Carnegie Mellon
School of Computer Science

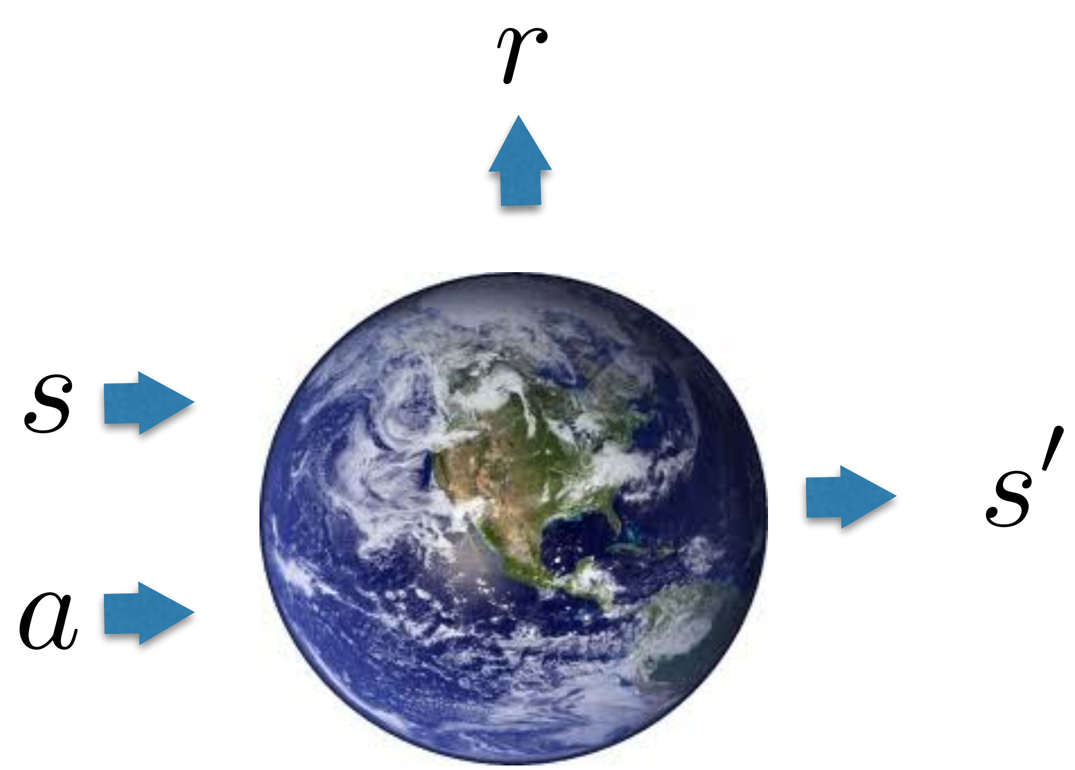
Deep Reinforcement Learning and Control

Model Based Reinforcement Learning

Katerina Fragkiadaki

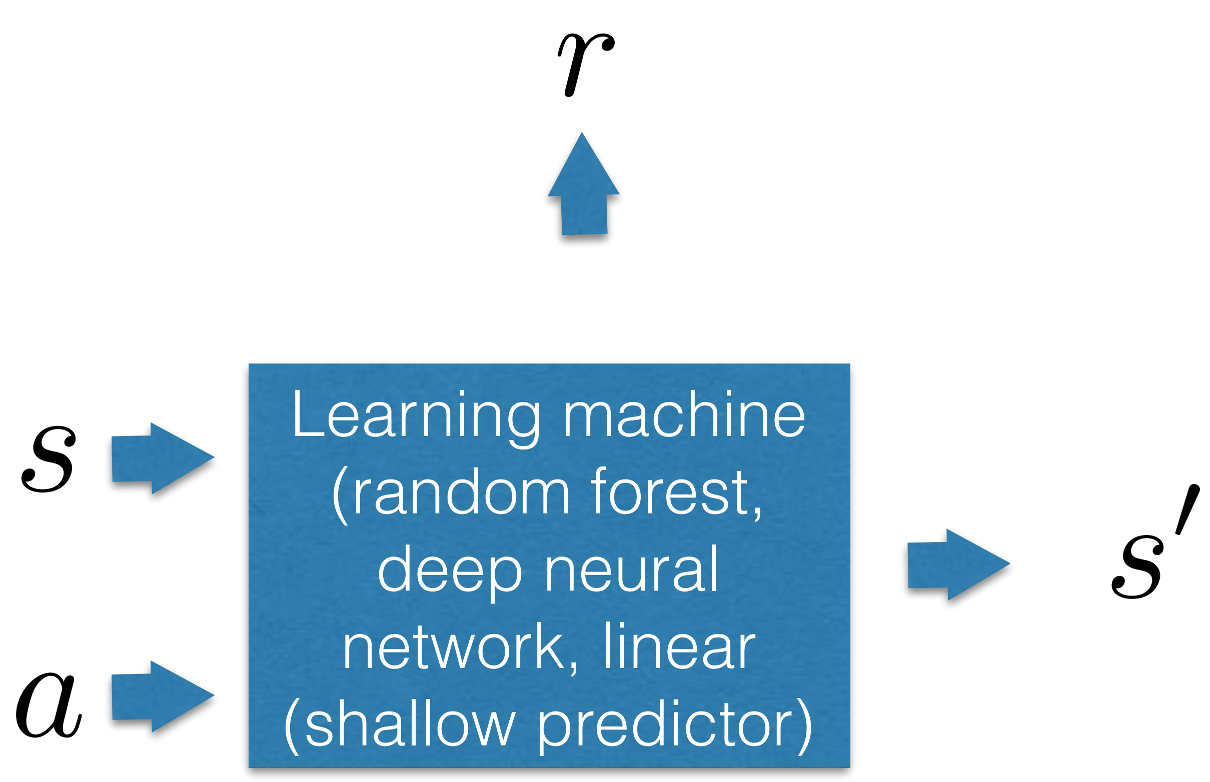
Model

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.



Learning Dynamics

Newtonian Physics equations

VS

general parametric form (no prior from Physics knowledge)

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

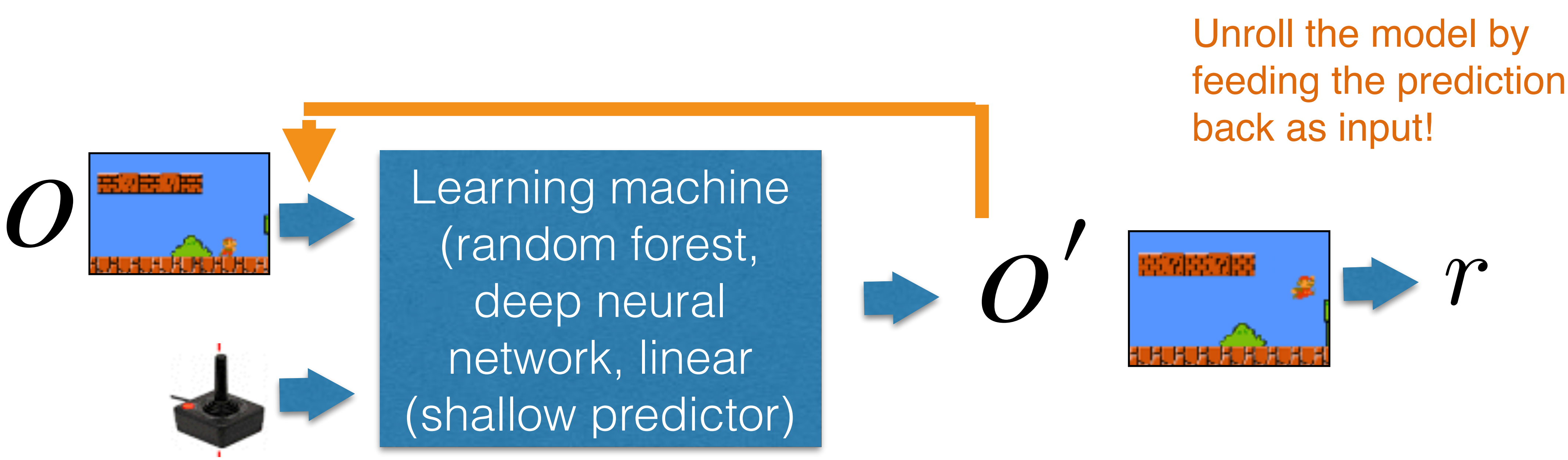
Much easier to learn but suffers from under-modeling, bad models

Neural networks: *lots* of unknown parameters

Very flexible, very hard to get it to generalize

Observation prediction

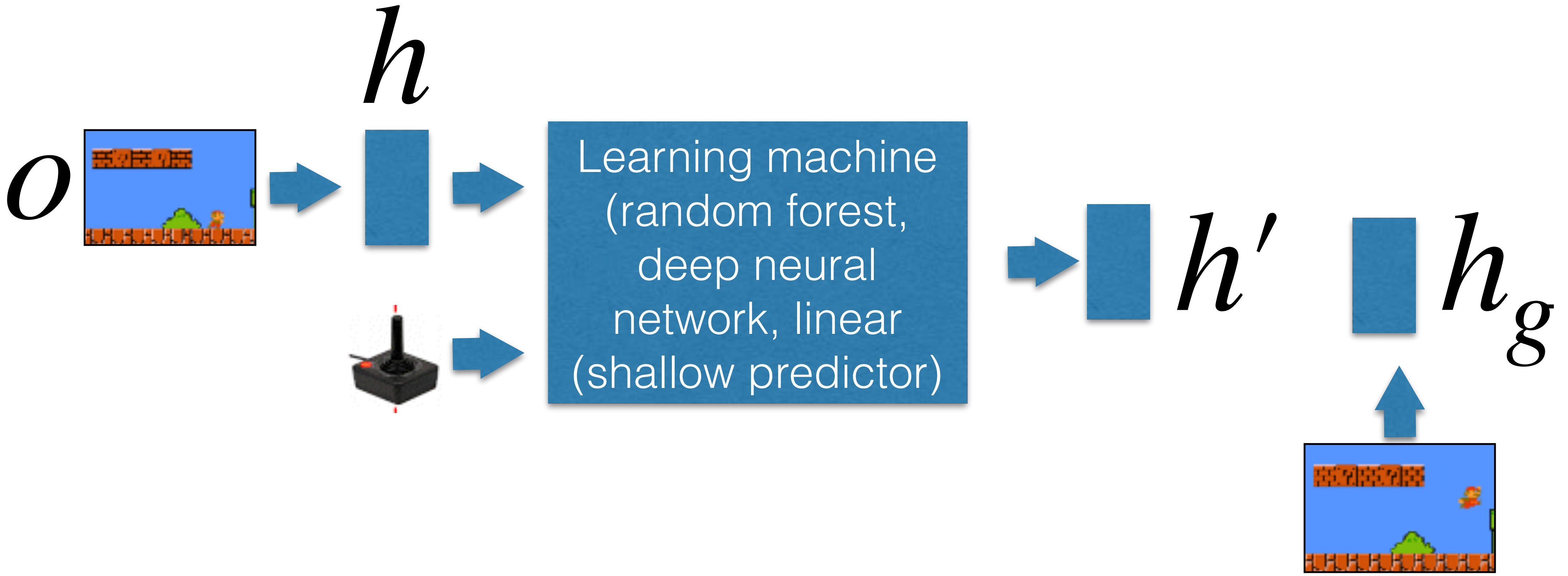
Our model tries to predict the observations. Why?
Because MANY different rewards can be computed once I have access to the future visual observation, e.g., make Mario jump, make Mario move to the right, to the left, lie down, make Mario jump on the well and then jump back down again etc..
If I was just predicting rewards, then I can only plan towards that specific goal, e.g., win the game, same in the model-free case.



Prediction in a latent space

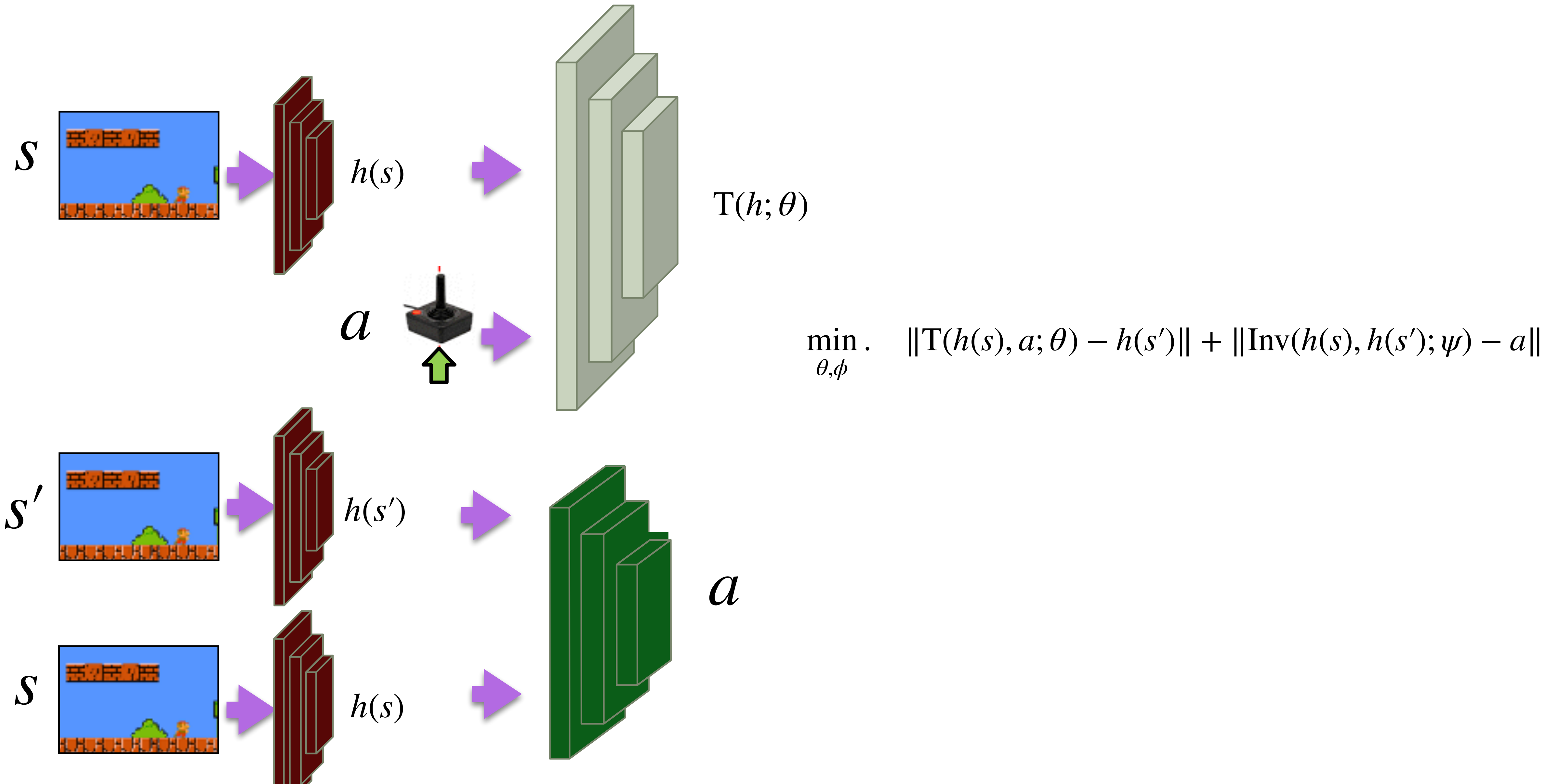
Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.

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



Prediction in a latent space

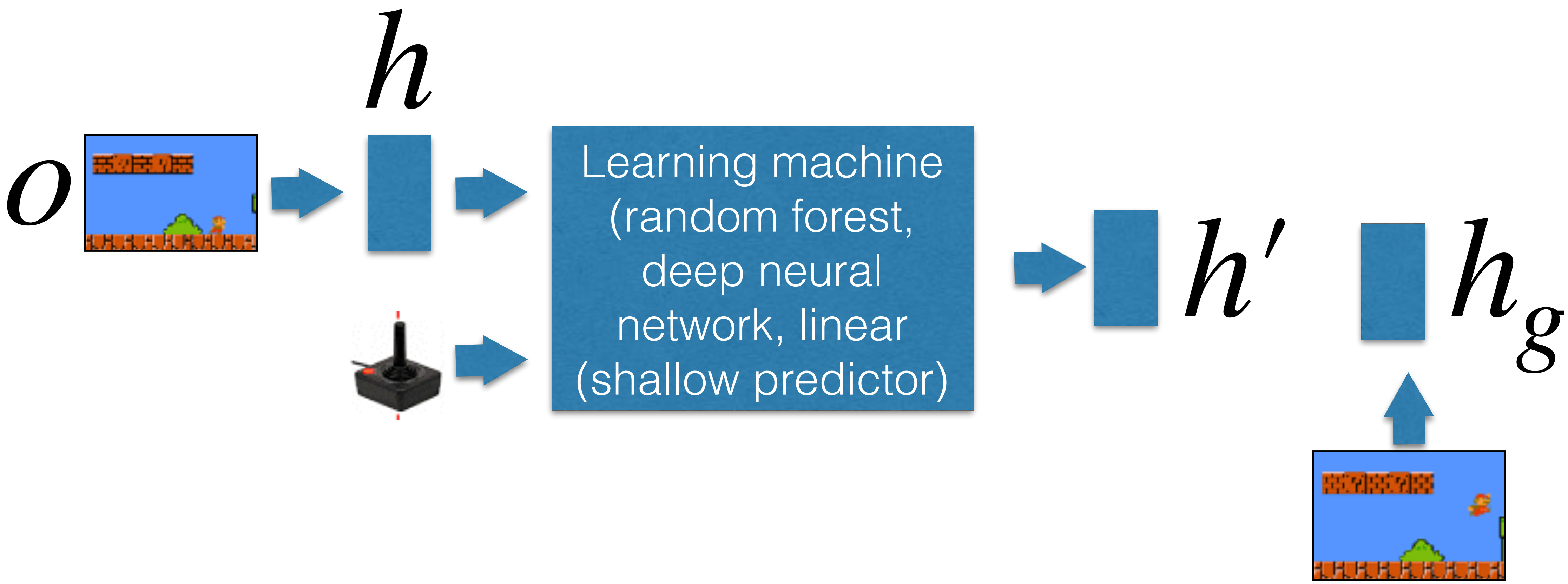
Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.
One such feature encoding we have seen is the one that keep from the observation ONLY whatever is controllable by the agent.



Prediction in a latent space

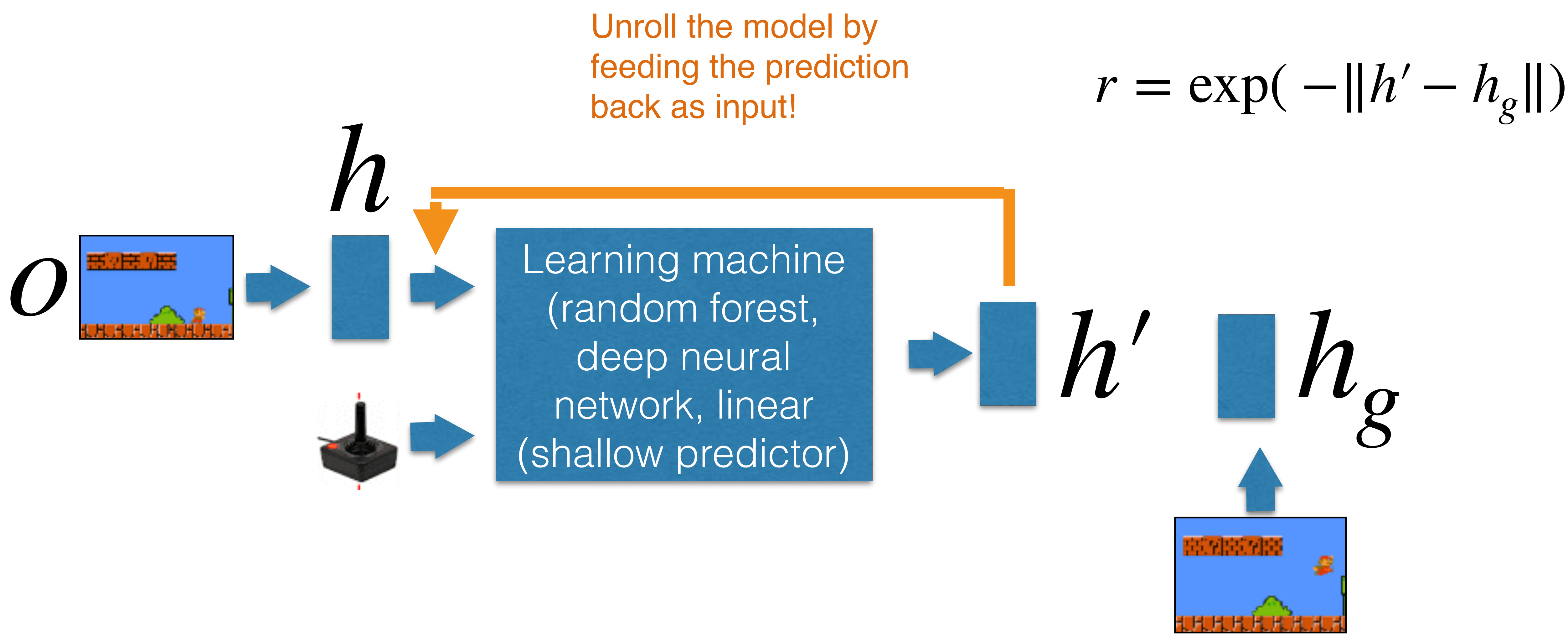
Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.

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



Prediction in a latent space

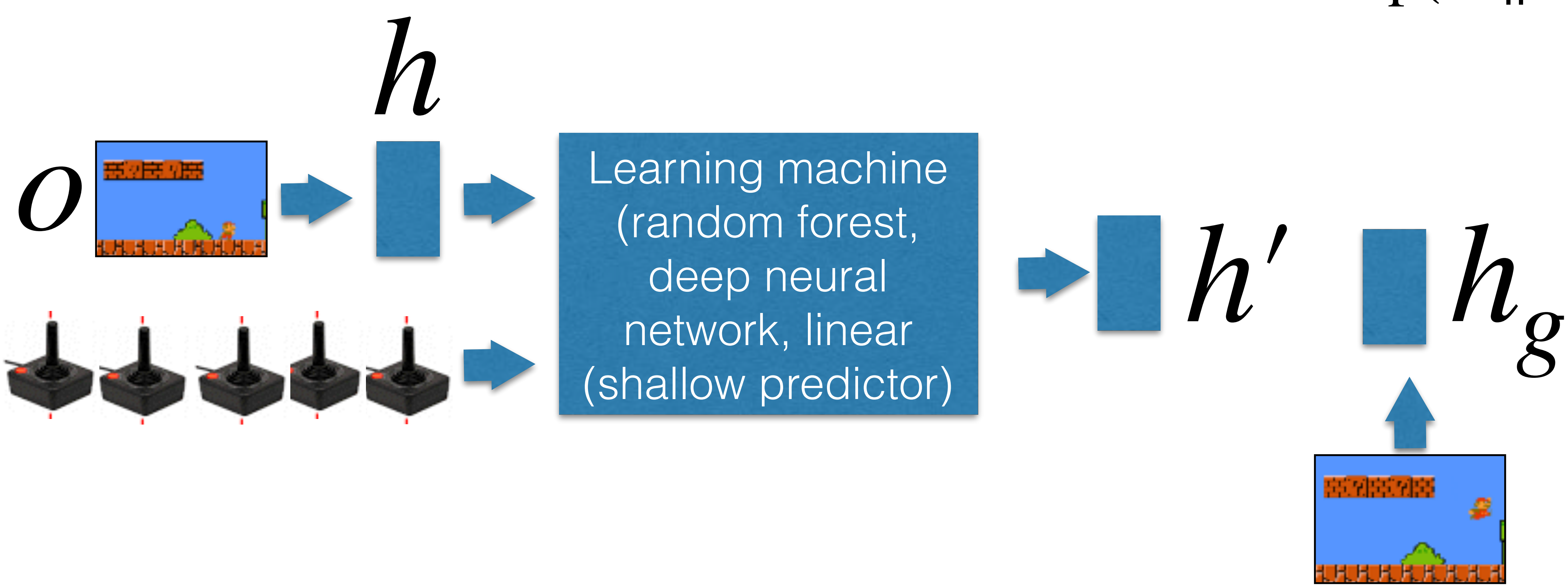
Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.



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

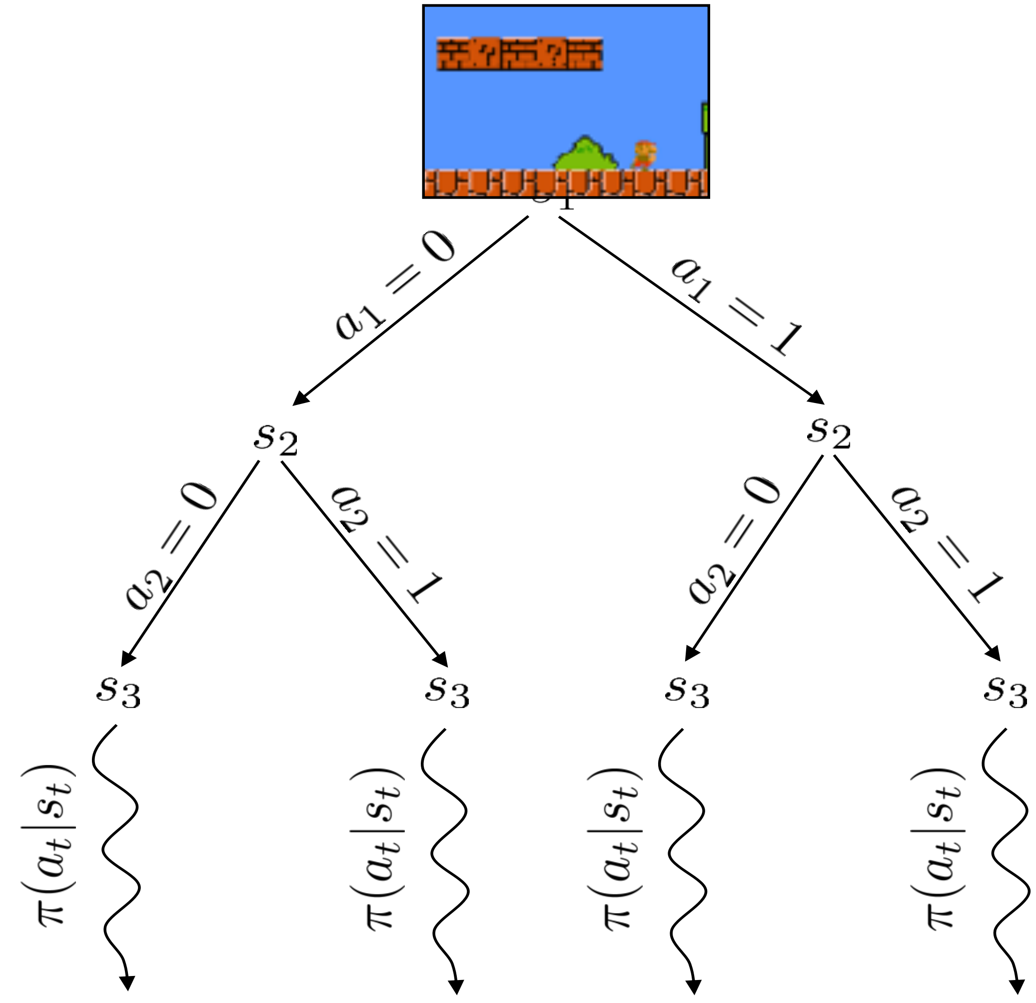
Why model learning

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

Given a state I unroll my model forward and seek the action that results in the highest reward. How do I select this action?

1. I discretize my action space and perform tree-search
2. I use continuous gradient descent to optimize over actions

a_t



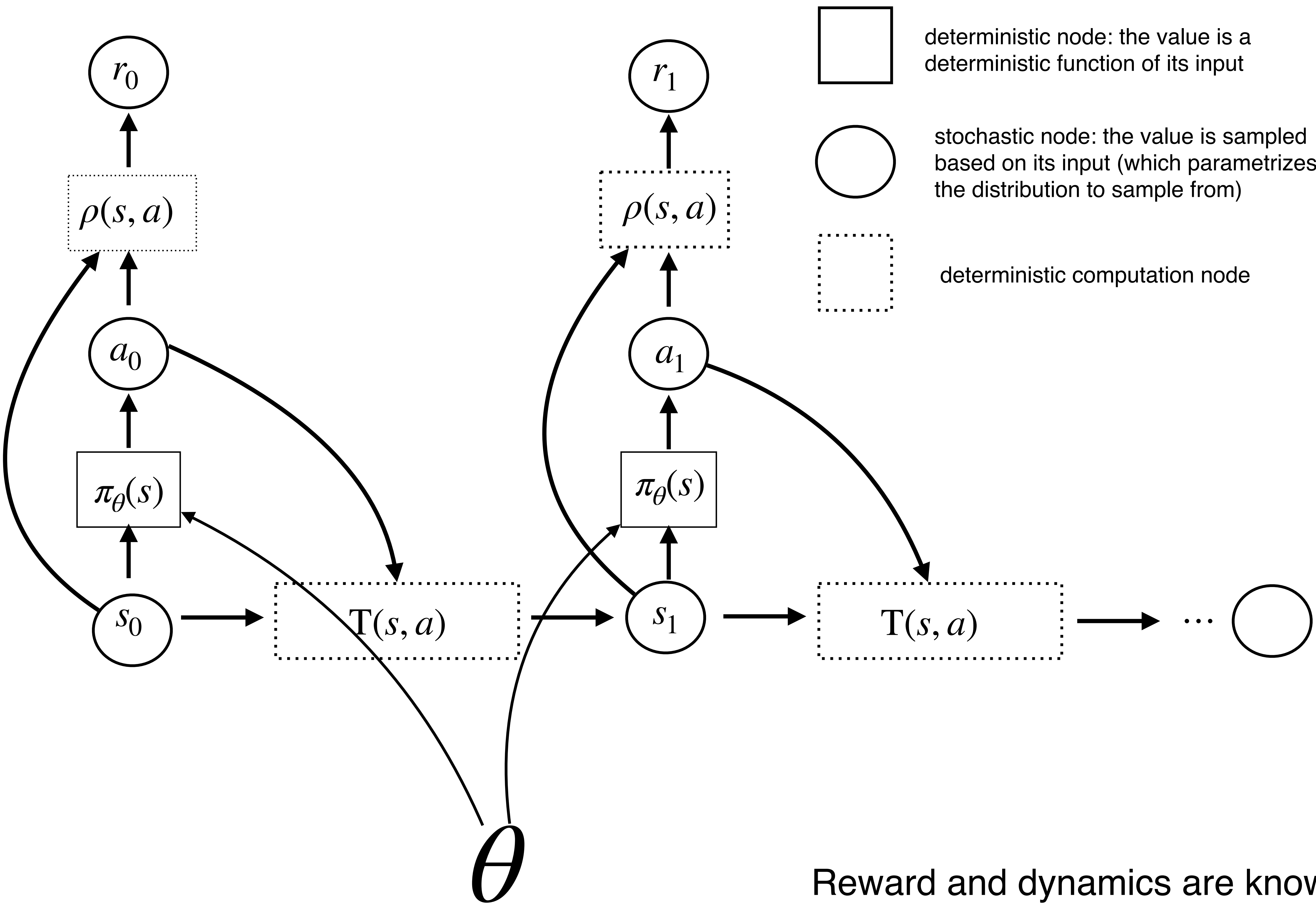
Why model learning

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

Given a state I unroll my model forward and seek the action that results in the highest reward. How do I select this action?

1. I discretize my action space and perform tree-search
2. I use continuous gradient descent to optimize over actions

Bachpropagate to actions

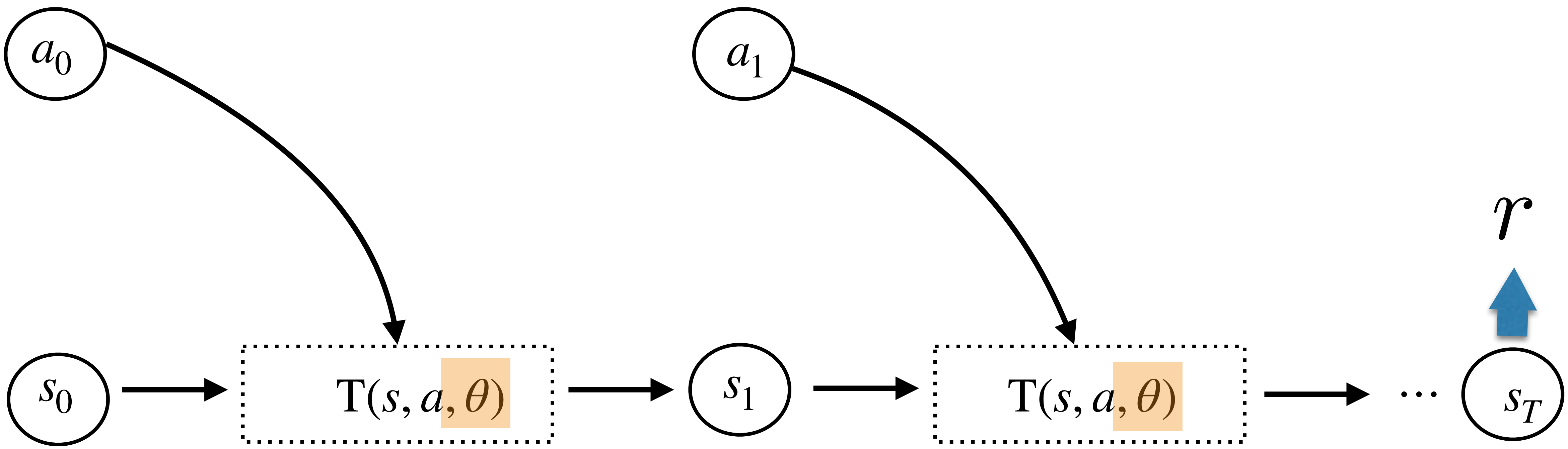


Reward and dynamics are known

Bachpropagate to actions

Given a state I unroll my model forward and seek the action that results in the highest reward. How do I select this action?

1. I discretize my action space and perform tree-search
2. I use continuous gradient descent to optimize over actions



No policy learned, action selection directly by backpropagating through the dynamics, the continuous analog of online planning

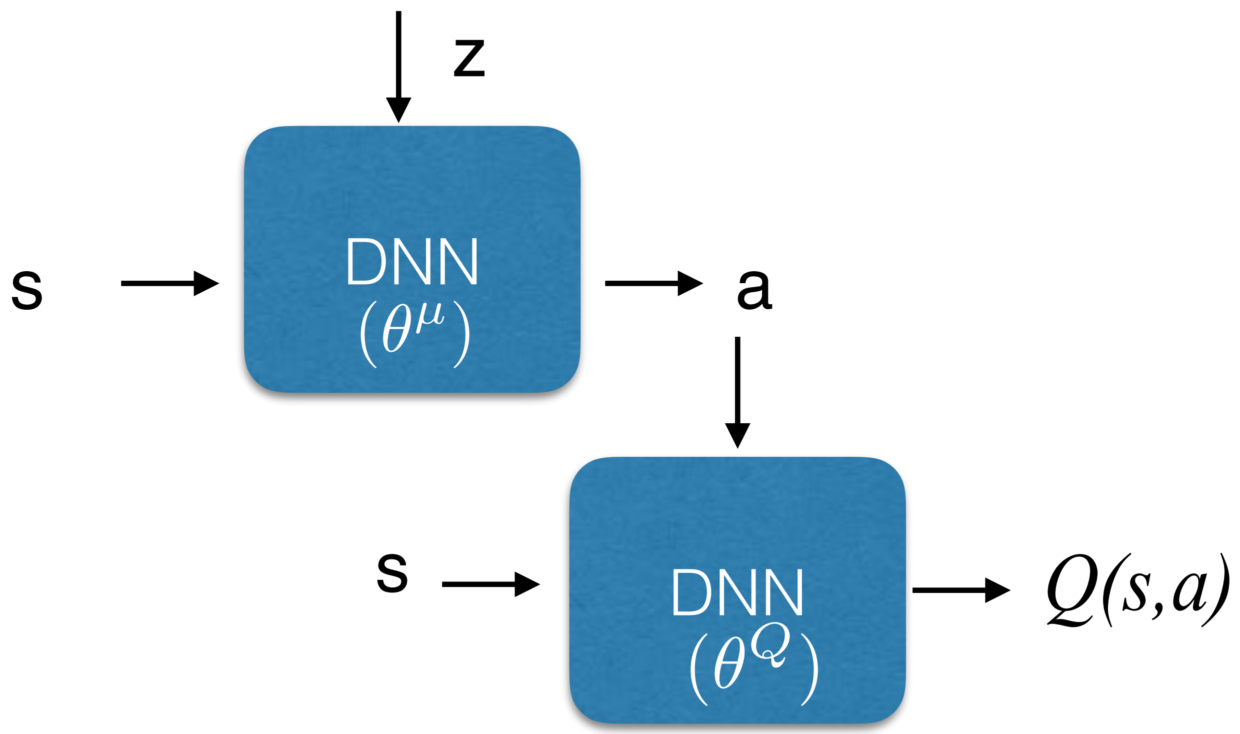
dynamics are frozen, we backpropagate to actions directly

Why model learning

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

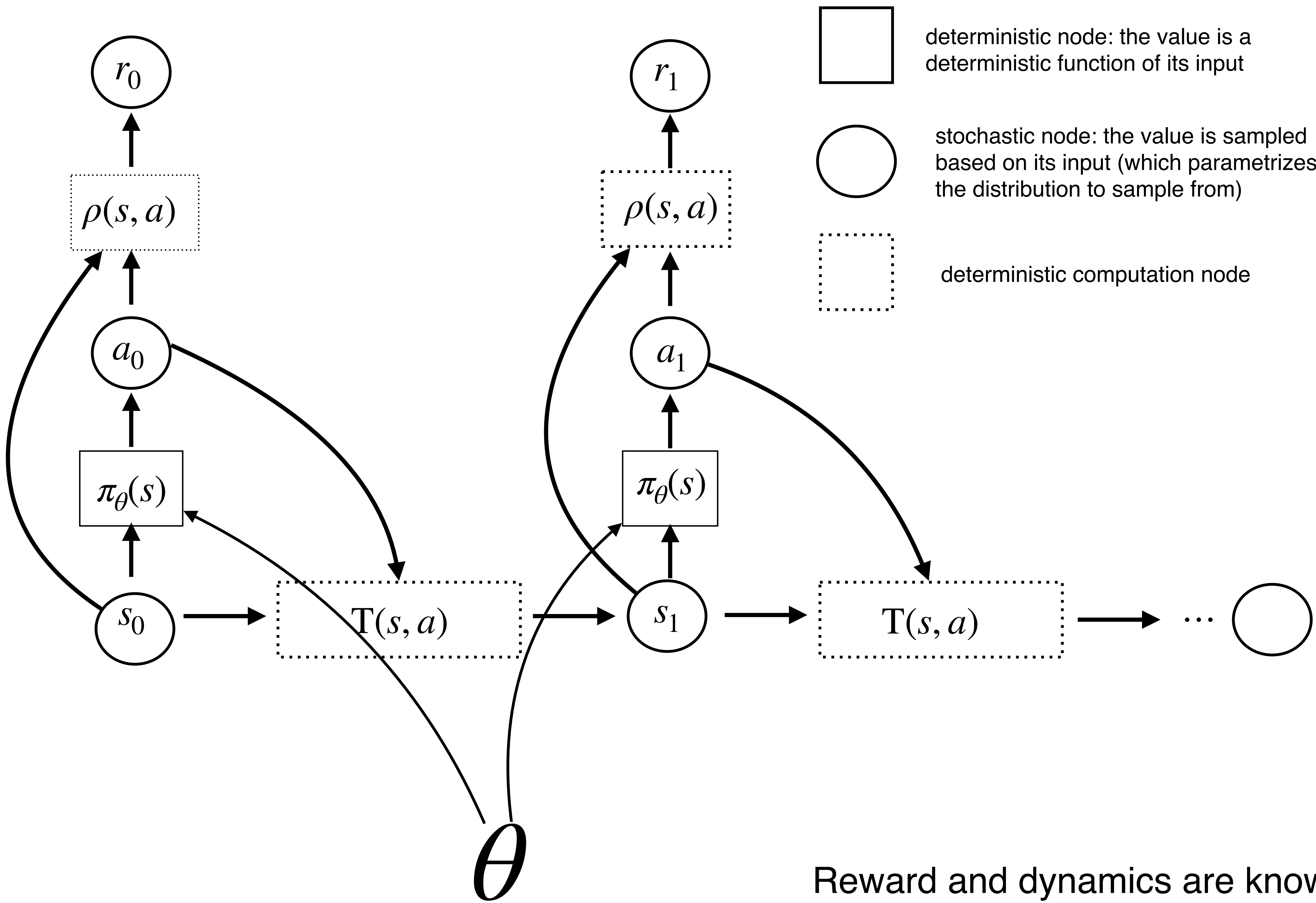
Remember: Stochastic Value Gradients V0

$$z \sim \mathcal{N}(0, 1)$$



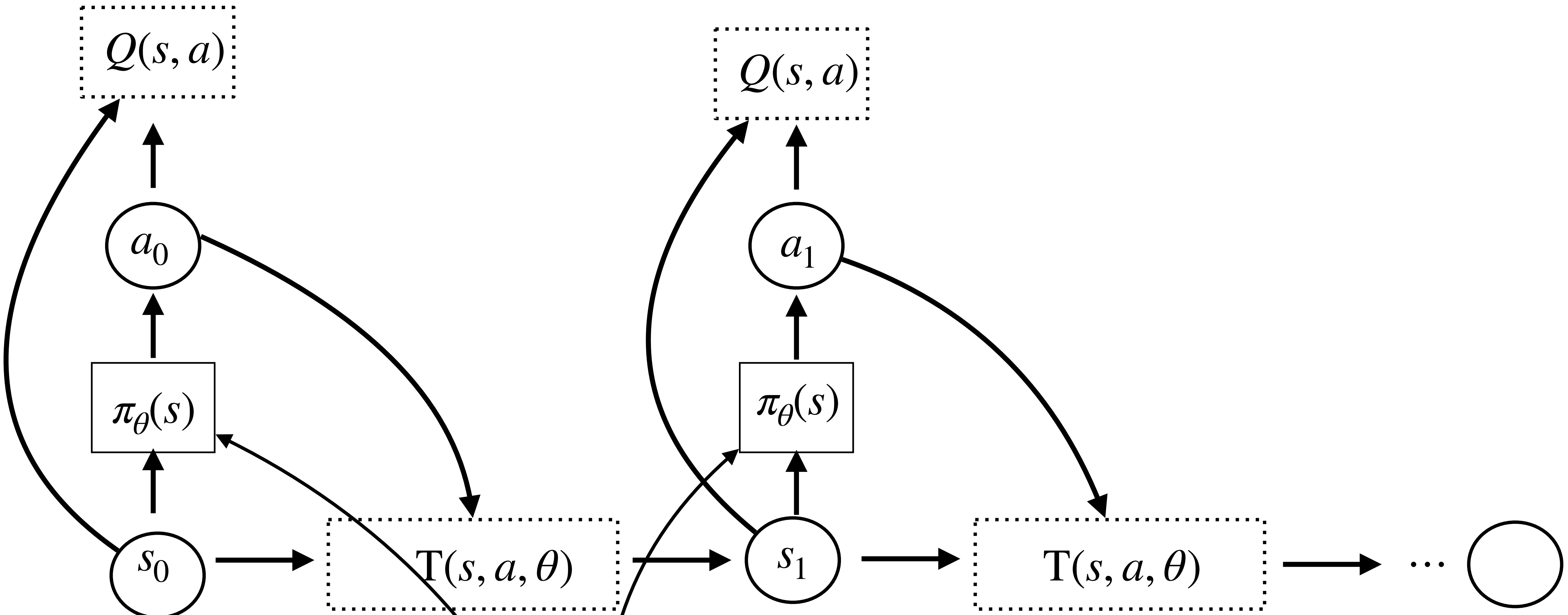
$$a = \mu(s;\theta) + z\sigma(s;\theta)$$

Bachpropagate to the policy

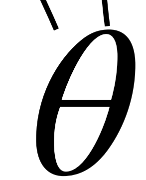


Reward and dynamics are known

Bachpropagate to the policy



dynamics are frozen, backprogate to the policy directly by maximizing Q within a time horizon



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

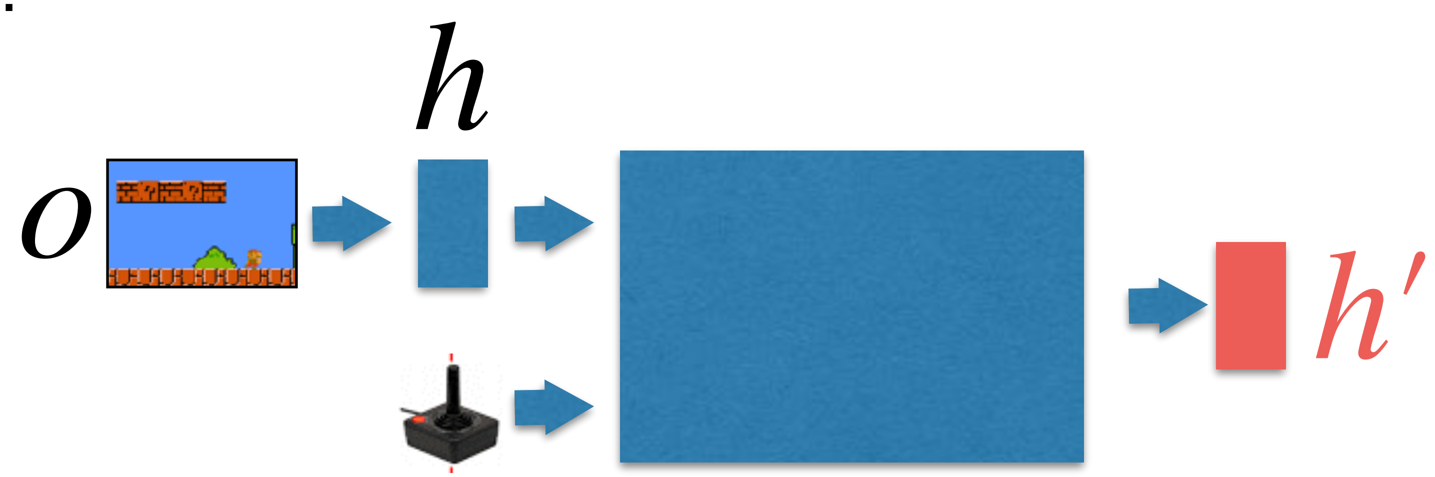
Answers:

- 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)

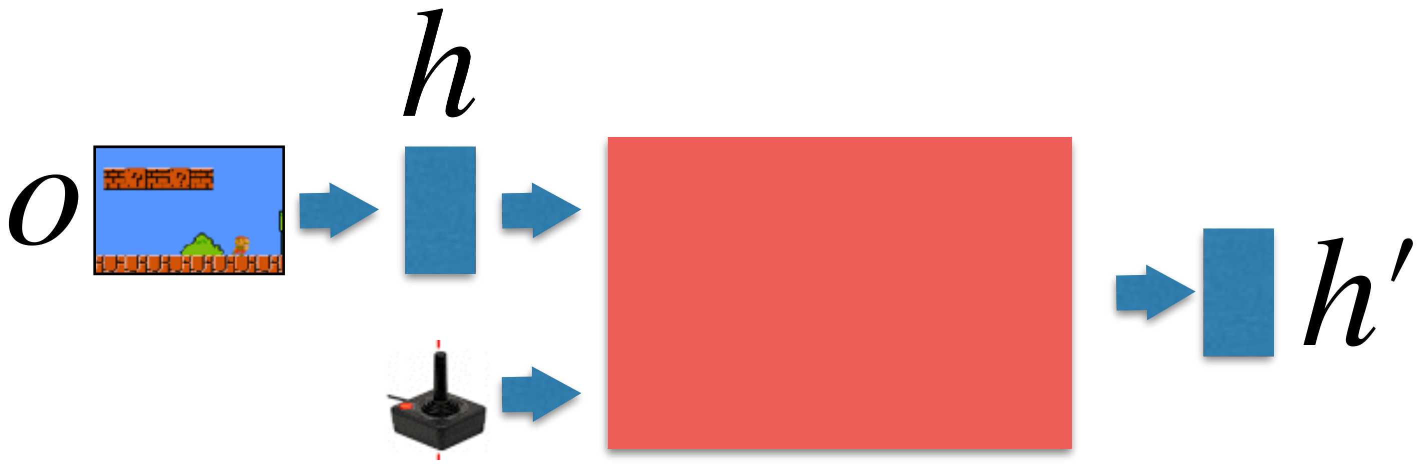
Model Learning

Three questions always in mind

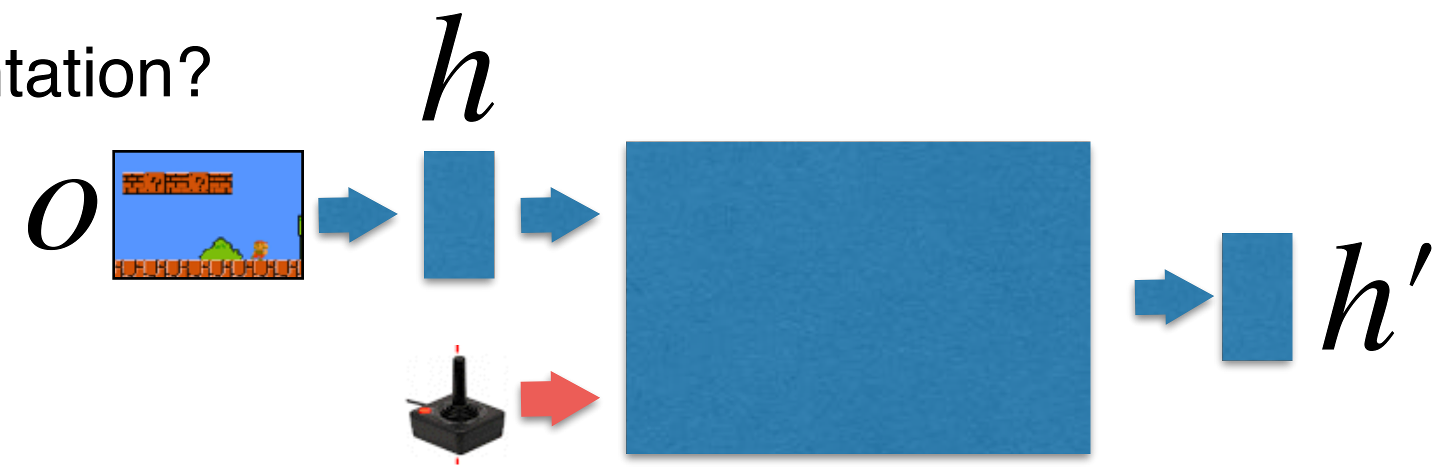
- What shall we be predicting?



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

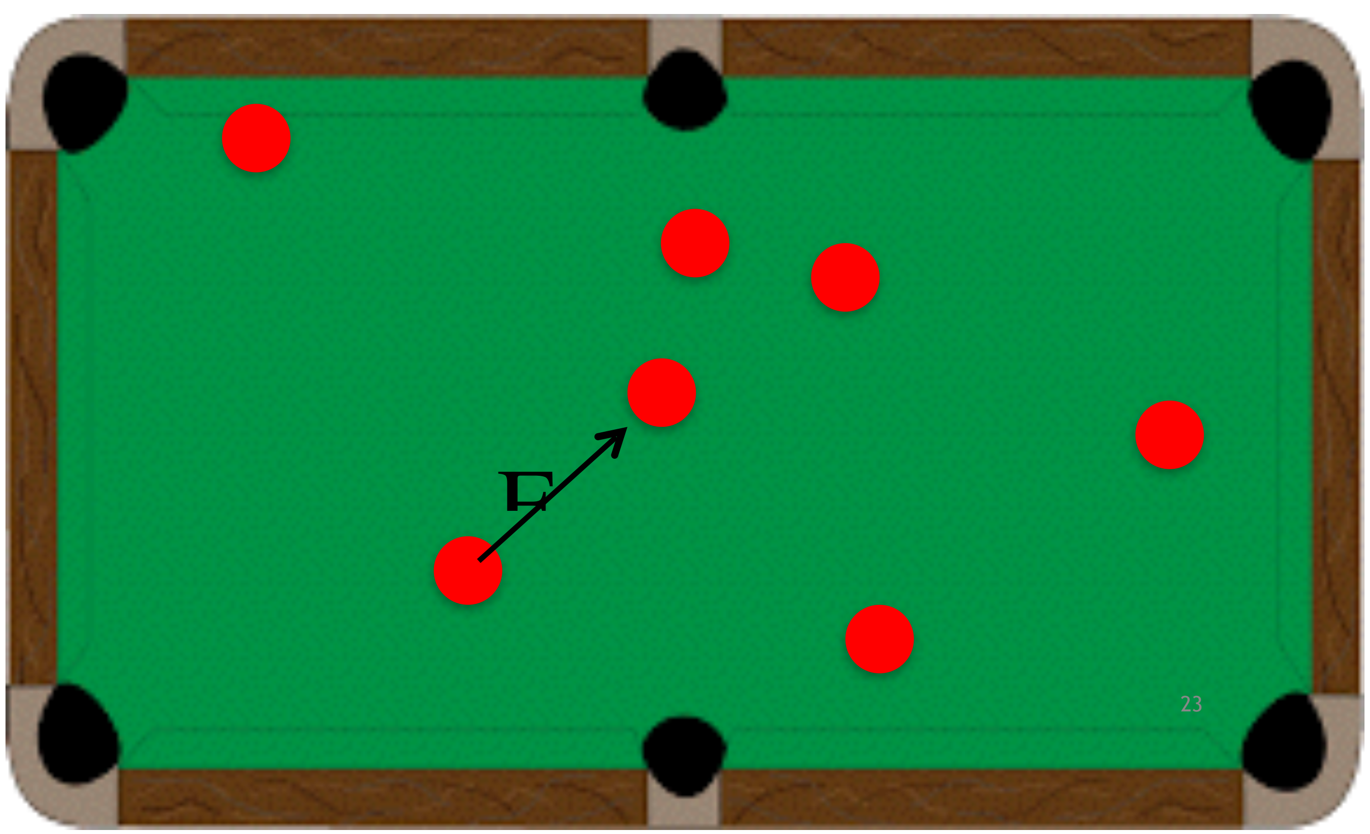


- What is the action representation?

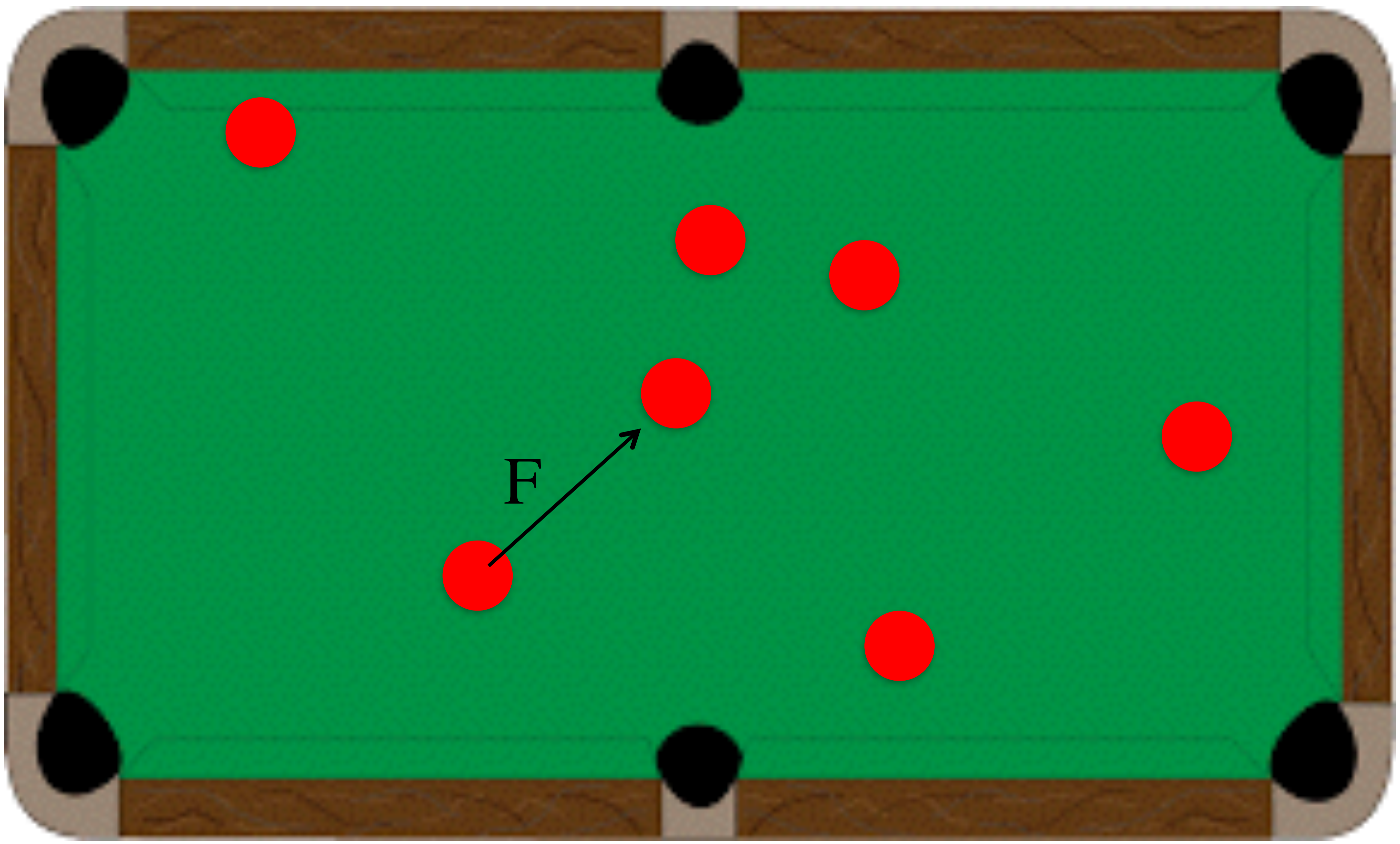


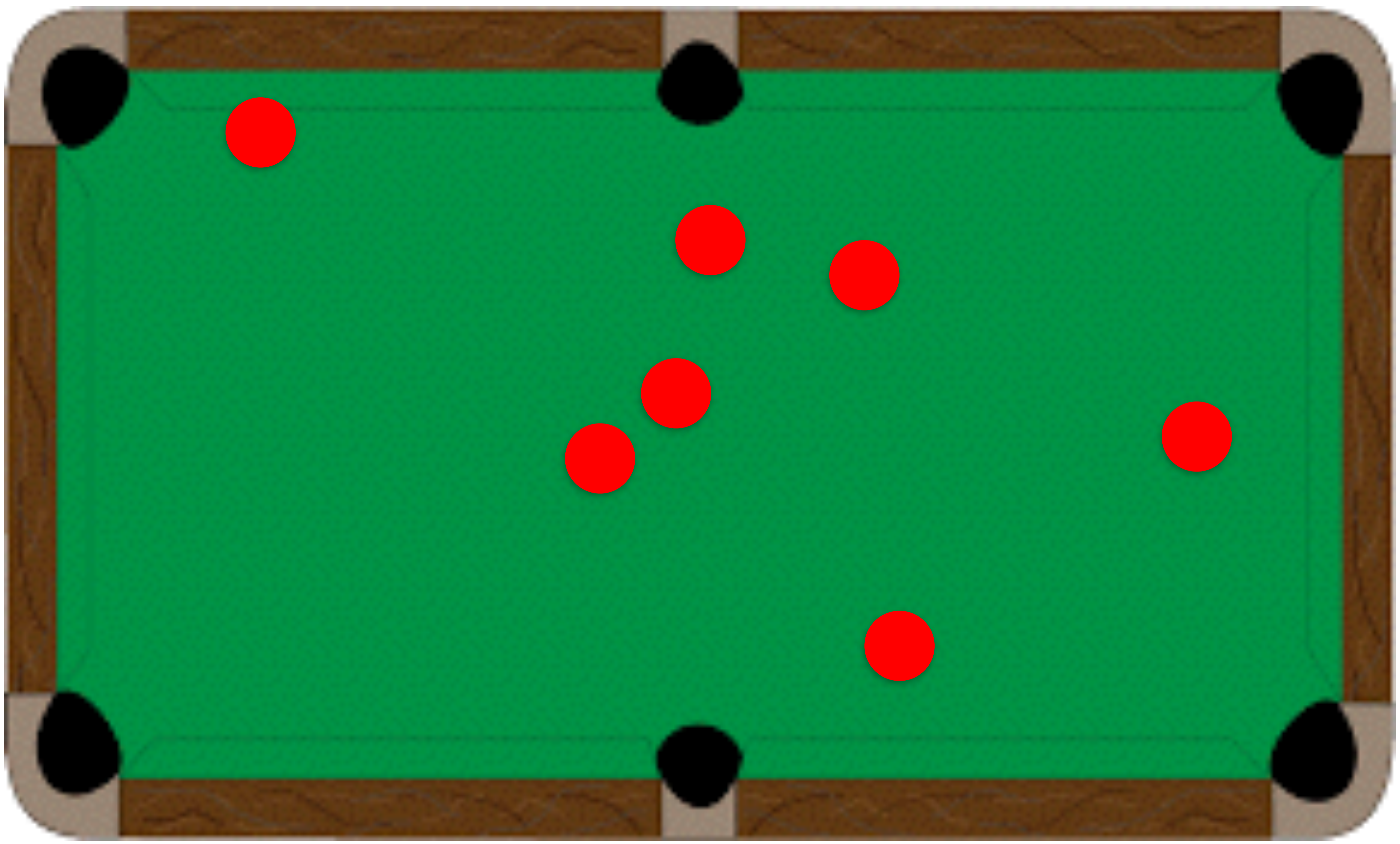
How do we learn to play Billiards?

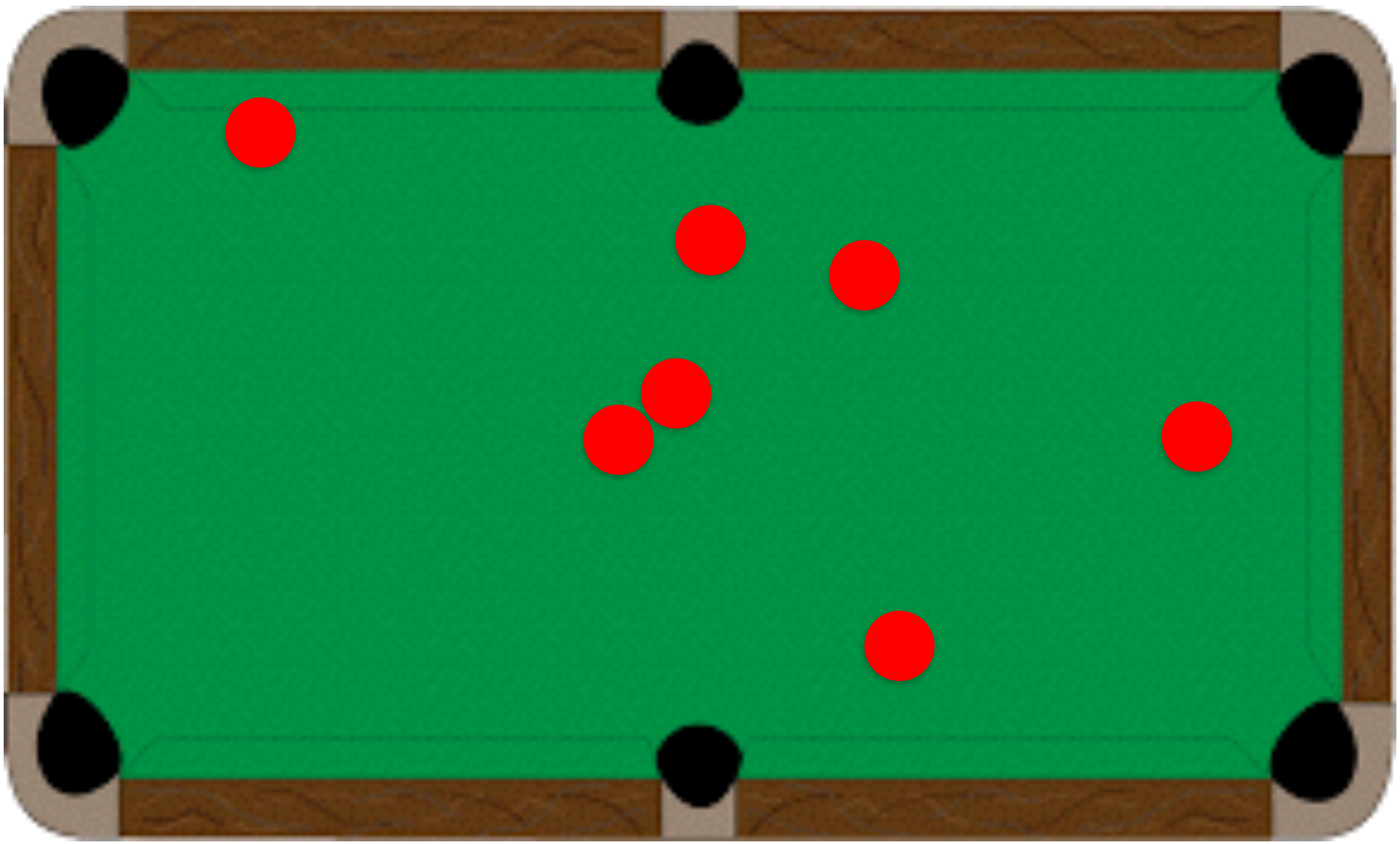
- First, we tranfer all knowledge about how objects move, that we have accumulated so far.
- Second, we watch other people play and practise ourselves, to finetune such model knowledge

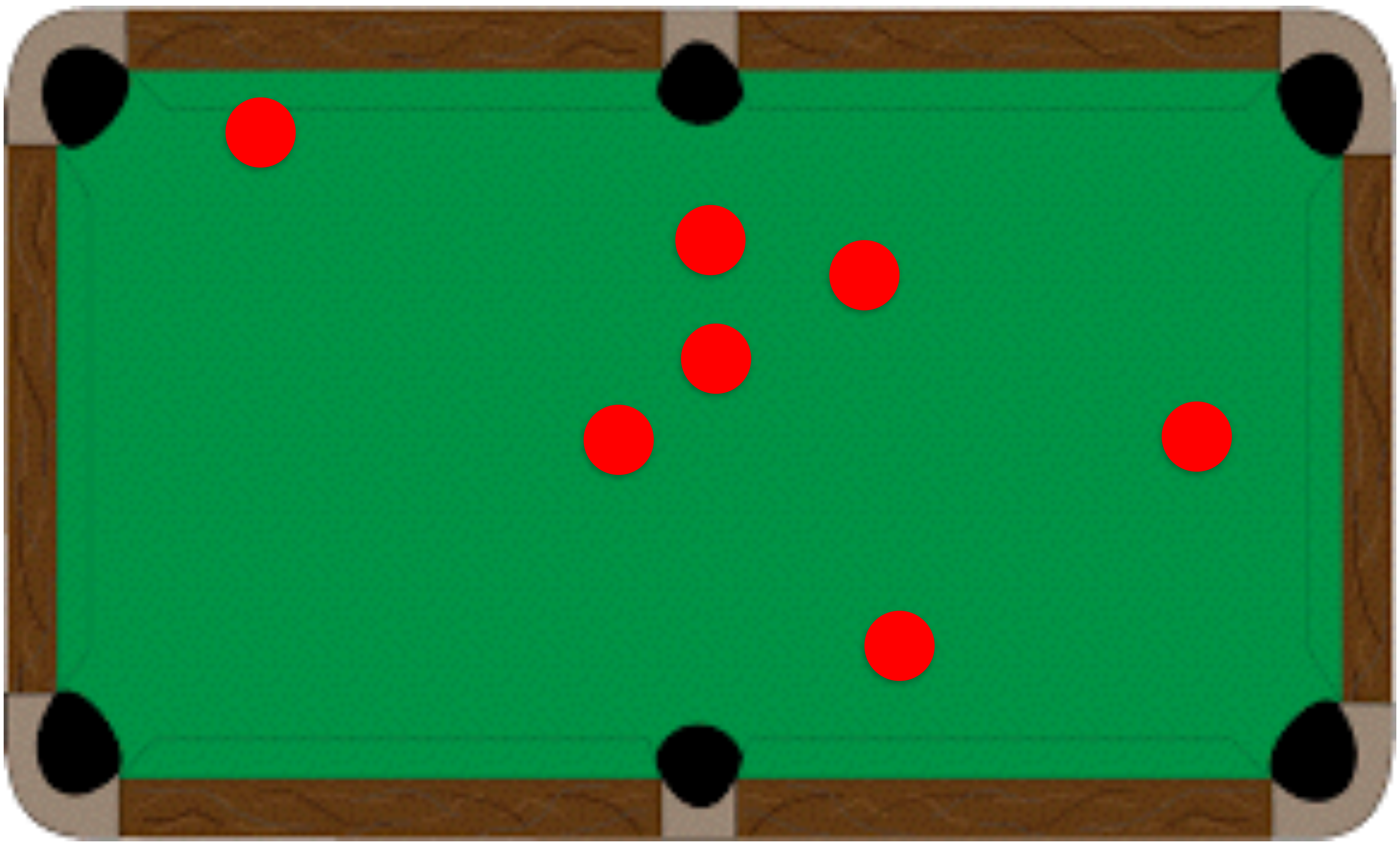


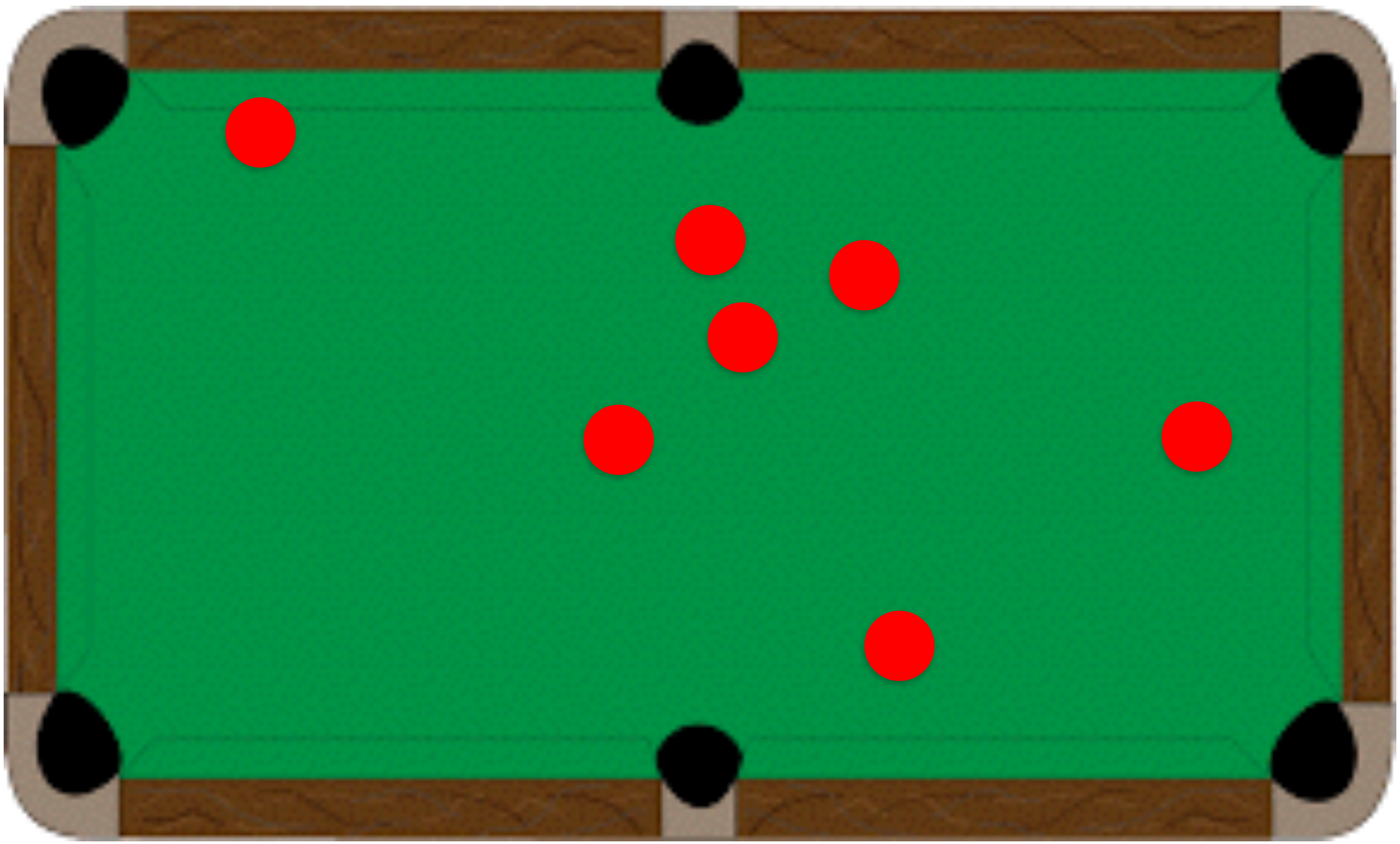
How do we learn to play Billiards?





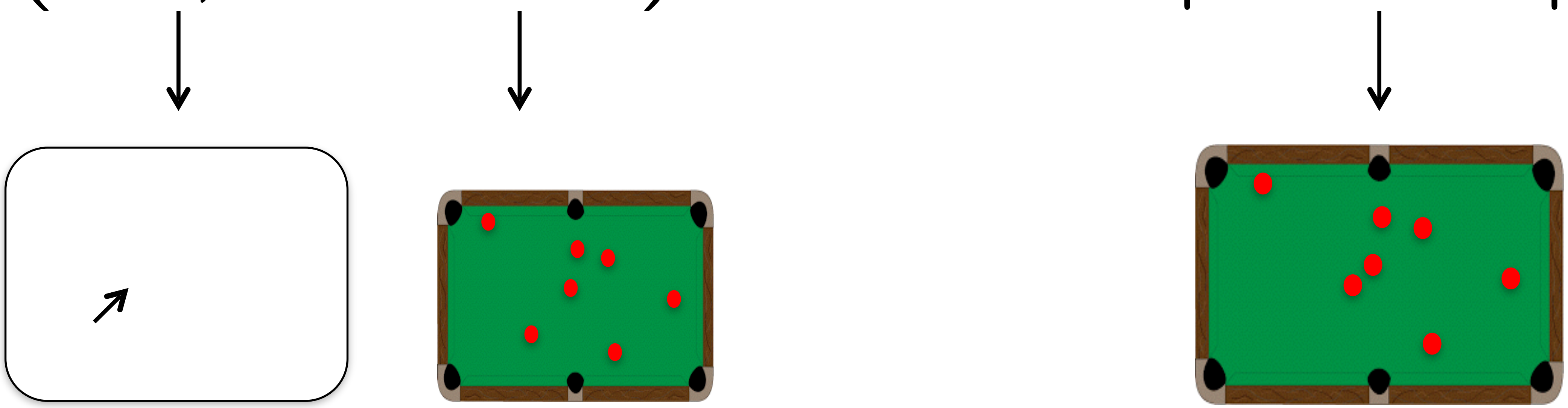






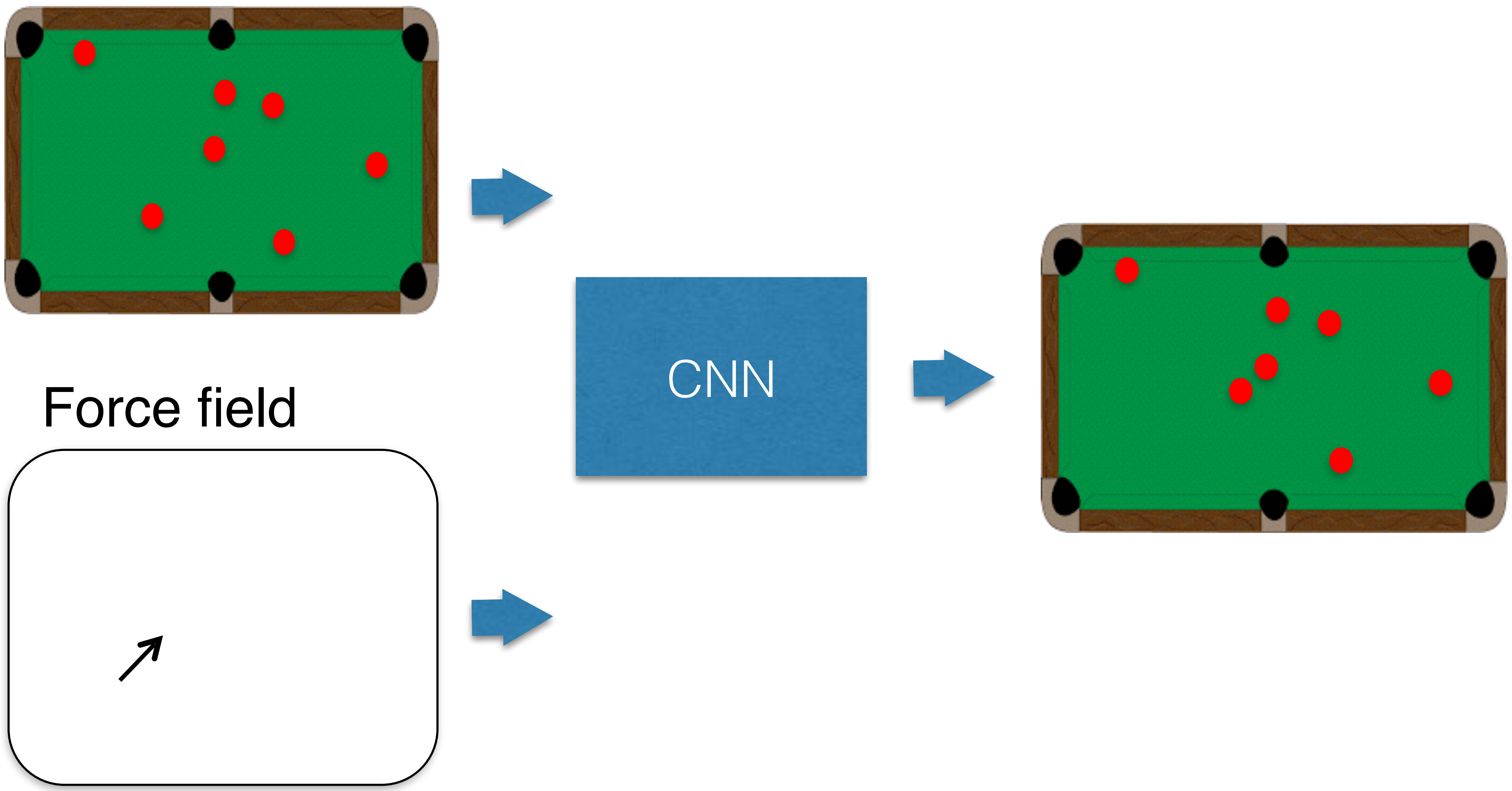
Learning Action-Conditioned Billiard Dynamics

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



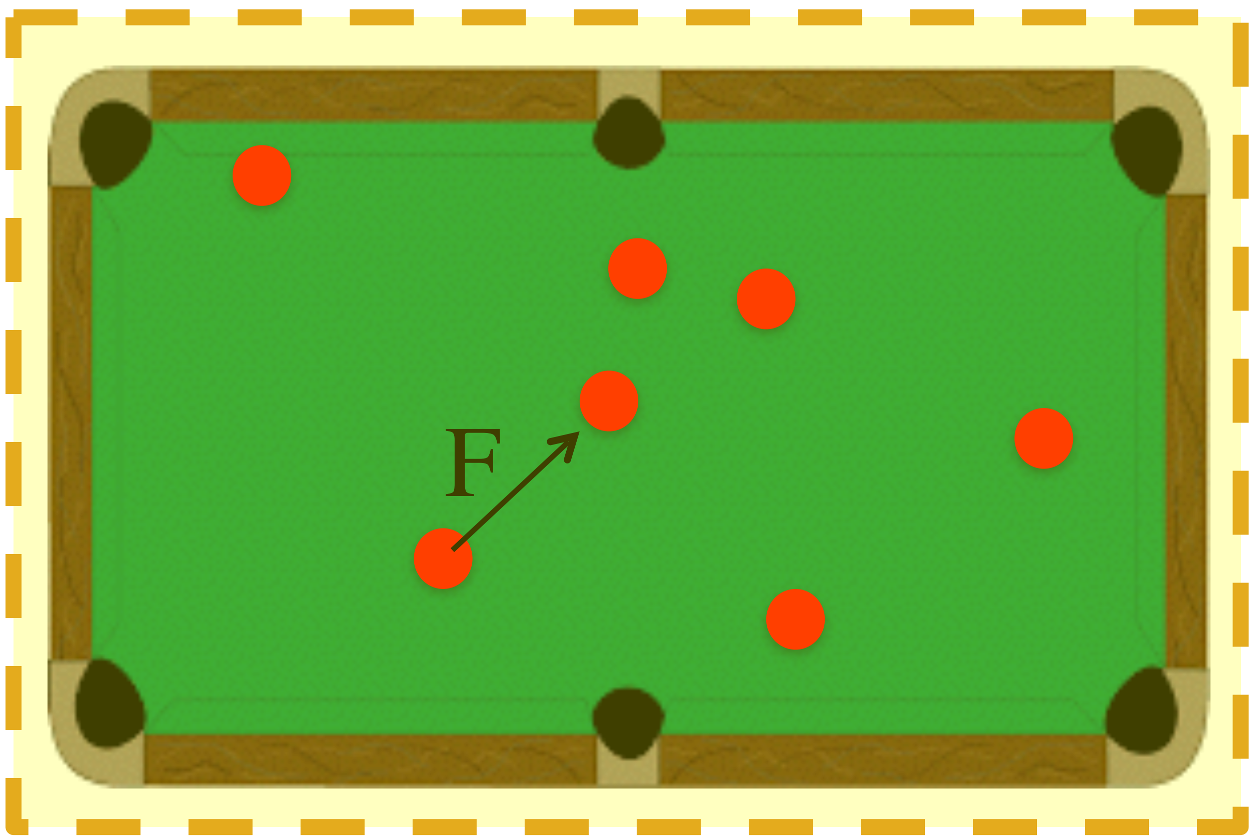
Predictive Visual Models of Physics for Playing Billiards, K.F. et al. ICLR 2016

Learning Action-Conditioned Billiard Dynamics

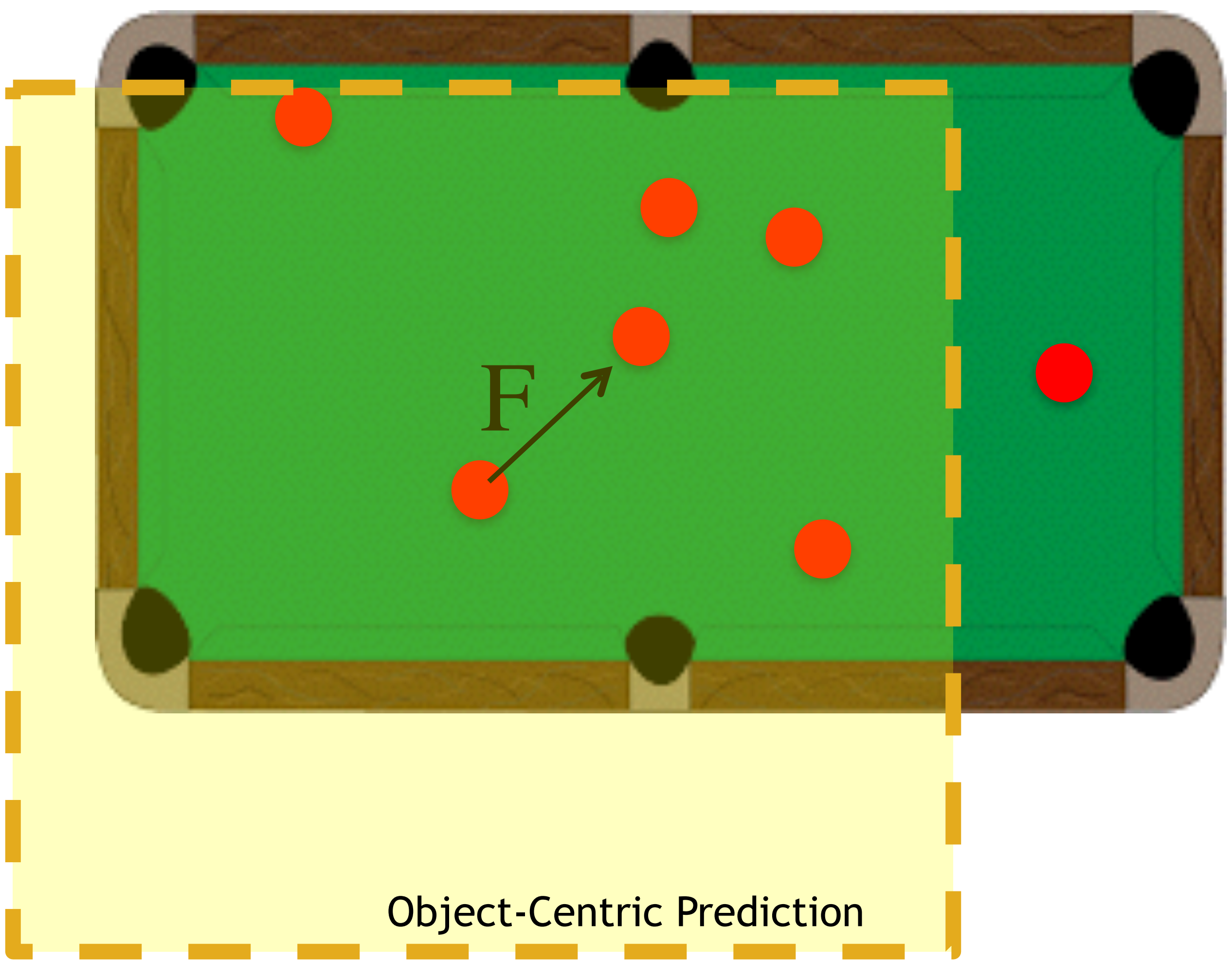


Q: will our model be able to generalize across different number of balls present?

Learning Action-Conditioned Billiard Dynamics

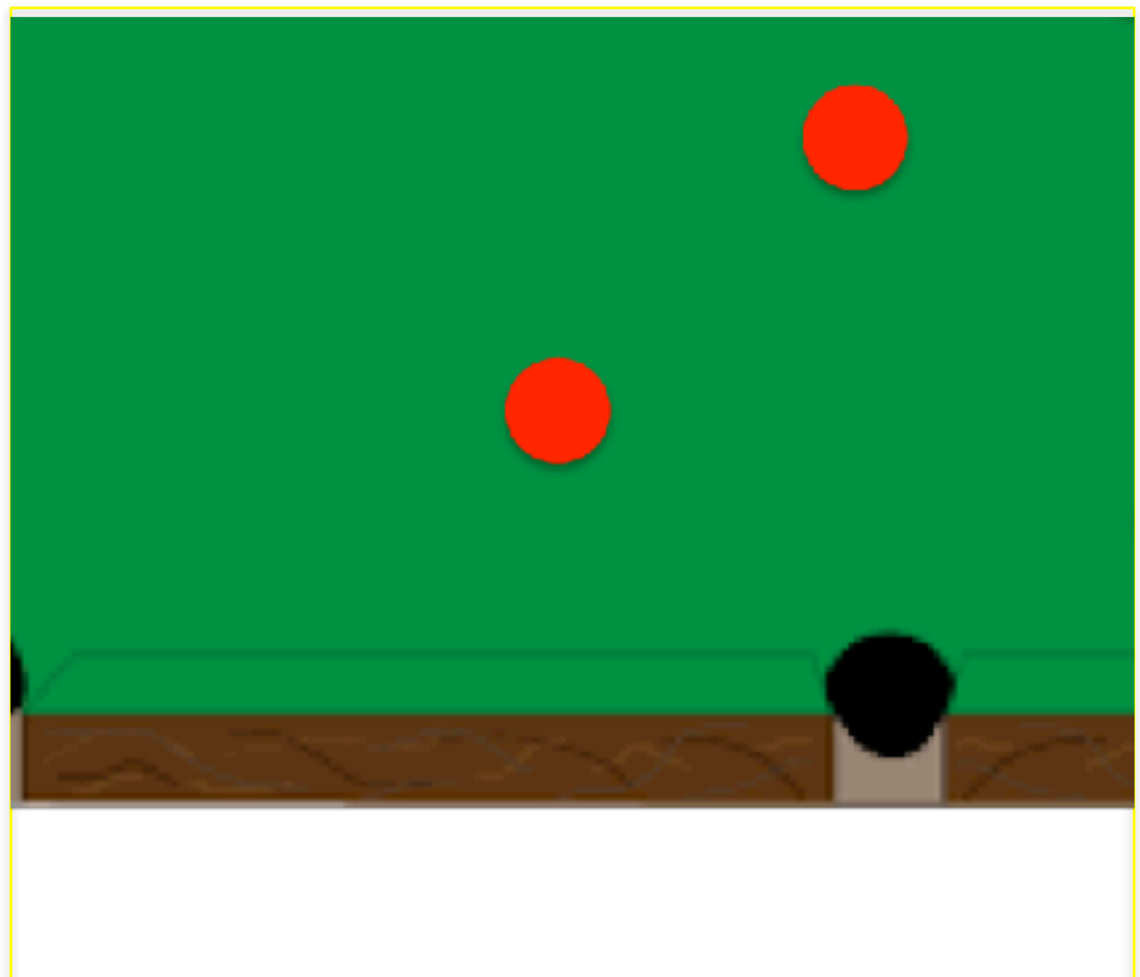


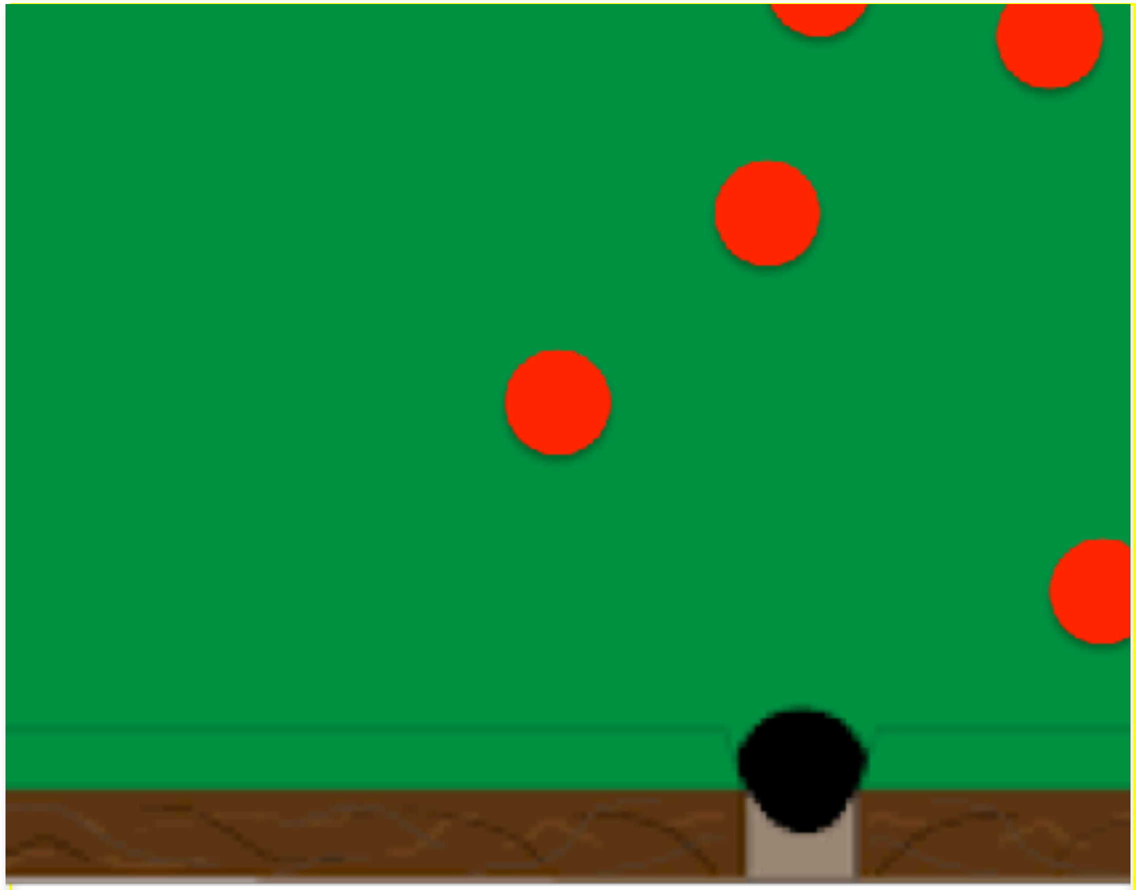
World-Centric Prediction

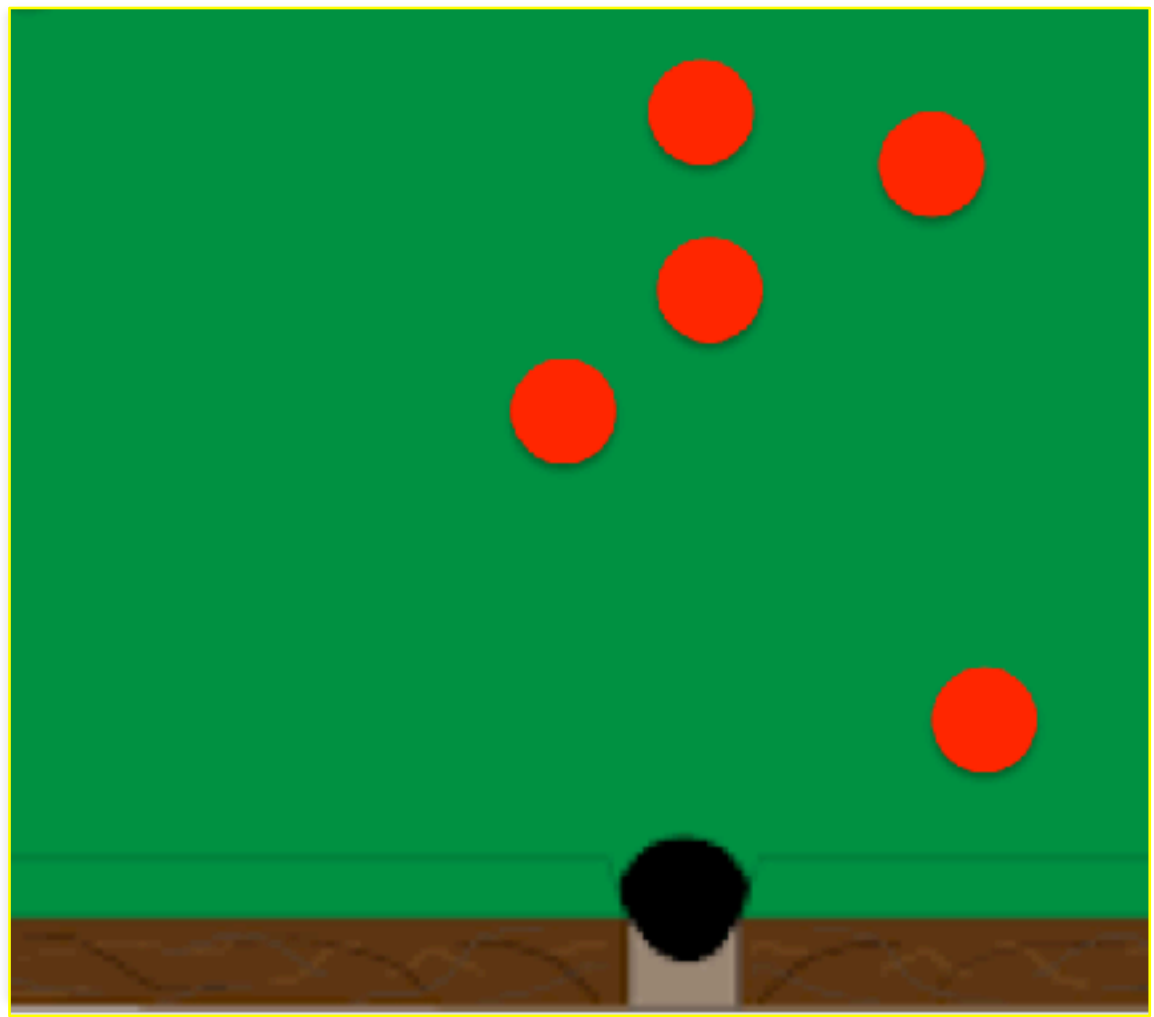


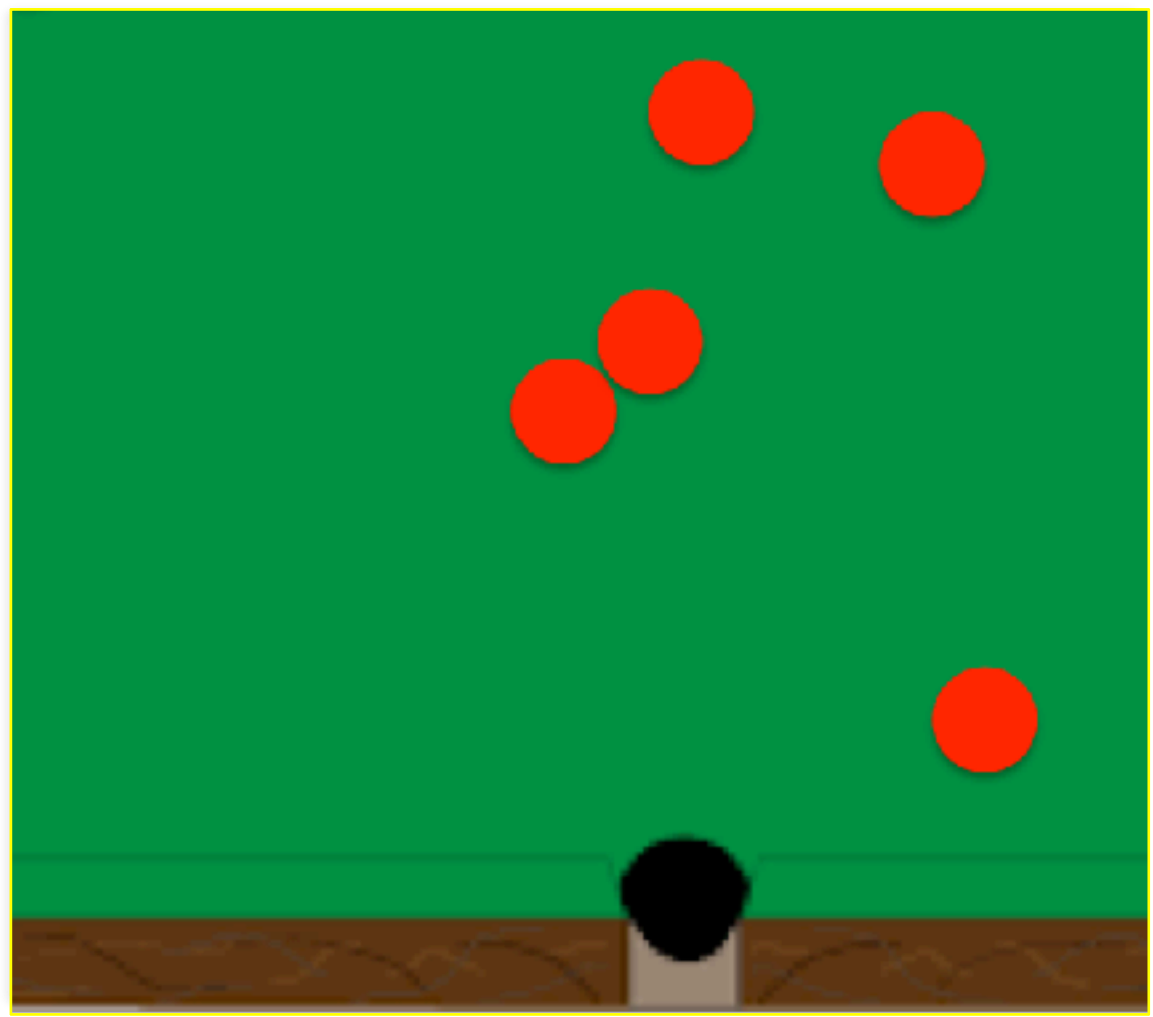
Object-Centric Prediction

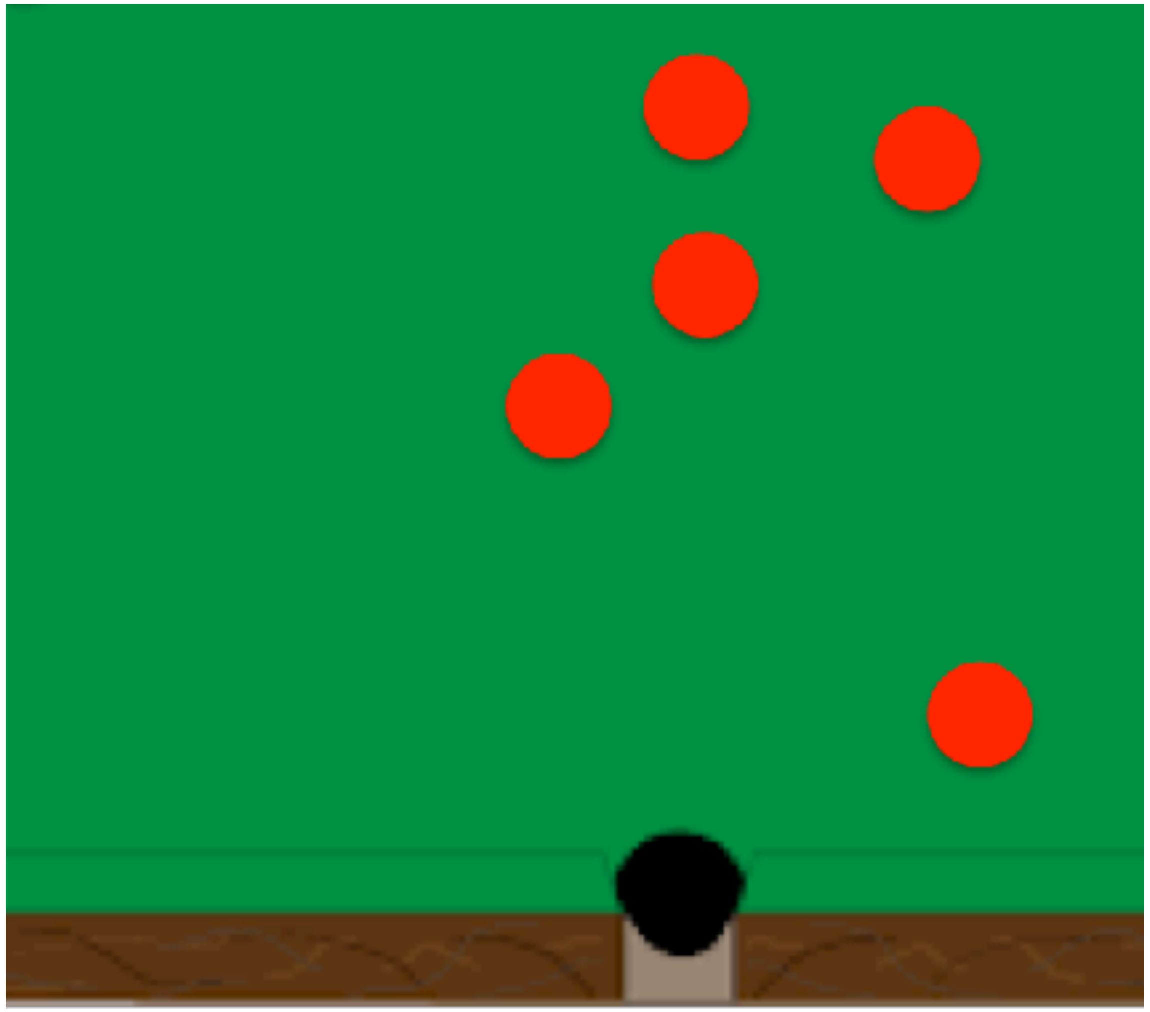
Q: will our model be able to generalize across different number of balls present?

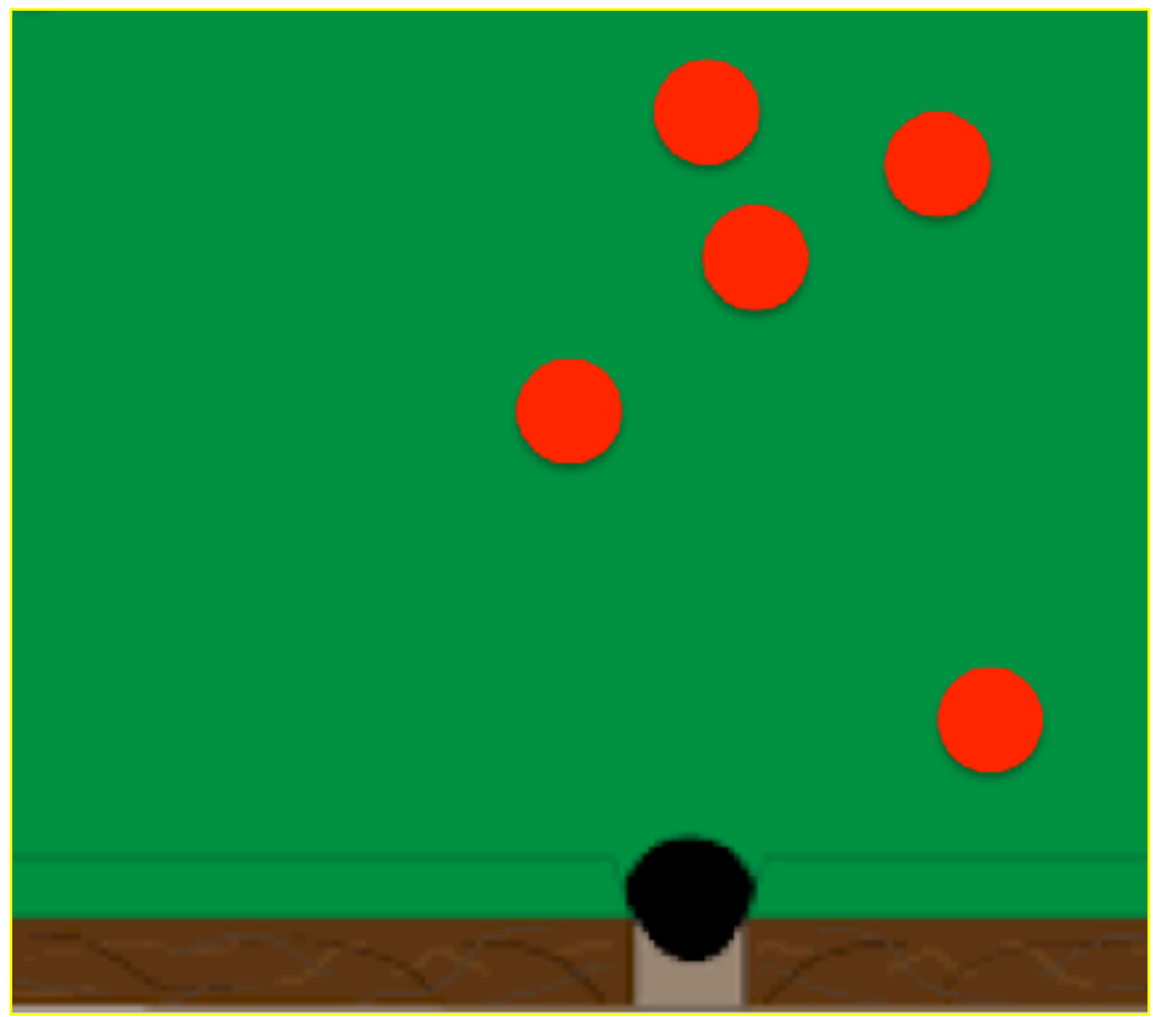




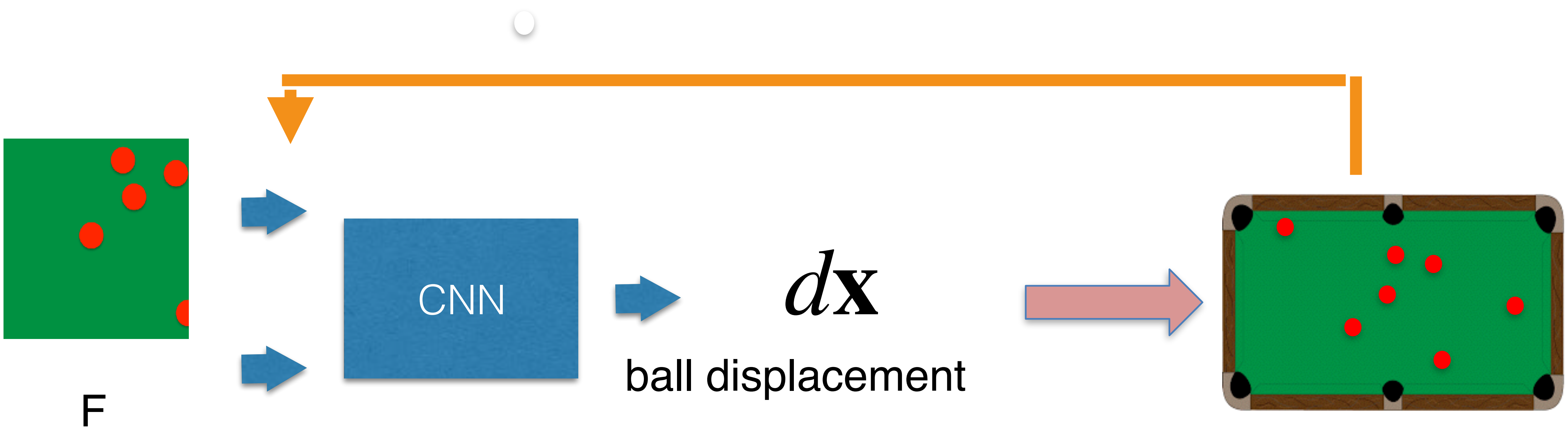




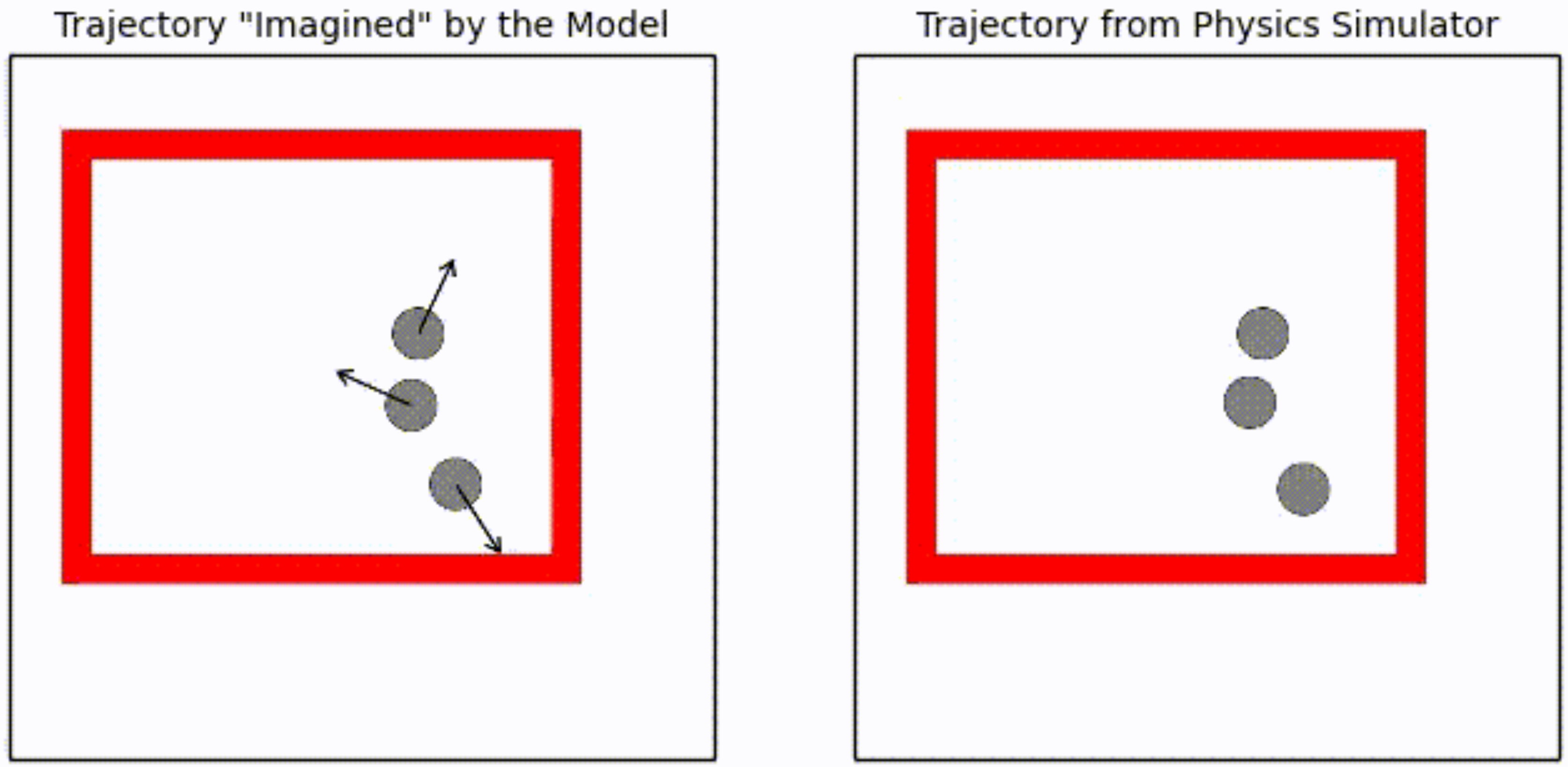




Object-centric Billiard Dynamics

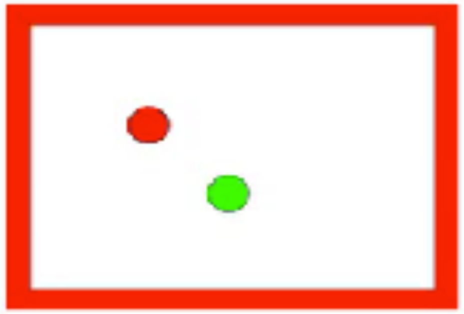


The object-centric CNN is shared across all objects in the scene.
We apply it one object at a time to predict the object's future displacement.
We then copy paste the ball at the predicted location, and feed back as input.


Trajectory "Imagined" by the Model
Trajectory from Physics Simulator

fi
id

Playing Billiards



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

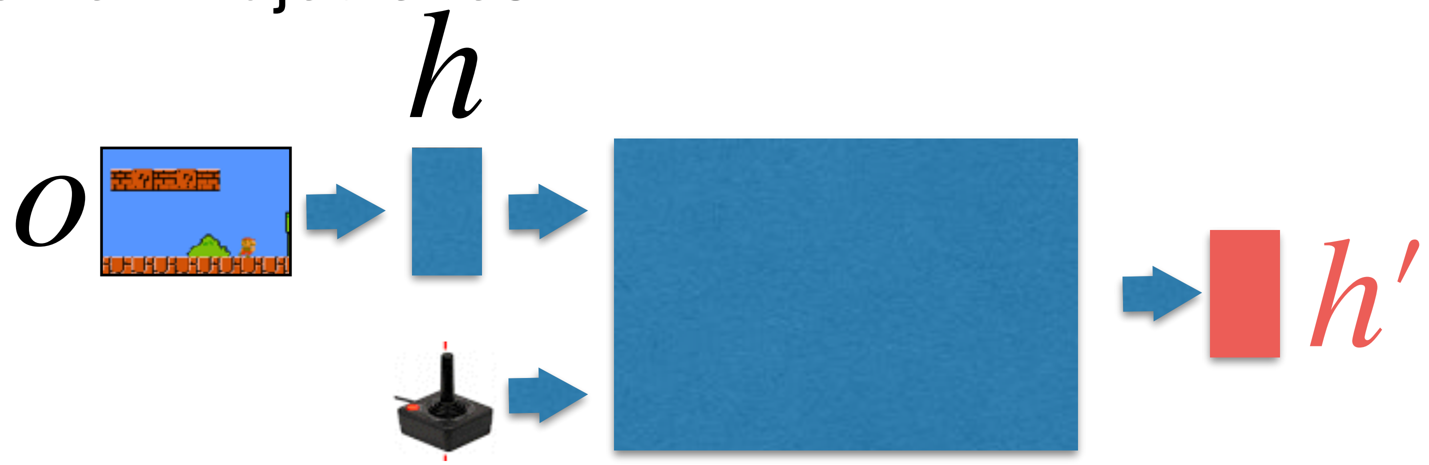
Learning Dynamics

Two good ideas so far:

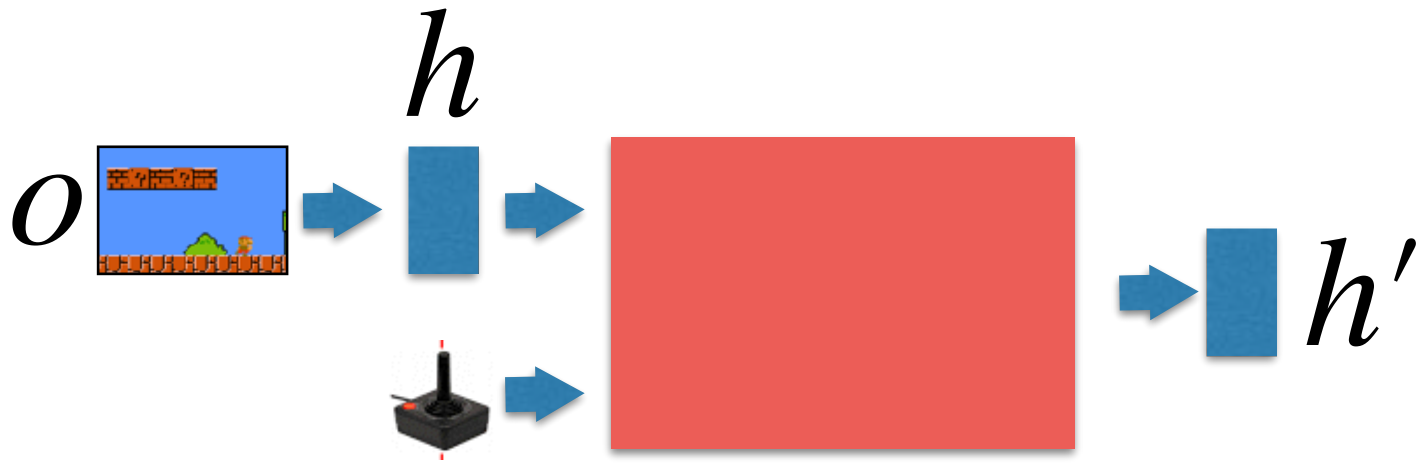
1) object graphs instead of images. Such encoding allows to generalize across different number of entities in the scene.
2) predict motion instead of appearance. Since appearance does not change, predicting motion suffices. Let's predict only the dynamic properties and keep the static one fixed.

Billiards

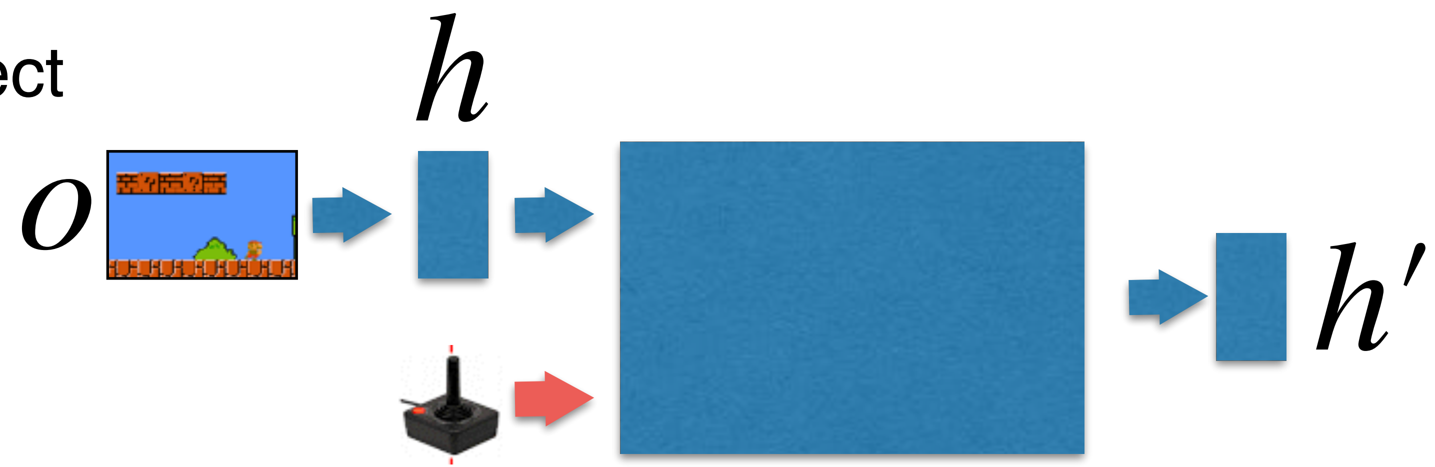
- We predicted object displacement trajectories



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

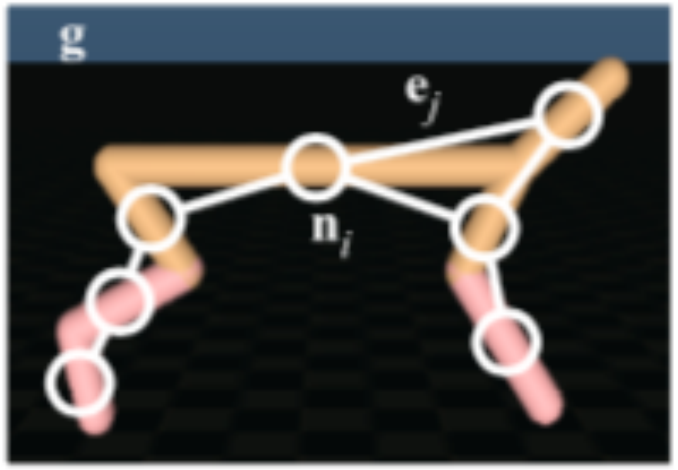


- A force applied to each object



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?

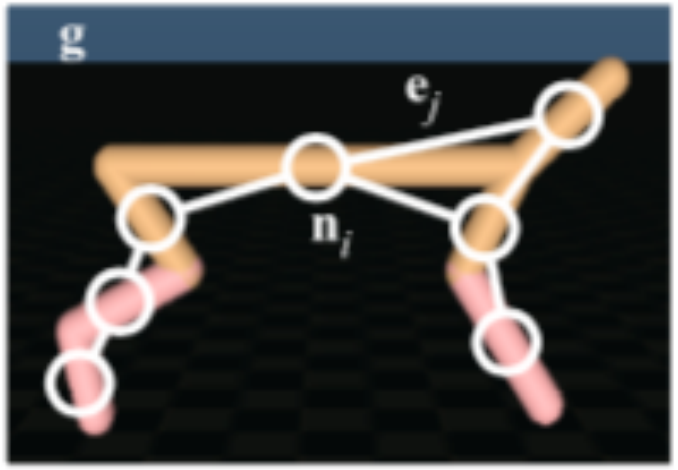


We will encode a robotic agent as a graph, where nodes are the different bodies of the agent and edges are the joints, links between the bodies

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

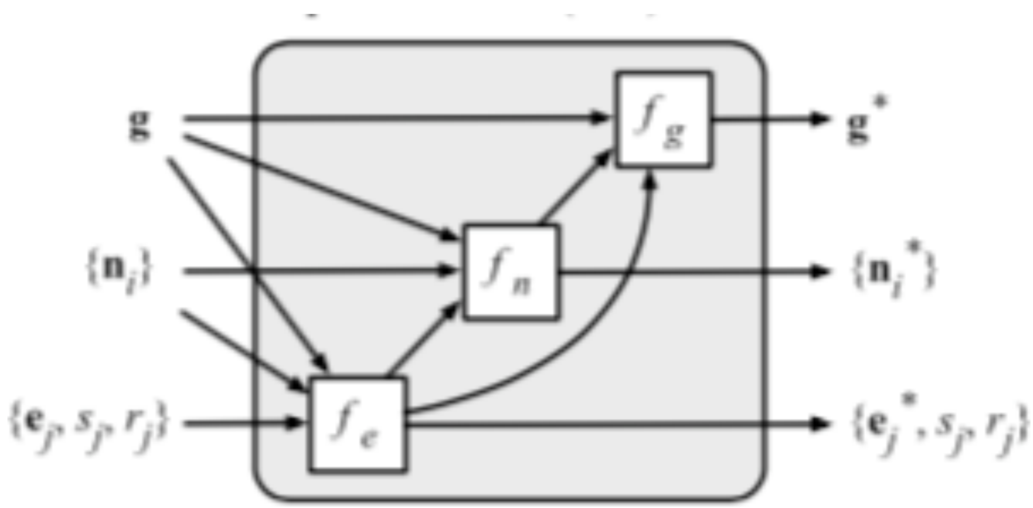
- Observable/dynamic: 3D position, 4D quaternion orientation, linear and angular velocities
- Unobservable/static: mass, inertia tensor
- Actions: forces applied on the joints

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

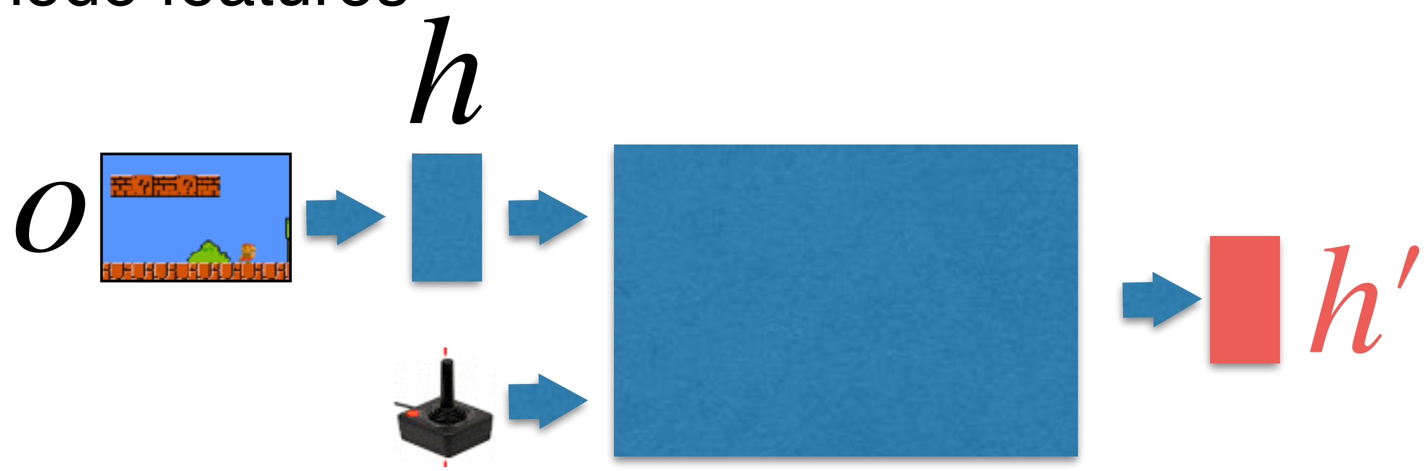
Input: Graph, $G = (\mathbf{g}, \{\mathbf{n}_i\}, \{\mathbf{e}_j, s_j, r_j\})$
for each edge $\{\mathbf{e}_j, s_j, r_j\}$ **do**
  Gather sender and receiver nodes $\mathbf{n}_{s_j}, \mathbf{n}_{r_j}$
  Compute output edges, $\mathbf{e}_j^* = f_e(\mathbf{g}, \mathbf{n}_{s_j}, \mathbf{n}_{r_j}, \mathbf{e}_j)$
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.

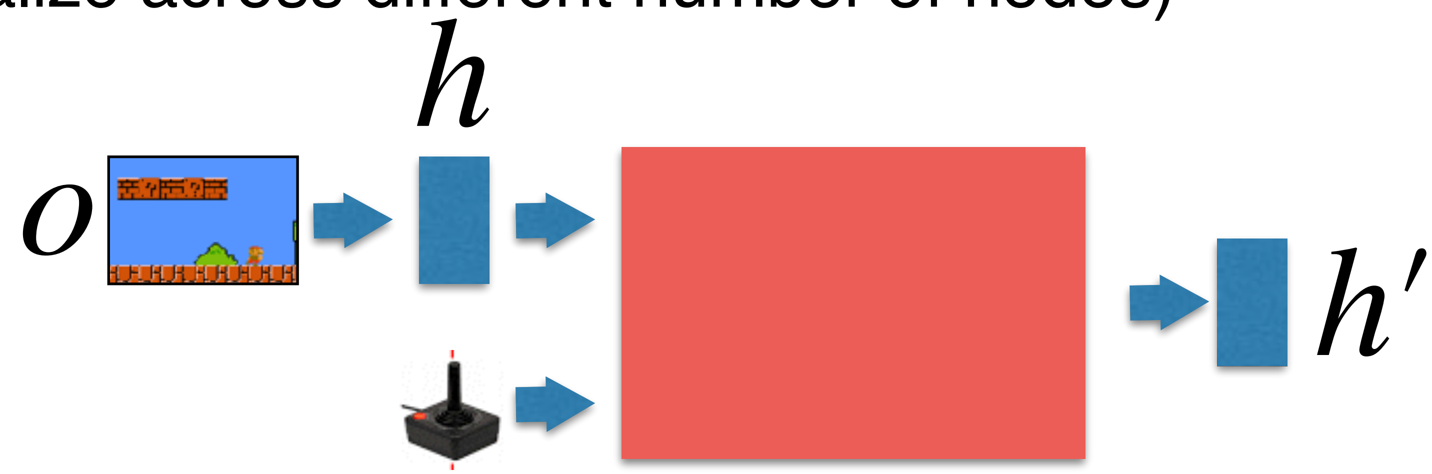
Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

Robots as graphs

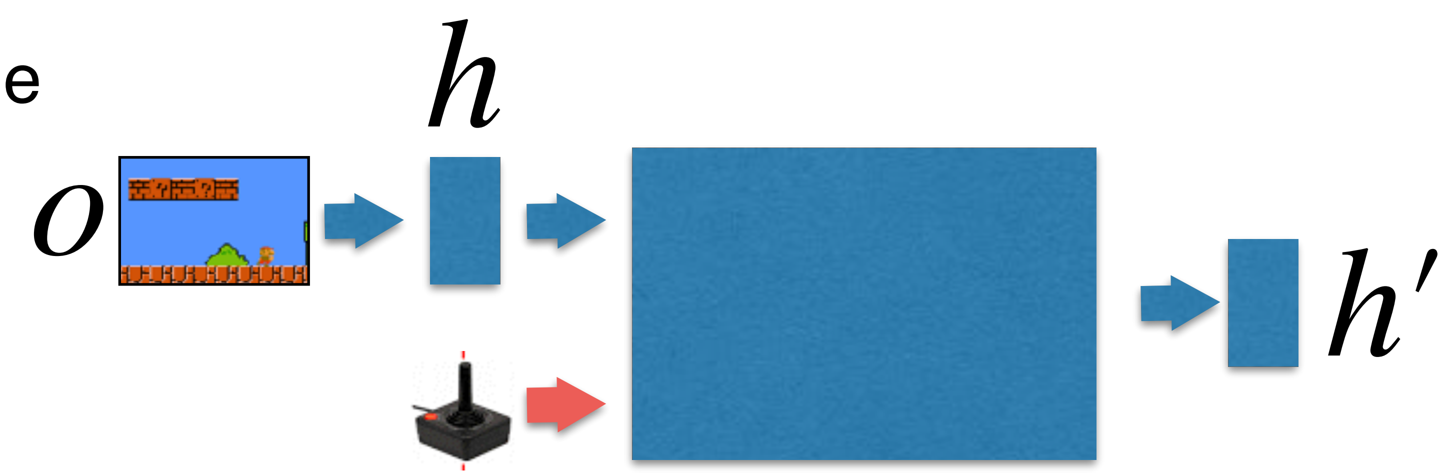
- We predicted dynamic only node features



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



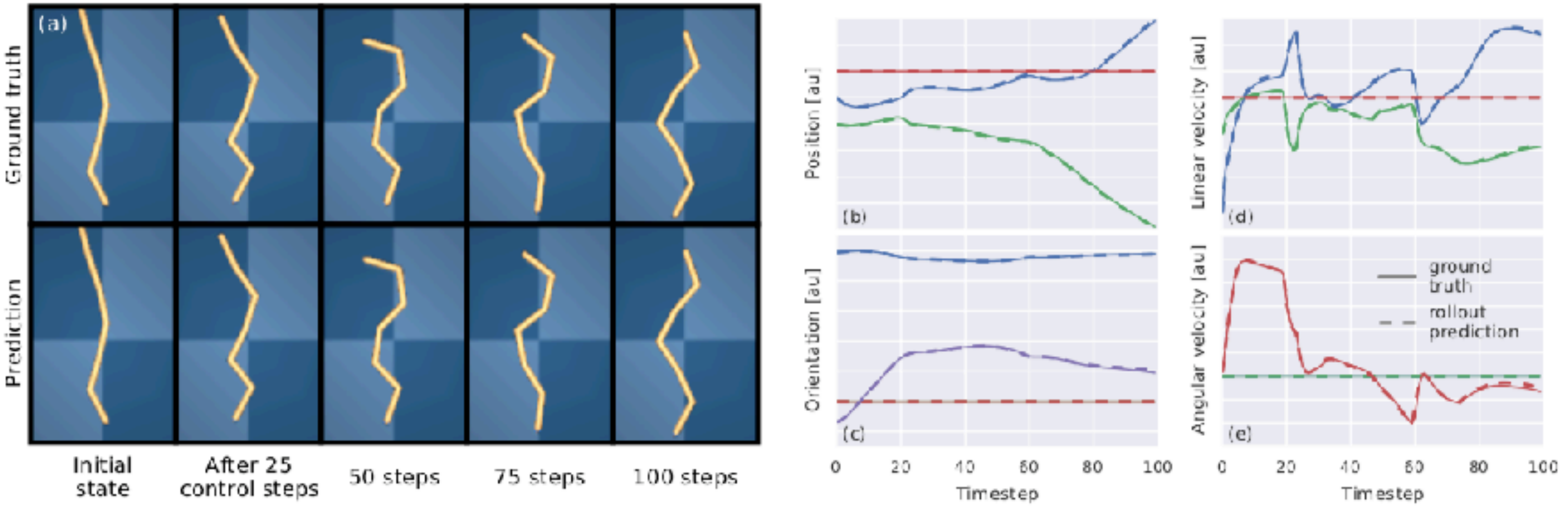
- Forces applied to each node



Graph Forward Dynamics

Node features

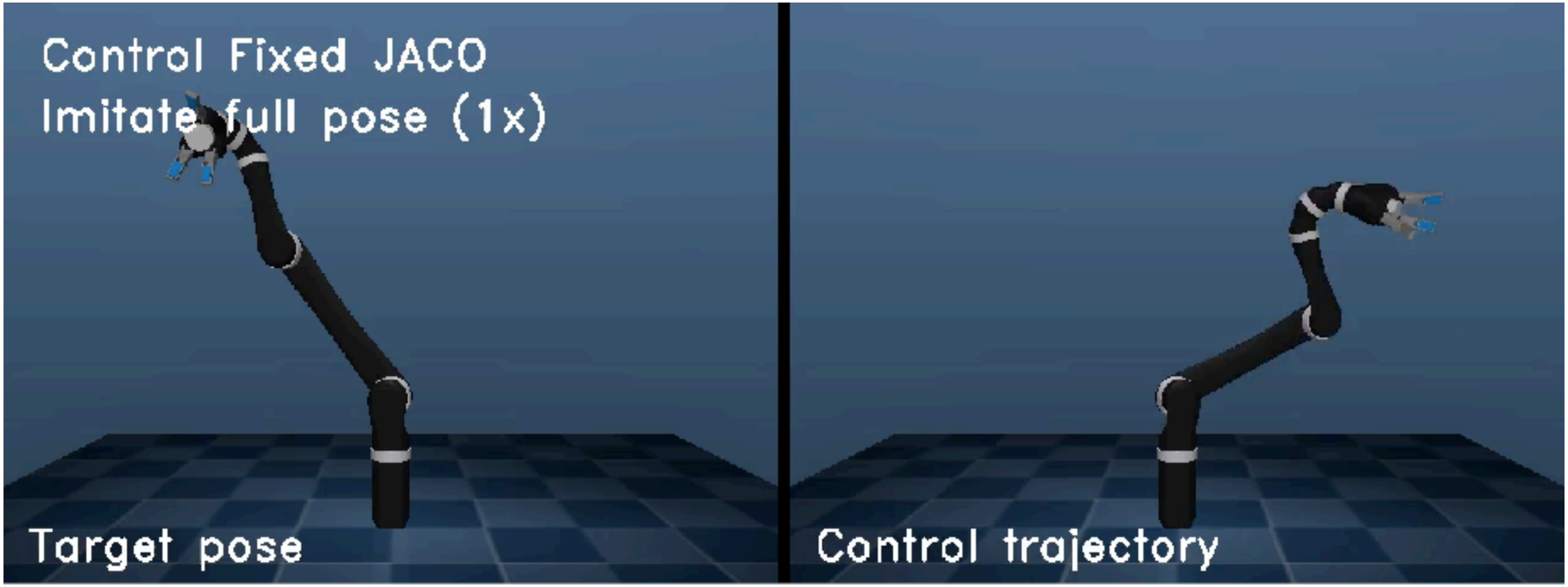
- Observable/dynamic: 3D position, 4D quaternion orientation, linear and angular velocities
- Unobservable/static: mass, inertia tensor
- Actions: forces applied on the joints



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



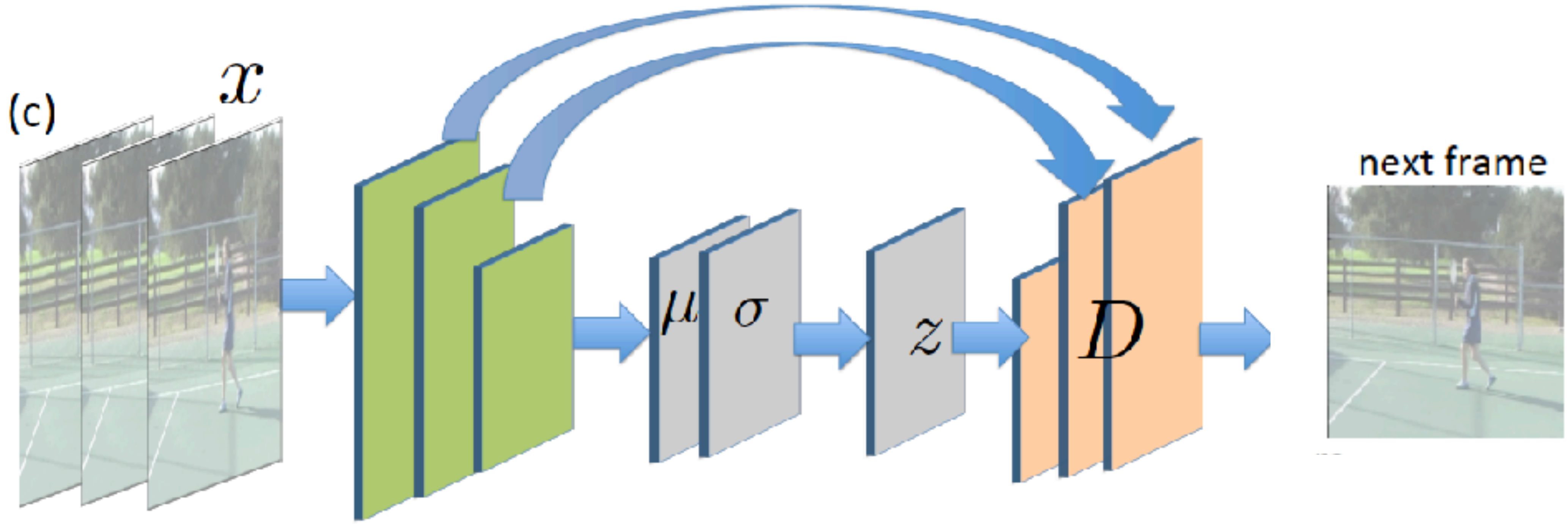
Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

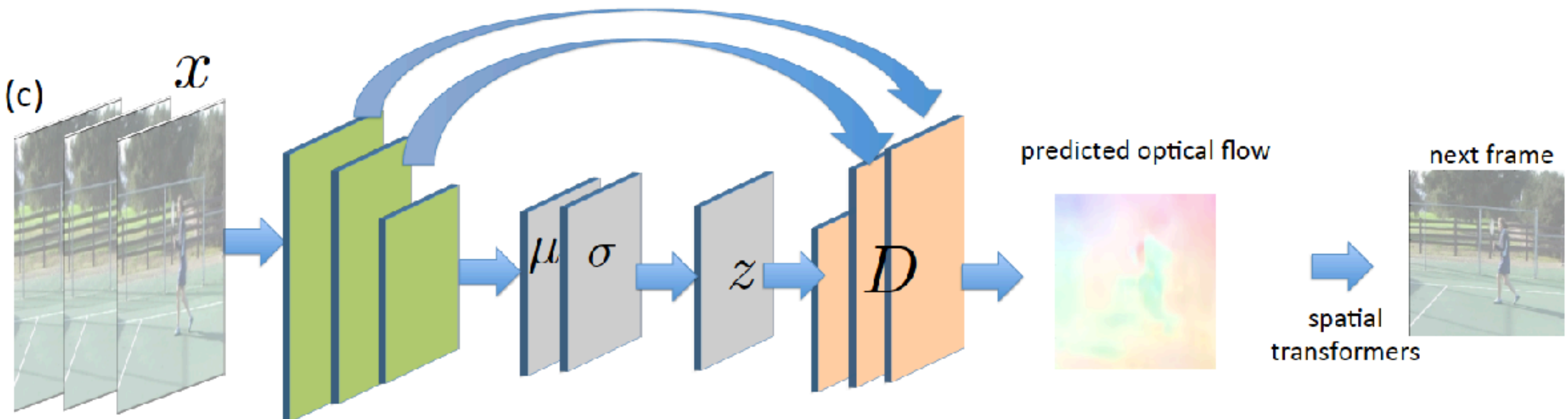
Learning Dynamics

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

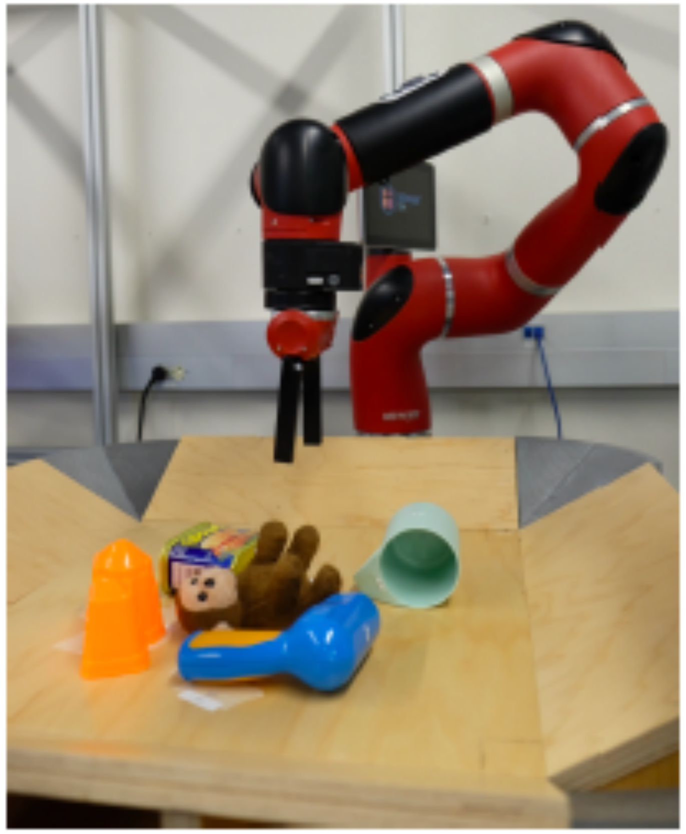


Figure 1: The robot learns to move new objects from self-supervised experience.

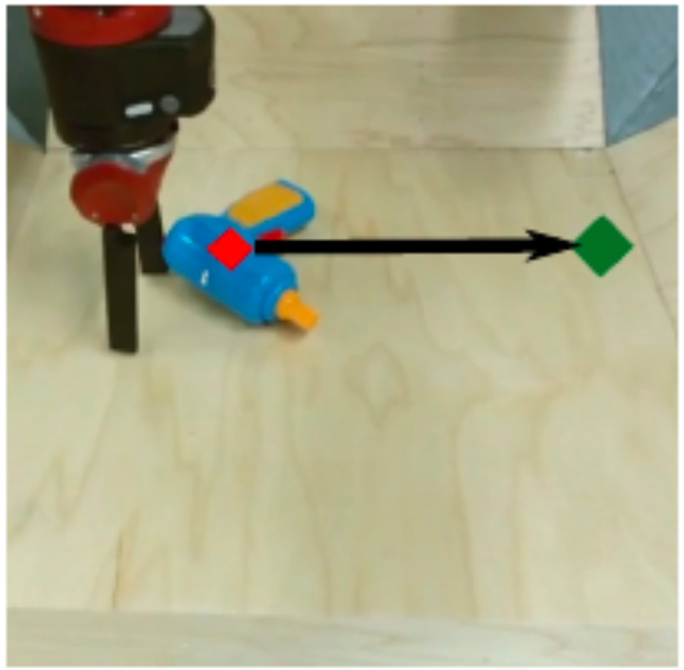
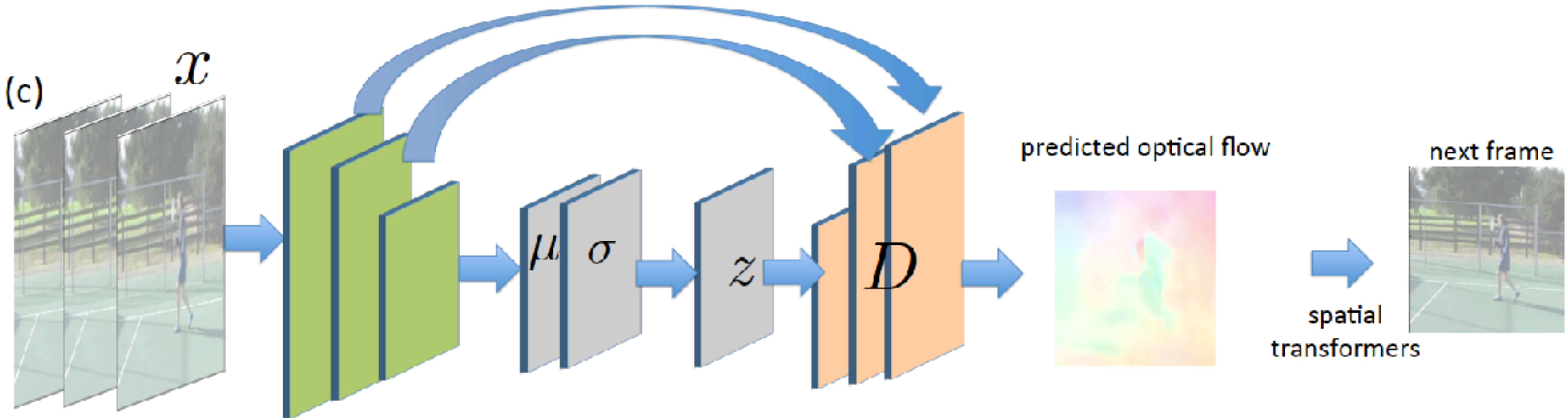


Figure 7: Pushing task. The designated pixel (red diamond) needs to be pushed to the green circle.

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

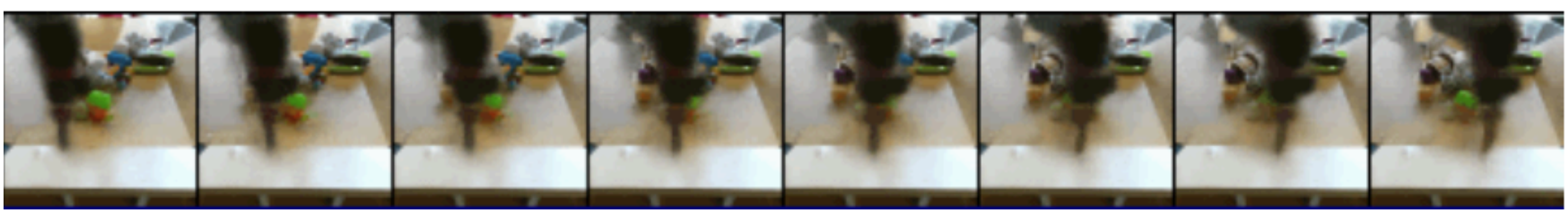
We will learn a model of pixel motion displacements

Visual dynamics using motion transformation

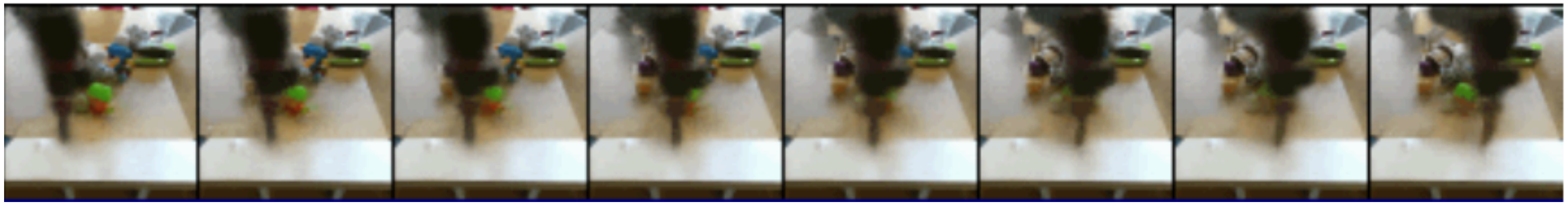


Differentiable warping

Can I use this model?



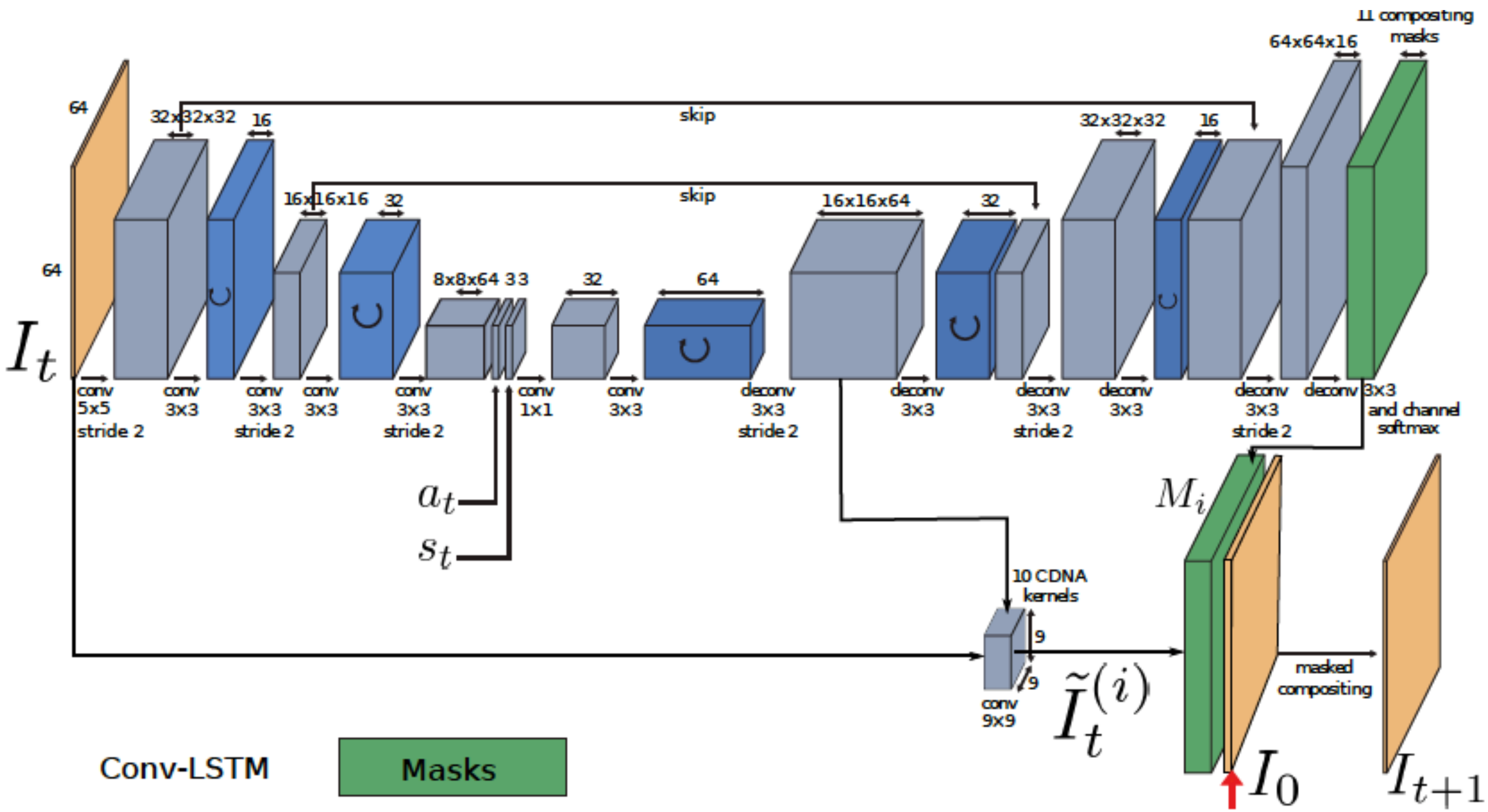
Visual dynamics using motion transformation



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

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

Visual dynamics using motion transformation



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

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

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

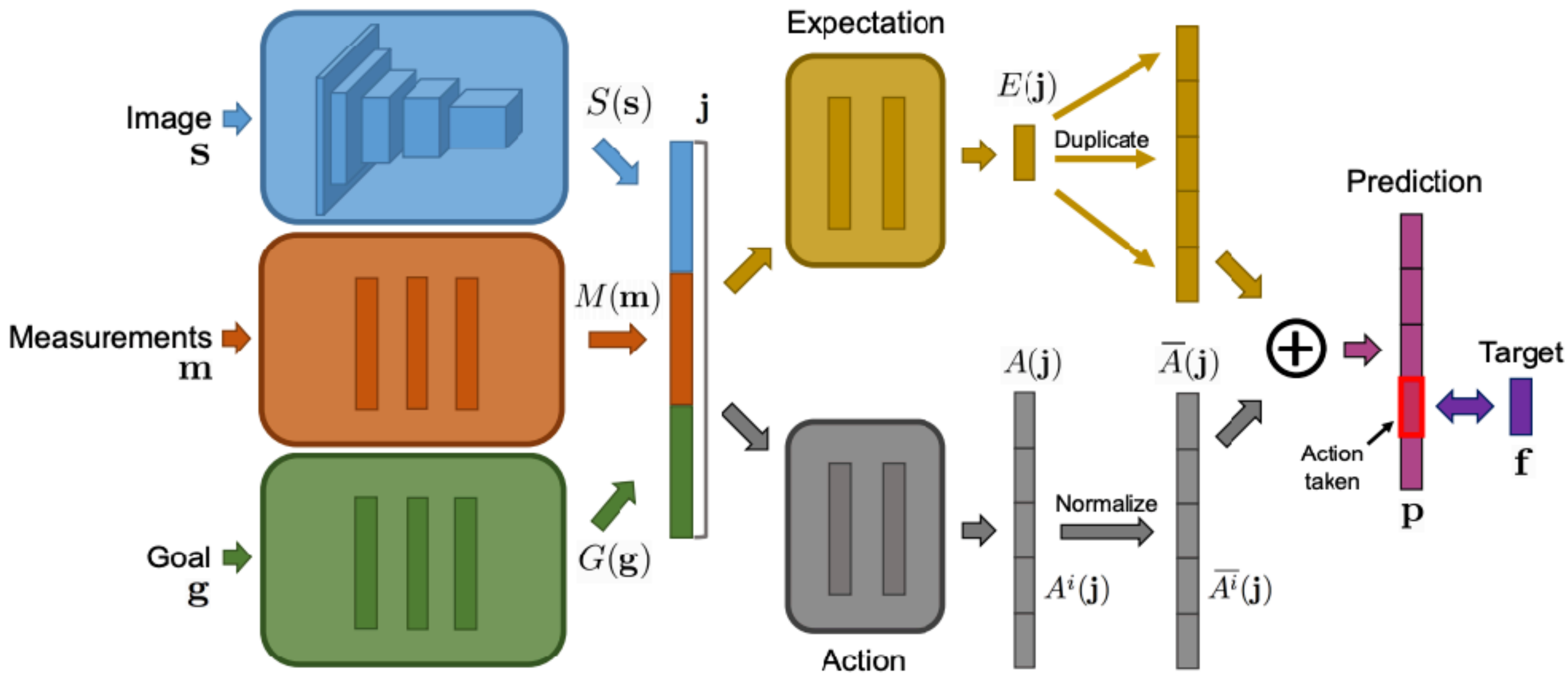
Train a deep predictor. No unrolling! One shot prediction of future values:

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

No policy, direct action selection:

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

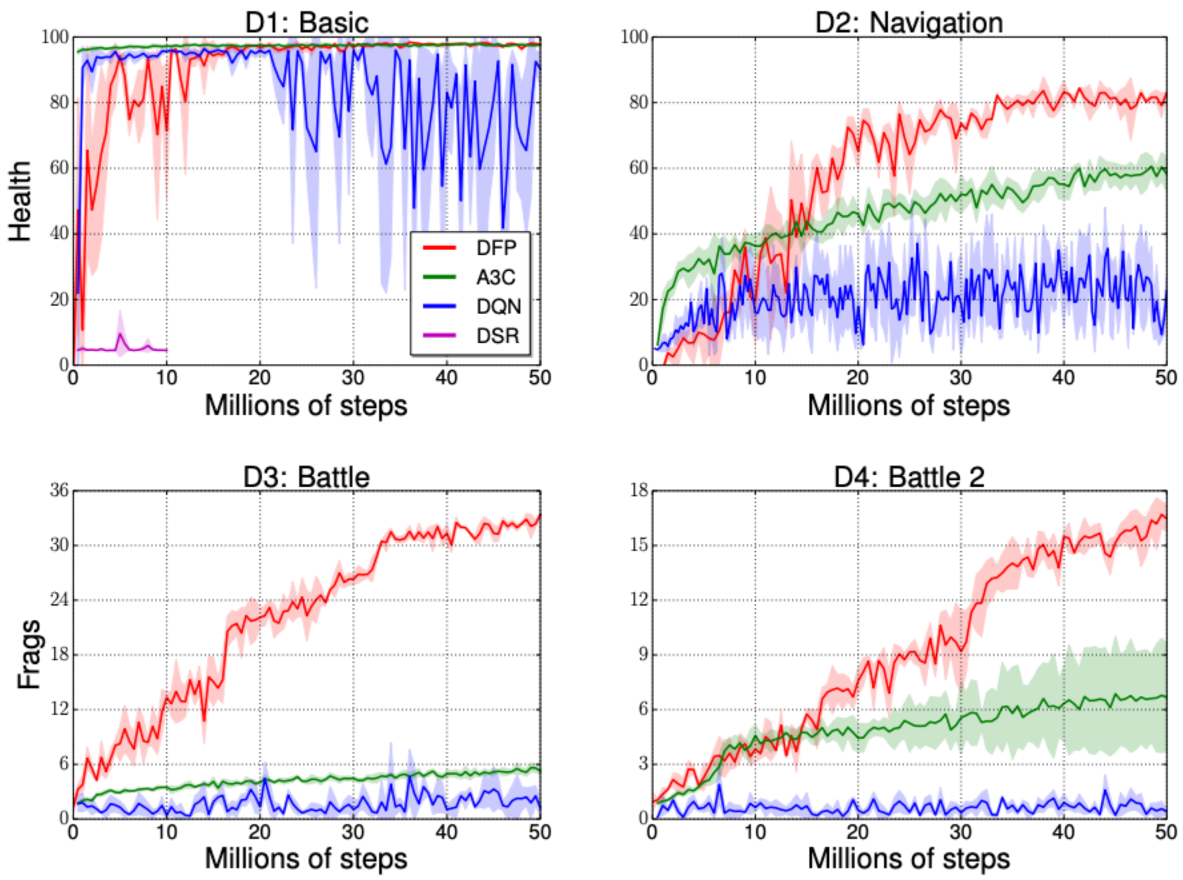
Learning dynamics of goal-related measurements



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.

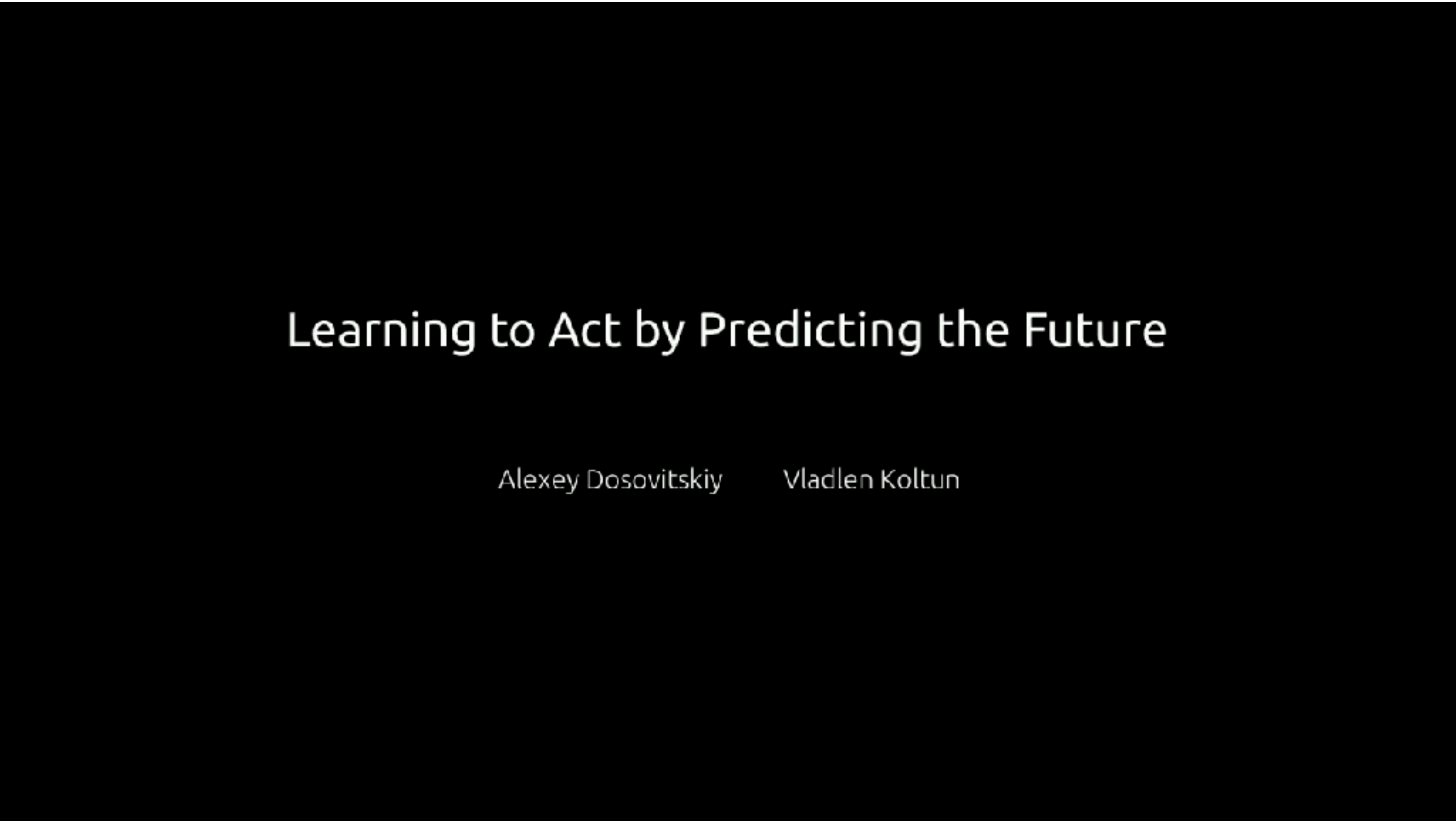
Learning dynamics of goal-related measurements



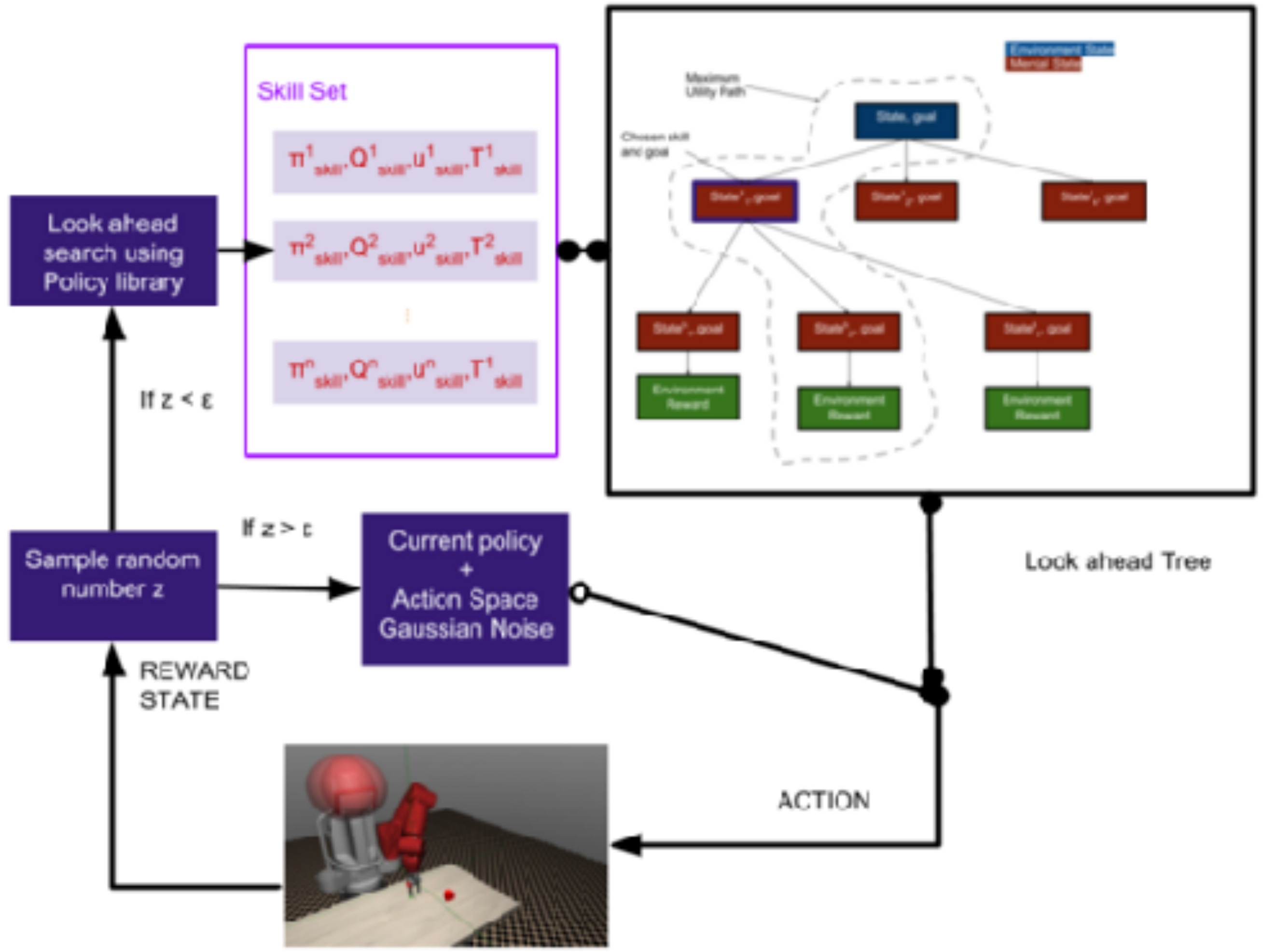
	D1 (health)	D2 (health)	D3 (frags)	D4 (frags)	steps/day
DQN	89.1 ± 6.4	25.4 ± 7.8	1.2 ± 0.8	0.4 ± 0.2	7M
A3C	$\mathbf{97.5 \pm 0.1}$	59.3 ± 2.0	5.6 ± 0.2	6.7 ± 2.9	80M
DSR	4.6 ± 0.1	–	–	–	1M
DFP	$\mathbf{97.7 \pm 0.4}$	$\mathbf{84.1 \pm 0.6}$	$\mathbf{33.5 \pm 0.4}$	$\mathbf{16.5 \pm 1.1}$	70M

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



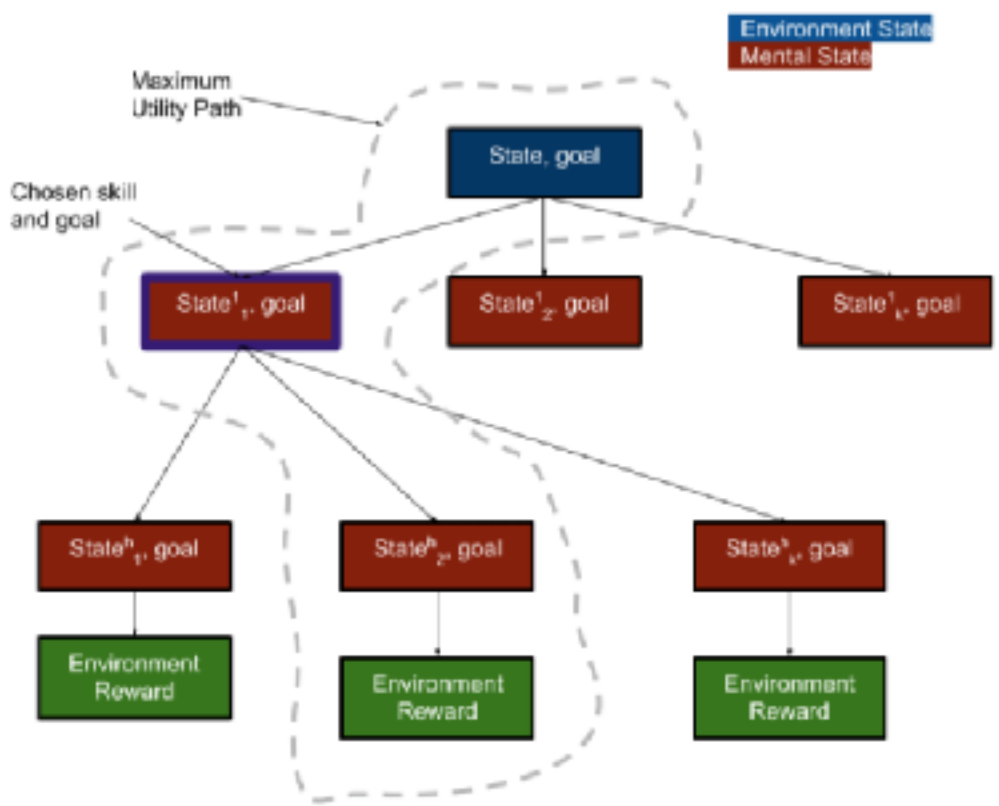
Exploration by Planning



1. Learn a set of skills, namely, grasp, reach and transfer, using HER
2. For each skill, we have a multistep inverse model $\pi(g, s)$
3. For each skill, we further train a forward model T(s,g)->s’
4. In each exploration step, we look-ahead by chaining multistep skills, as opposed to single step.

Skill-guided Look-ahead Exploration for Reinforcement Learning of Manipulation Policies, submitted

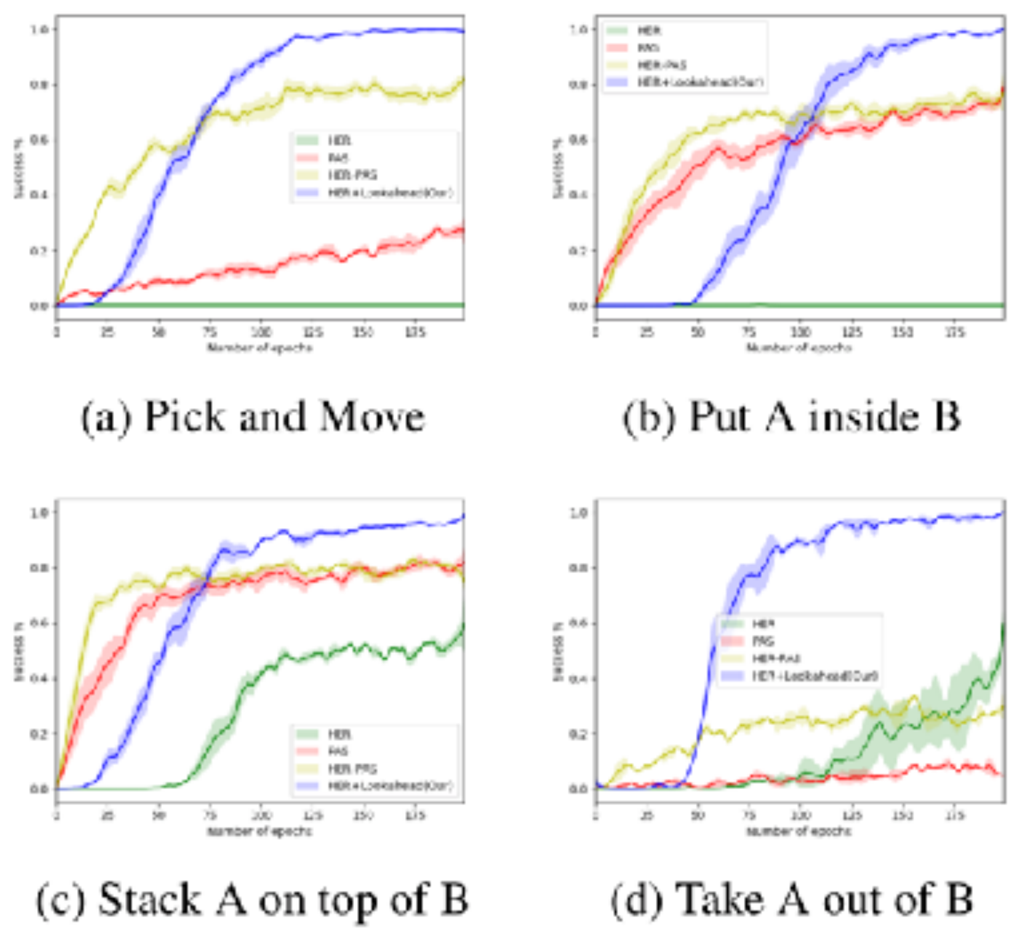
Exploration by Planning



1. Learn a set of skills, namely, grasp, reach and transfer, using HER
2. For each skill, we have a multistep inverse model $\pi(g, s)$
3. For each skill, we further train a forward model T(s,g)->s'
4. In each exploration step, we look-ahead by chaining multistep skills, as opposed to single step.

Skill-guided Look-ahead Exploration for Reinforcement Learning of Manipulation Policies, submitted

Exploration by Planning



(a) Pick and Move (b) Put A inside B (c) Stack A on top of B (d) Take A out of B

1. Learn a set of skills, namely, grasp, reach and transfer, using HER
2. For each skill, we have a multistep inverse model $\pi(g, s)$
3. For each skill, we further train a forward model T(s,g)->s'
4. In each exploration step, we look-ahead by chaining multistep skills, as opposed to single step.

Skill-guided Look-ahead Exploration for Reinforcement Learning of Manipulation Policies, submitted