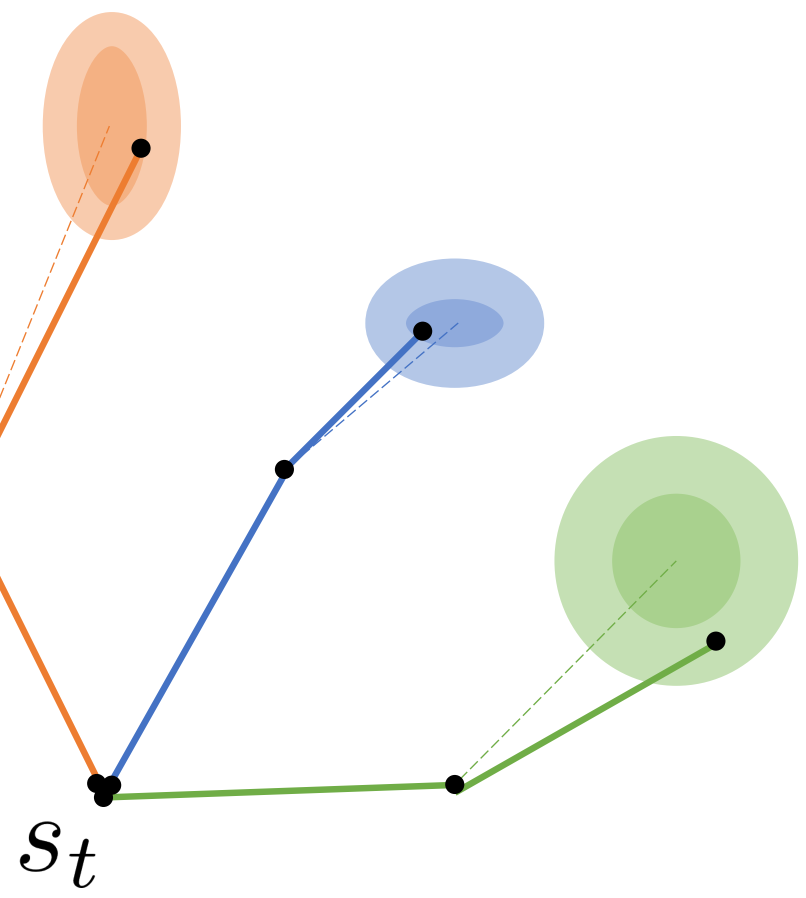
Propagating Particles



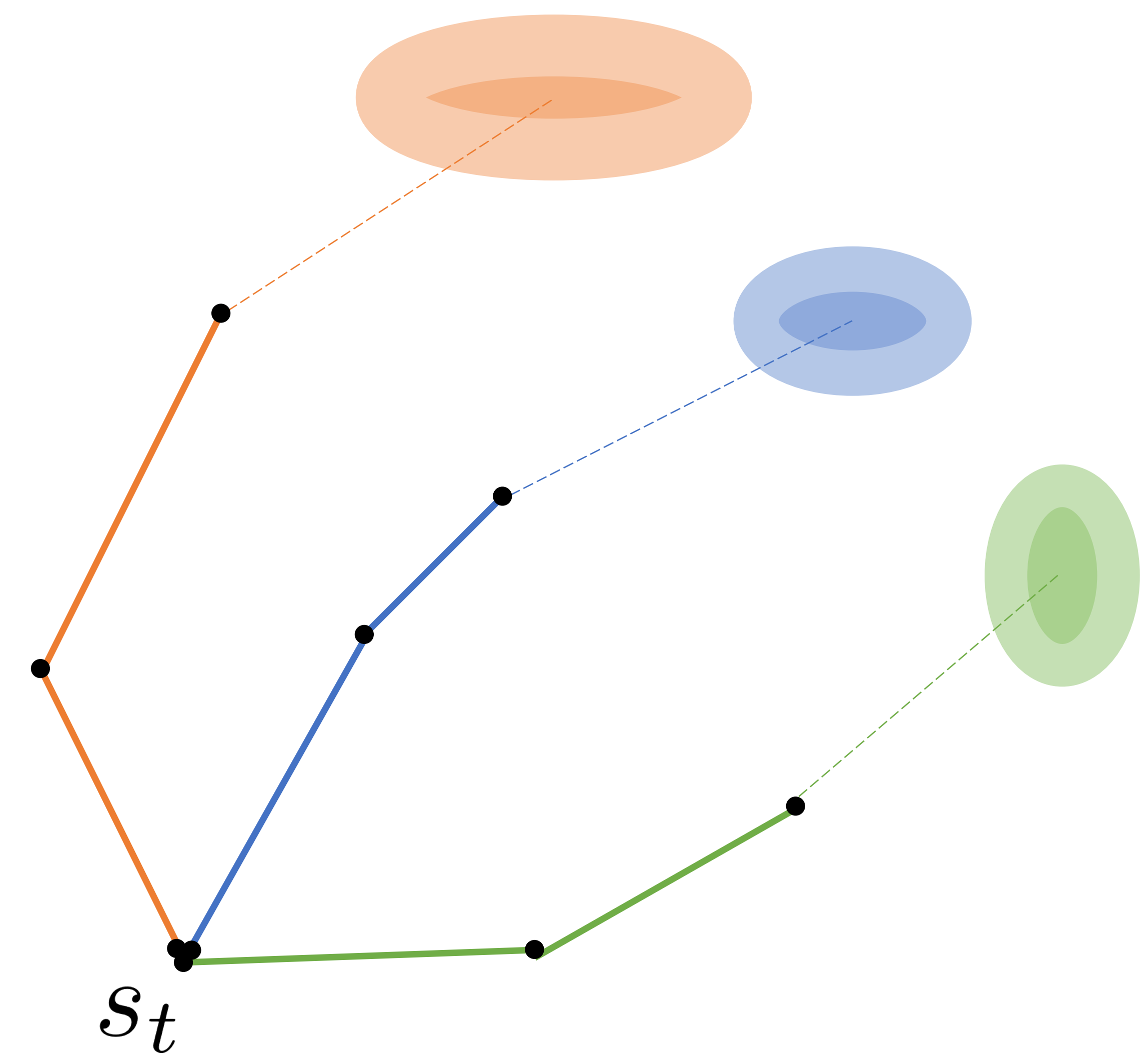
Propagating Particles



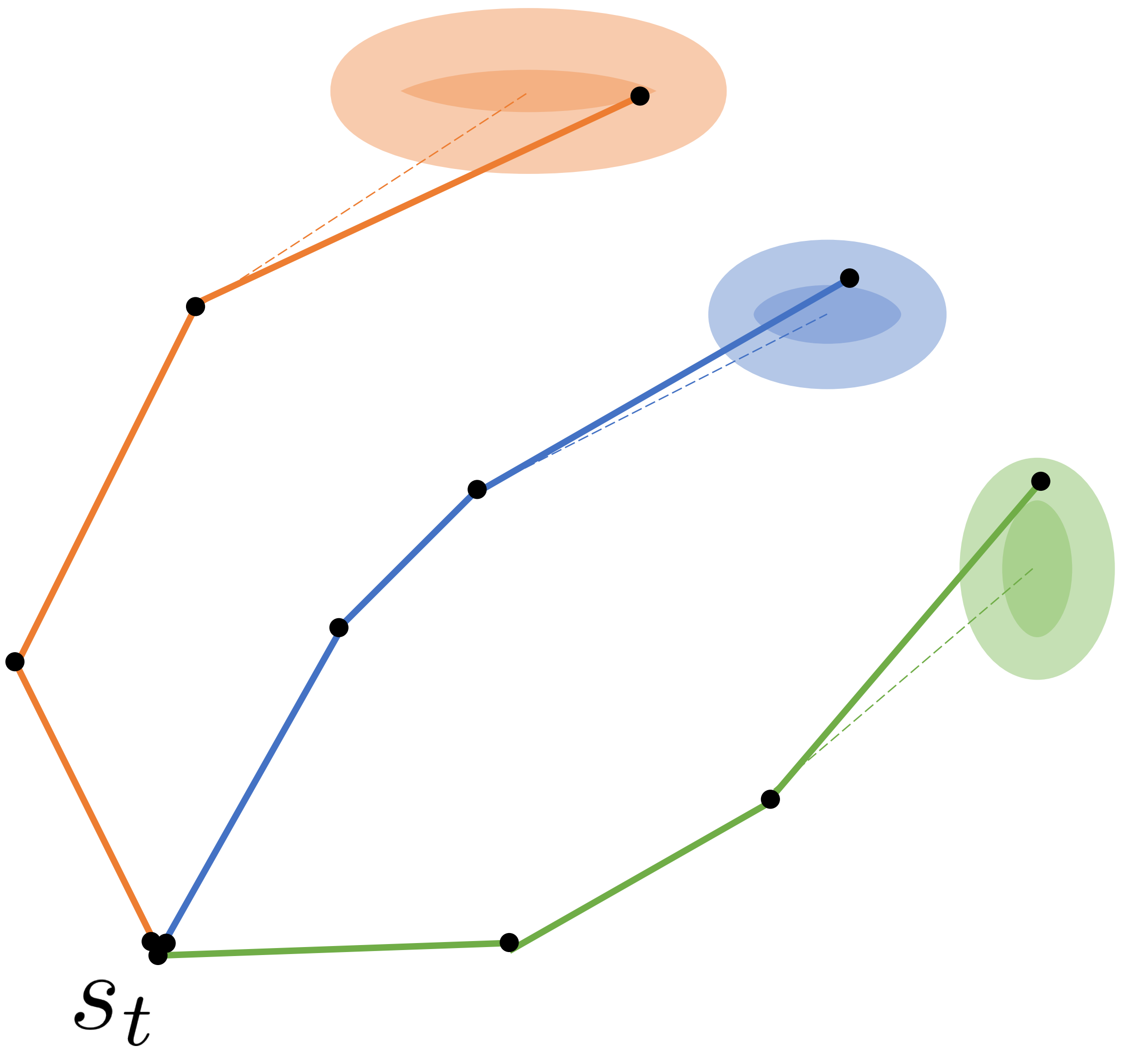
Propagating Particles



Propagating Particles

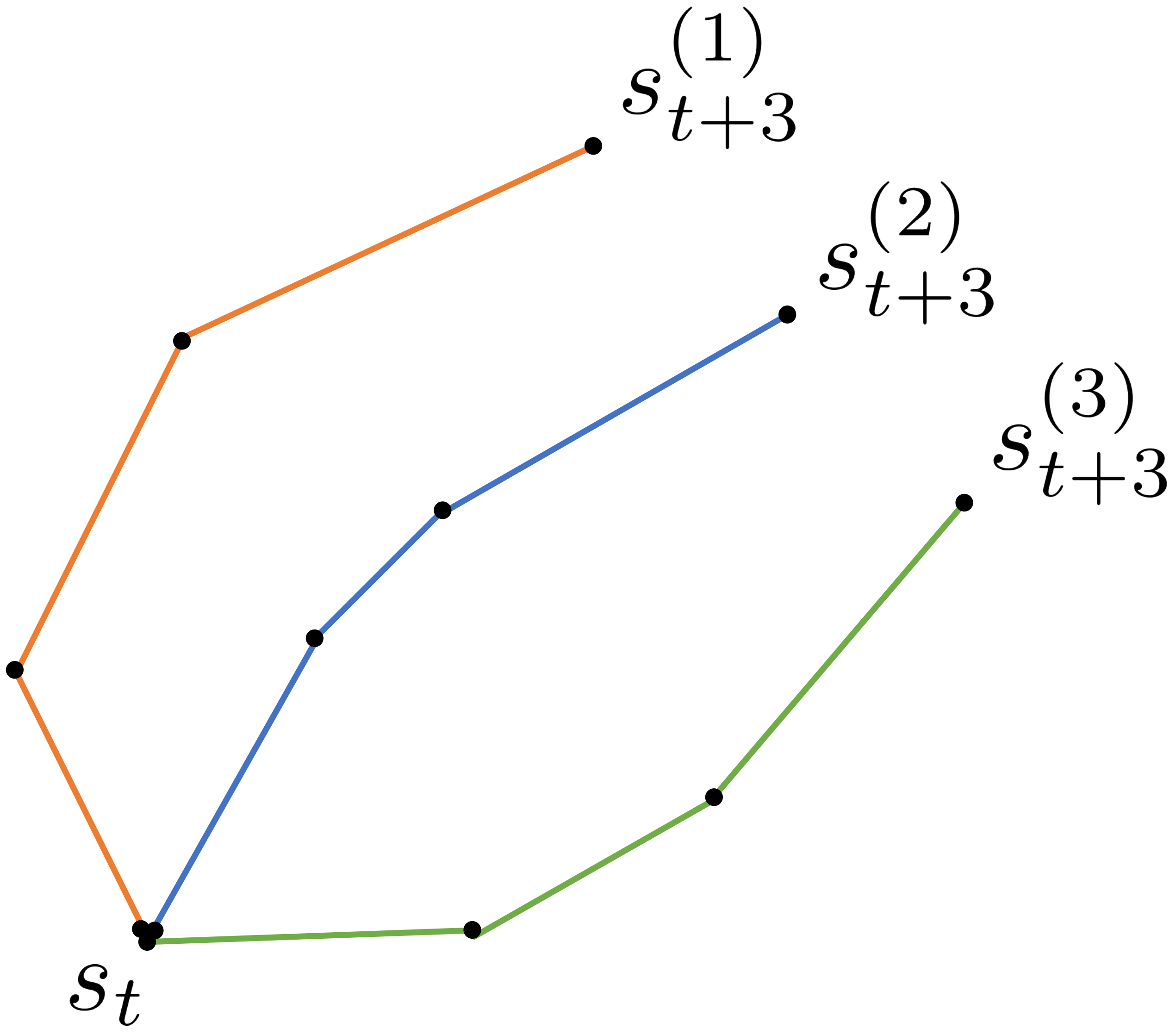


Propagating Particles



Propagating Particles

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



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

Results

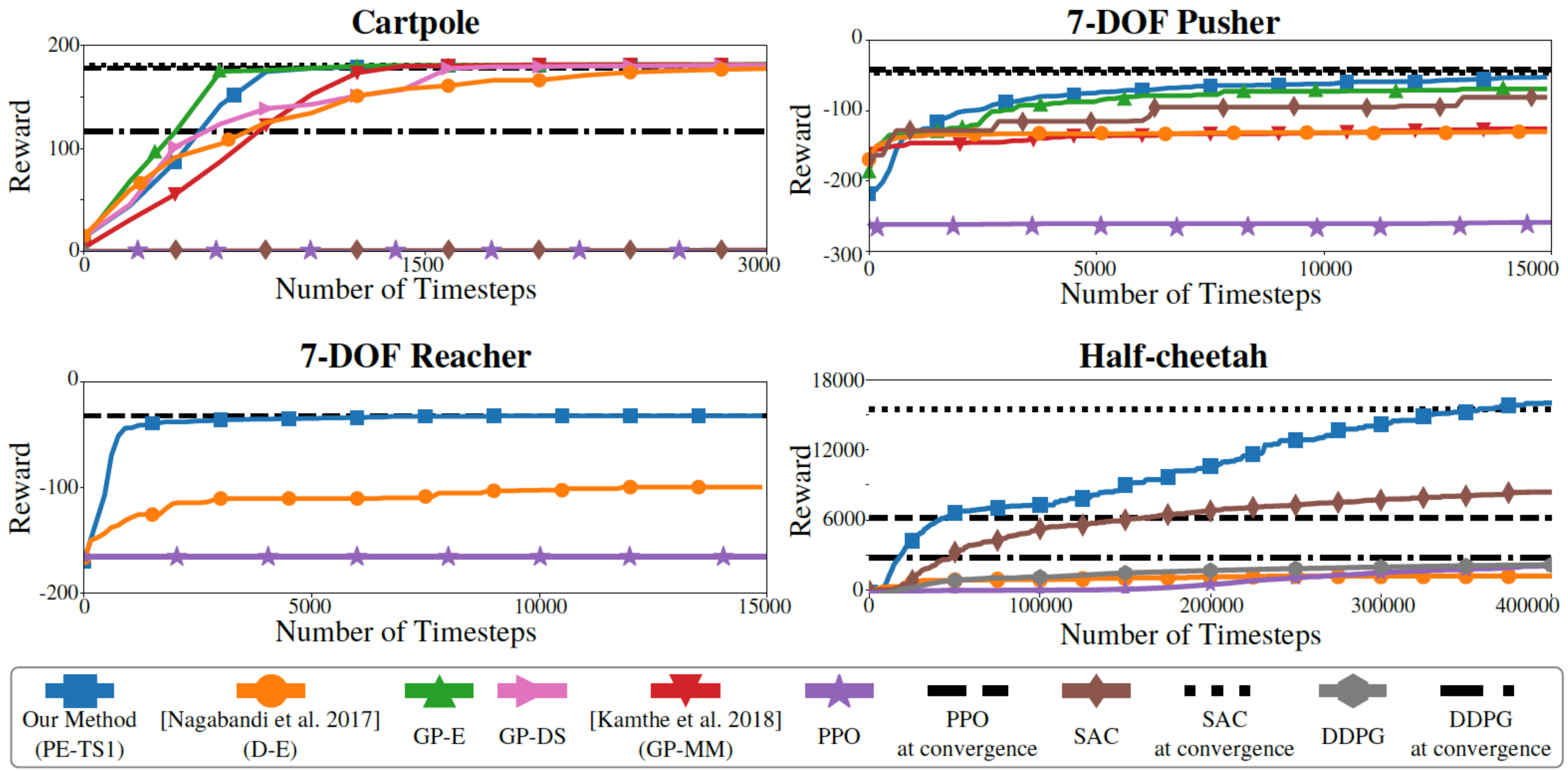
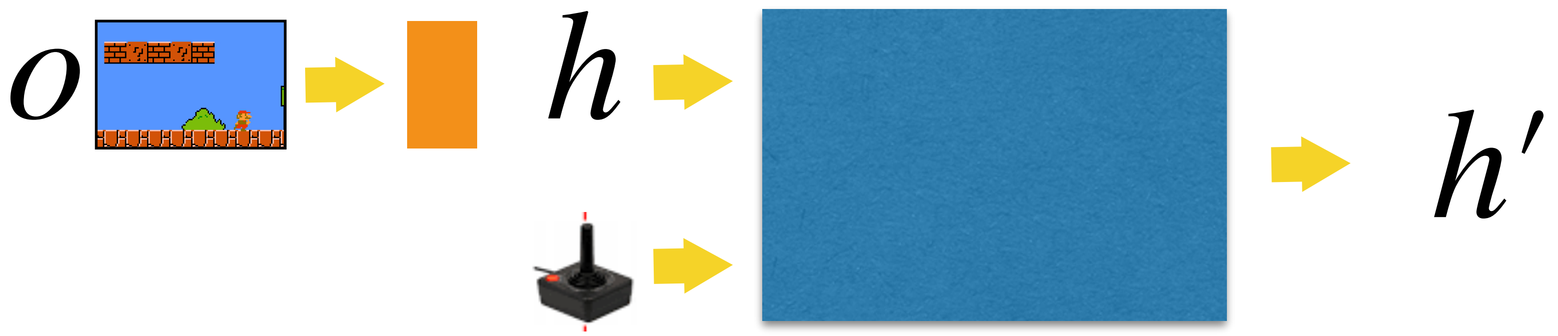


Figure 3: Learning curves for different tasks and algorithm. For all tasks, our algorithm learns in under 100K time steps or 100 trials. With the exception of Cartpole, which is sufficiently low-dimensional to efficiently learn a GP model, our proposed algorithm significantly outperform all other baselines.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

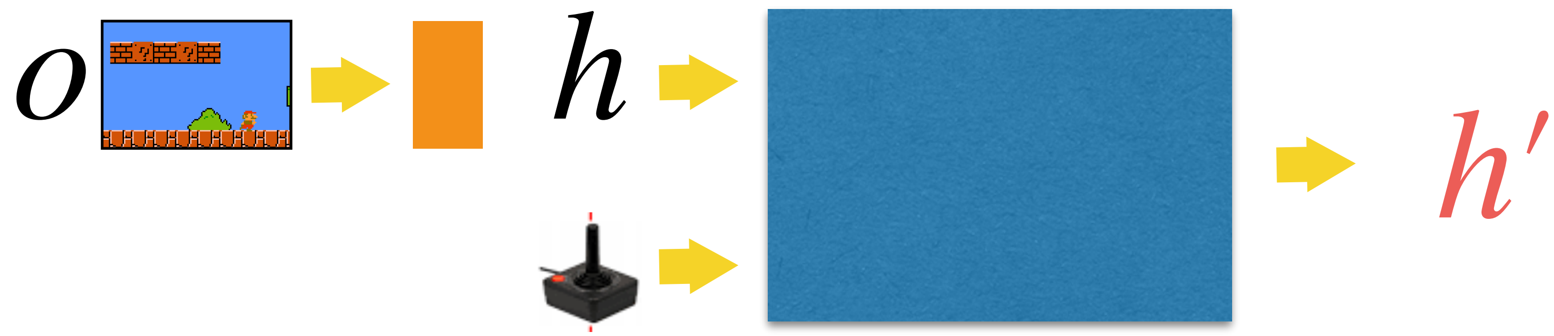
Model-based RL in sensory space

Model Learning - 3 Qs always in mind

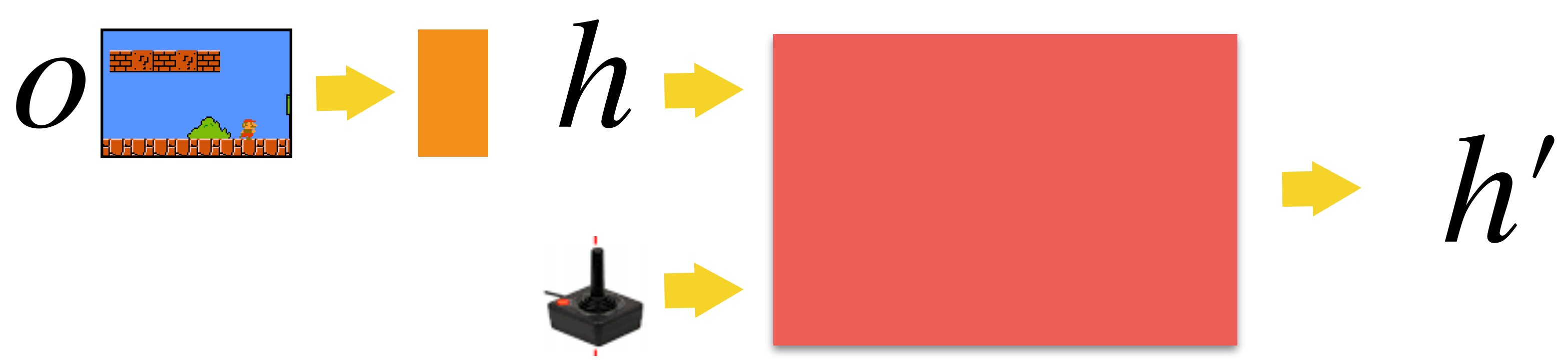


Model Learning - 3 Qs always in mind

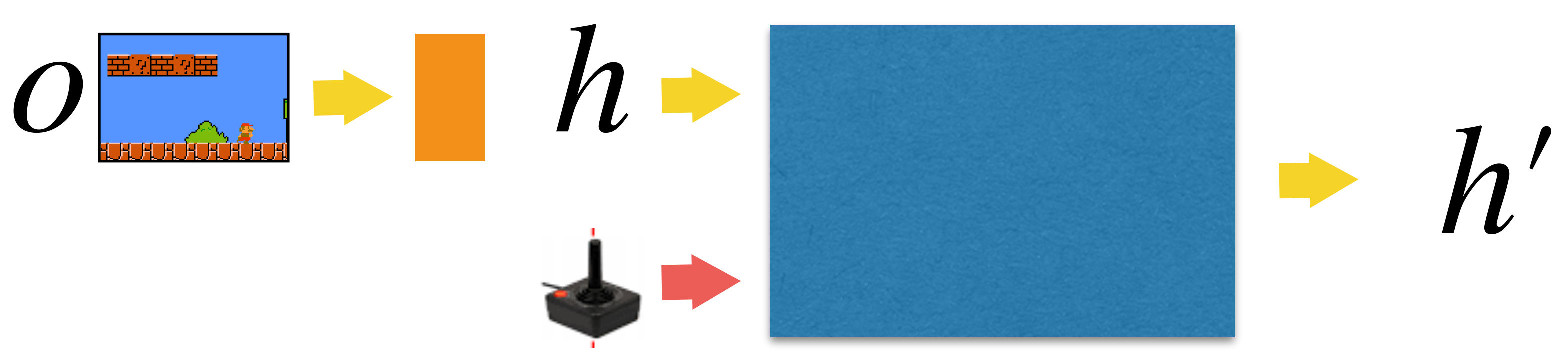
- What shall we be predicting?



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

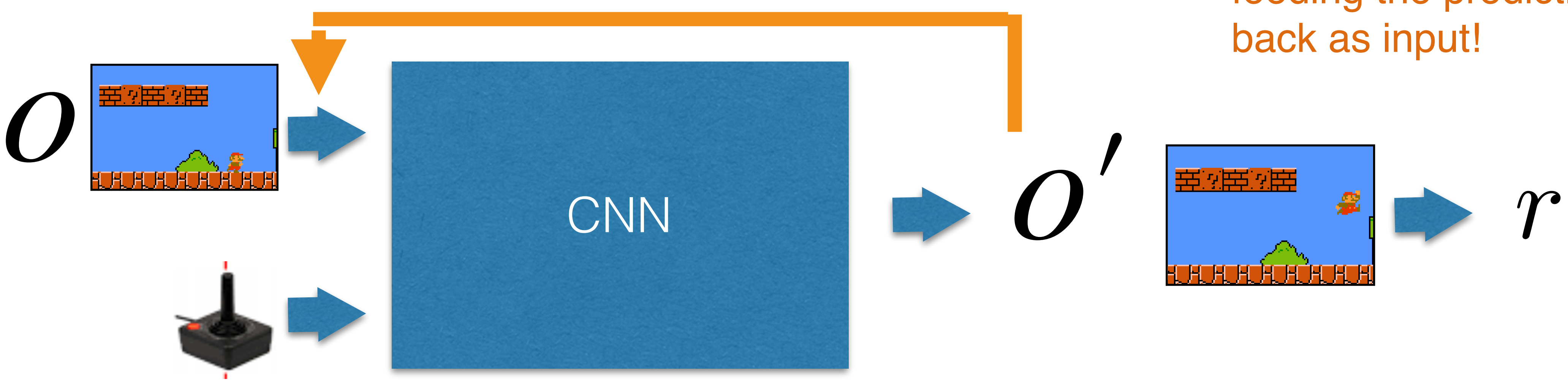


- What is the action representation?



Model learning in image space

Unroll the model by feeding the prediction back as input!

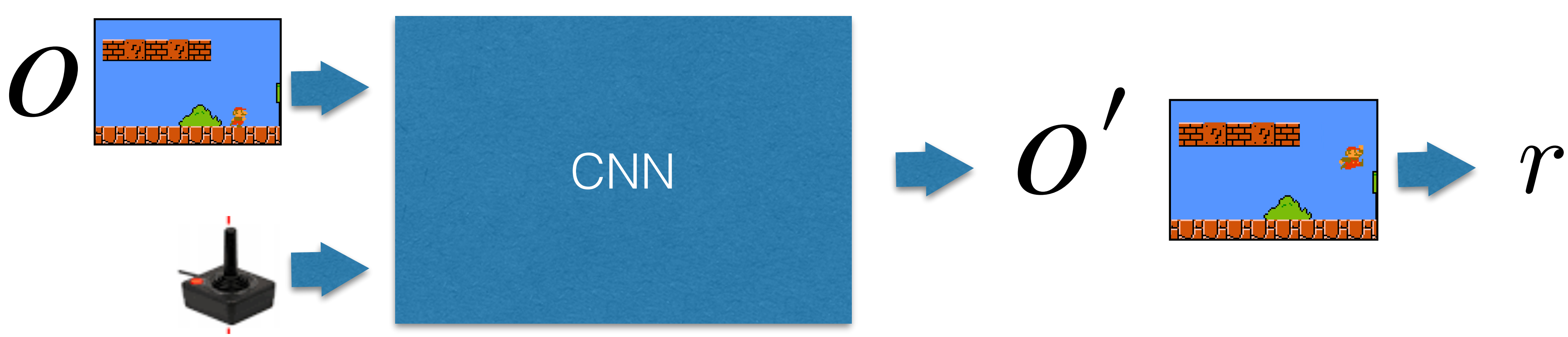


MANY different rewards can be computed from the future visual observation, e.g., make Mario jump, make Mario move to the right, to the left, lie down, make Mario jump on the well and then jump back down again etc..

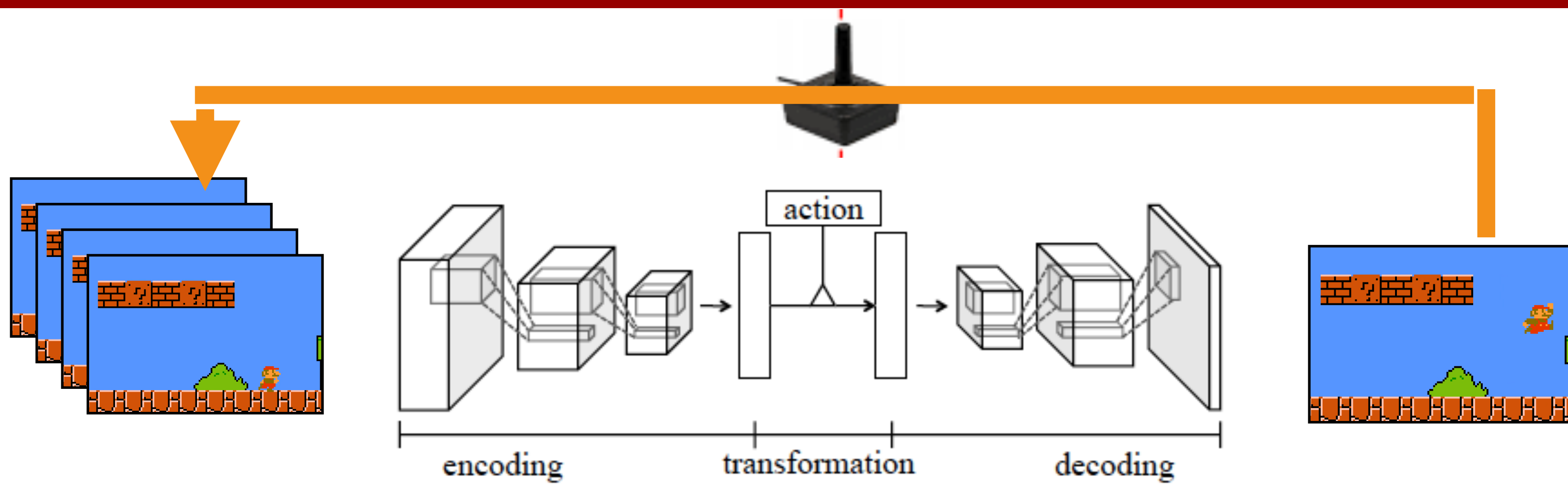
Action-Conditional Video Prediction using Deep Networks in Atari Games

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
- (Use this frame prediction to come up with an exploratory behavior in DQN: choose the action that leads to frames that are most dissimilar to a buffer of recent frames)



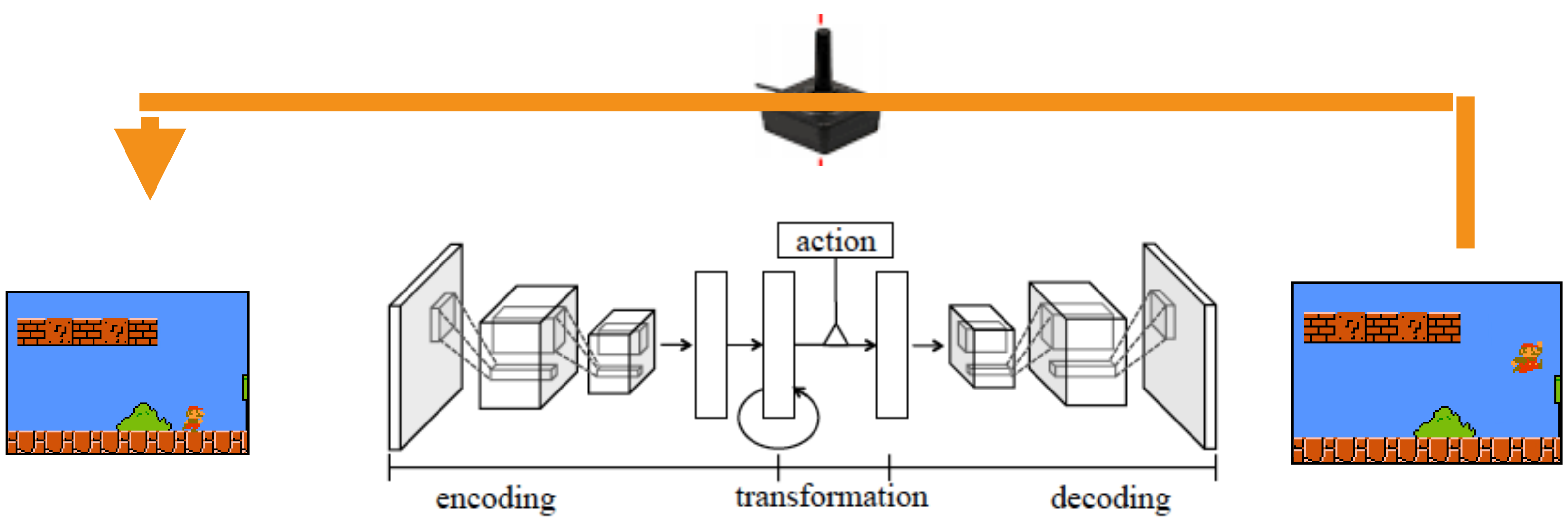
Model learning in image space



(a) Feedforward encoding

Multiplicative interactions between action and hidden state (not concatenation):

$$h^{dec}_{t,i} = \sum_{j,l} W_{ijl} h^{enc}_{t,j} a_{t,l} + b_i,$$



(b) Recurrent encoding

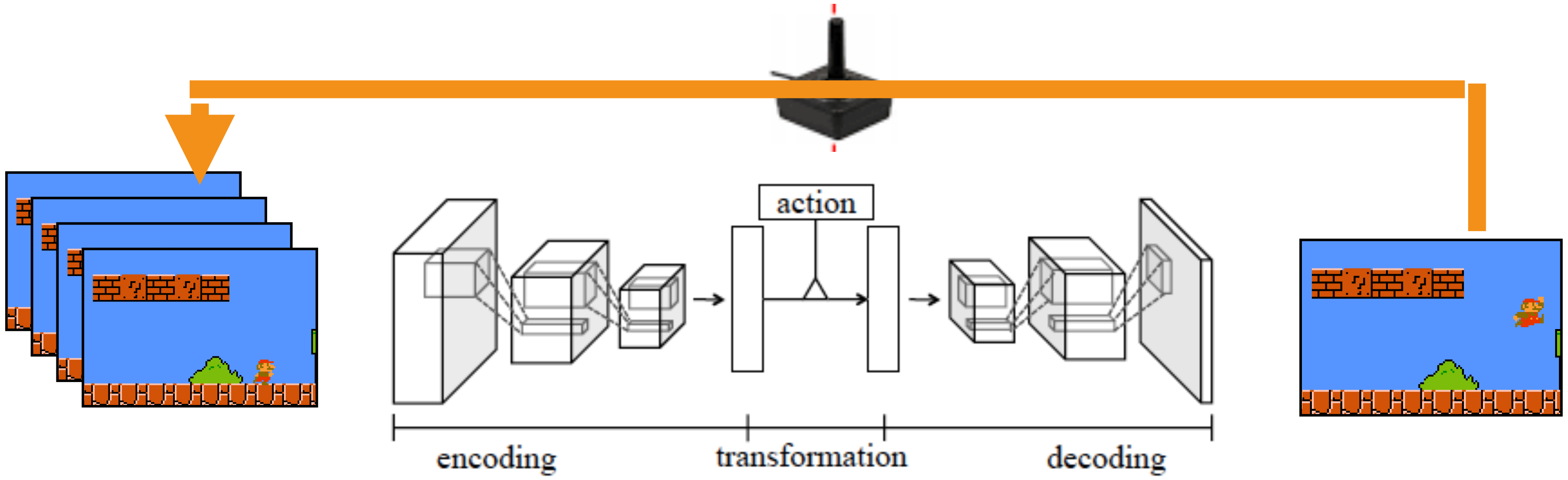
Unroll the model by feeding the prediction back as input!

Progressively increase k (the length of the conditioning history) so that we do not feed garbage predictions as input to the predictive model:

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

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

How to train our model so that unrolling works



(a) Feedforward encoding

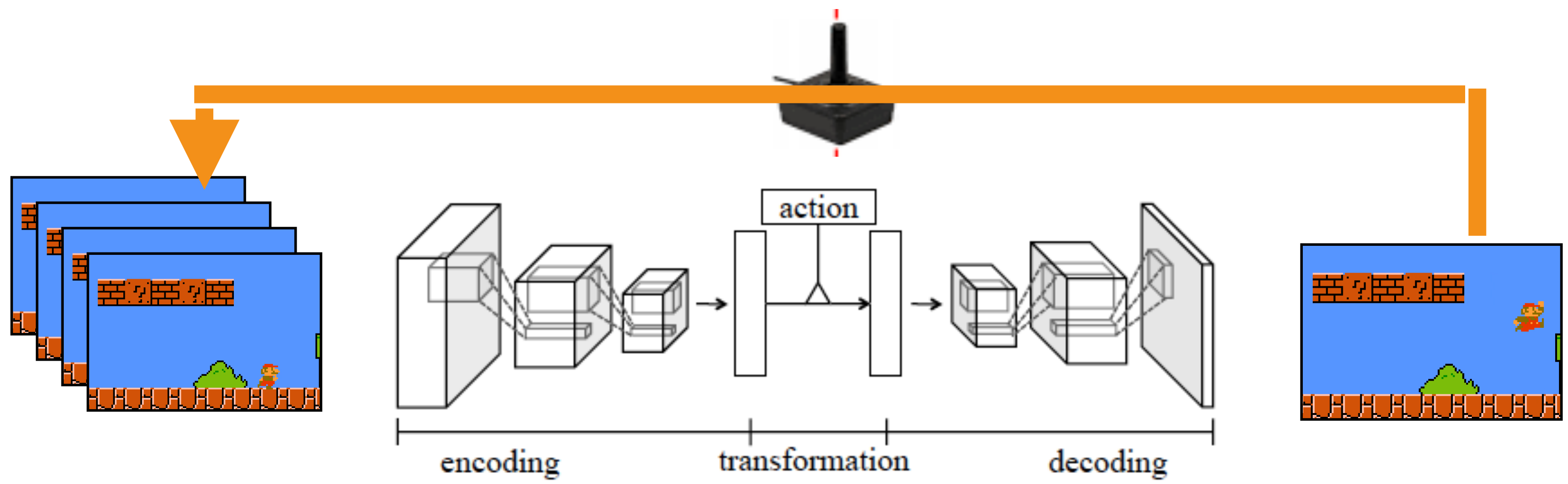
Q: Can I train my model using tupples (o,a,o') and I test time unroll it in time?
A: no, we will have distribution shift, same as in imitation learning: tiny mistakes will soon cause the model to diverge

Solution: Progressively increase the unroll length k at training time so that the model learns to correct 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

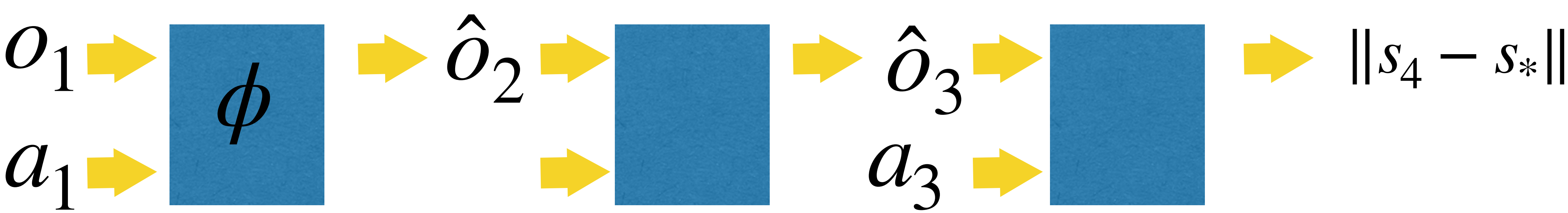


(a) Feedforward encoding

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

Solution: Progressively increase the unroll length k at training time so that the model learns to correct its mistakes:

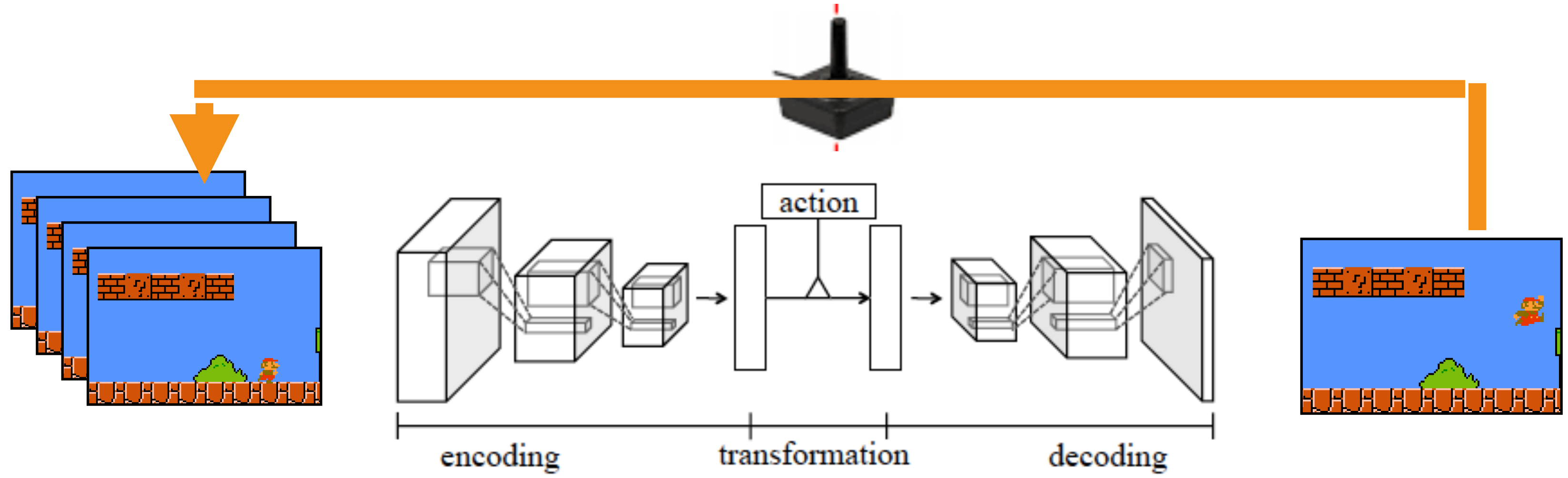
$$\mathcal{L}(\phi) = \frac{1}{N}\sum_{i=1}^{N} \|f(a_2^i, f(a_1^i, o_1^i; \phi); \phi) - o_3^i\| + \|f(a_1^i, o_1^i; \phi) - o_2^i\|$$



$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{o}_2^i - o_2^i\|$$

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

How to train our model so that unrolling works

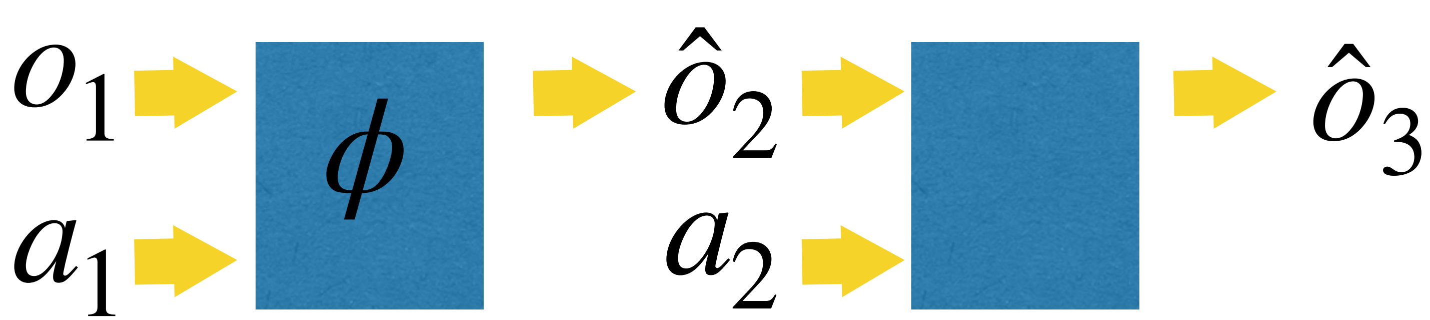


(a) Feedforward encoding

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

Solution: Progressively increase the unroll length k at training time so that the model learns to correct its mistakes:

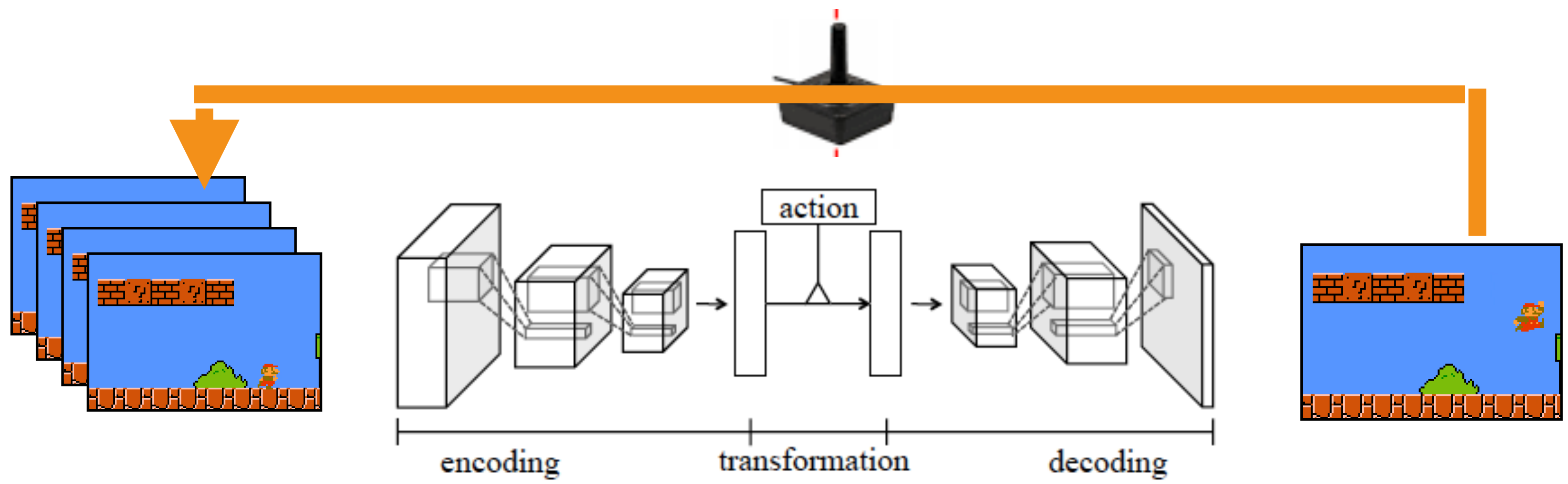
$$\mathscr{L}(\phi) = \frac{1}{N}\sum_{i=1}^{N} \|f(a_2^i, f(a_1^i, o_1^i; \phi); \phi) - o_3^i\| + \|f(a_1^i, o_1^i; \phi) - o_2^i\|$$



$$\mathscr{L} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{o}_2^i - o_2^i\|$$

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 I test time unroll it in time?
A: no, we will have distribution shift, same as in imitation learning: tiny mistakes will soon cause the model to diverge

Solution: Progressively increase the unroll length k at training time so that the model learns to correct its mistakes:

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

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 task-relevant)

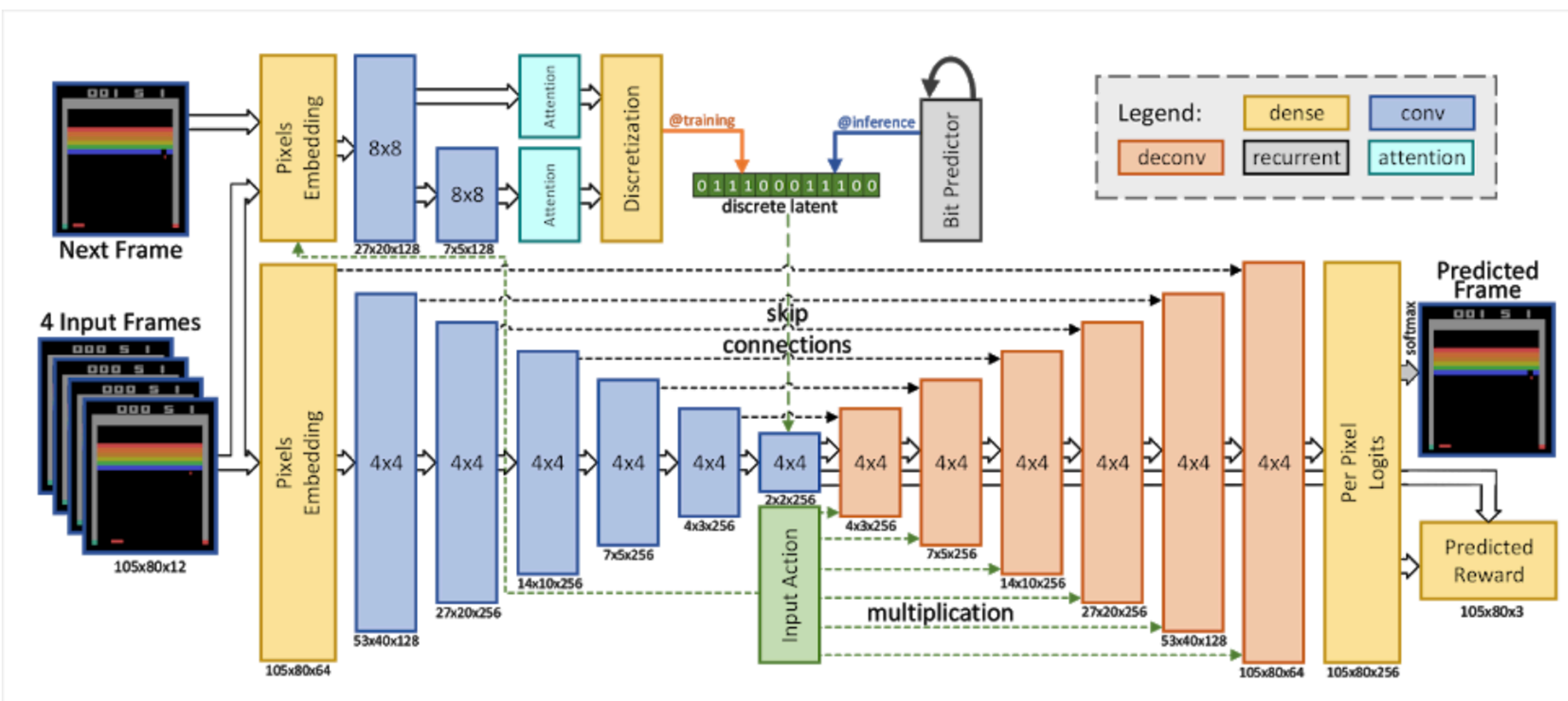
Model-Based Reinforcement Learning for Atari

Łukasz Kaiser Ryan Sepassi
Google Brain

Henryk Michalewski Piotr Miłoś
University of Warsaw

Błażej Osiński
deepsense.ai

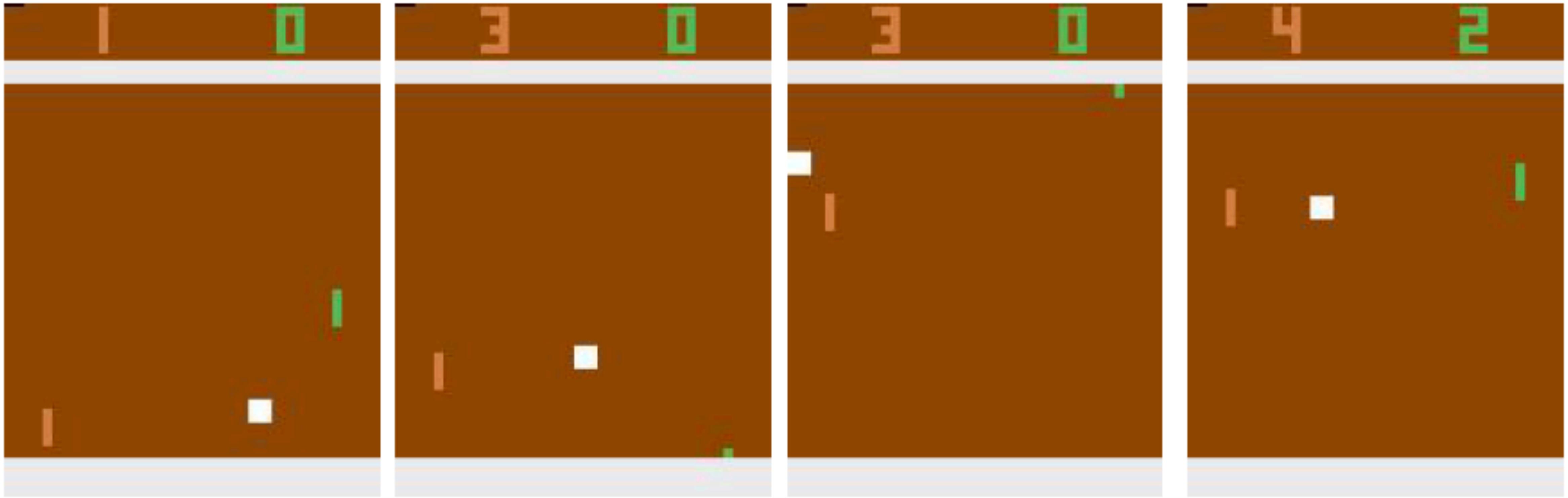
Similar architecture as before but..



Reward-aware 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 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 :-( )

doesn’t this require super long unrolls?



results

Results

- Number of frames required to reach human performance

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

results

Predicting Raw Sensory Input (Pixels)

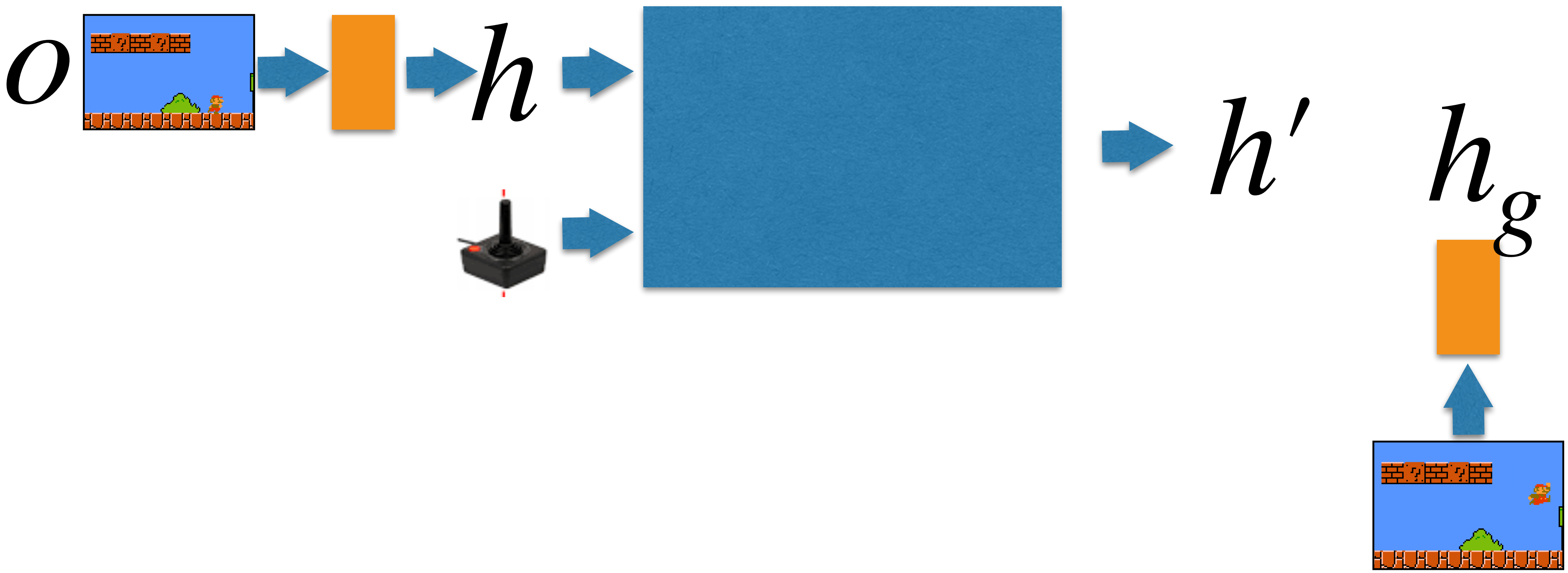
Should our prediction model be predicting the input observations?

- Observation prediction is difficult especially for high dimensional observations, such as images.
- Observation contains a lot of information unnecessary for planning, e.g., dynamically changing backgrounds that the agent cannot control and/or are irrelevant to the reward.

Prediction in a latent space

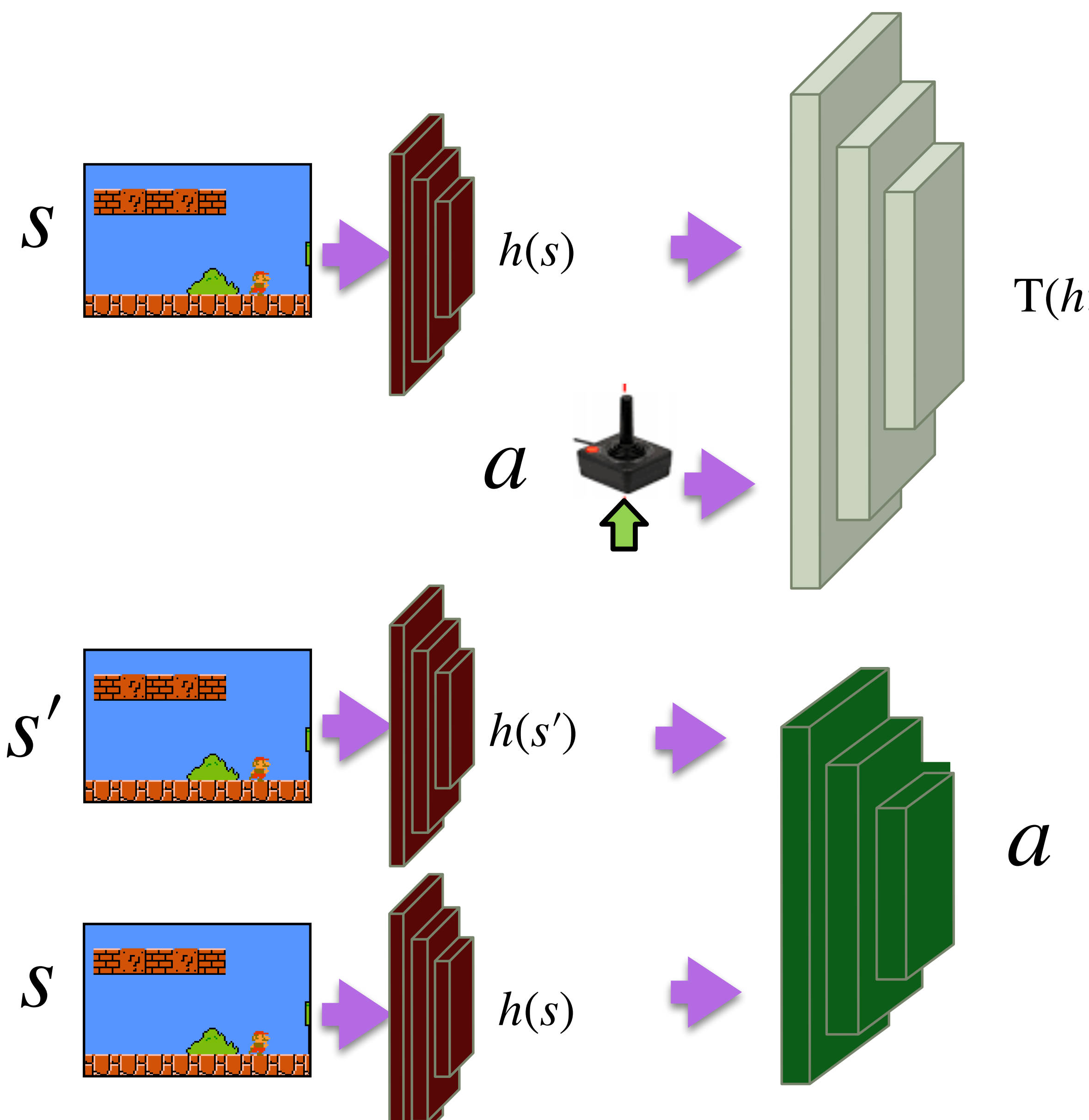
Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.

$$r = \exp(-\|h' - h_g\|)$$



Prediction in a latent space

Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.
One such feature encoding we have seen is the one that keep from the observation ONLY whatever is controllable by the agent.

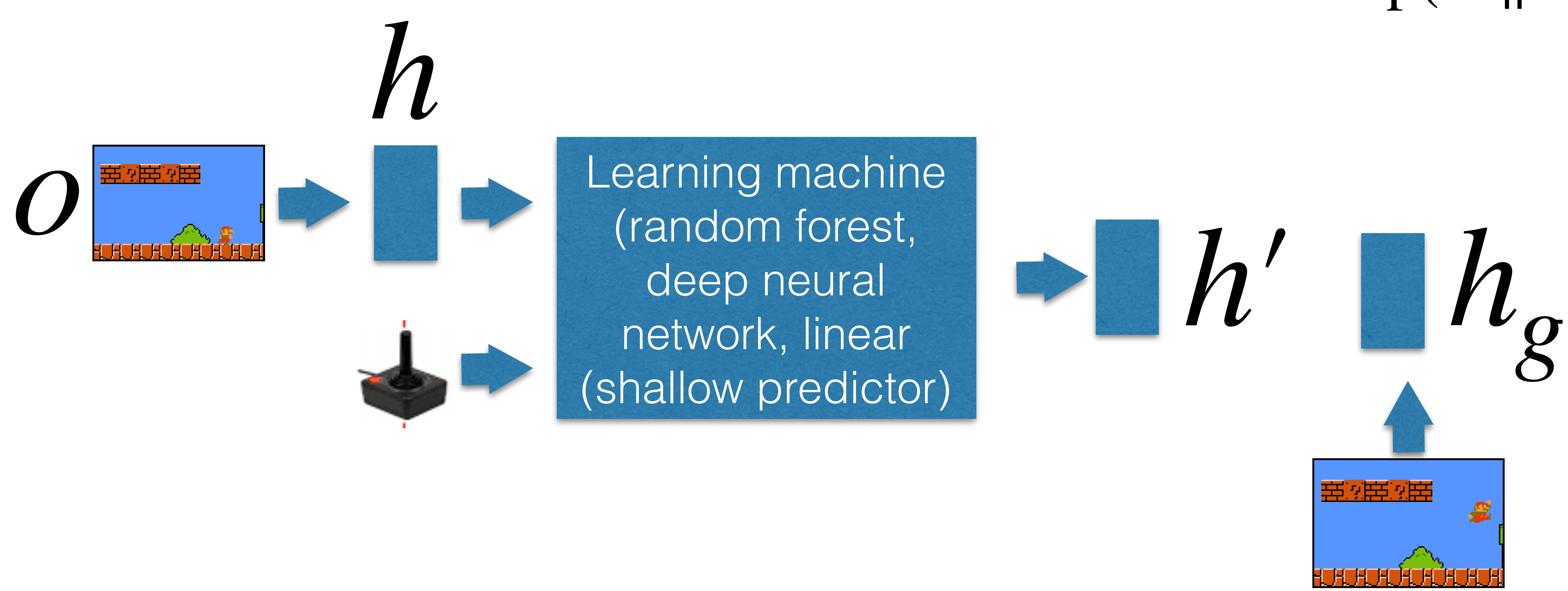


$$\min_{\theta,\phi} . \quad \|T(h(s), a; \theta) - h(s')\| + \|\mathrm{Inv}(h(s), h(s'); \psi) - a\|$$

Prediction in a latent space

Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.

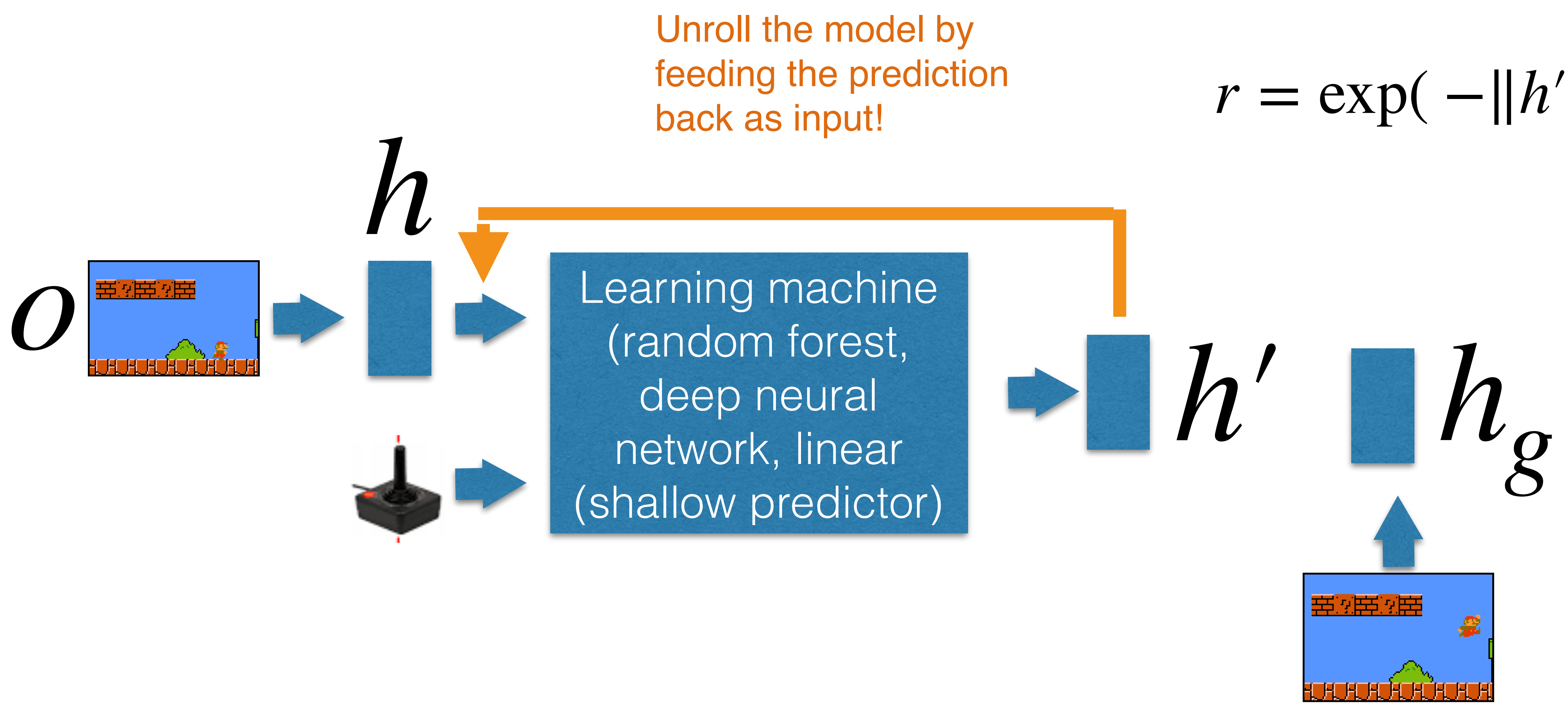
$$r = \exp(-\|h' - h_g\|)$$



Prediction in a latent space

Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.

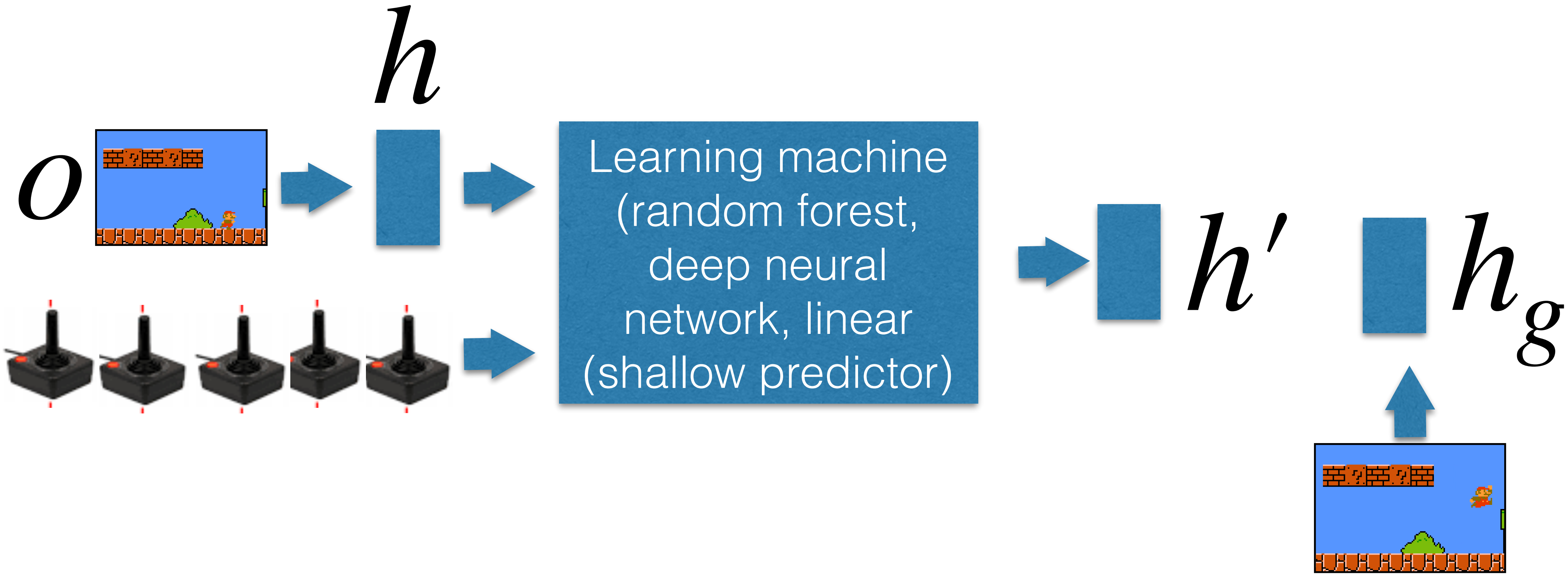
$$r = \exp(-\|h' - h_g\|)$$



Avoid or minimize unrolling

Unrolling quickly causes errors to accumulate. We can instead consider coarse models, where we input a long sequences of actions and predict the final embedding in one shot, without unrolling.

$$r = \exp(-\|h' - h_g\|)$$



Why model learning

- Online Planning at test time - Model predictive Control
- Model-based RL: training policies using simulated experience
- Efficient Exploration

Challenges

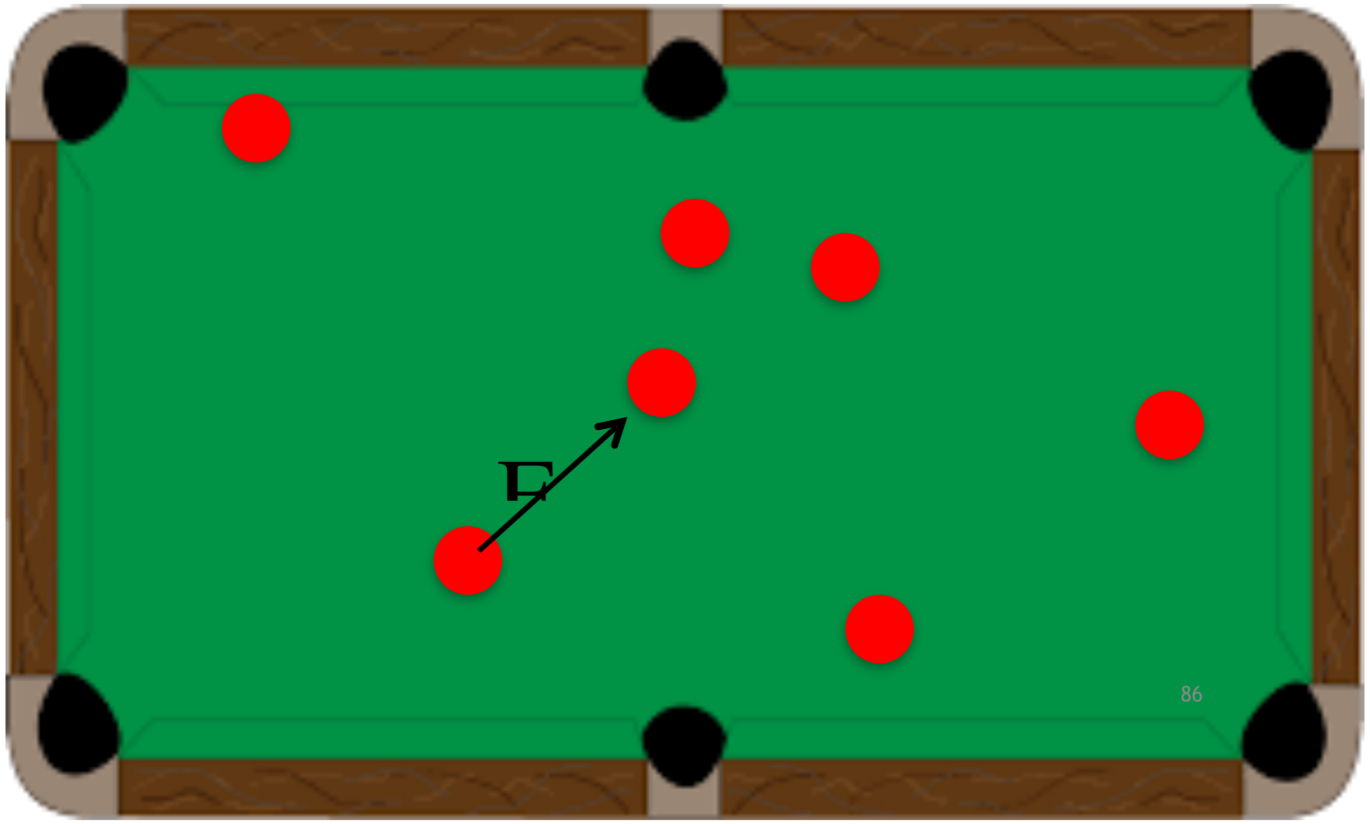
- Errors accumulate during unrolling
- Policy learnt on top of an inaccurate model is upperbounded by the accuracy of the model
- Policies exploit model errors be being overly optimistic
- With lots of experience, model-free methods would always do better

Answers:

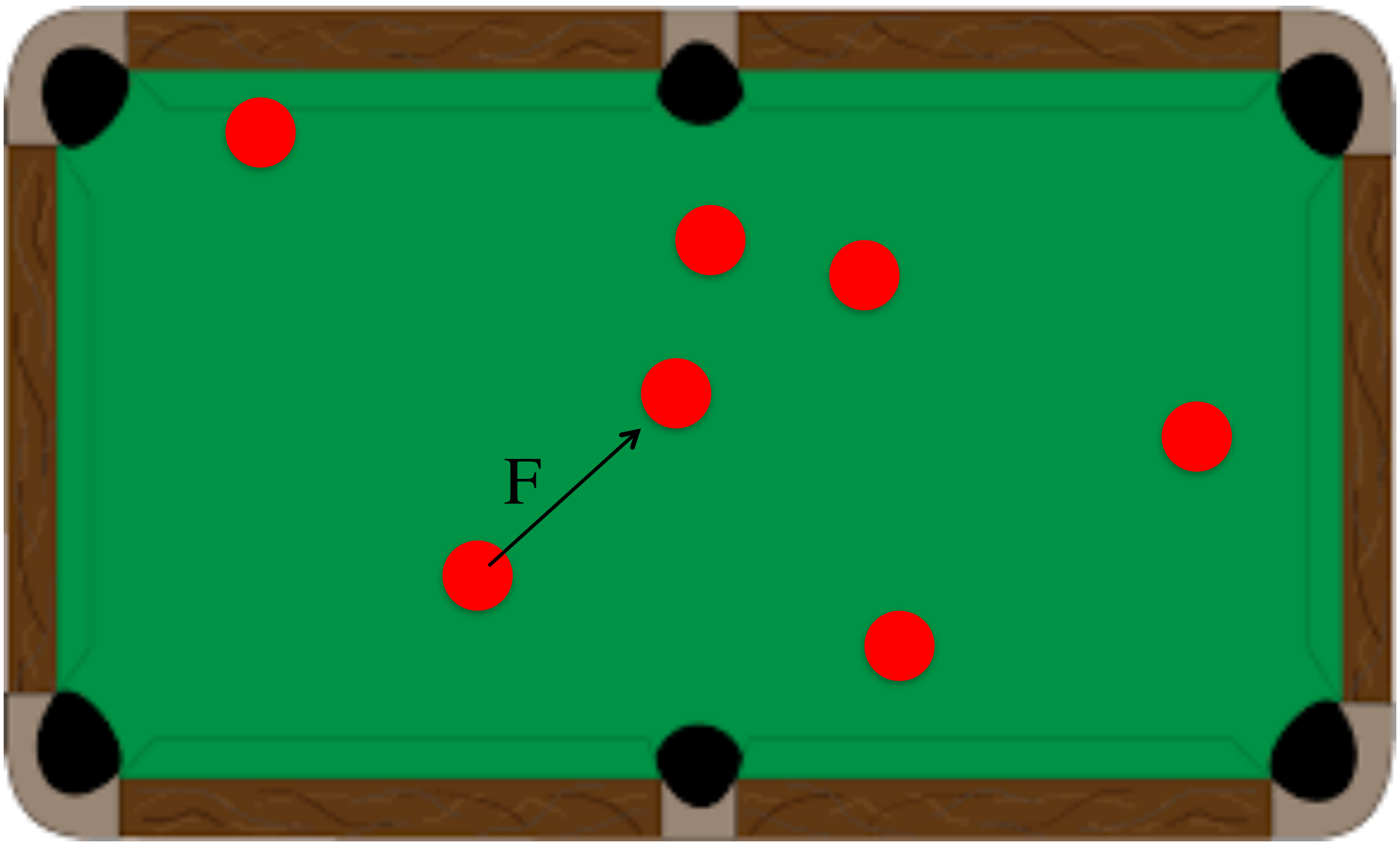
- Use model to pre-train your polic, finetune while being model-free
- Use model to explore fast, but always try actions not suggested by the model so you do not suffer its biases
- Build a model on top of a latent space which is succinct and easily predictable
- Abandon global models and train local linear models, which do not generalize but help you solve your problem fast, then distill the knowledge of the actions to a general neural network policy (next week)

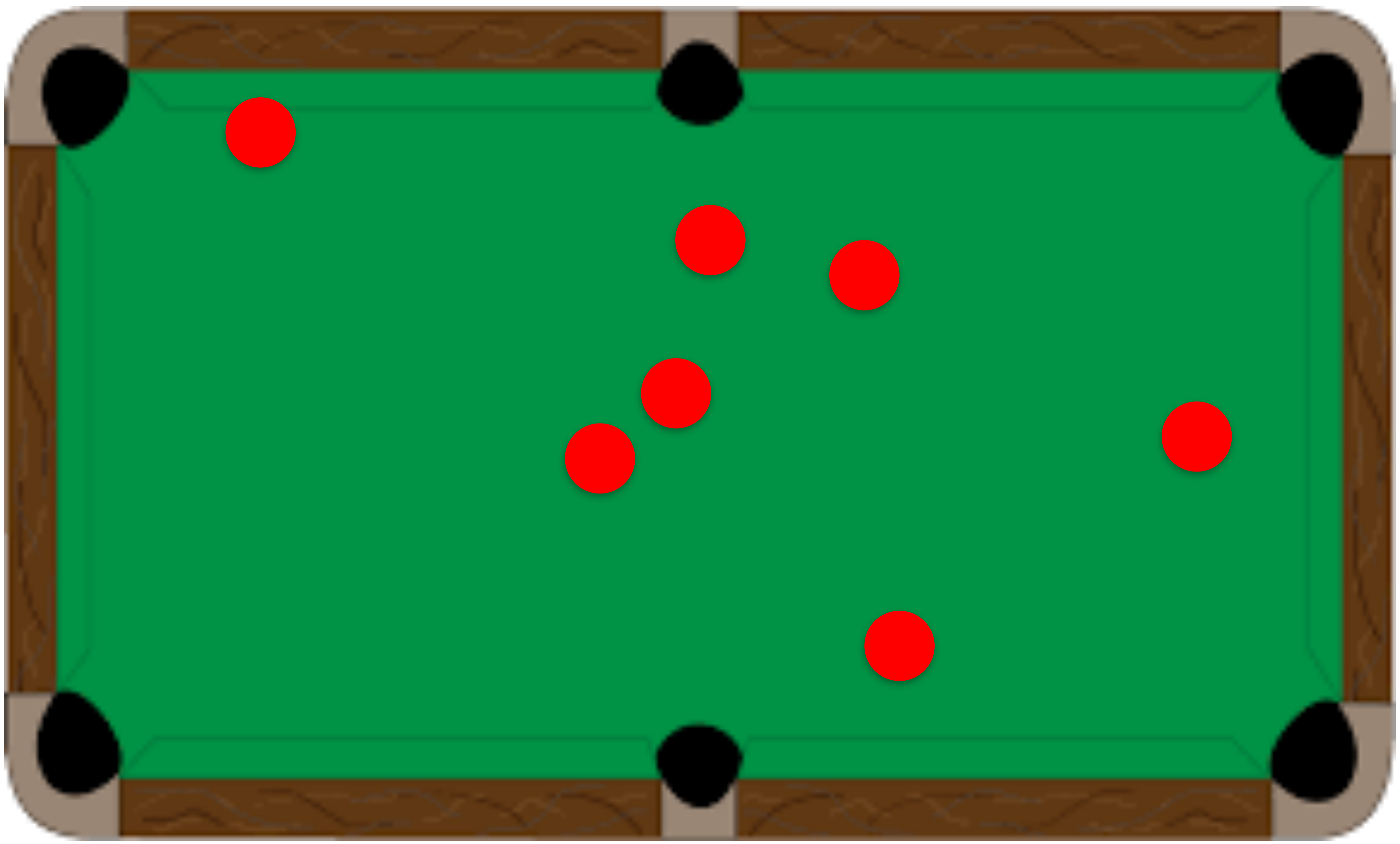
How do we learn to play Billiards?

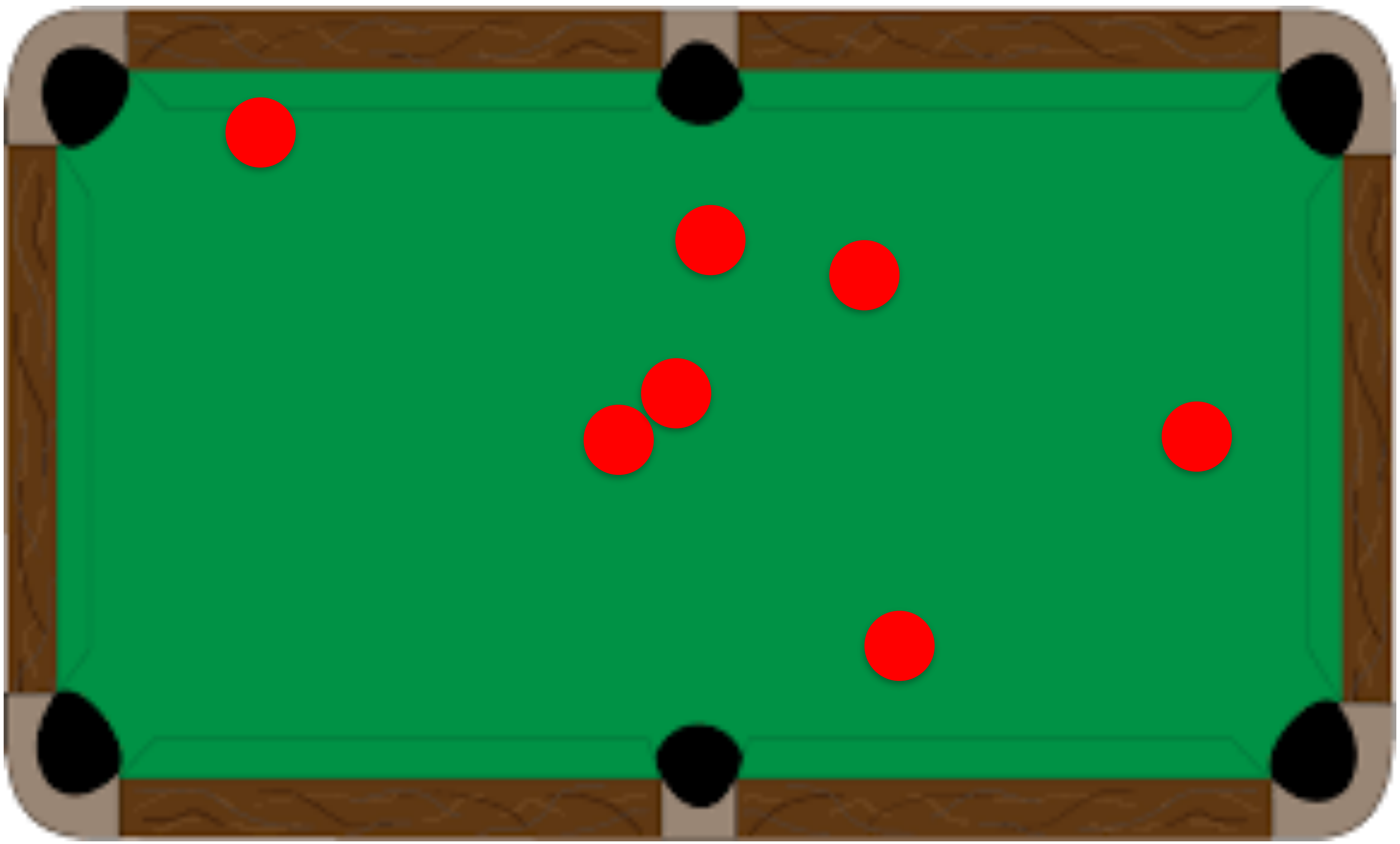
- First, we tranfer all knowledge about how objects move, that we have accumulated so far.
- Second, we watch other people play and practise ourselves, to finetune such model knowledge

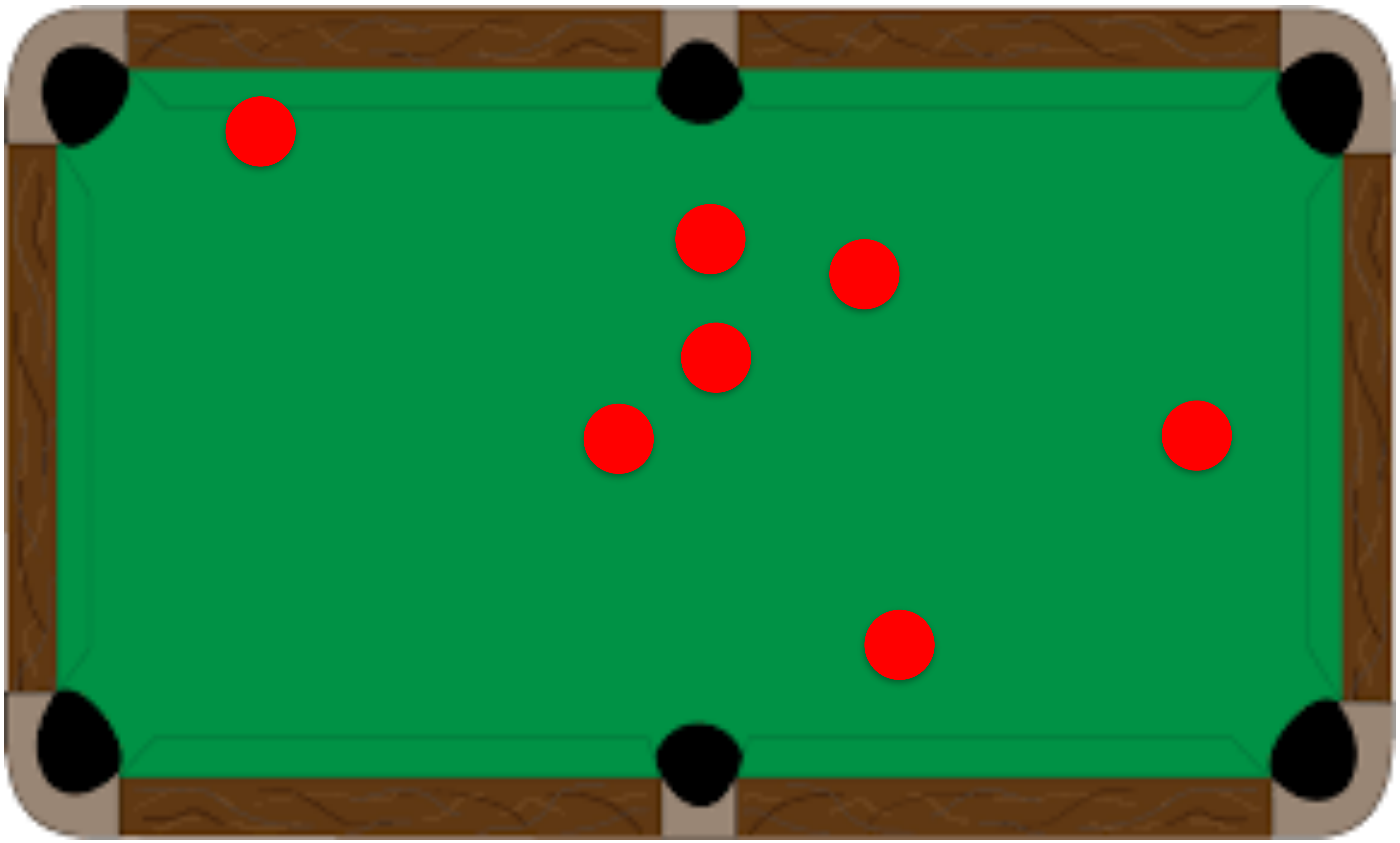


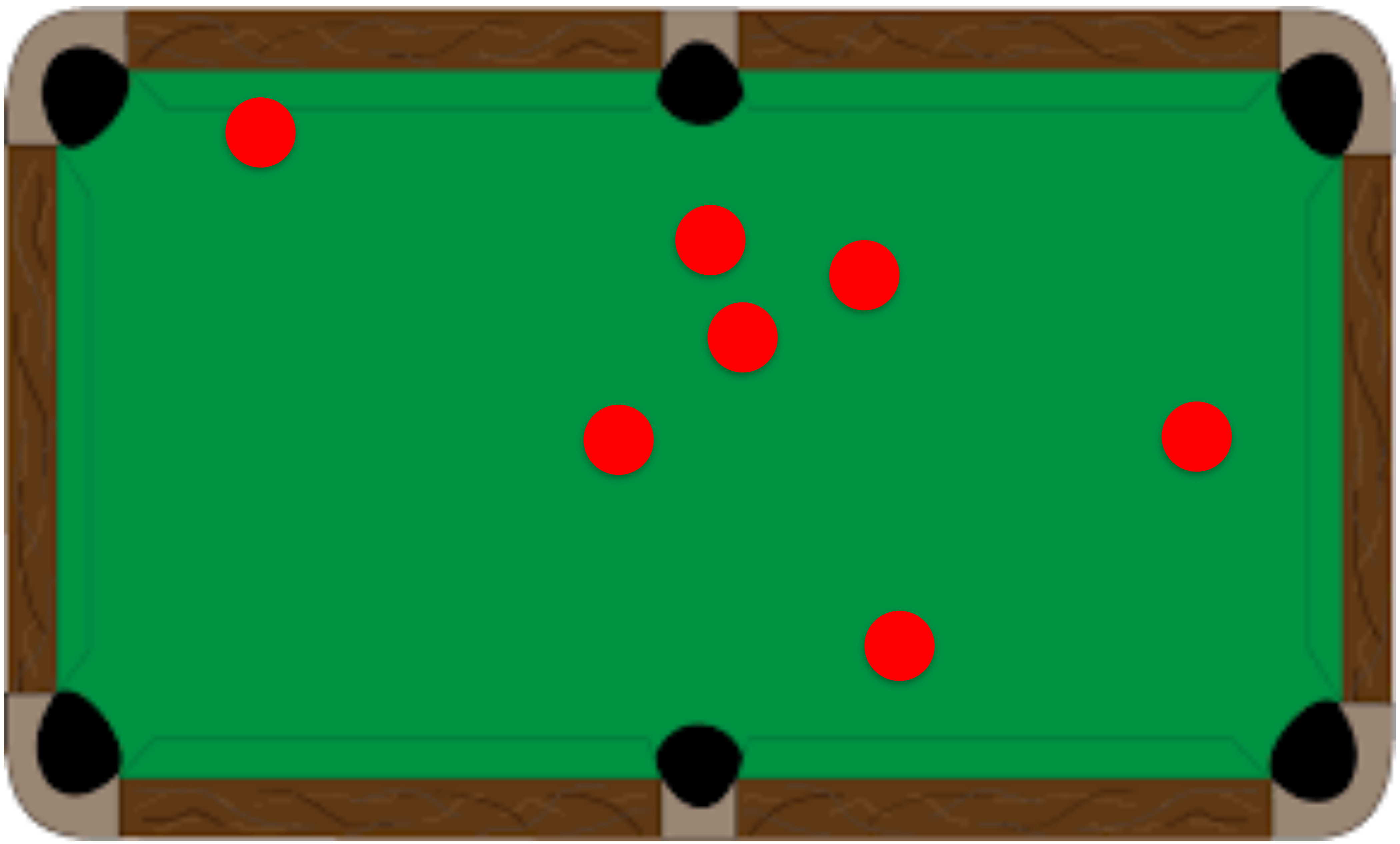
How do we learn to play Billiards?





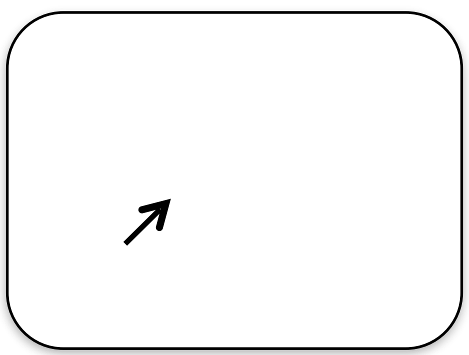


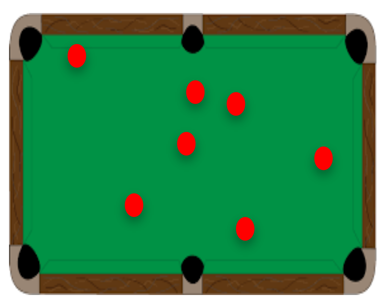


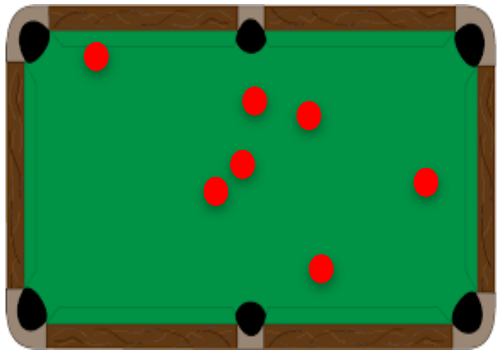


Learning Action-Conditioned Billiard Dynamics

$$(\mathbf{F}, \mathbf{x}_{1 \cdots t}) \rightarrow \mathbf{x}_{t+1 \cdots t+T}$$

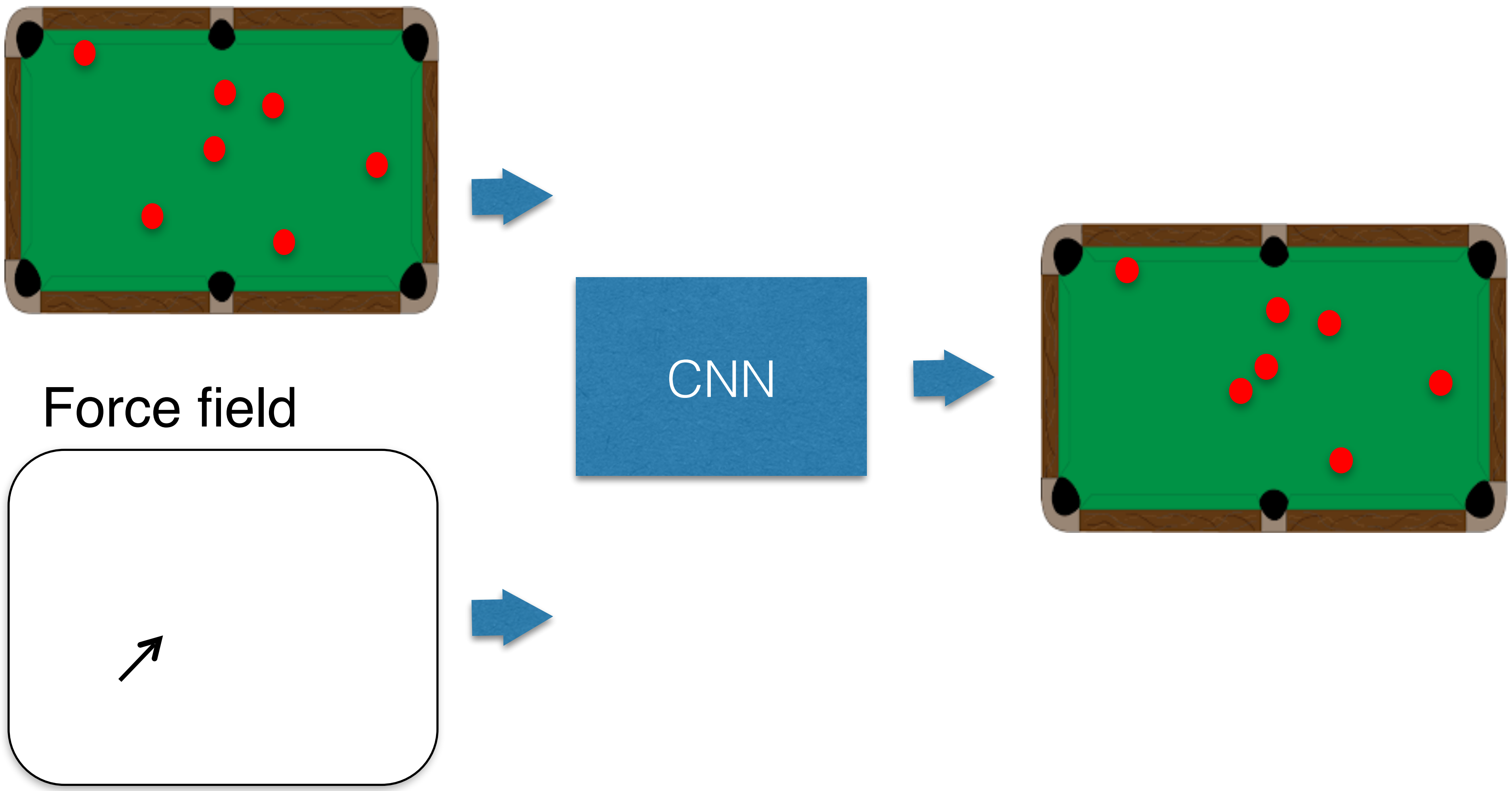






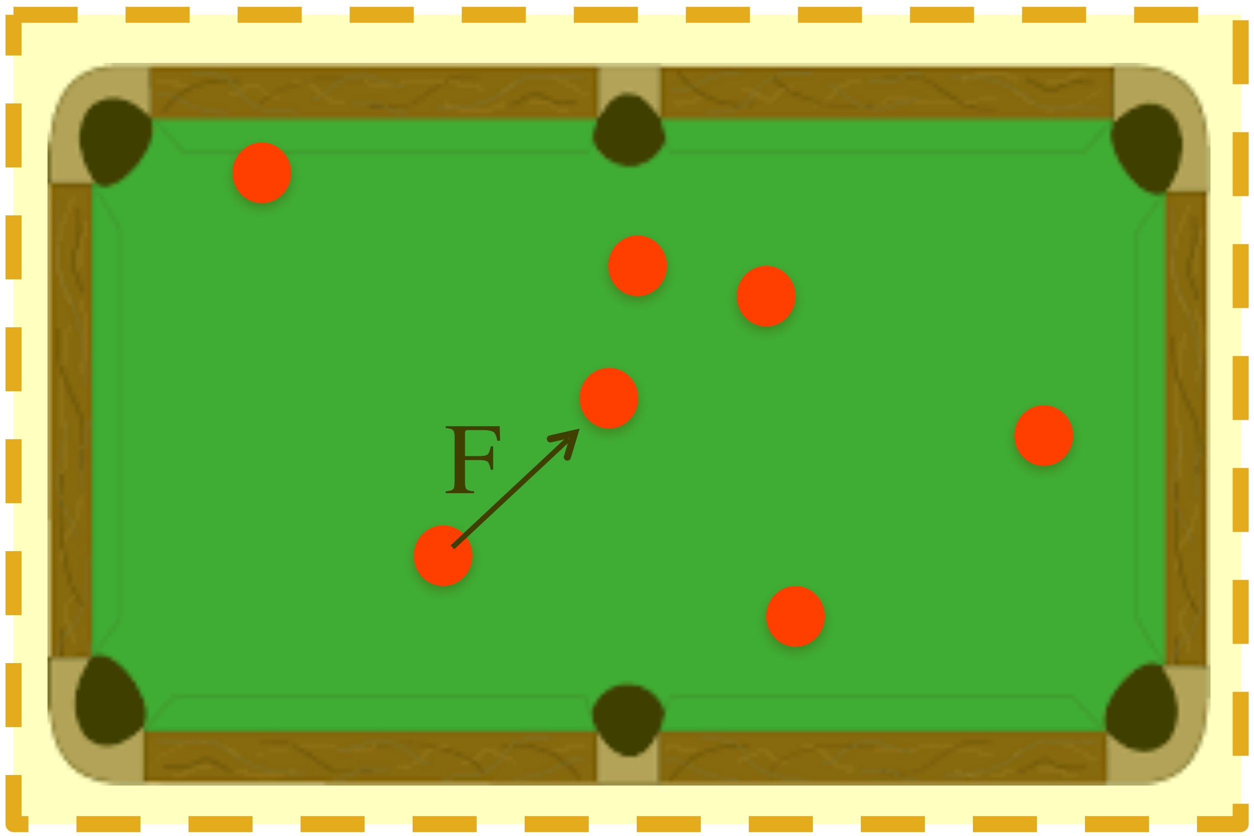
Predictive Visual Models of Physics for Playing Billiards, K.F. et al. ICLR 2016

Learning Action-Conditioned Billiard Dynamics

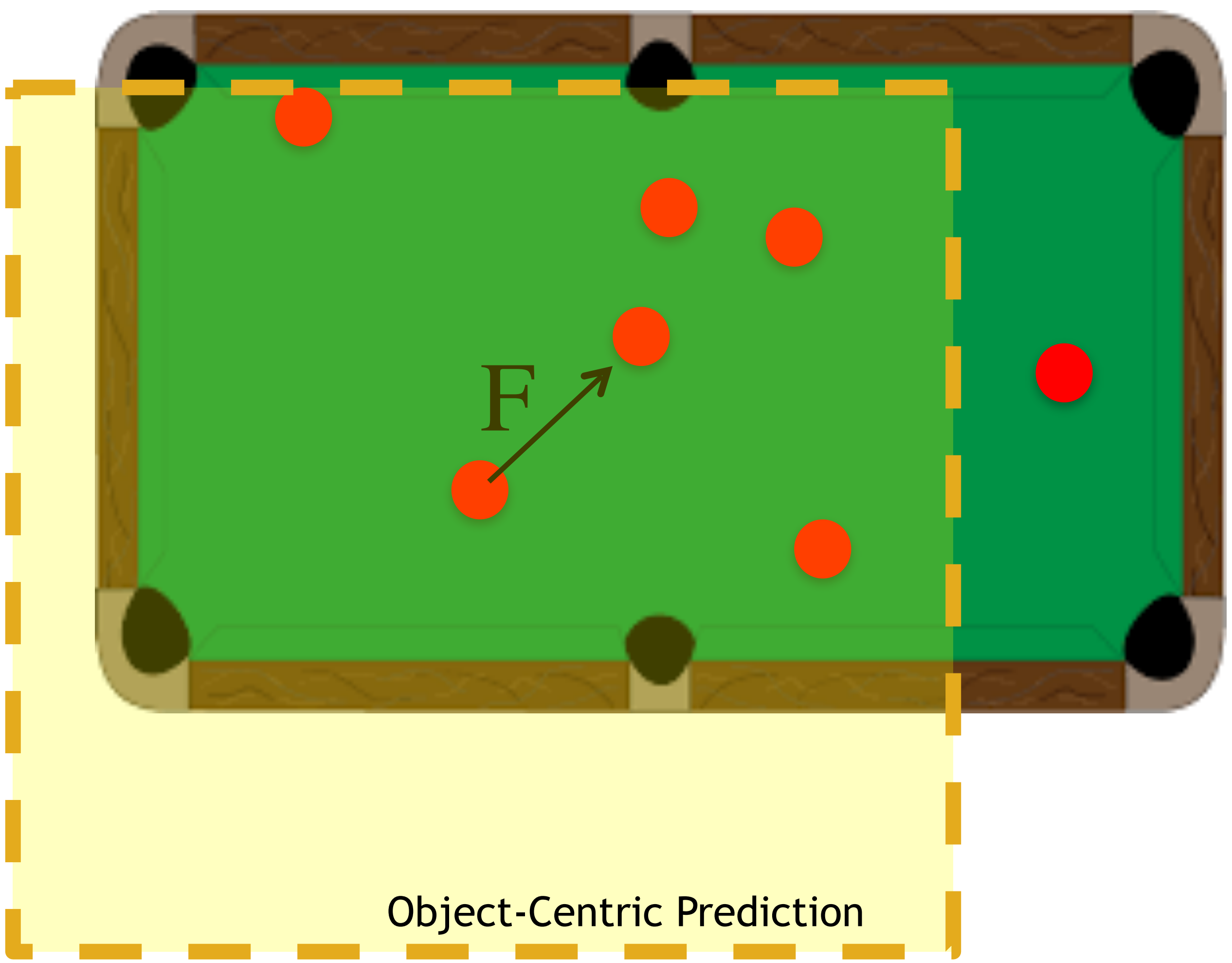


Q: will our model be able to generalize across different number of balls present?

Learning Action-Conditioned Billiard Dynamics

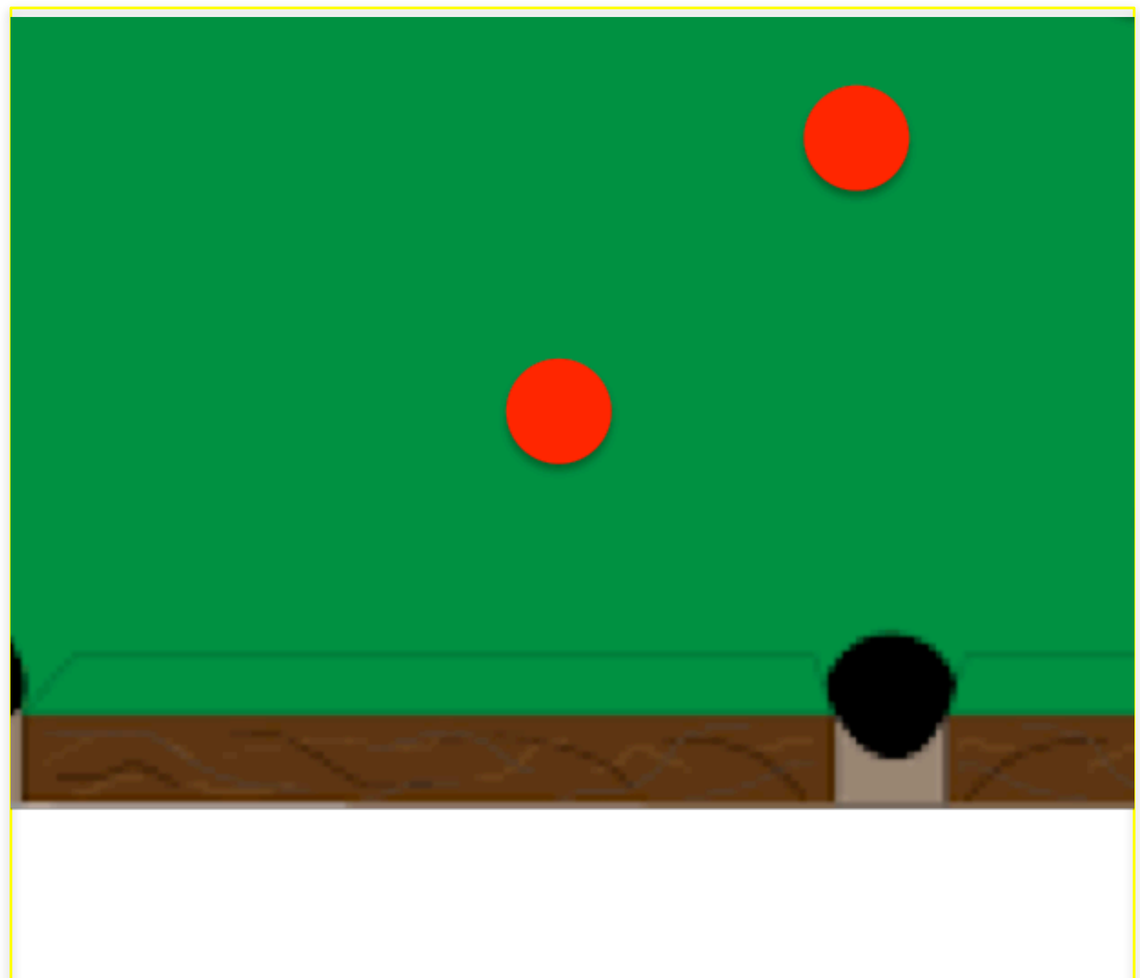


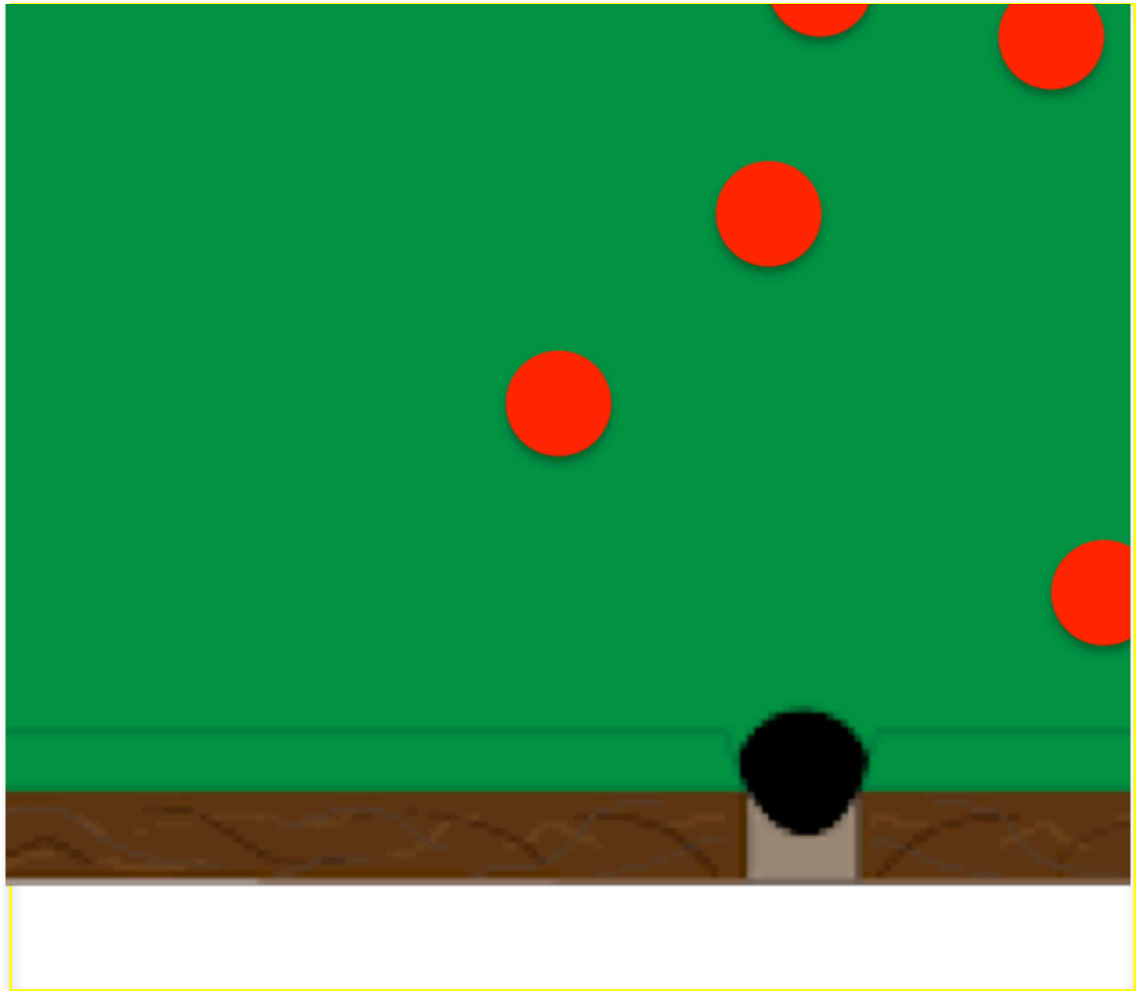
World-Centric Prediction

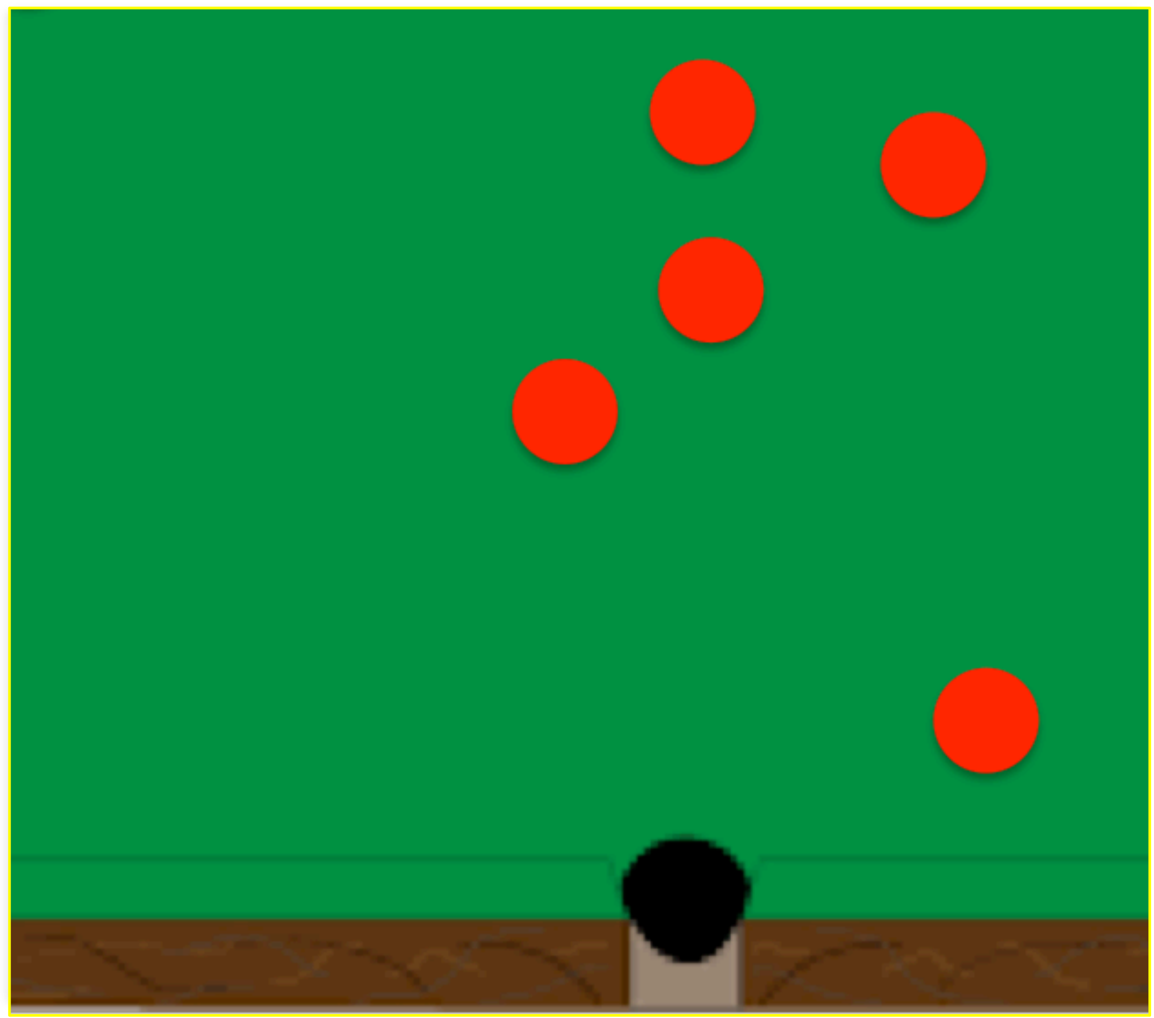


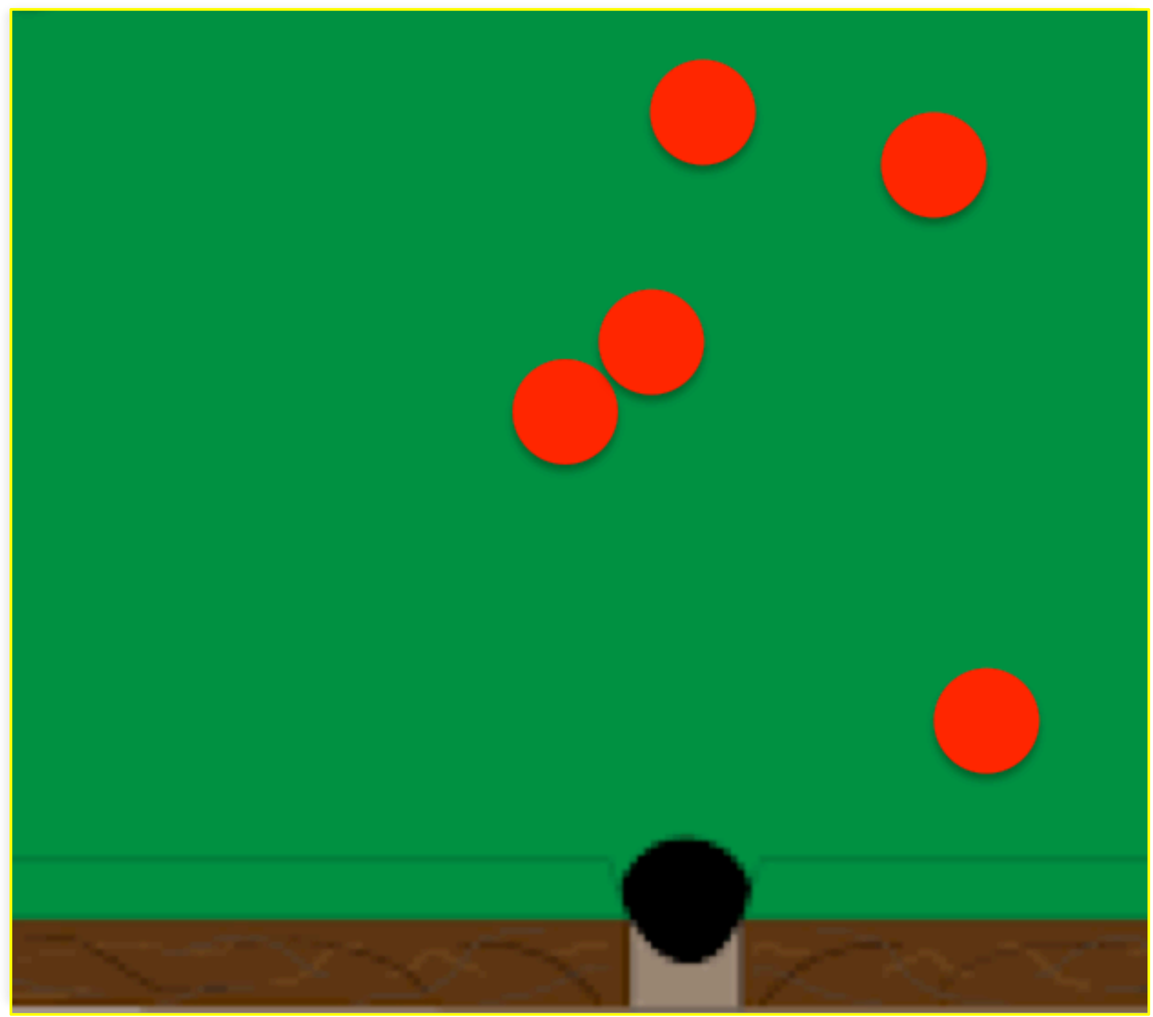
Object-Centric Prediction

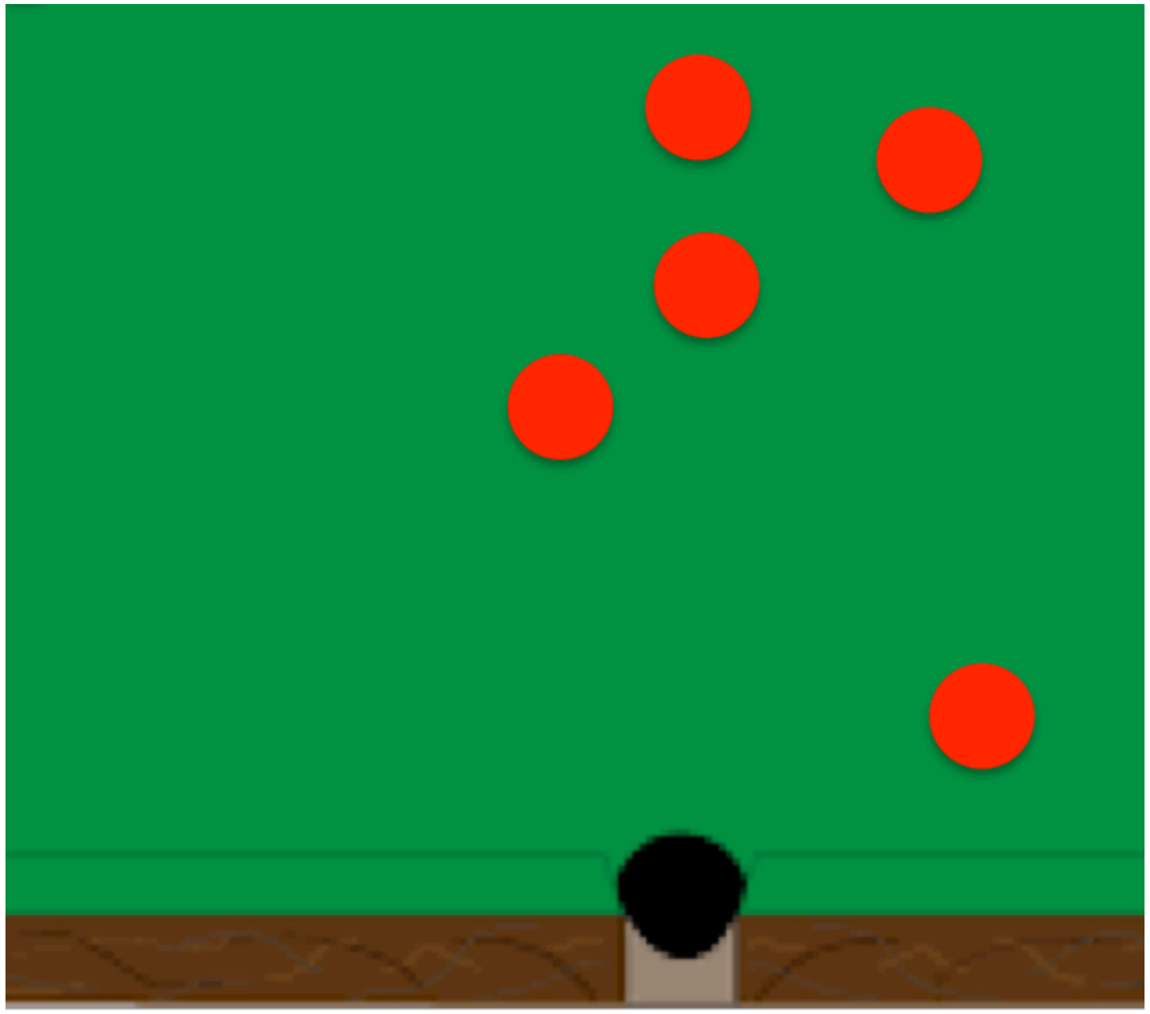
Q: will our model be able to generalize across different number of balls present?

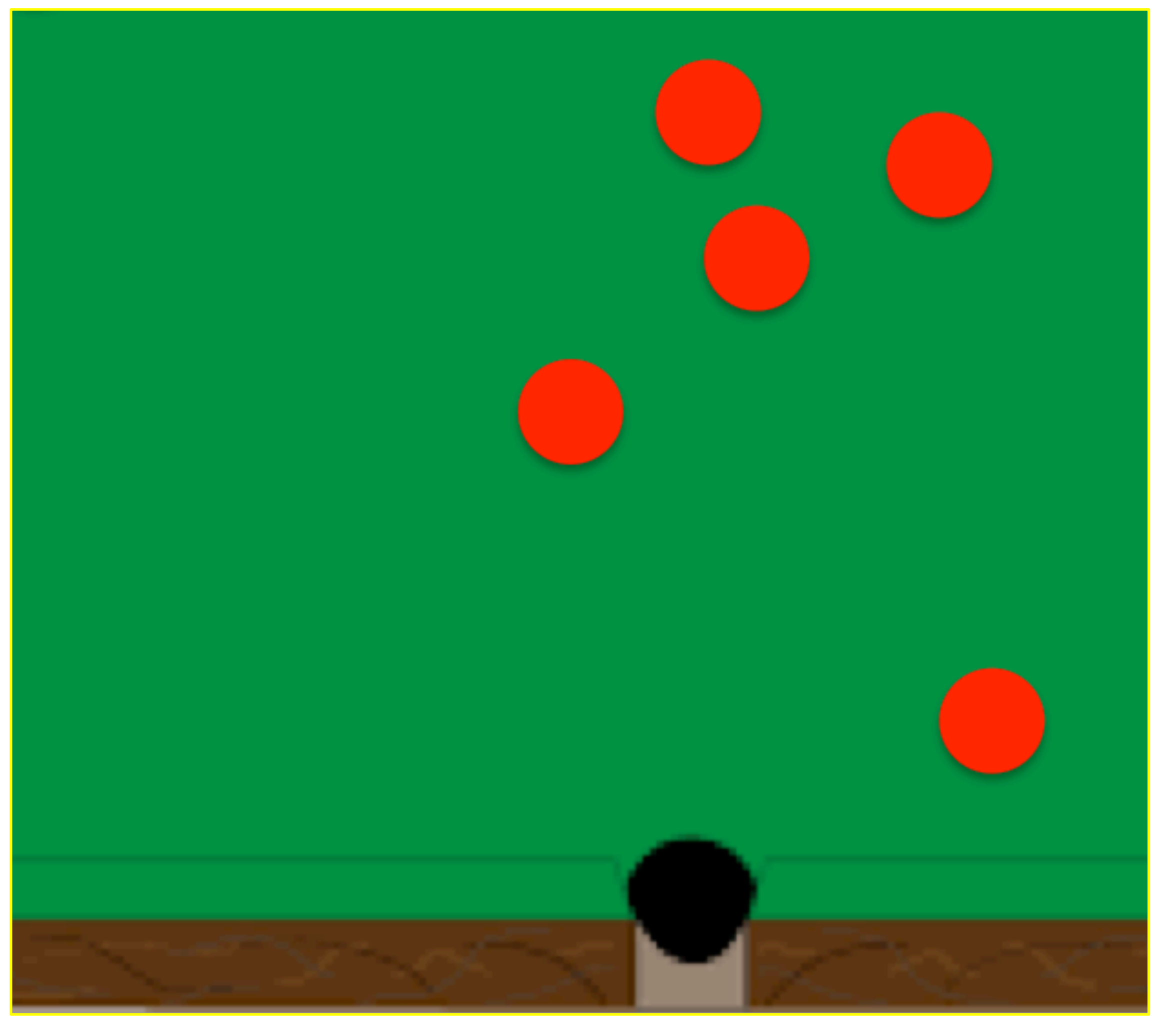




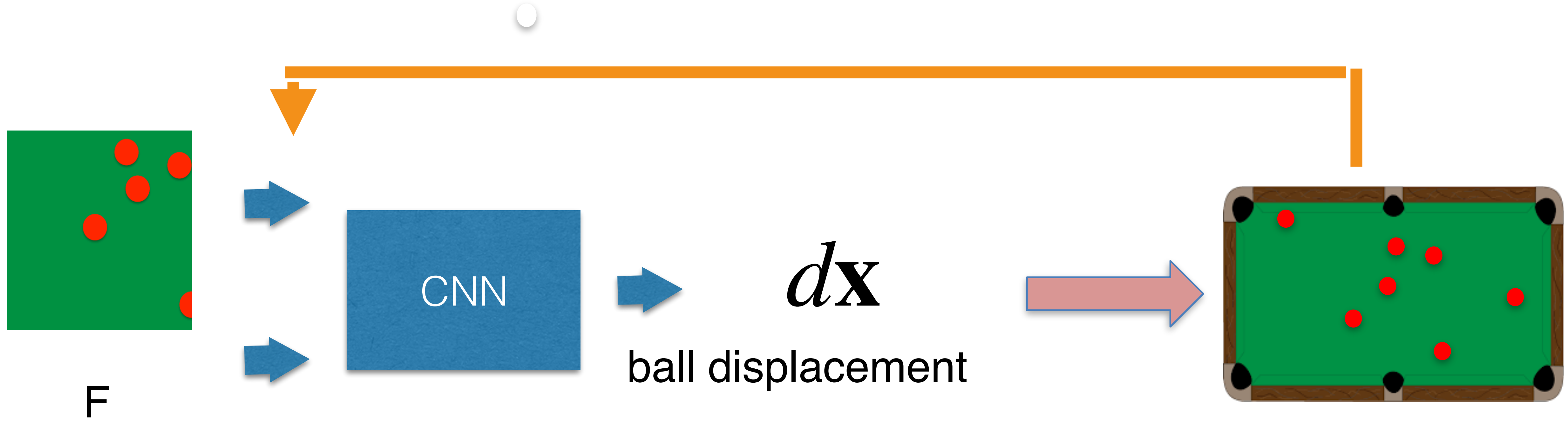




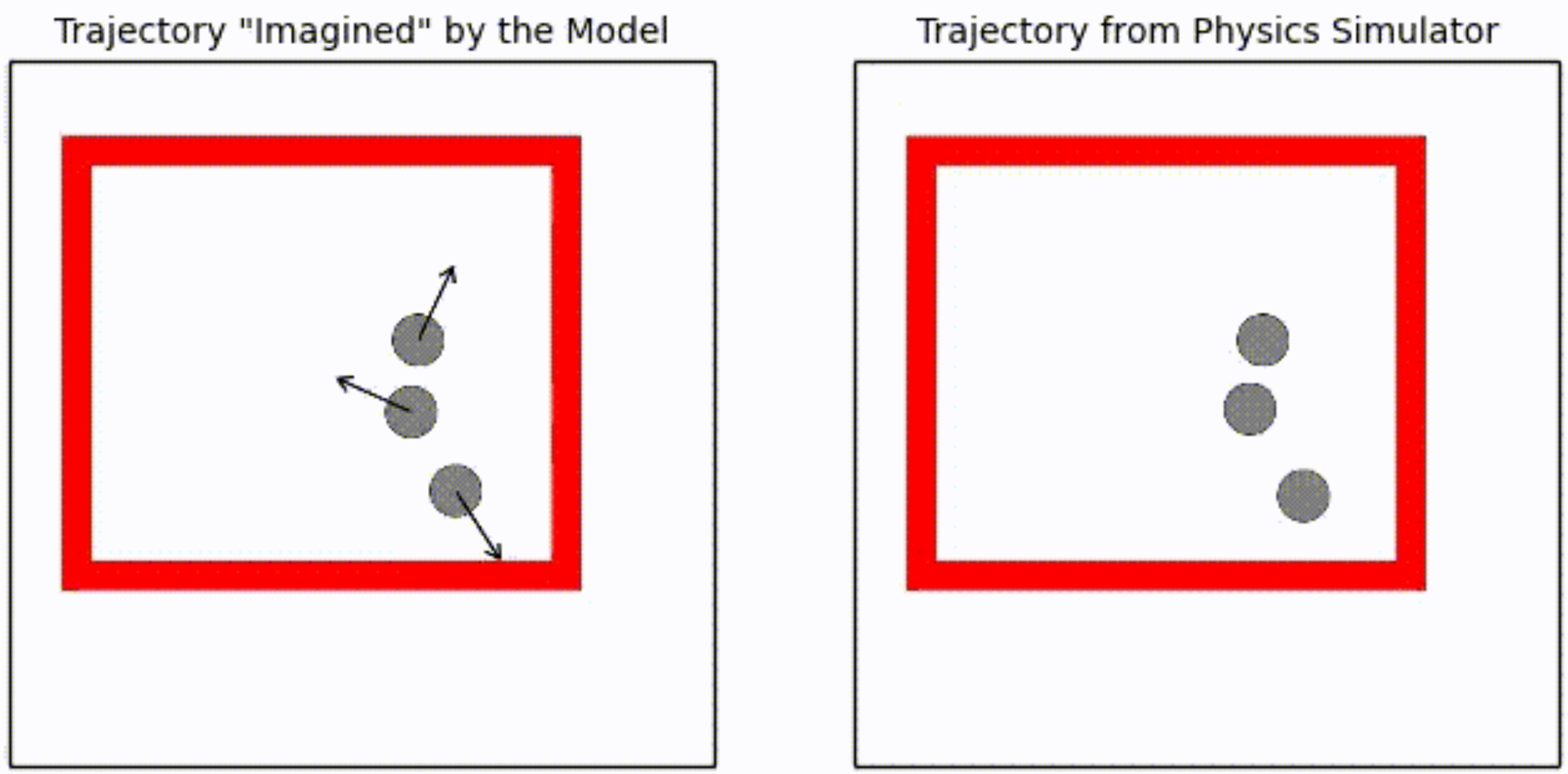




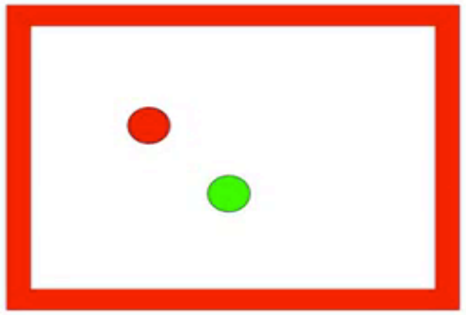
Object-centric Billiard Dynamics



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

fi
ic



How should I push the red ball so that it collides with the green on?
Cme for searching in the force space

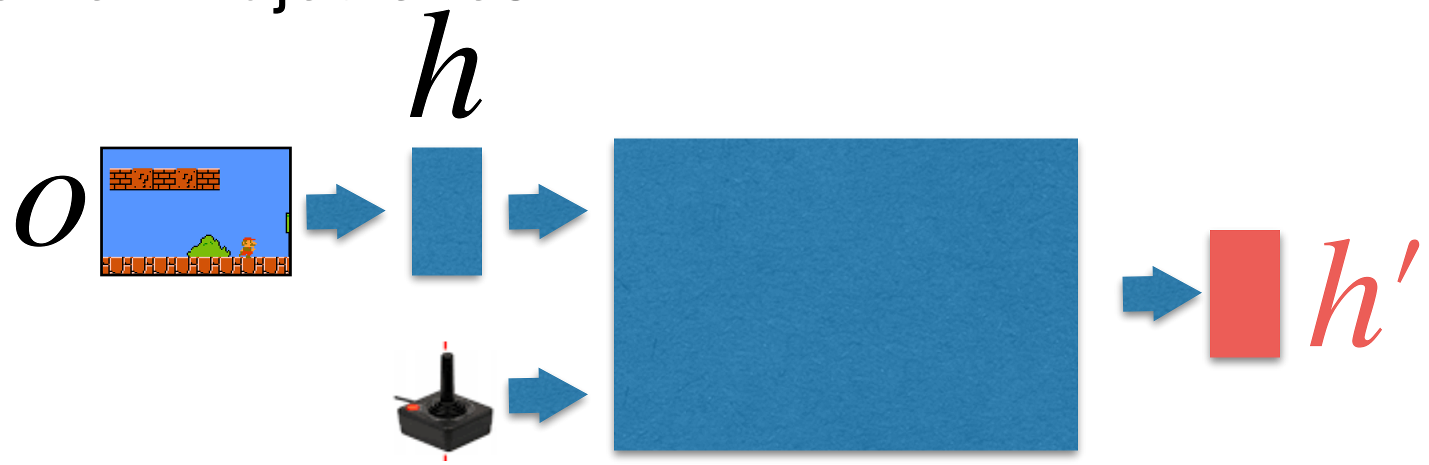
Learning Dynamics

Two good ideas so far:

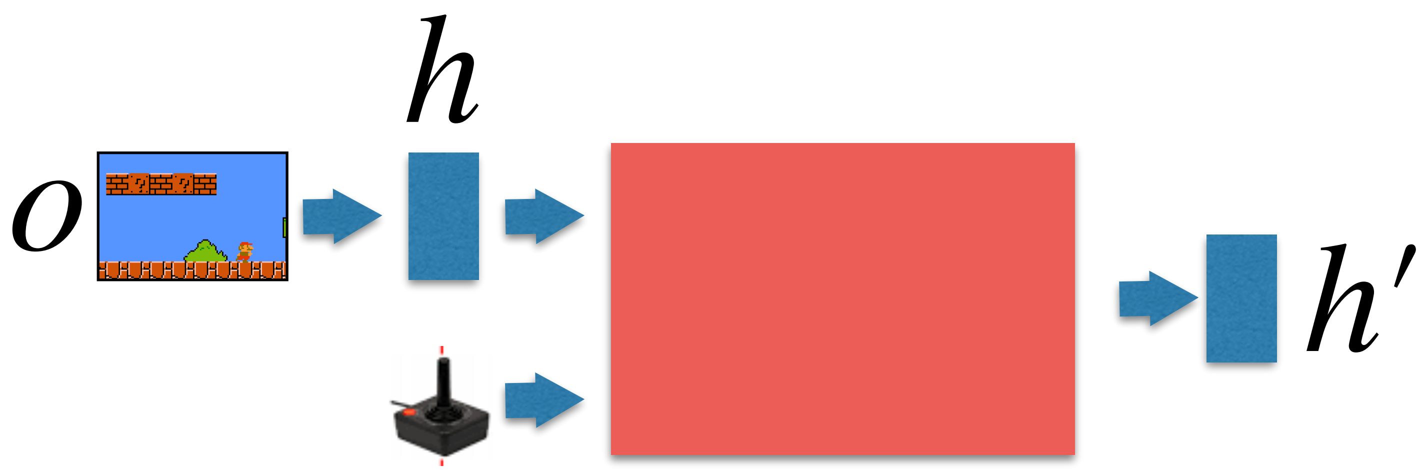
1) object graphs instead of images. Such encoding allows to generalize across different number of entities in the scene.
2) predict motion instead of appearance. Since appearance does not change, predicting motion suffices. Let’s predict only the dynamic properties and keep the static one fixed.

Billiards

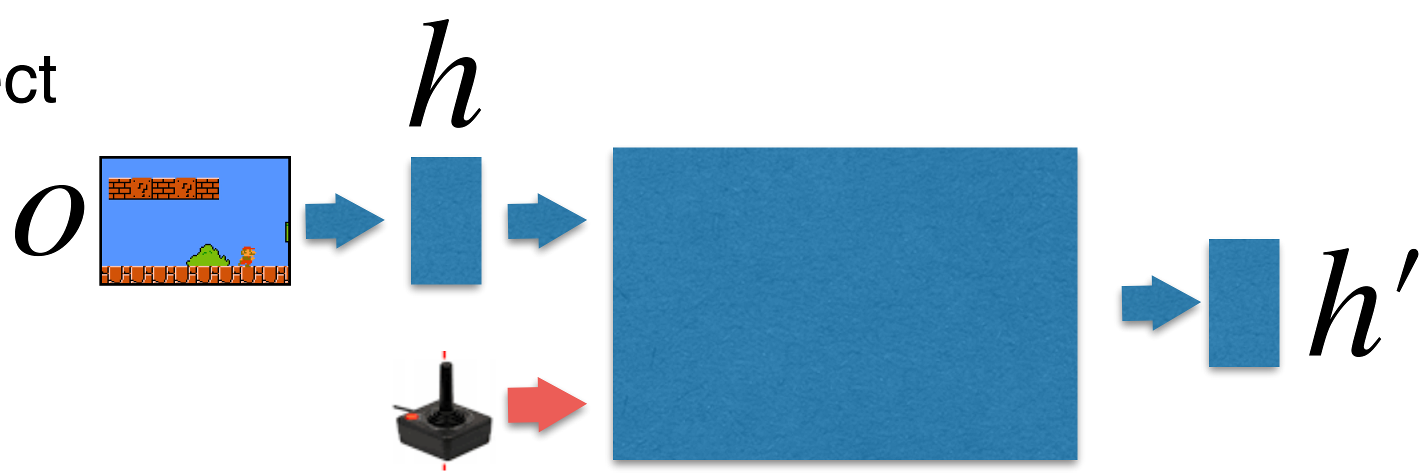
- We predicted object displacement trajectories



- We had one CNN per object in the scene, shared the weights across objects

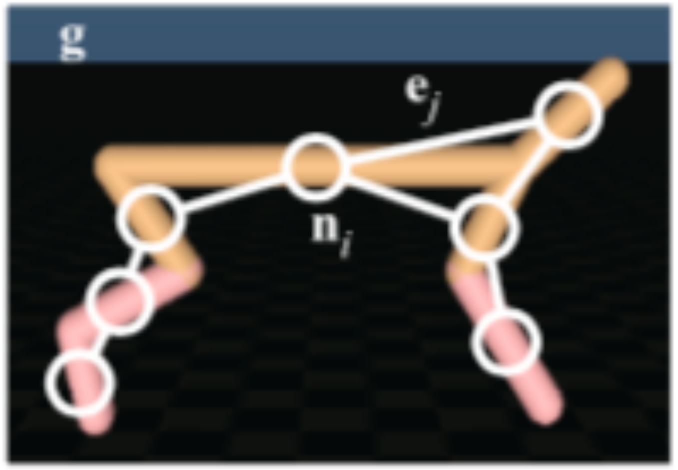


- A force applied to each object



Graph Encoding

In the Billiard case, object computations were coordinated by using a large enough context around each object (node). What if we explicitly send each node's computations to neighboring nodes to be taken account when computing their features?

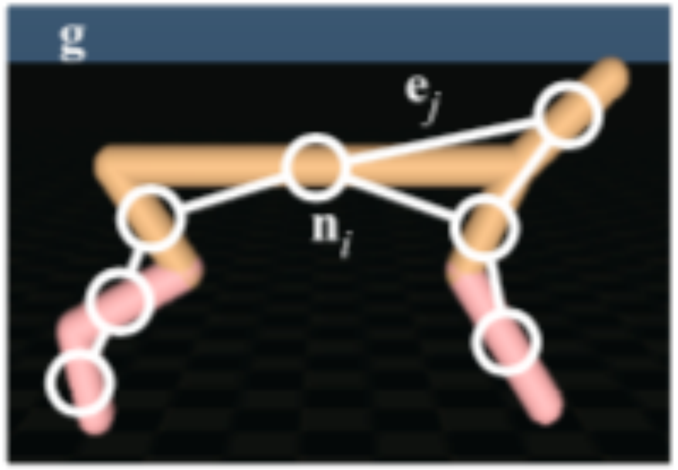


We will encode a robotic agent as a graph, where nodes are the different bodies of the agent and edges are the joints, links between the bodies

Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

Graph Encoding

In the Billiard case, object computations were coordinated by using a large enough context around each object (node). What if we explicitly send each node's computations to neighboring nodes to be taken account when computing their features?



Node features

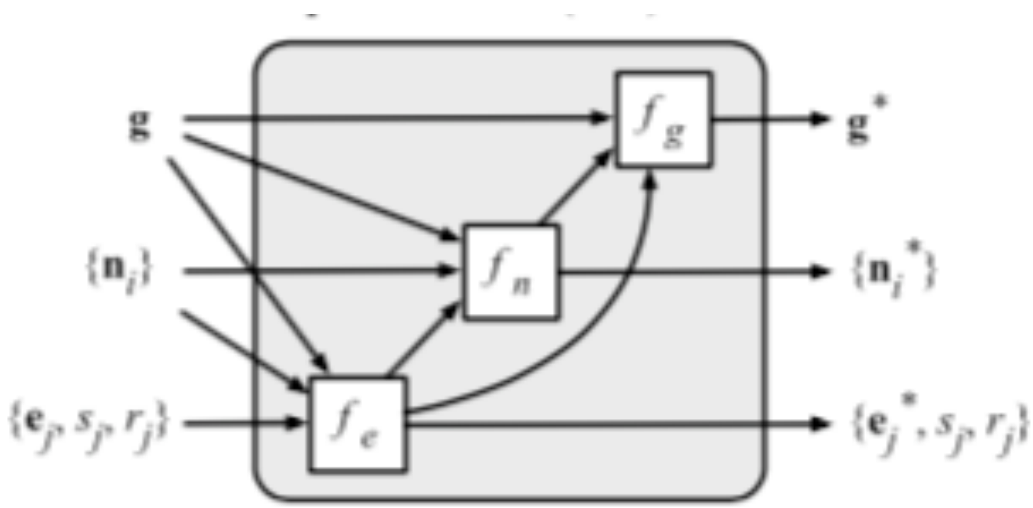
- Observable/dynamic: 3D position, 4D quaternion orientation, linear and angular velocities
- Unobservable/static: mass, inertia tensor
- Actions: forces applied on the joints

Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

Graph Forward Dynamics

Node features

- Observable/dynamic: 3D position, 4D quaternion orientation, linear and angular velocities
- Unobservable/static: mass, inertia tensor
- Actions: forces applied on the joints
- No visual input here, much easier!



Algorithm 1 Graph network, GN

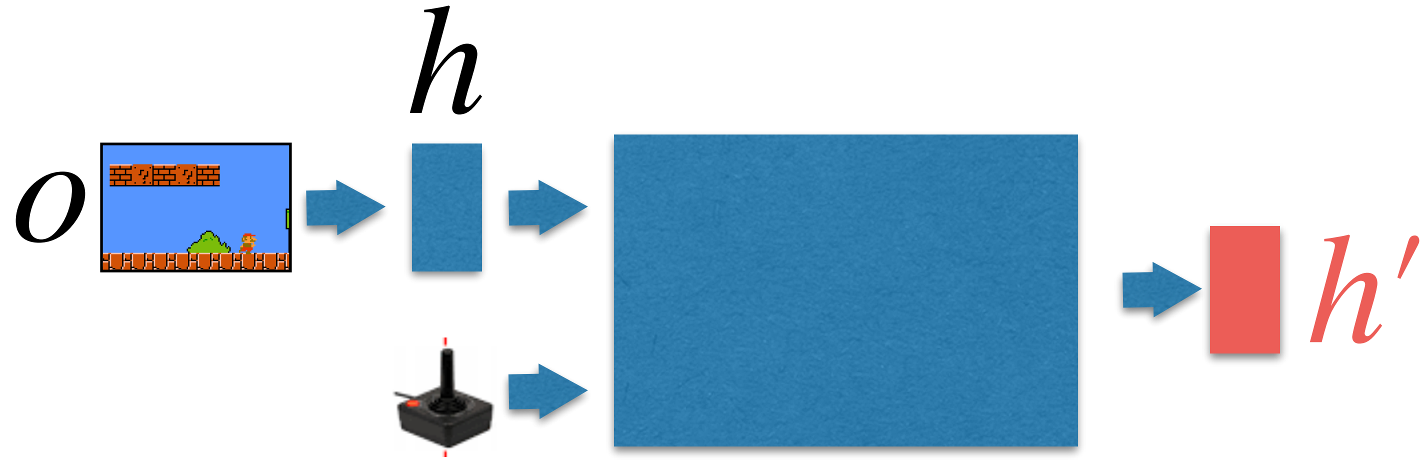
Input: Graph, $G = (\mathbf{g}, \{\mathbf{n}_i\}, \{\mathbf{e}_j, s_j, r_j\})$
for each edge $\{\mathbf{e}_j, s_j, r_j\}$ **do**
  Gather sender and receiver nodes $\mathbf{n}_{s_j}, \mathbf{n}_{r_j}$
  Compute output edges, $\mathbf{e}_j^* = f_e(\mathbf{g}, \mathbf{n}_{s_j}, \mathbf{n}_{r_j}, \mathbf{e}_j)$
end for
for each node $\{\mathbf{n}_i\}$ **do**
  Aggregate $\mathbf{e}_j^*$ per receiver, $\hat{\mathbf{e}}_i = \sum_{j/r_j=i} \mathbf{e}_j^*$
  Compute node-wise features, $\mathbf{n}_i^* = f_n(\mathbf{g}, \mathbf{n}_i, \hat{\mathbf{e}}_i)$
end for
Aggregate all edges and nodes $\hat{\mathbf{e}} = \sum_j \mathbf{e}_j^*$, $\hat{\mathbf{n}} = \sum_i \mathbf{n}_i^*$
Compute global features, $\mathbf{g}^* = f_g(\mathbf{g}, \hat{\mathbf{n}}, \hat{\mathbf{e}})$
Output: Graph, $G^* = (\mathbf{g}^*, \{\mathbf{n}_i^*\}, \{\mathbf{e}_j^*, s_j, r_j\})$

Predictions: I predict only the dynamic features, their temporal difference. Train with regression.

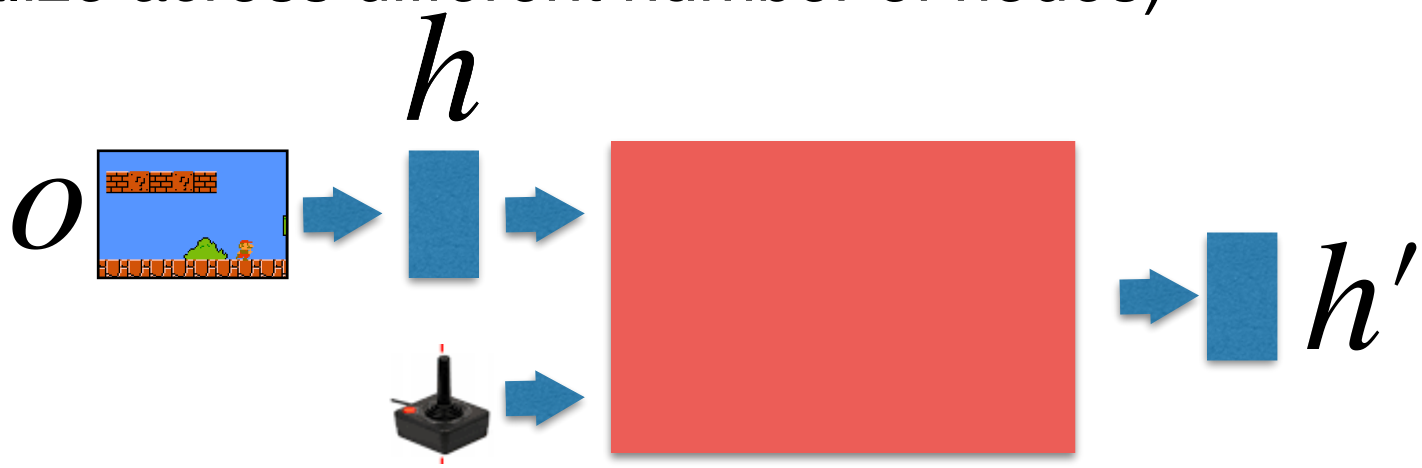
Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

Robots as graphs

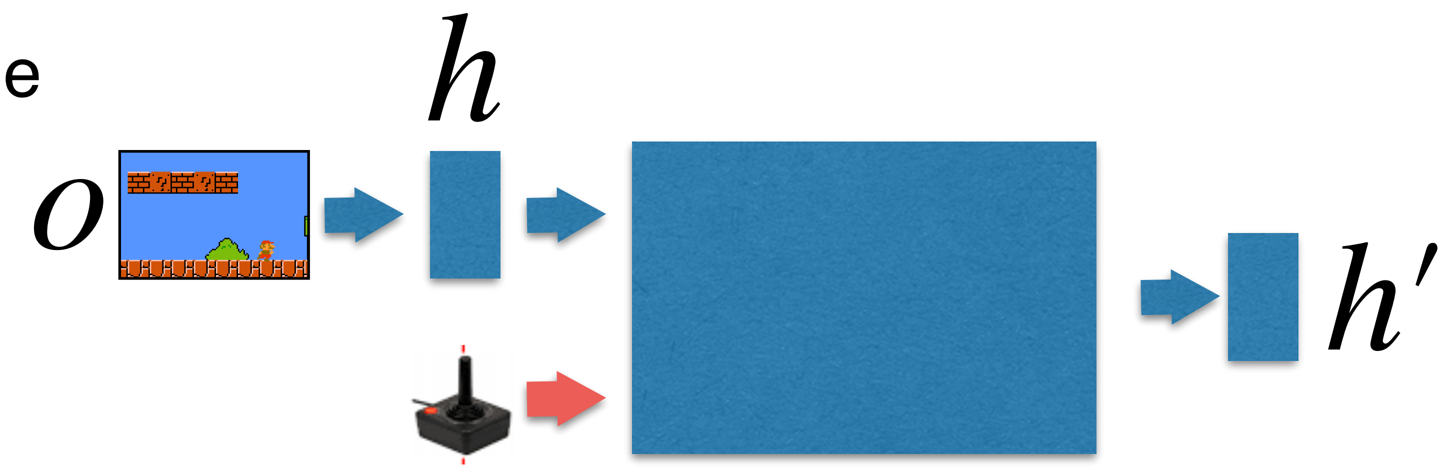
- We predicted dynamic only node features



- Our CNN is a Graph network, the node update function is shared across all nodes (thus we can generalize across different number of nodes)



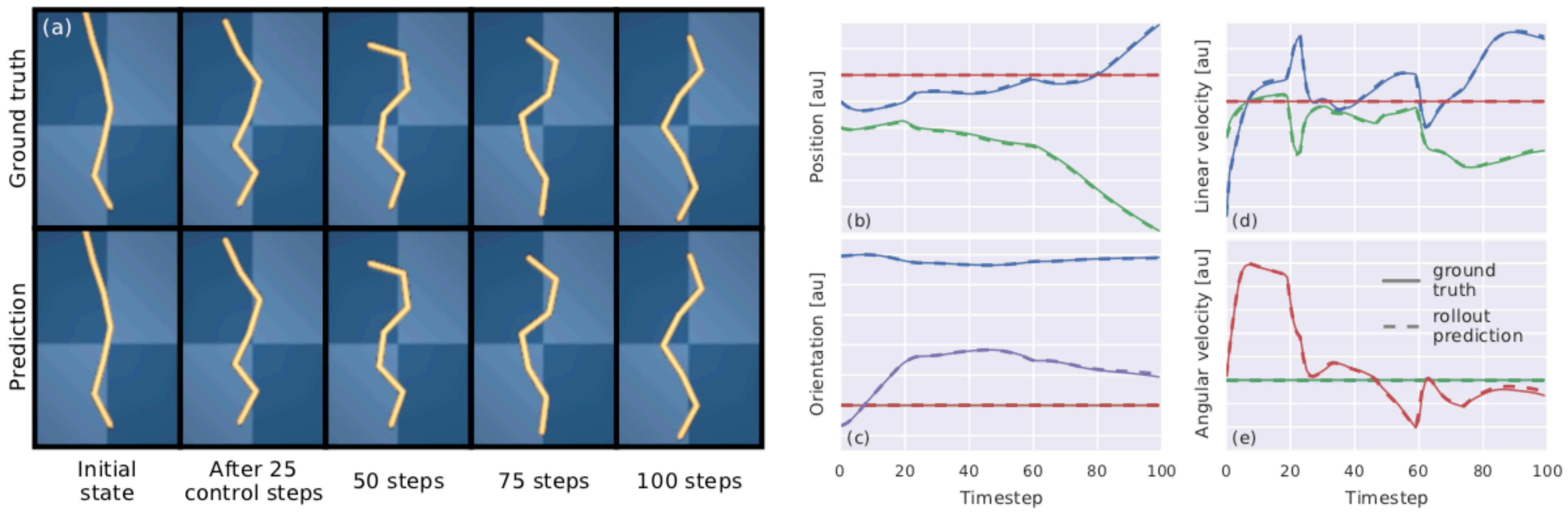
- Forces applied to each node



Graph Forward Dynamics

Node features

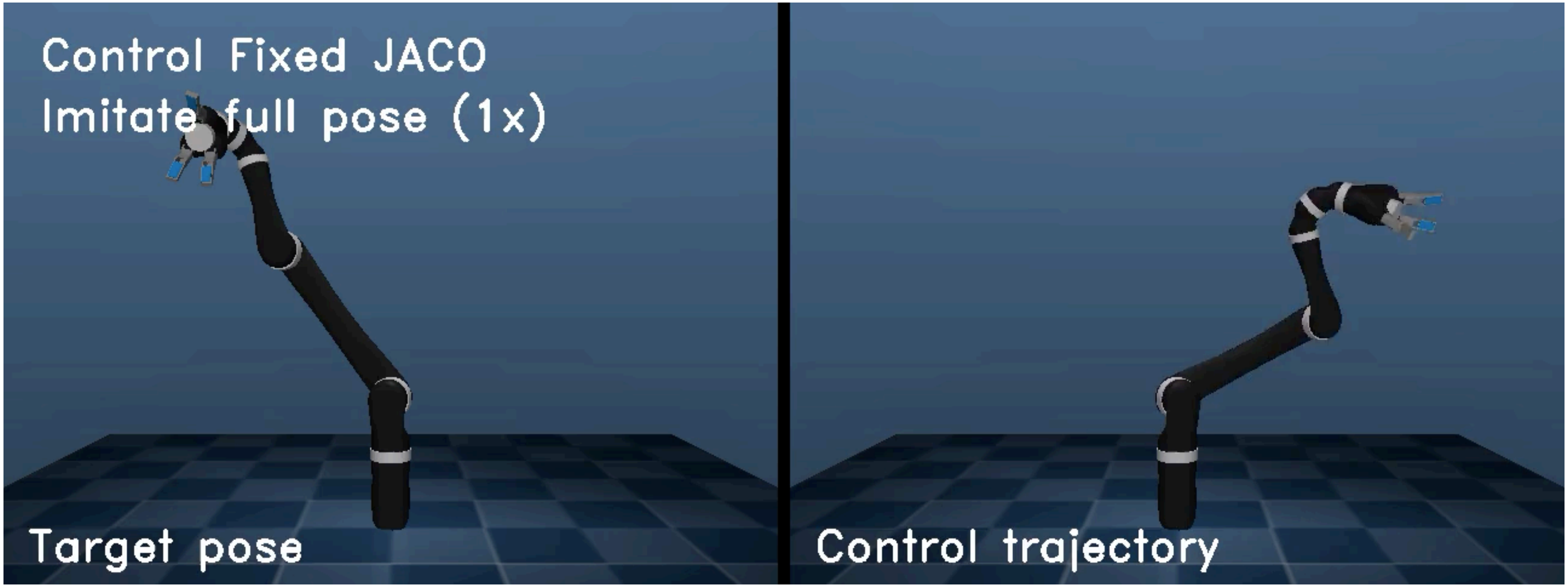
- Observable/dynamic: 3D position, 4D quaternion orientation, linear and angular velocities
- Unobservable/static: mass, inertia tensor
- Actions: forces applied on the joints



Predictions: I predict only the dynamic features, their temporal difference:

Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

Graph Model Predictive Control



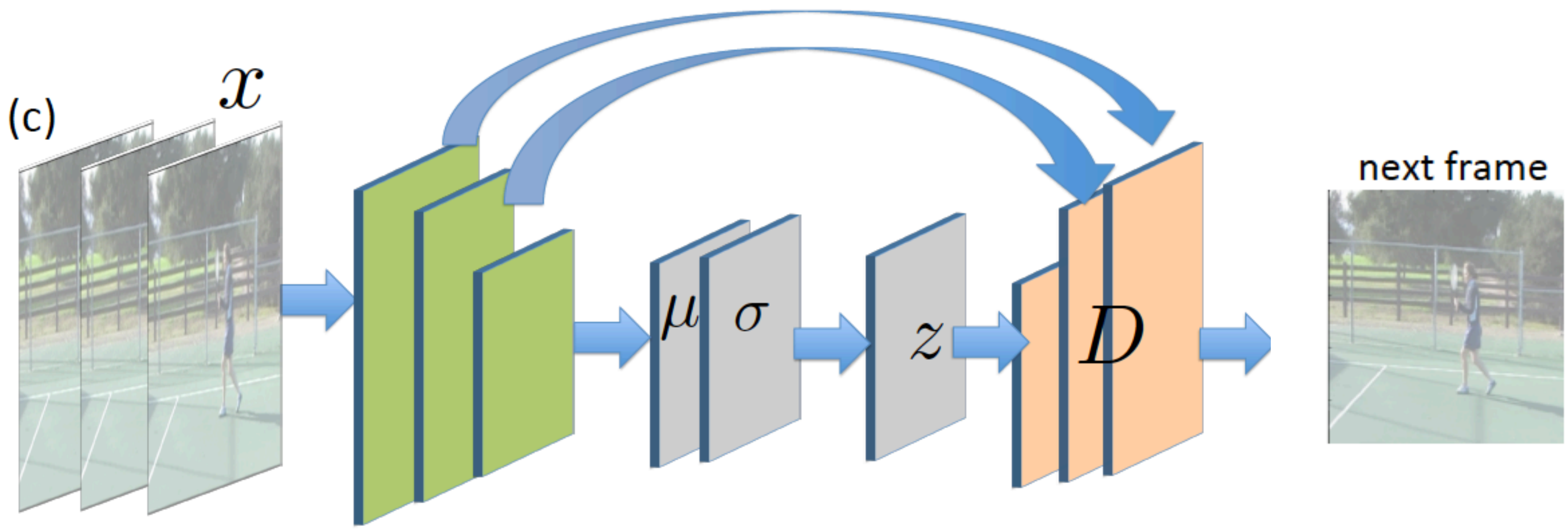
Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

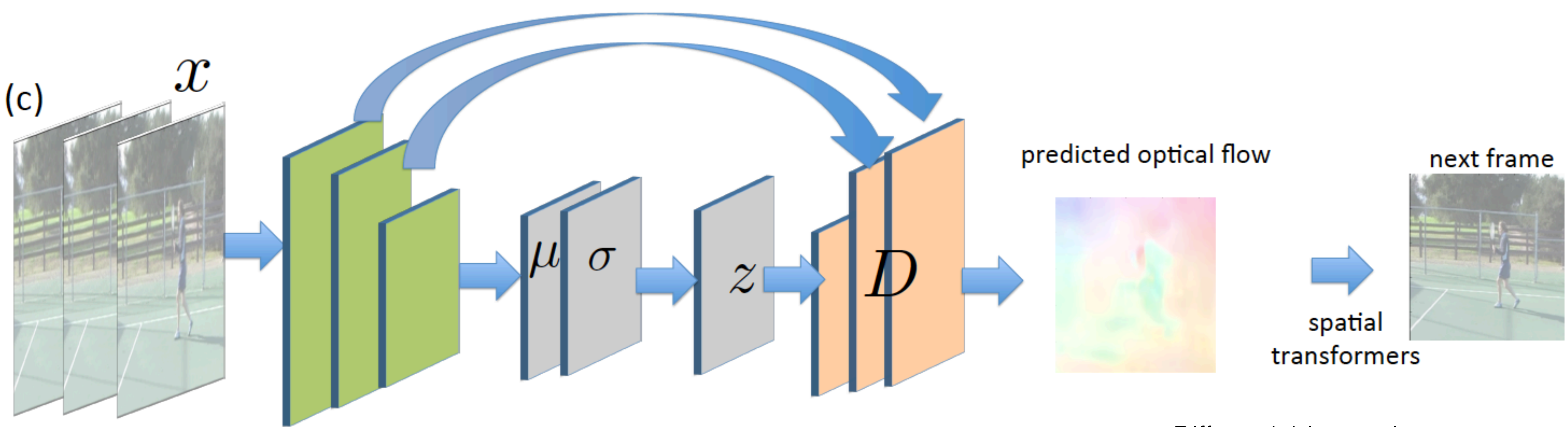
Learning Dynamics

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

Visual dynamics using motion transformation

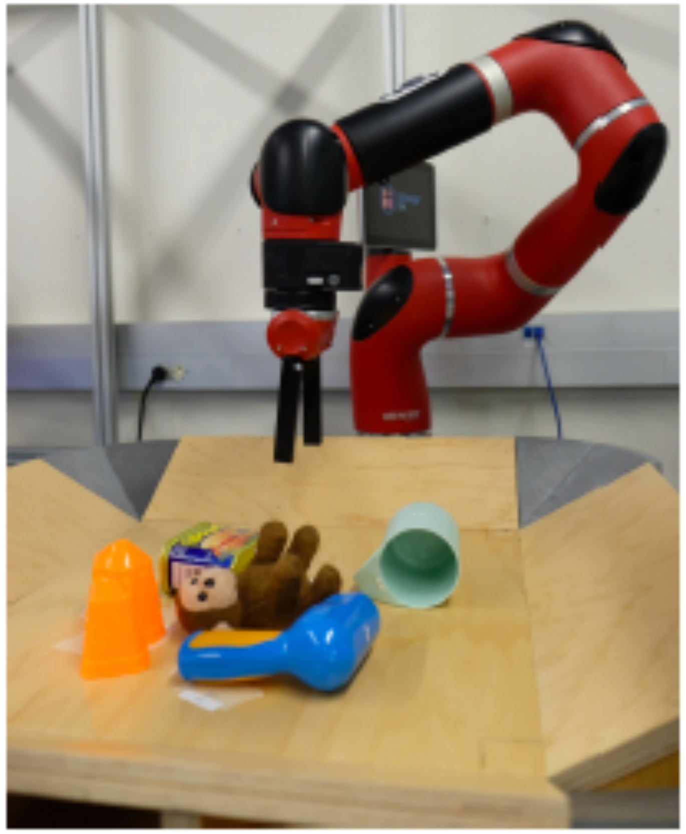


Figure 1: The robot learns to move new objects from self-supervised experience.

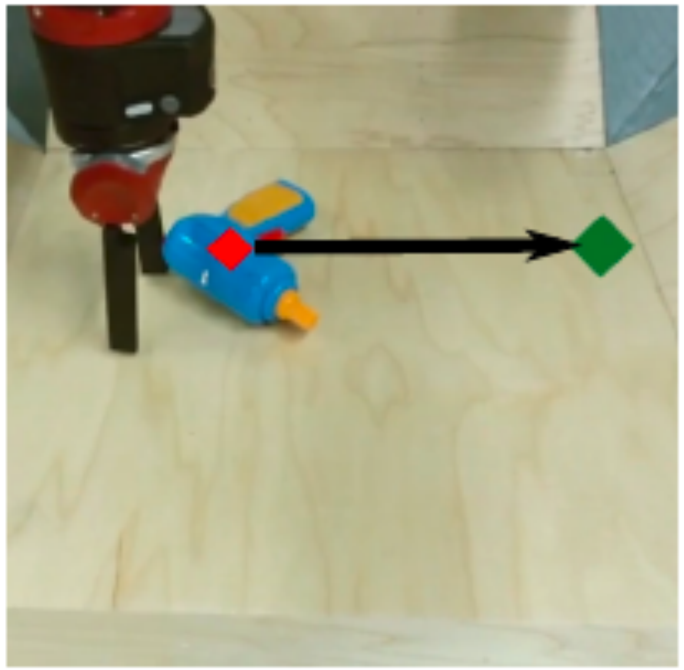
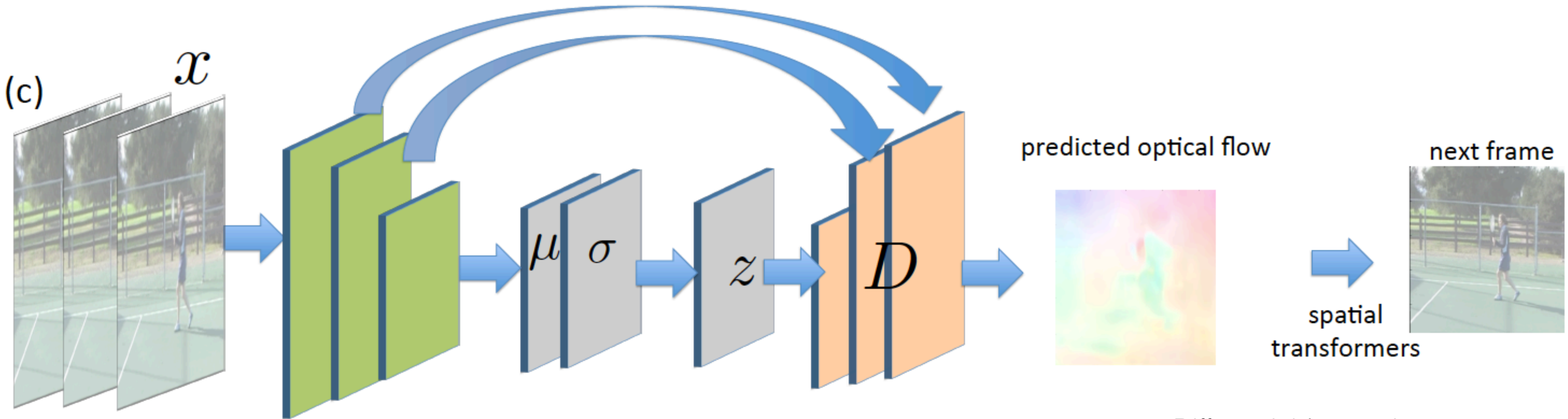


Figure 7: Pushing task. The designated pixel (red diamond) needs to be pushed to the green circle.

Goal representation: move certain pixel of the initial image to desired locations

We will learn a model of pixel motion displacements

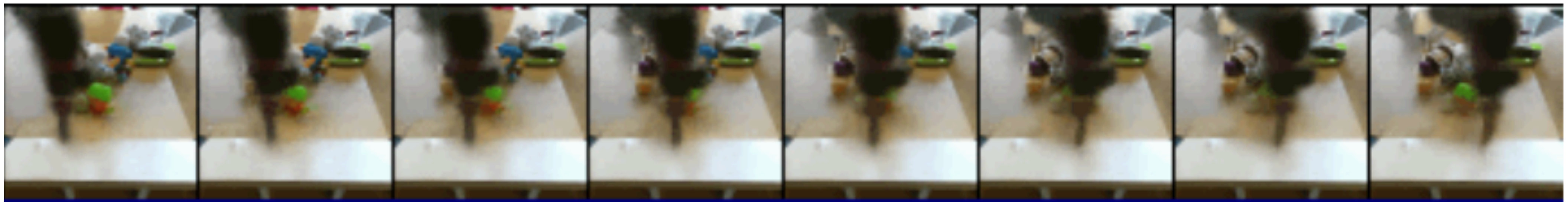
Visual dynamics using motion transformation



Differentiable warping

Can I use this model?

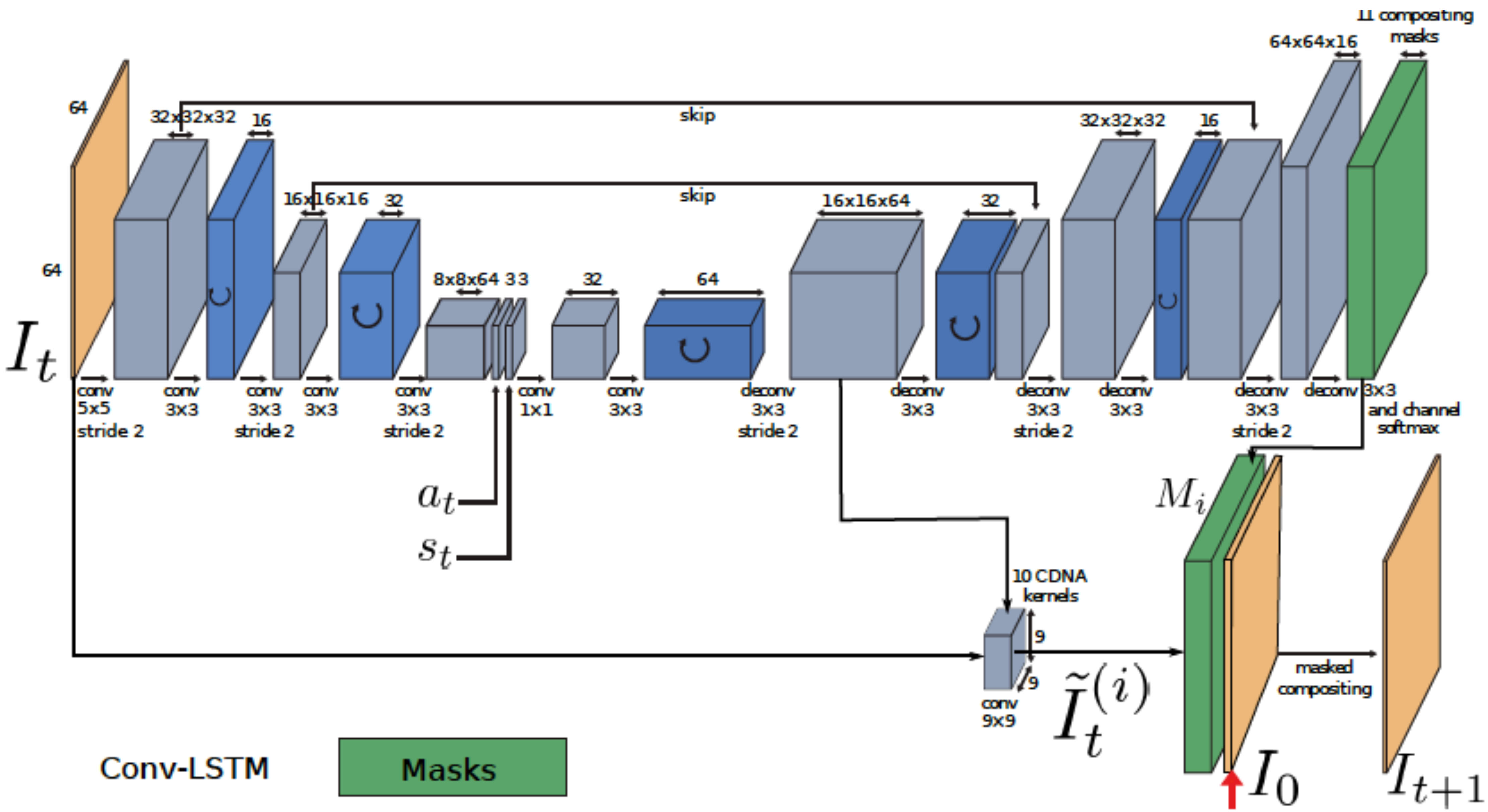
Visual dynamics using motion transformation



$$\hat{I}_{t+1} = I_0 \mathrm{M}_{N+1} + \sum_{i=1}^{N} \tilde{I}_t^{(i)} \mathrm{M}_i$$

$$\hat{I}_{t+1} = \sum_{i=1}^{N} \tilde{I}_t^{(i)} \mathbf{M}_i$$

Visual dynamics using motion transformation



$$\hat{I}_{t+1} = I_0 \mathbf{M}_{N+1} + \sum_{i=1}^{N} \tilde{I}_t^{(i)} \mathbf{M}_i$$

$$\hat{I}_{t+1} = \sum_{i=1}^{N} \tilde{I}_t^{(i)} \mathbf{M}_i$$

Self-Supervised Visual Planning with Temporal Skip Connections, Ebert et al.

Visual dynamics using motion transformation

https://sites.google.com/view/sna-visual-mpc

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 useful 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 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

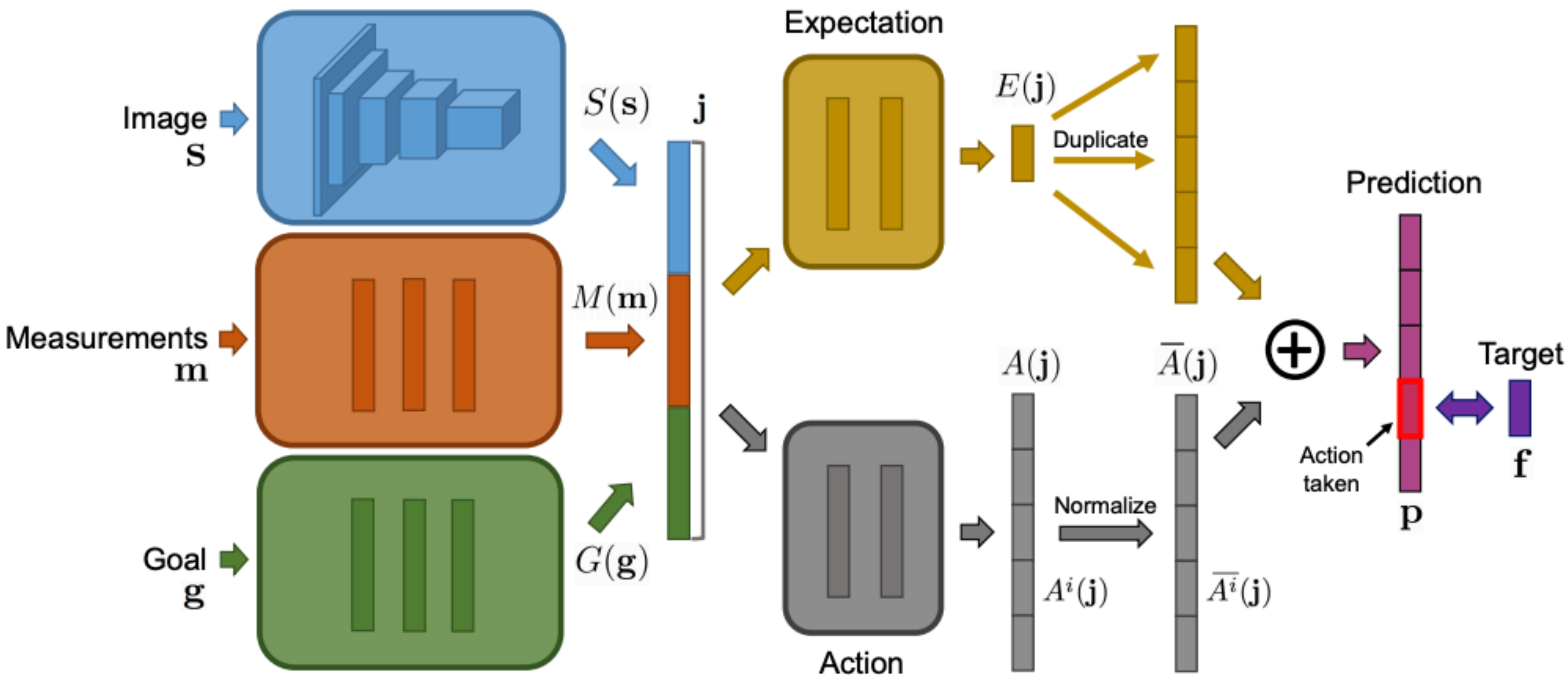
Train a deep predictor. No unrolling! One shot prediction of future values:

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

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

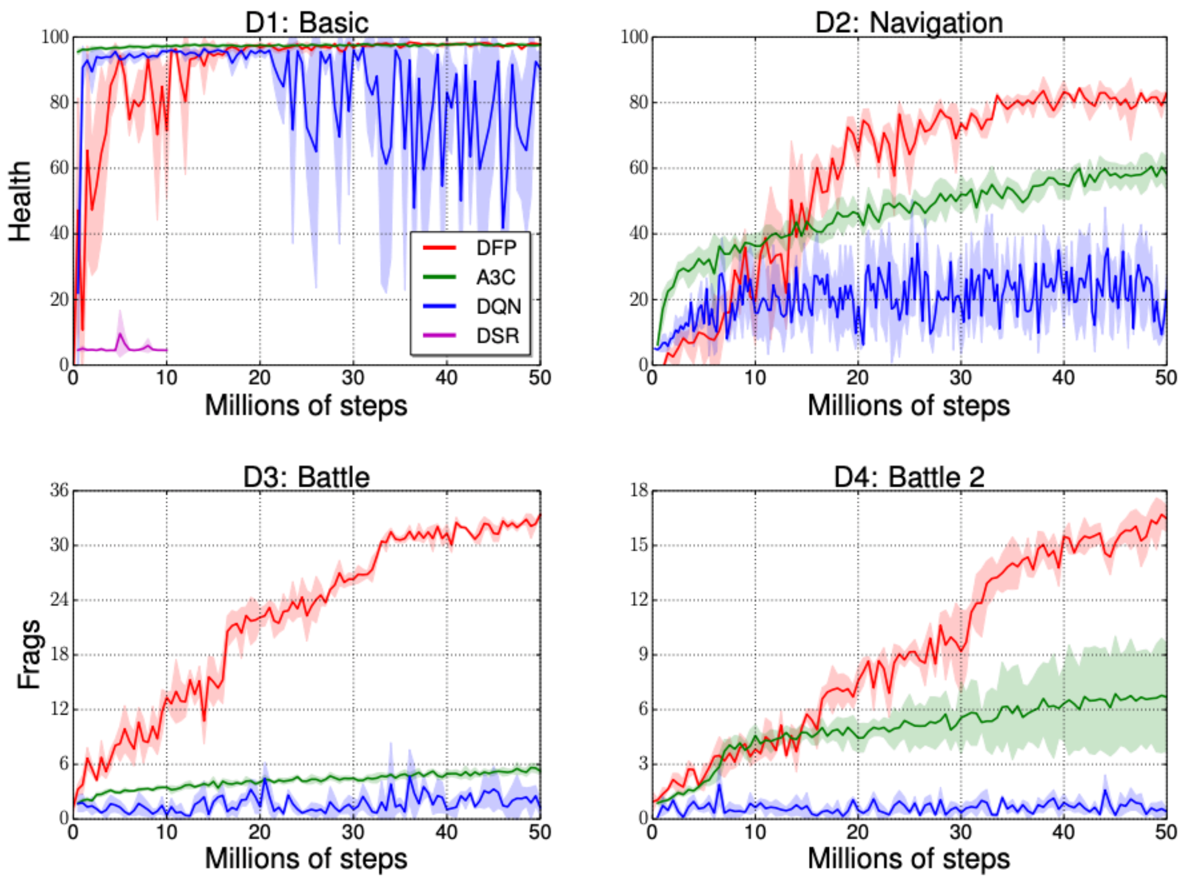


Action selection:

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

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