Carnegie Mellon
School of Computer Science

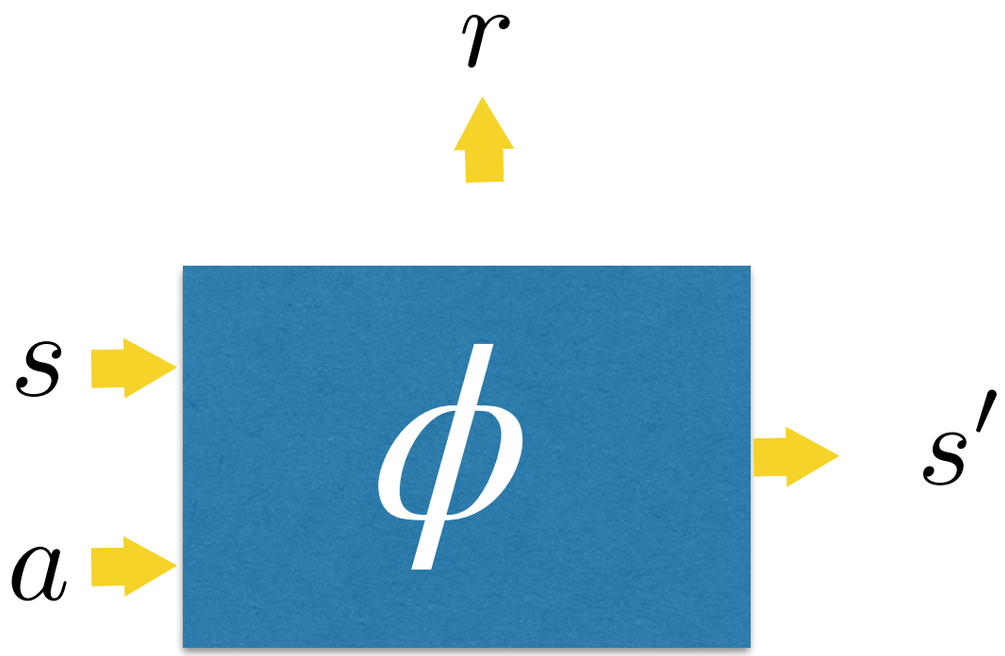
Deep Reinforcement Learning and Control

Model Based Reinforcement Learning II

Katerina Fragkiadaki

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

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

The model can be unrolled in time by feeding the prediction of the model back as input

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

Why model learning

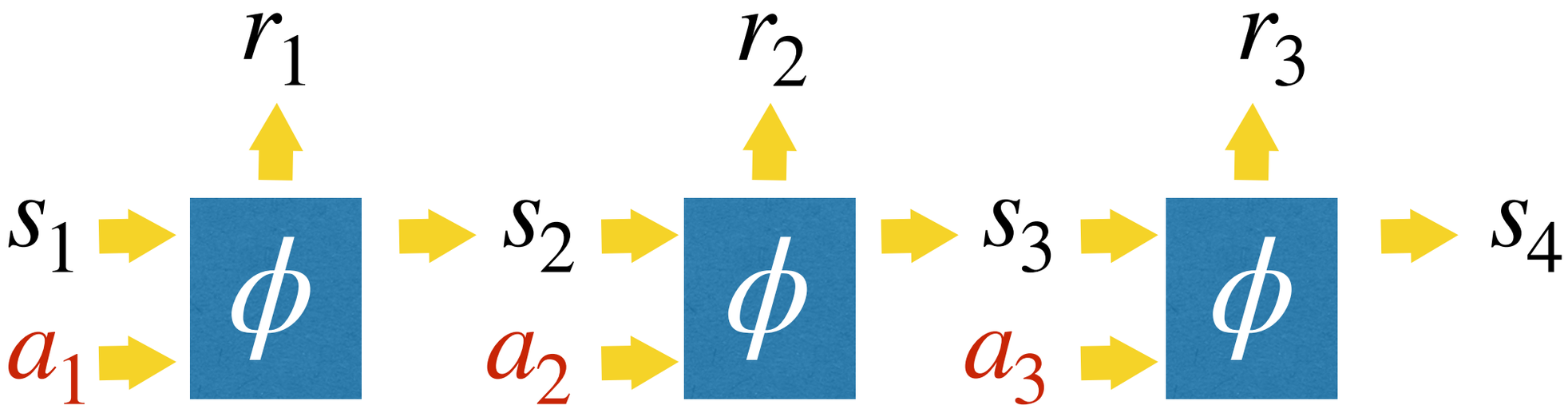
- Model-based control: given an initial state s_0 estimate **action sequence** to reach a desired goal or maximize reward by unrolling the model forward in time
- Model-based RL: train **policies** using:
    1. a model-free RL method using simulated experience (experience sampled from the model)
  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

Model-based control

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

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

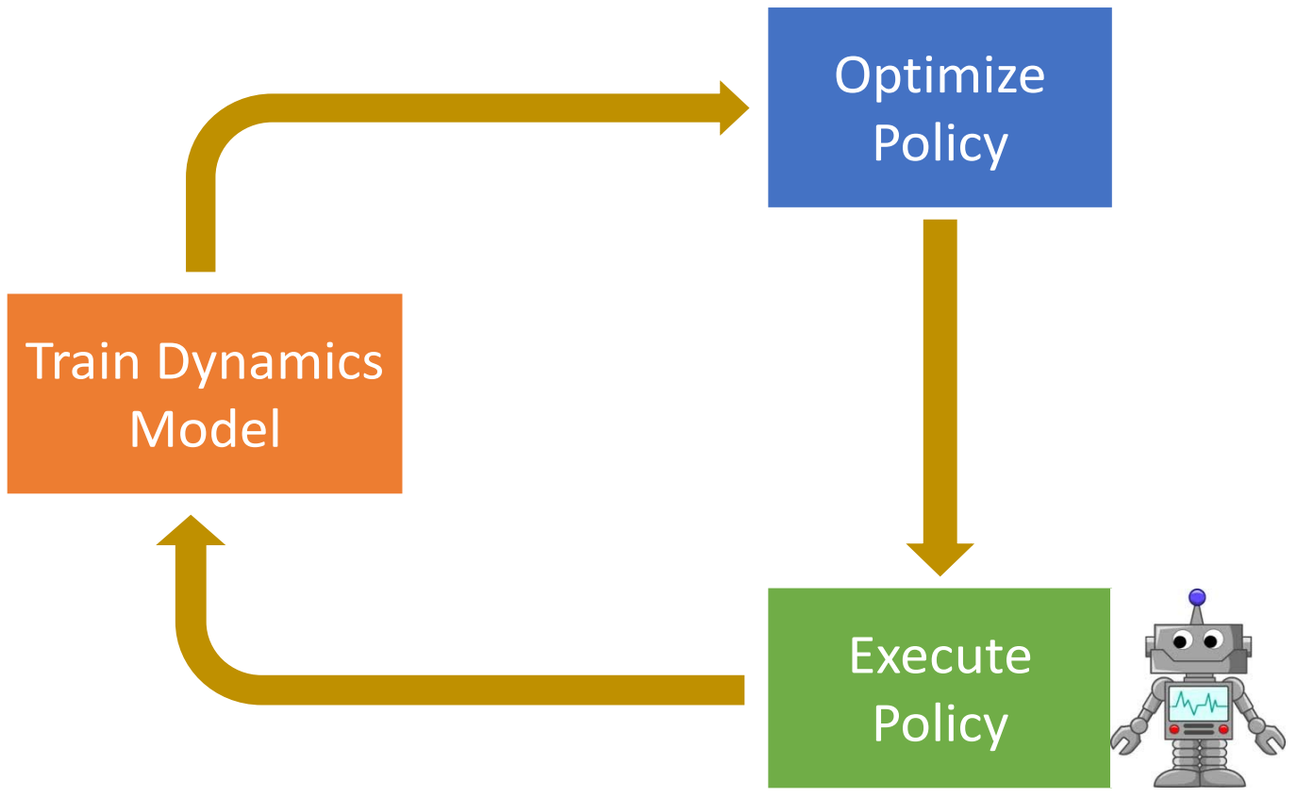
If the dynamics are non-linear and the loss is not a quadratic, this optimization is difficult. We can use SGD or evolutionary methods.



Alternating between model and policy learning

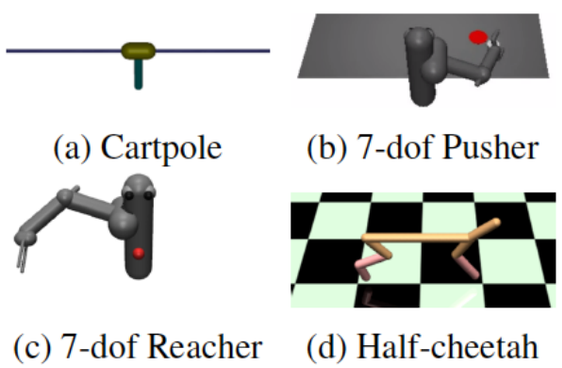
Initialize policy $\pi(s;\theta)$ and D={}.

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.

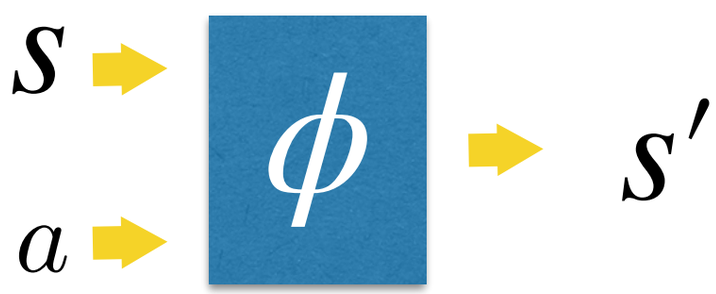


Model Learning

- Where a low dimensional state is observed and given:

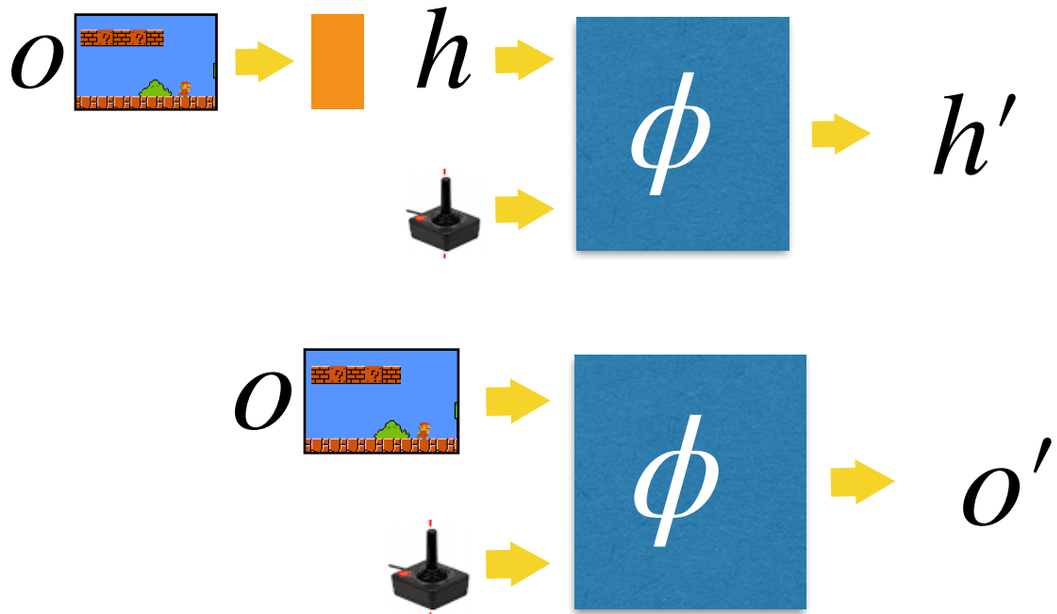


state can be 3D locations and 3D velocities of agent joints, actions can be torques



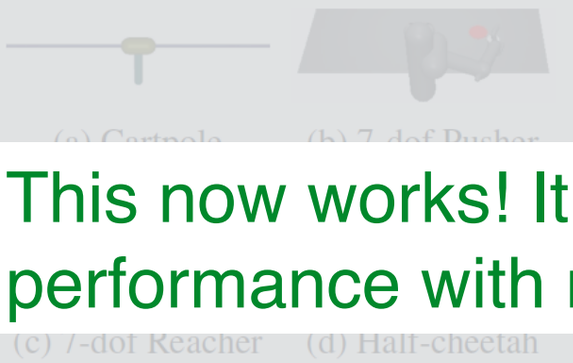
- Where we only have access to (high dim) sensory input, e.g., images:

e.g., Atari game playing



Model Learning

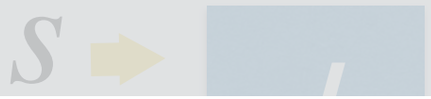
- Where a low dimensional state is observed and given:



(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

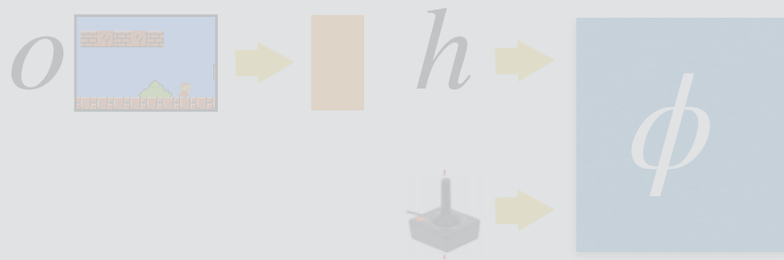


This now works! It outperforms model-free RL methods: reaches same final performance with much fewer samples! :-)

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

e.g., Atari game playing

Still an open problem :-(



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

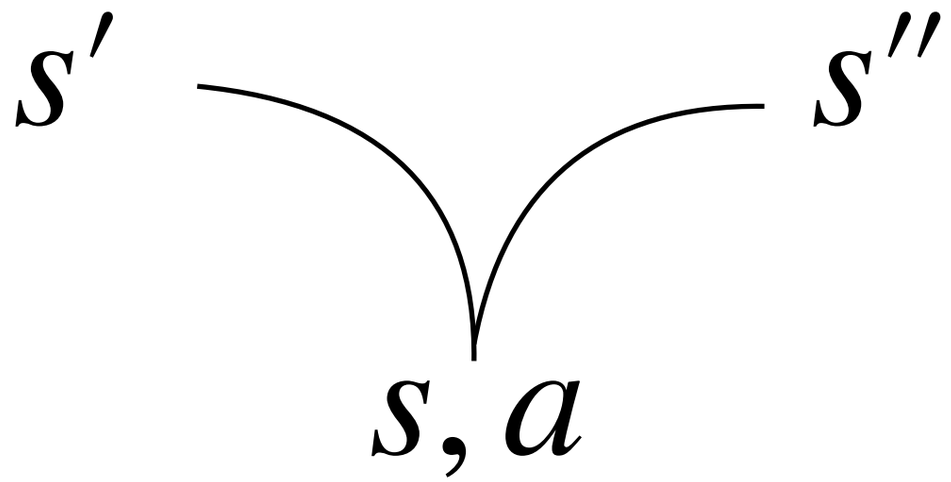
It's all about representing uncertainty. Two types of uncertainty:

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

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

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

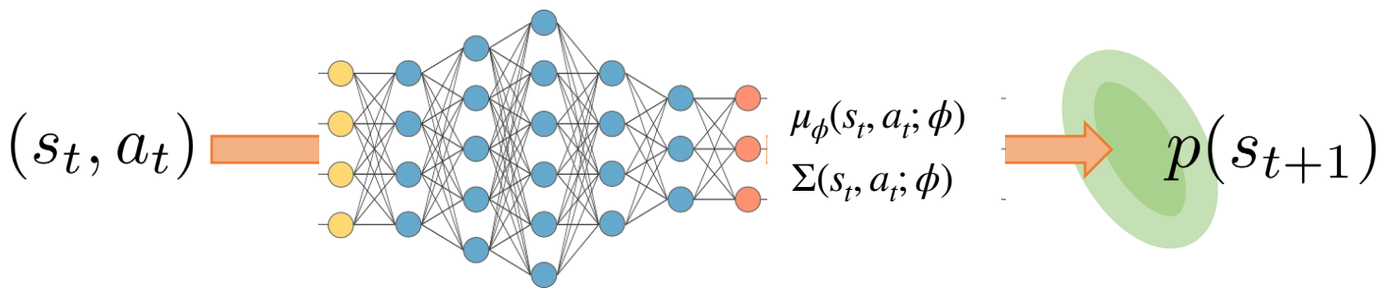
Training a probabilistic NN! Given a (s,a) as input, the NN outputs a mean vector and a set of variances, one for each dimension of the state vector. We train by maximizing log likelihood of our training set.

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

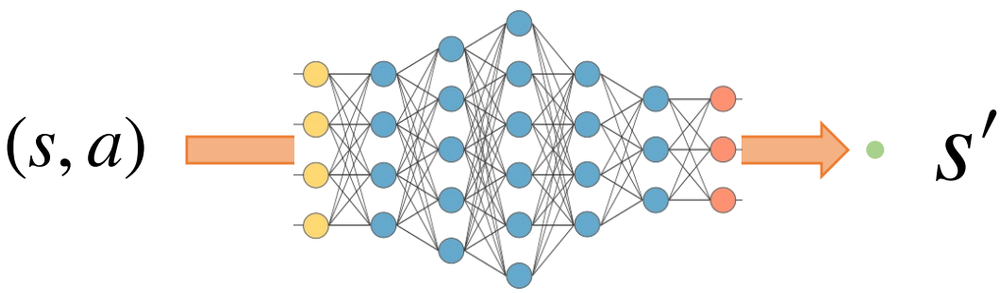
$$\mathscr{L}_\phi = -\frac{1}{N}\sum_{i=1}^{N} \log p(s_i'|s_i,a_i;\phi)$$

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

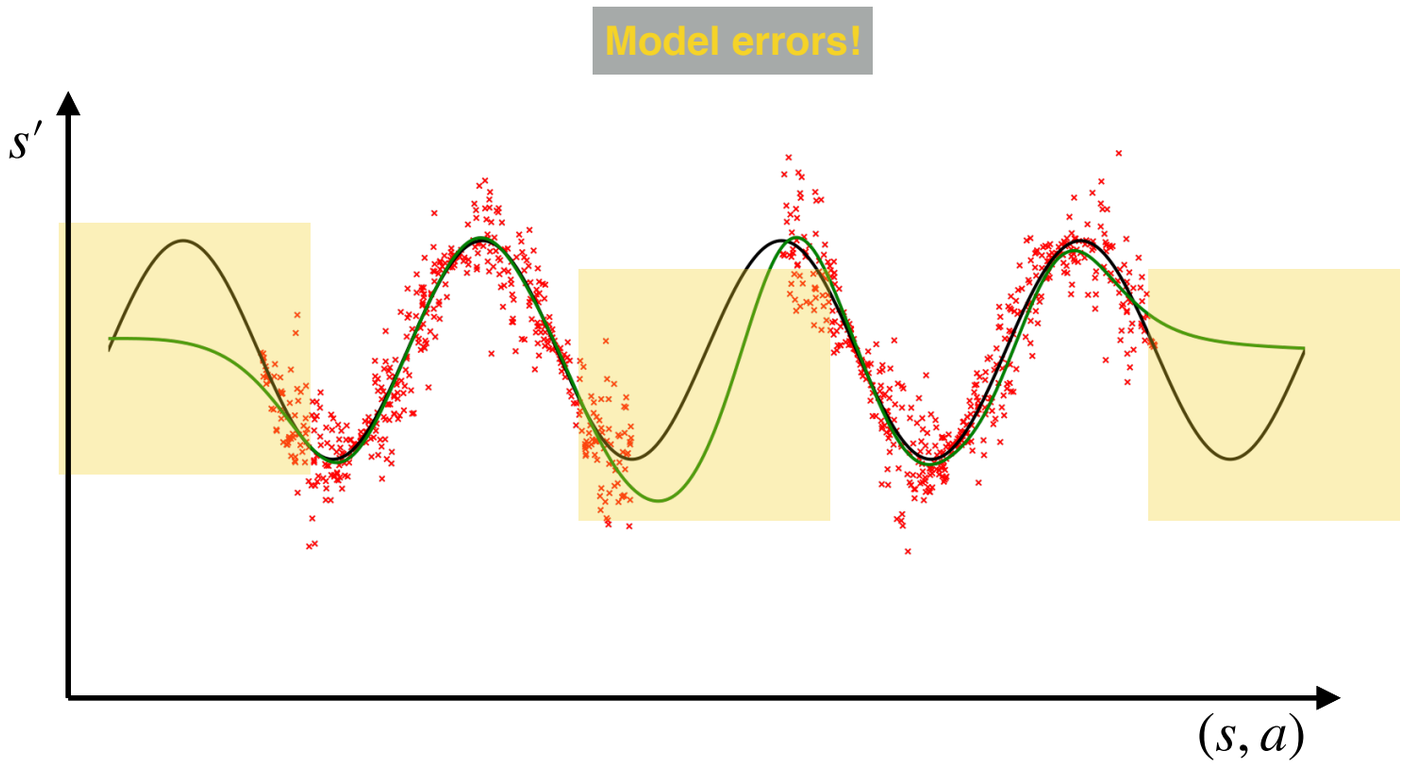
$$+\frac{1}{2}\log(\det \mathbf{\Sigma}(s_i,a_i;\phi)) + \text{ const.}$$



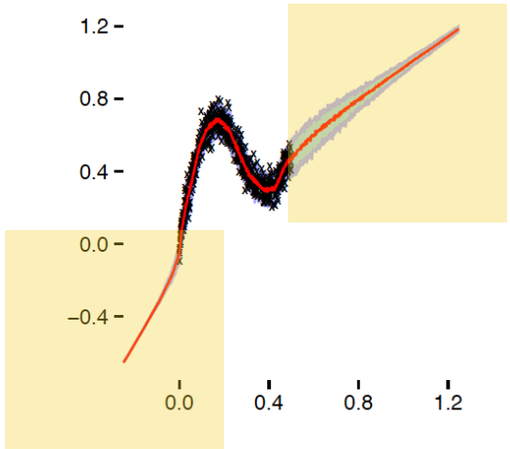
Epistemic uncertainty in Model Learning



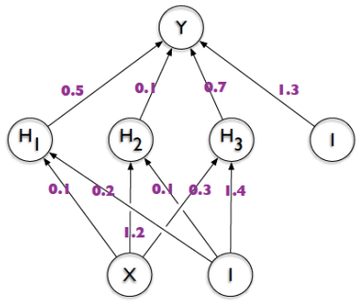
Fitting a deterministic neural network



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

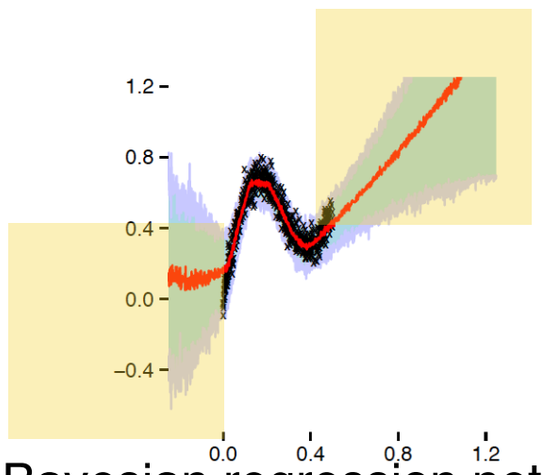


regression network



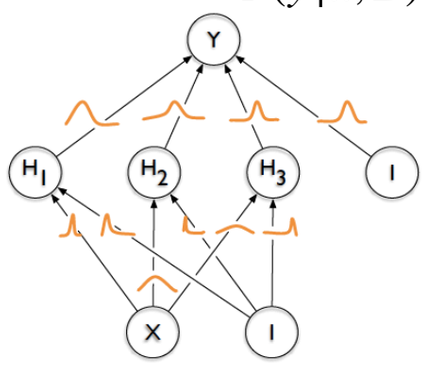
$$\phi^{MAP} = \arg\max_{\phi} \log P(\phi \mid D) = \arg\max_{\phi} \left(P(D \mid \phi) + \log P(\phi) \right)$$

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



Bayesian regression network

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

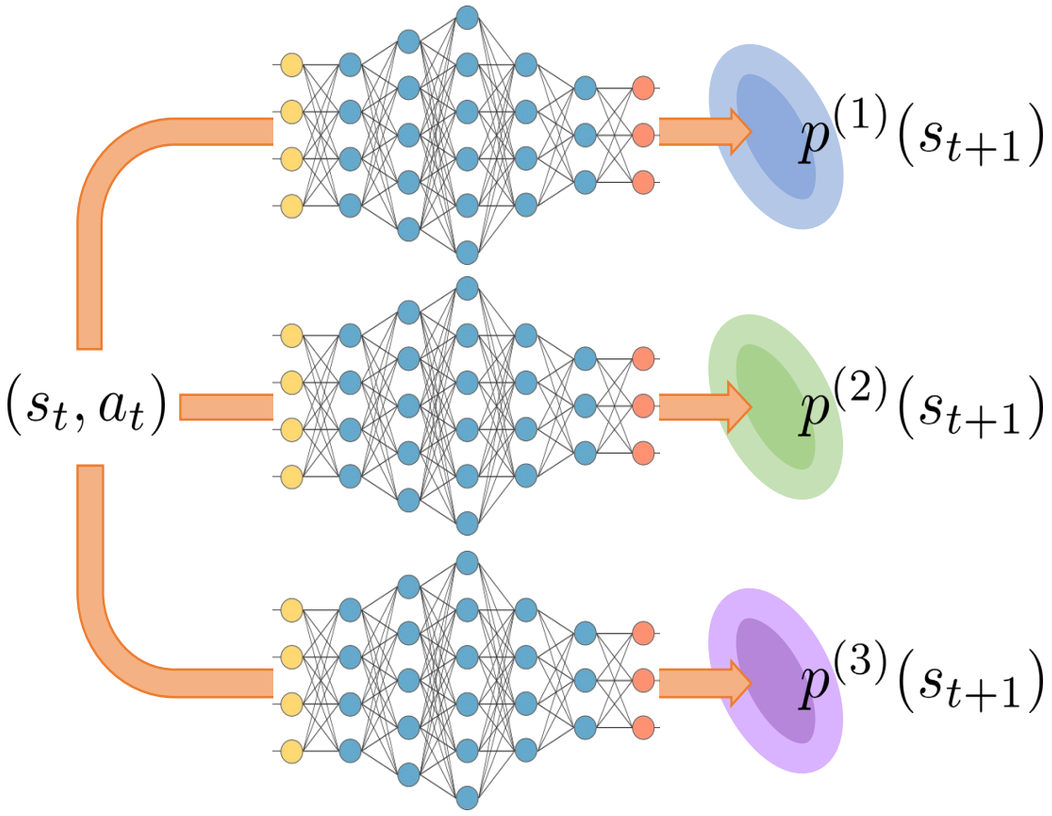


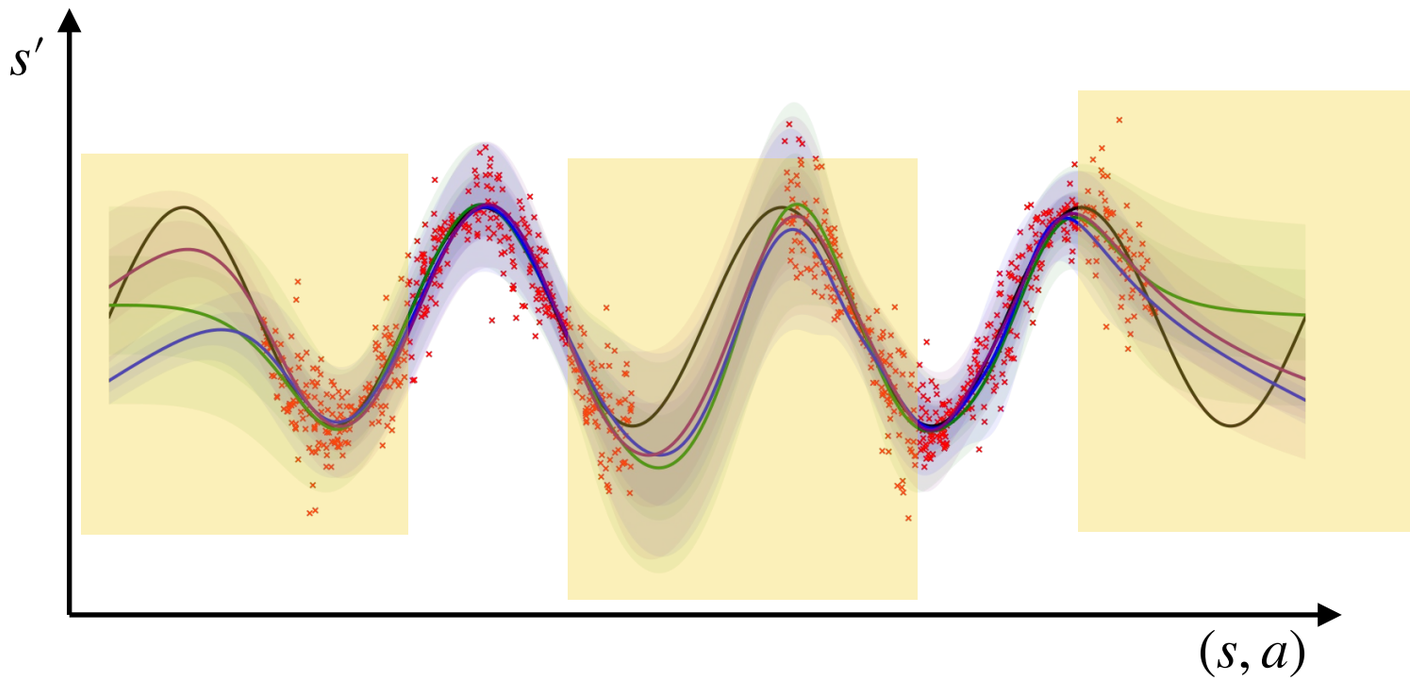
$P(\phi \mid D)$

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)
  - On the data they have not seen, each fails in its own way (high entropy of predictions)





Results

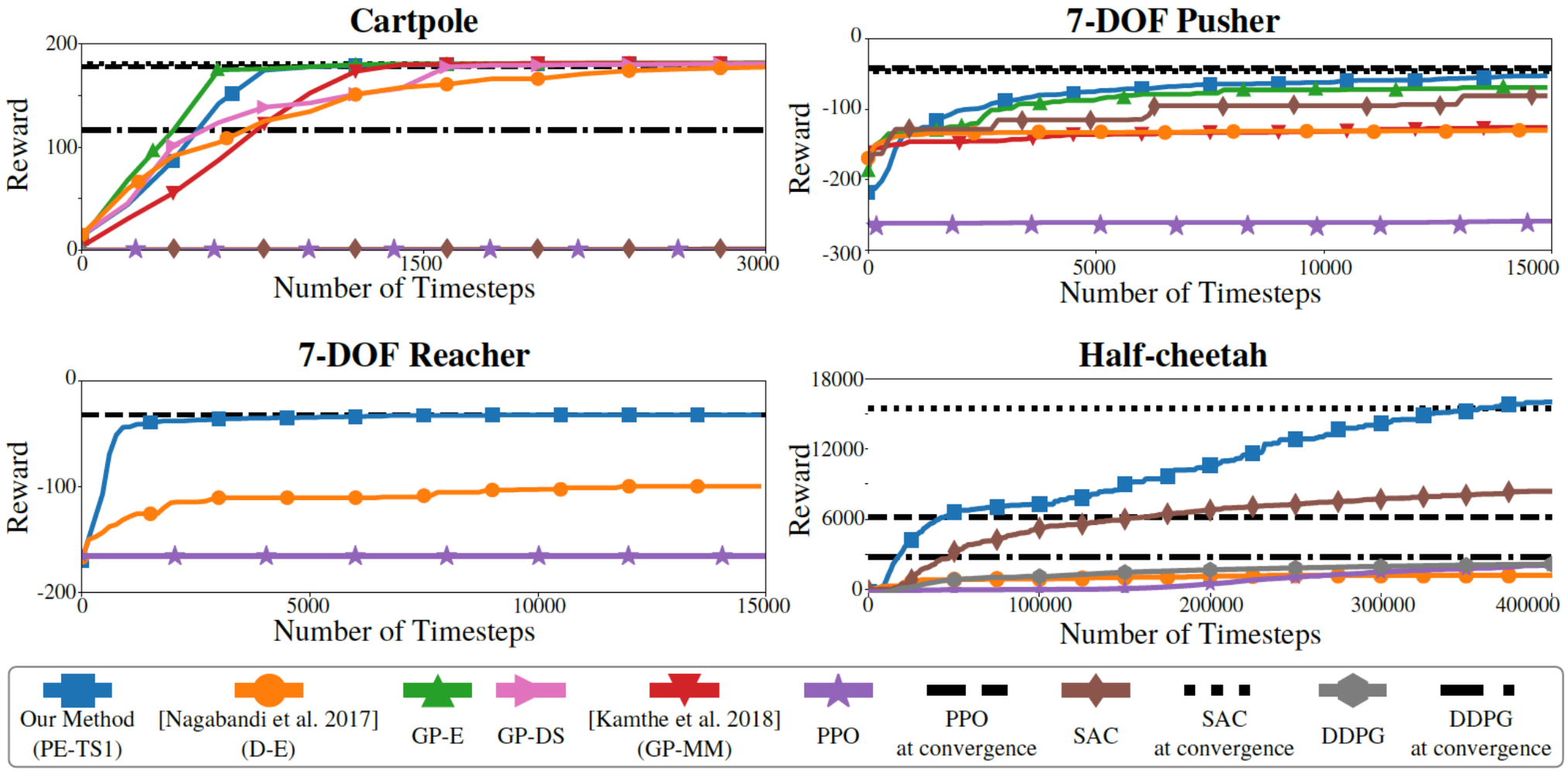


Figure 3: Learning curves for different tasks and algorithm. For all tasks, our algorithm learns in under 100K time steps or 100 trials. With the exception of Cartpole, which is sufficiently low-dimensional to efficiently learn a GP model, our proposed algorithm significantly outperform all other baselines.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

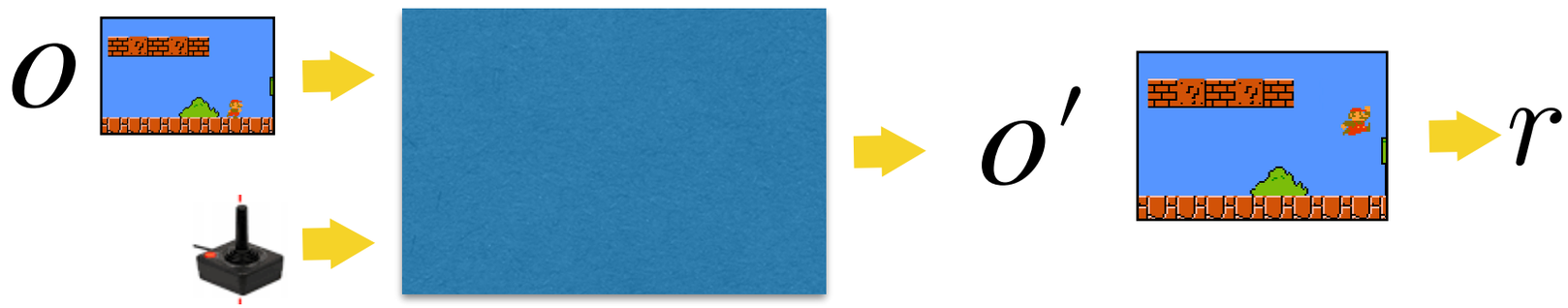
https://www.youtube.com/watch?time_continue=86&v=3d8ixUMSiL8

Model-based RL in sensory space

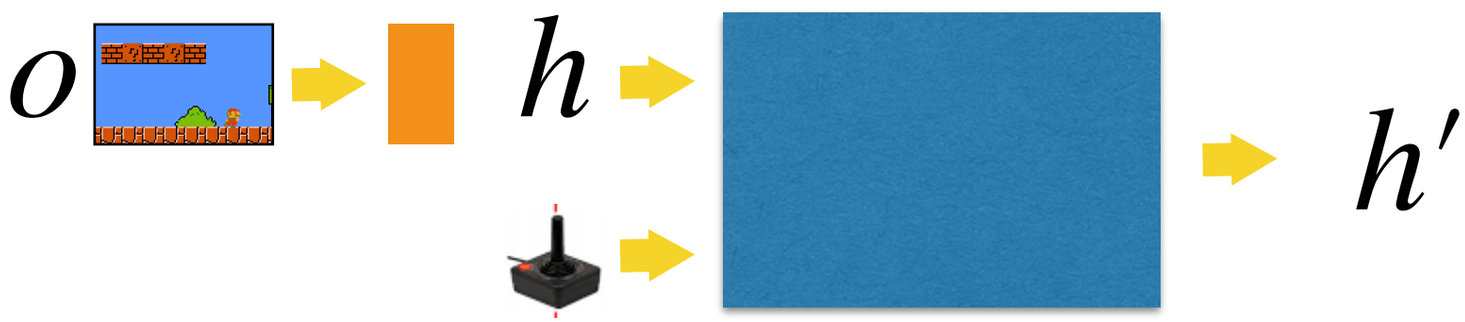
Model Learning in sensory space

Predicting action-conditioned dynamics directly in observation space

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.

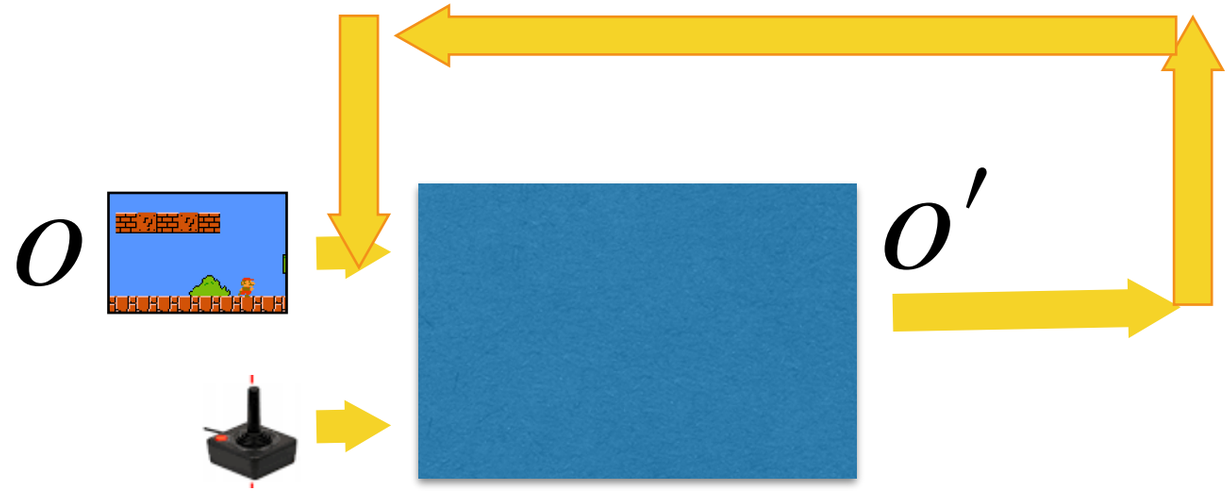


Predicting action-conditioned dynamics directly in a latent embedding space

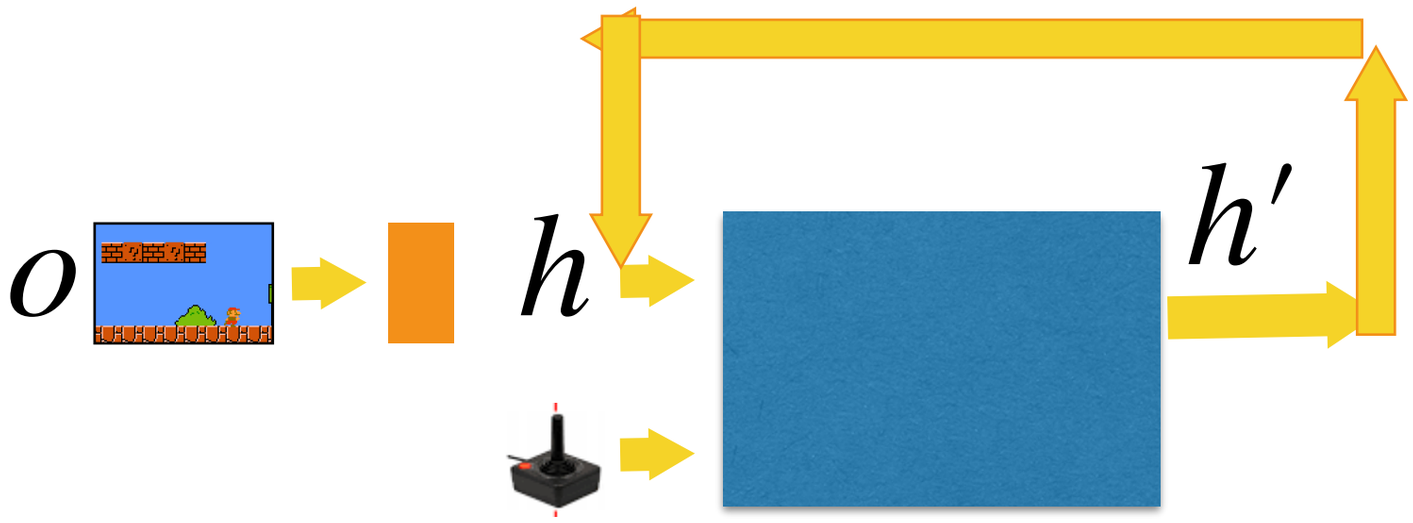


Model Learning in sensory space

Predicting action-conditioned dynamics directly in observation space

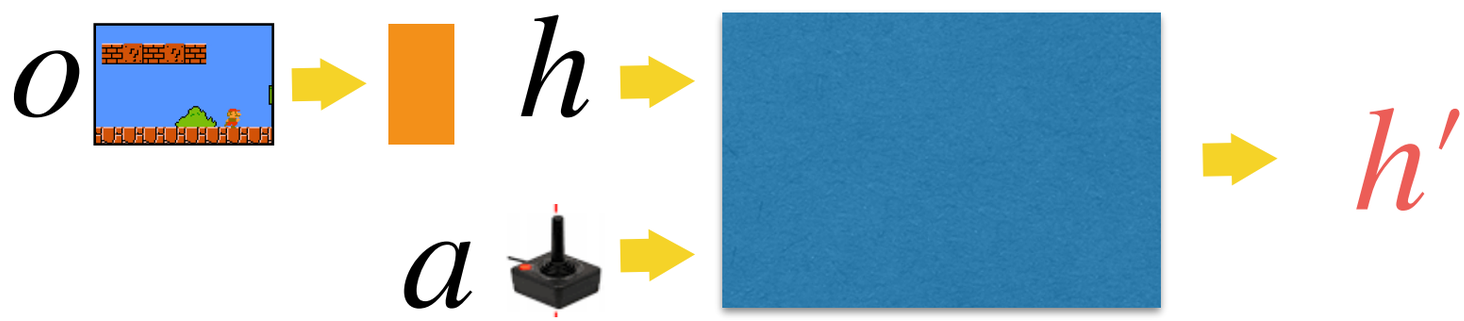


Predicting action-conditioned dynamics directly in a latent embedding space

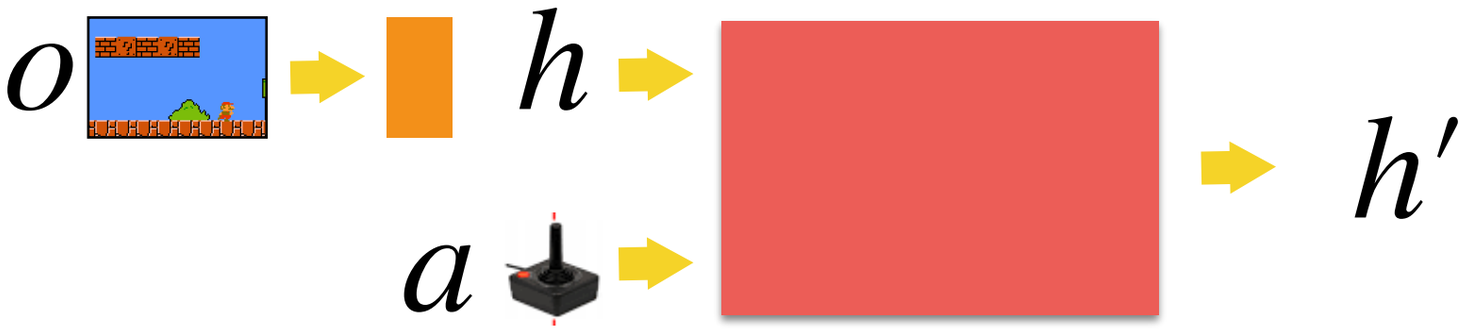


Model Learning - 4 Qs always in mind

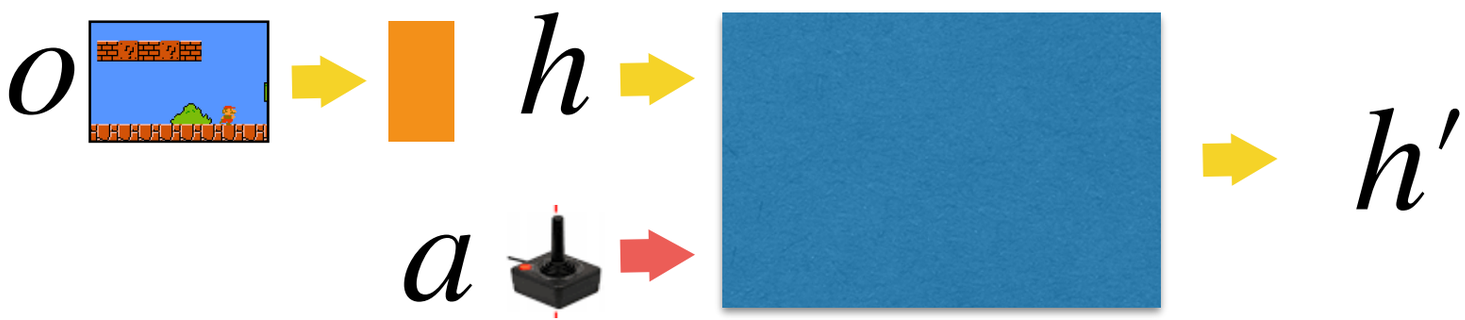
- What shall we be predicting?



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

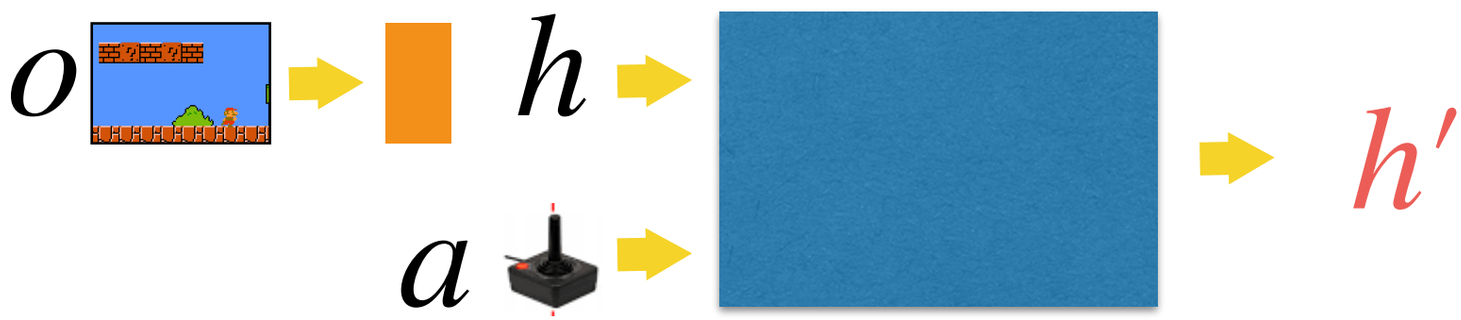


- What is the action representation?



Model Learning - 4 Qs always in mind

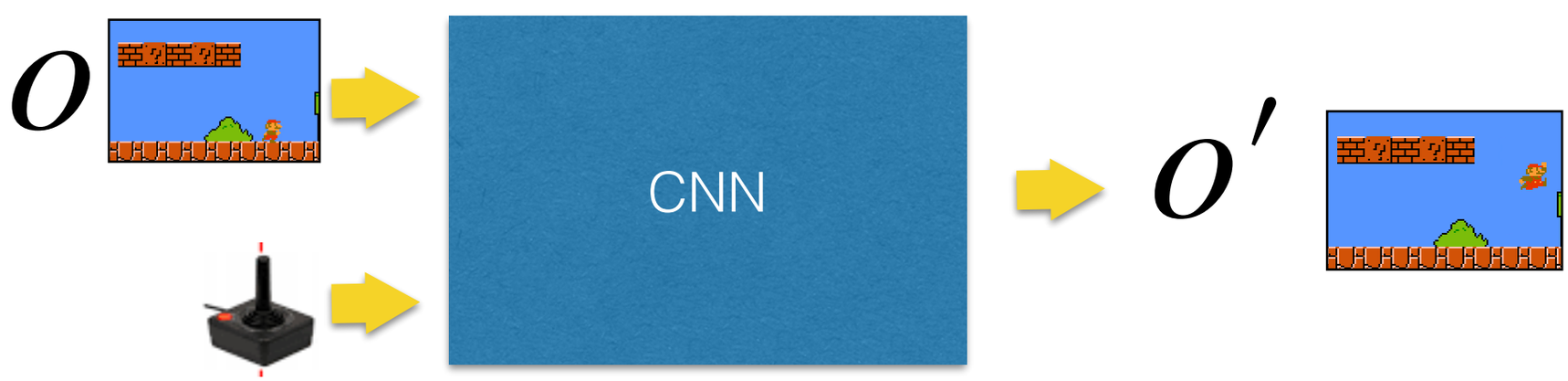
- What loss do we use?



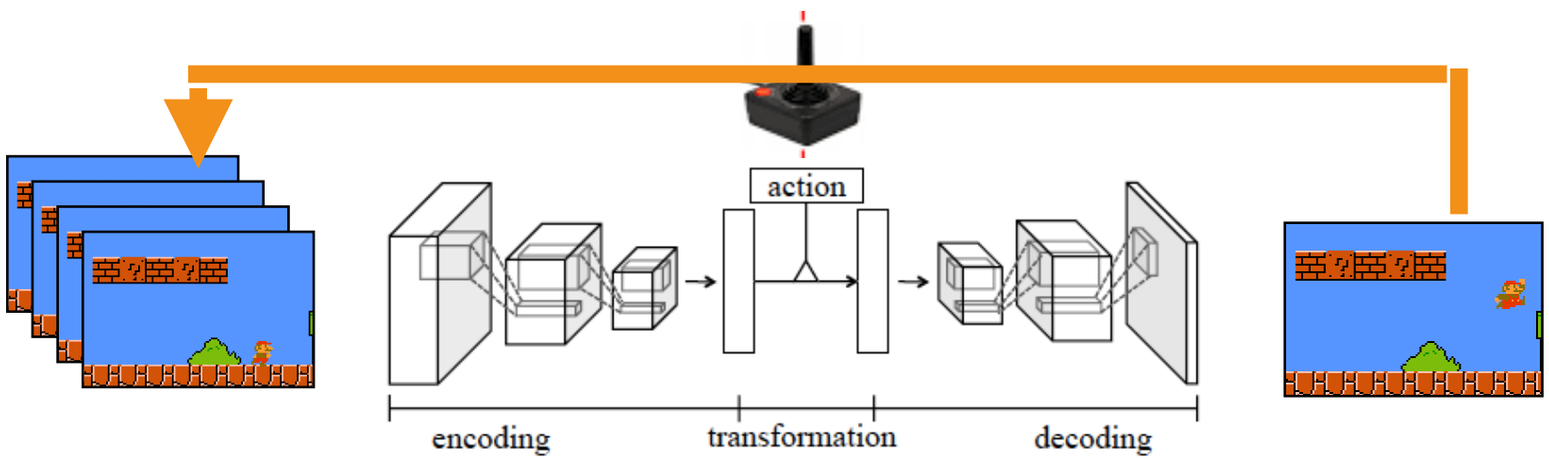
Action-Conditional Video Prediction using Deep Networks in Atari Games

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)

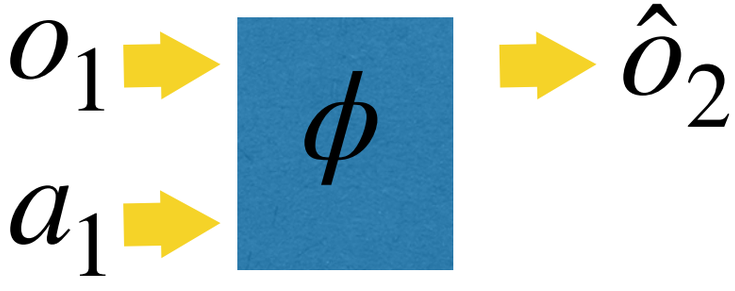


How to train our model so that unrolling works



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

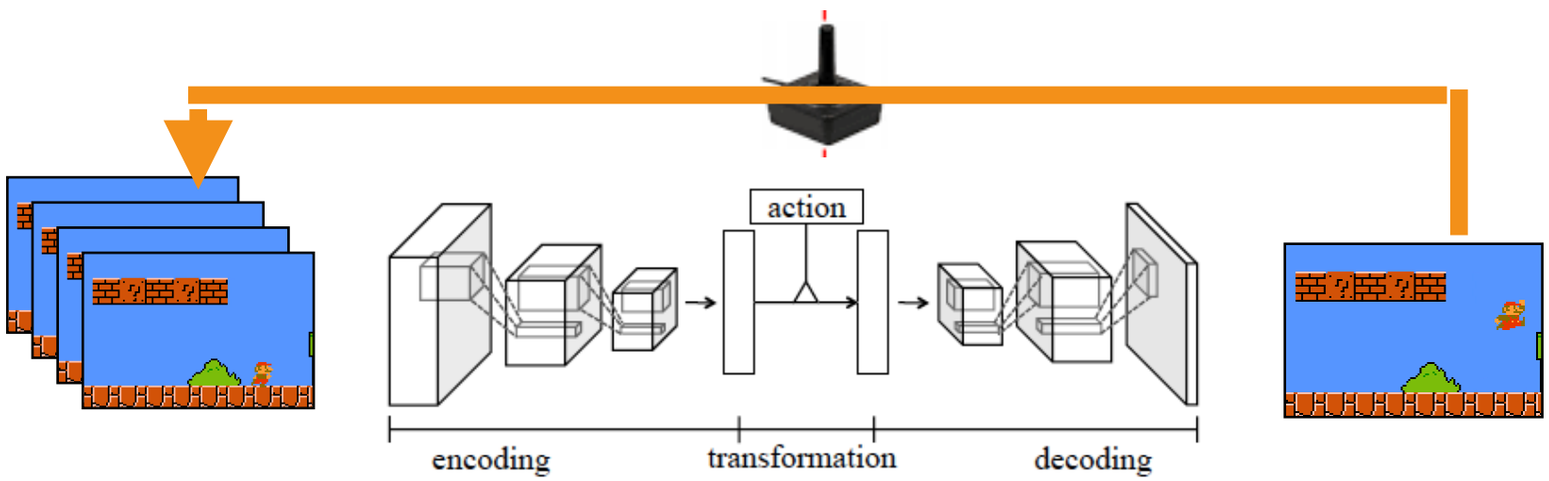
A solution: Progressively increase the unroll length k at training time so that the model learns to correct its own mistakes!



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

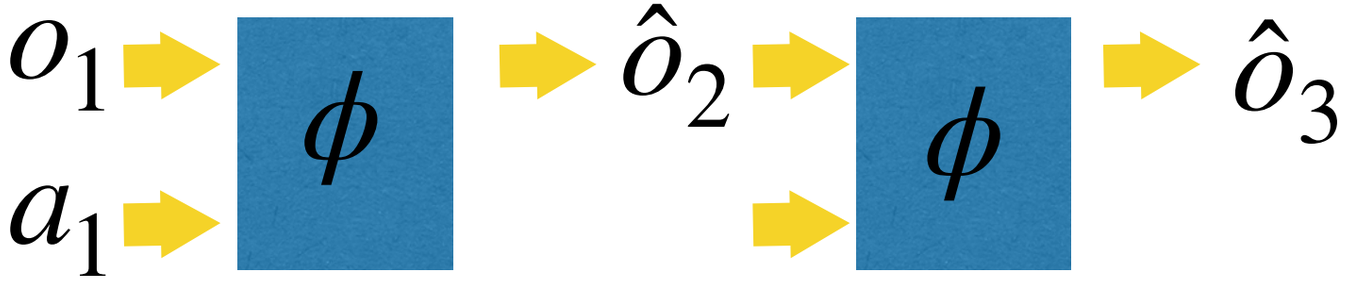
Action-Conditional Video Prediction using Deep Networks in Atari Games, Oh et al.

How to train our model so that unrolling works



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

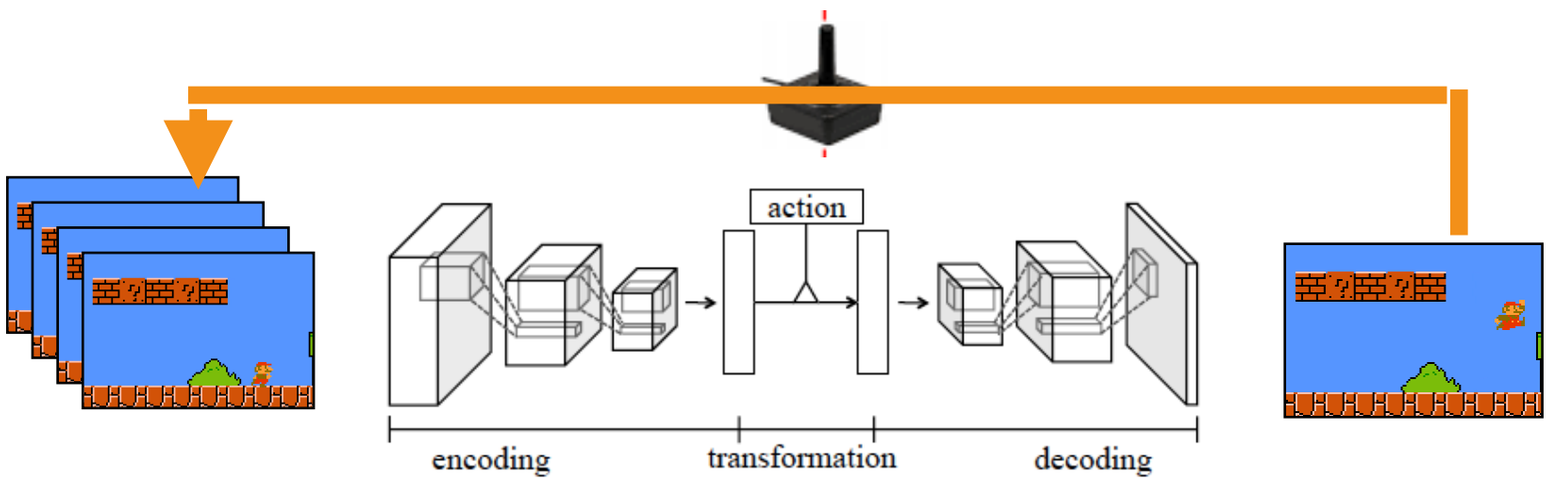
A solution: Progressively increase the unroll length k at training time so that the model learns to correct its own mistakes!



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

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

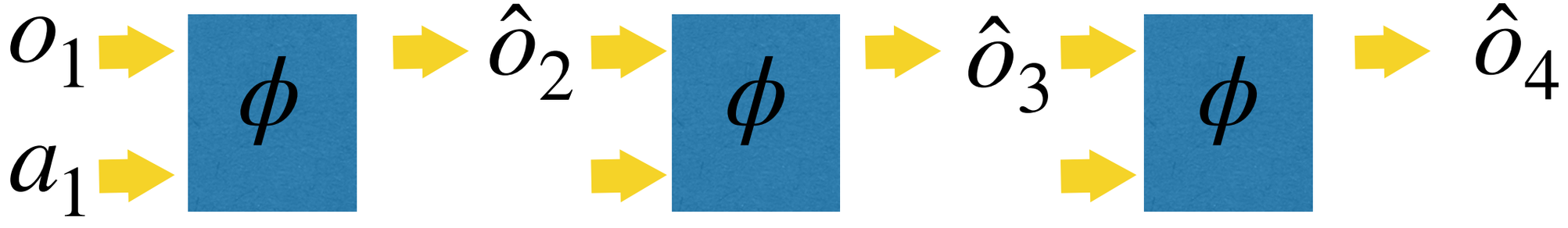
How to train our model so that unrolling works



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

A solution: Progressively increase the unroll length k at training time so that the model learns to correct its own mistakes!

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



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

Small objects are missed, e.g., the bullets.

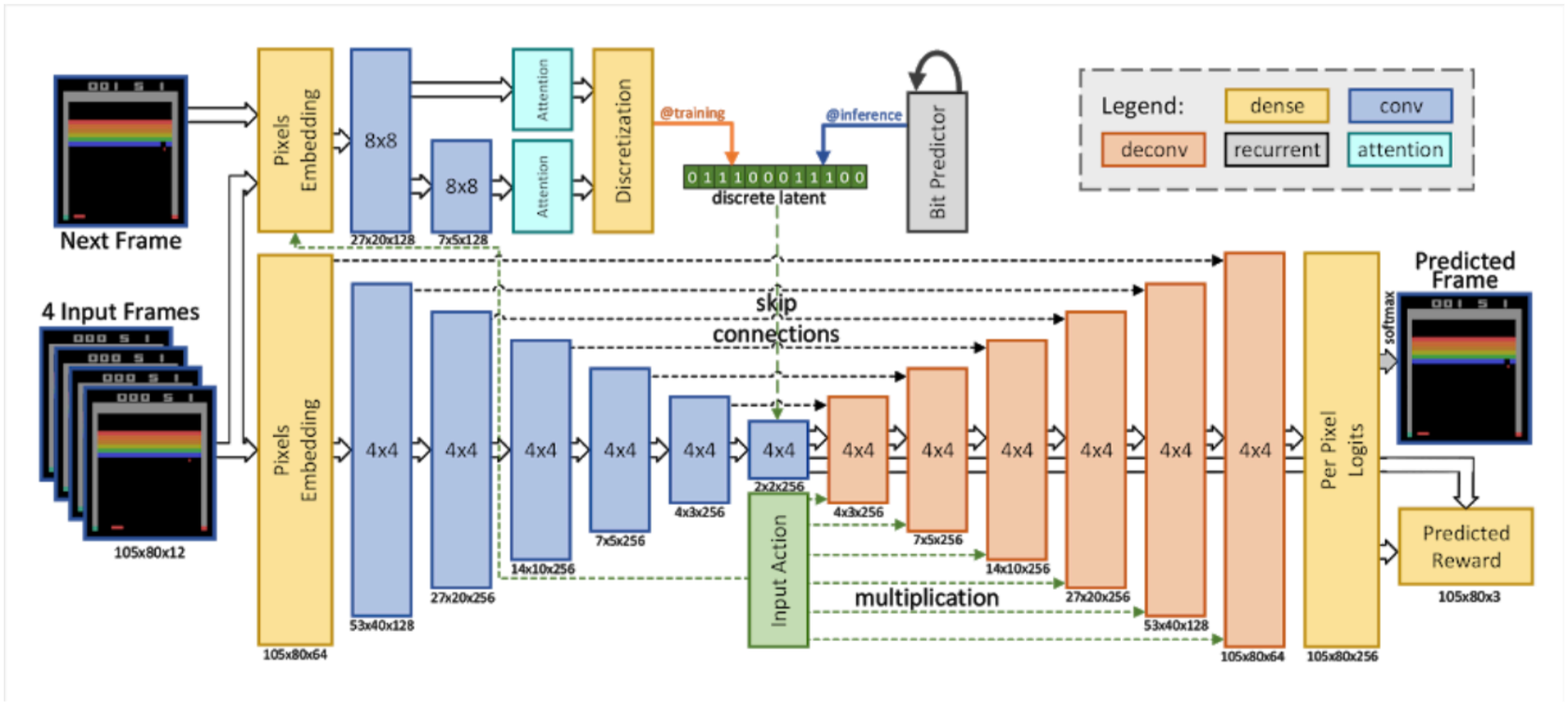
Q: Why?

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

Model-Based Reinforcement Learning for Atari

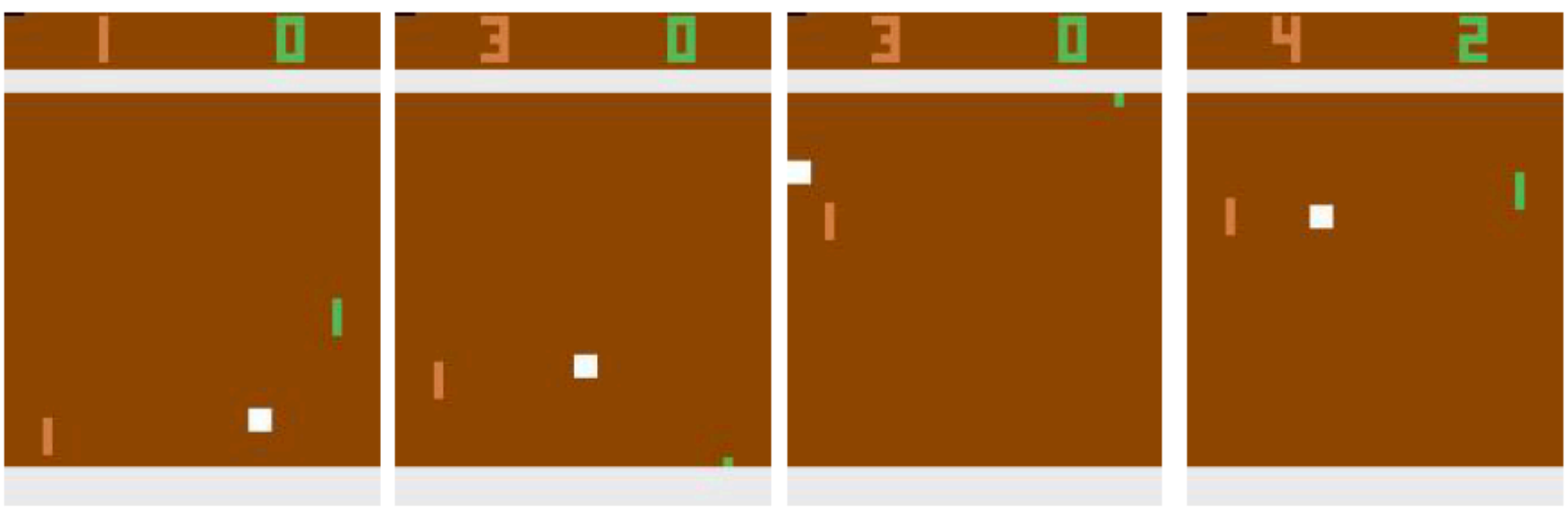
Łukasz Kaiser **Ryan Sepassi** Google Brain
Henryk Michalewski **Piotr Miłoś** University of Warsaw
Błażej Osiński deepsense.ai

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 :-( )



results

Results

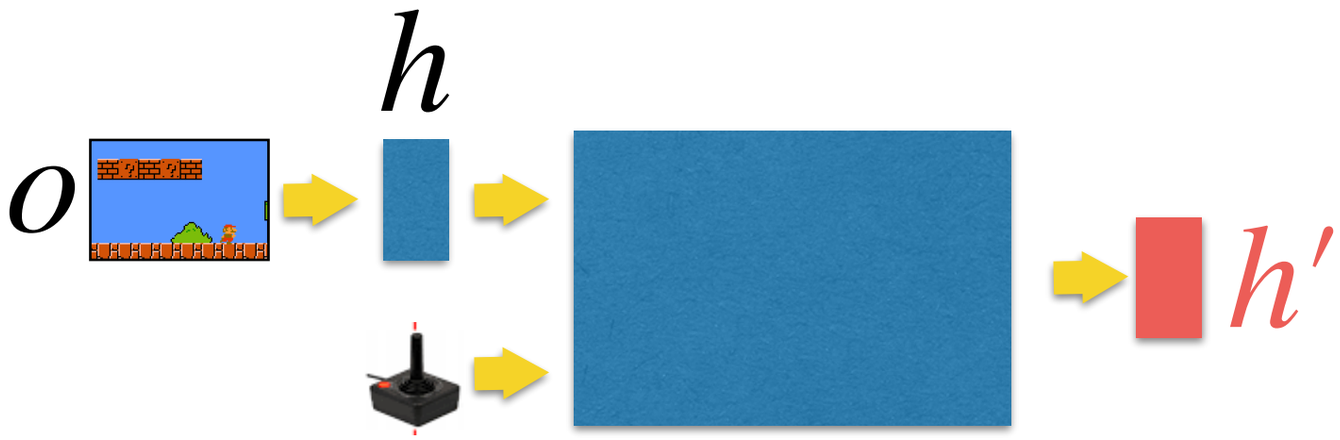
- Number of frames required to reach human performance.
- Model based first time wins also from pixels!!

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

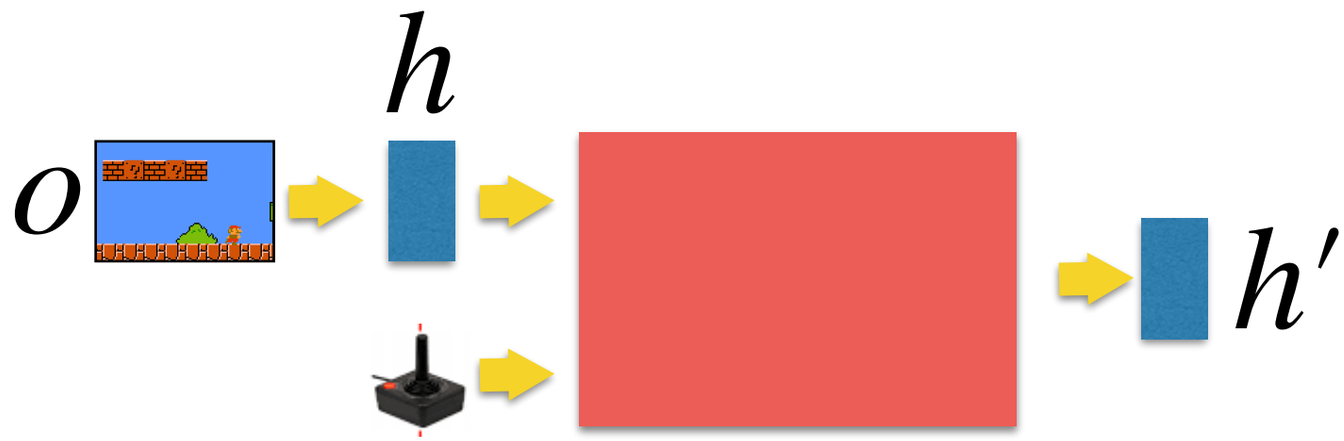
results

Model-based RL in Atari

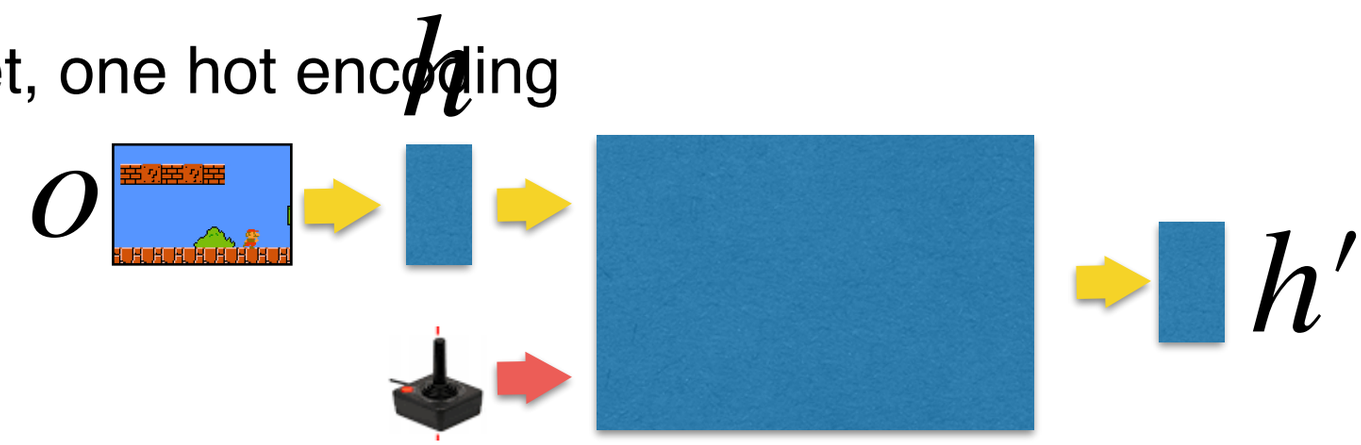
- We predicted next frame



- Our model was a CNN

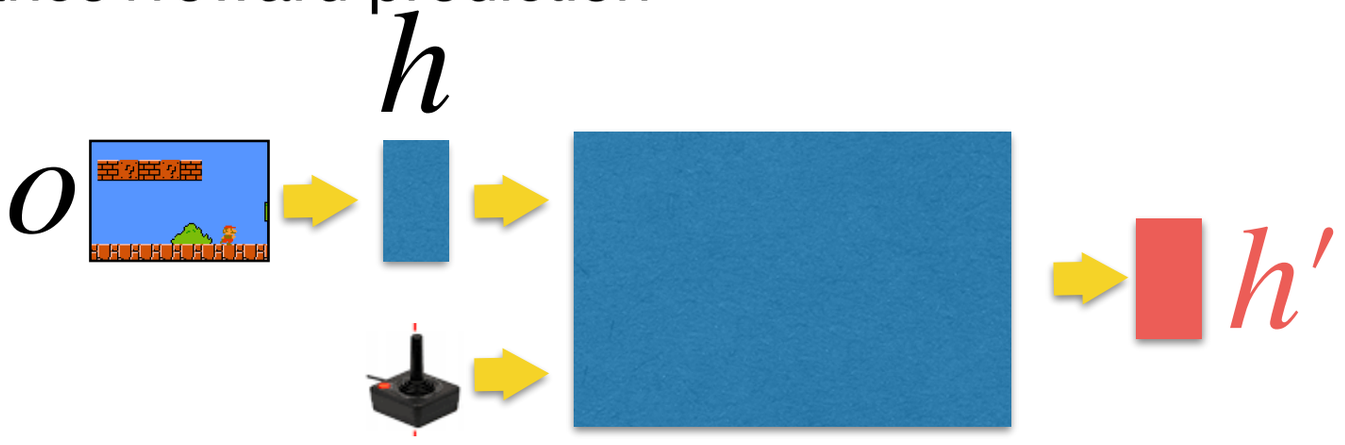


- The Atari discrete action set, one hot encoding



Model-based RL in Atari

- We mimimized L2 pixel distance+reward prediction



Predicting Raw Sensory Input (Pixels)

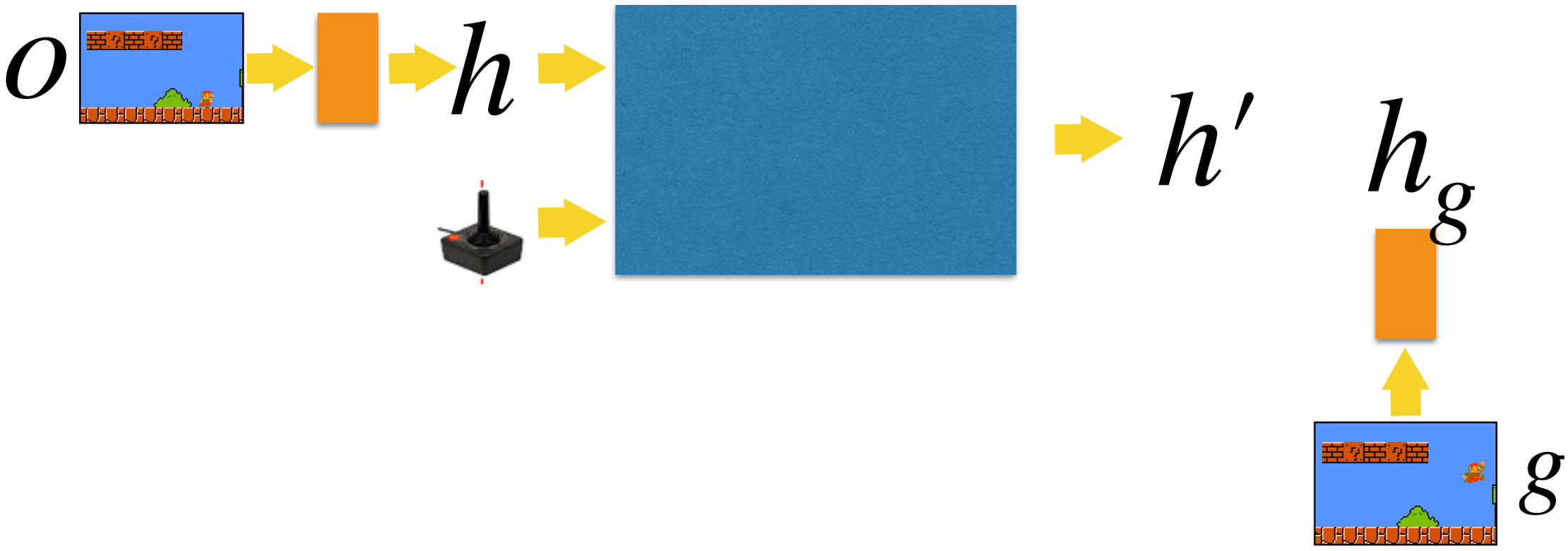
Should our prediction model be predicting the input observations?

- 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.

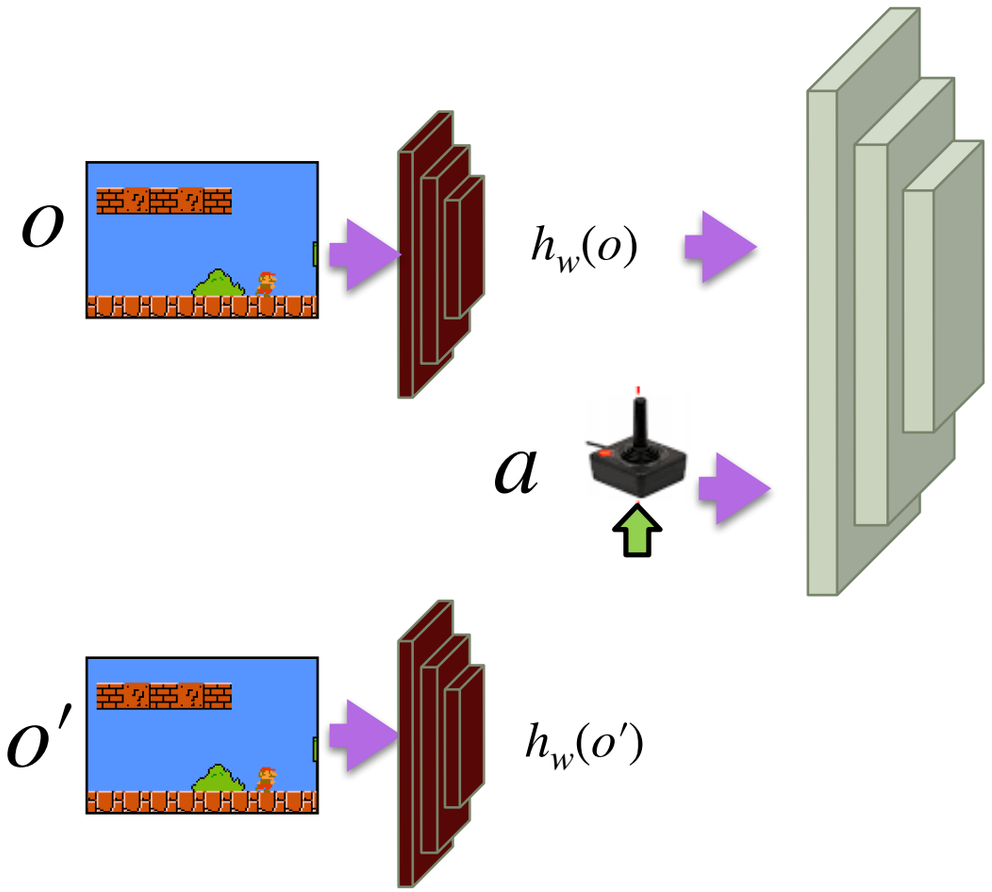
Prediction in a latent space

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



$$\min_{w,\phi} . \quad \|f(h(o;w),a;\phi) - h(o';w)\|$$

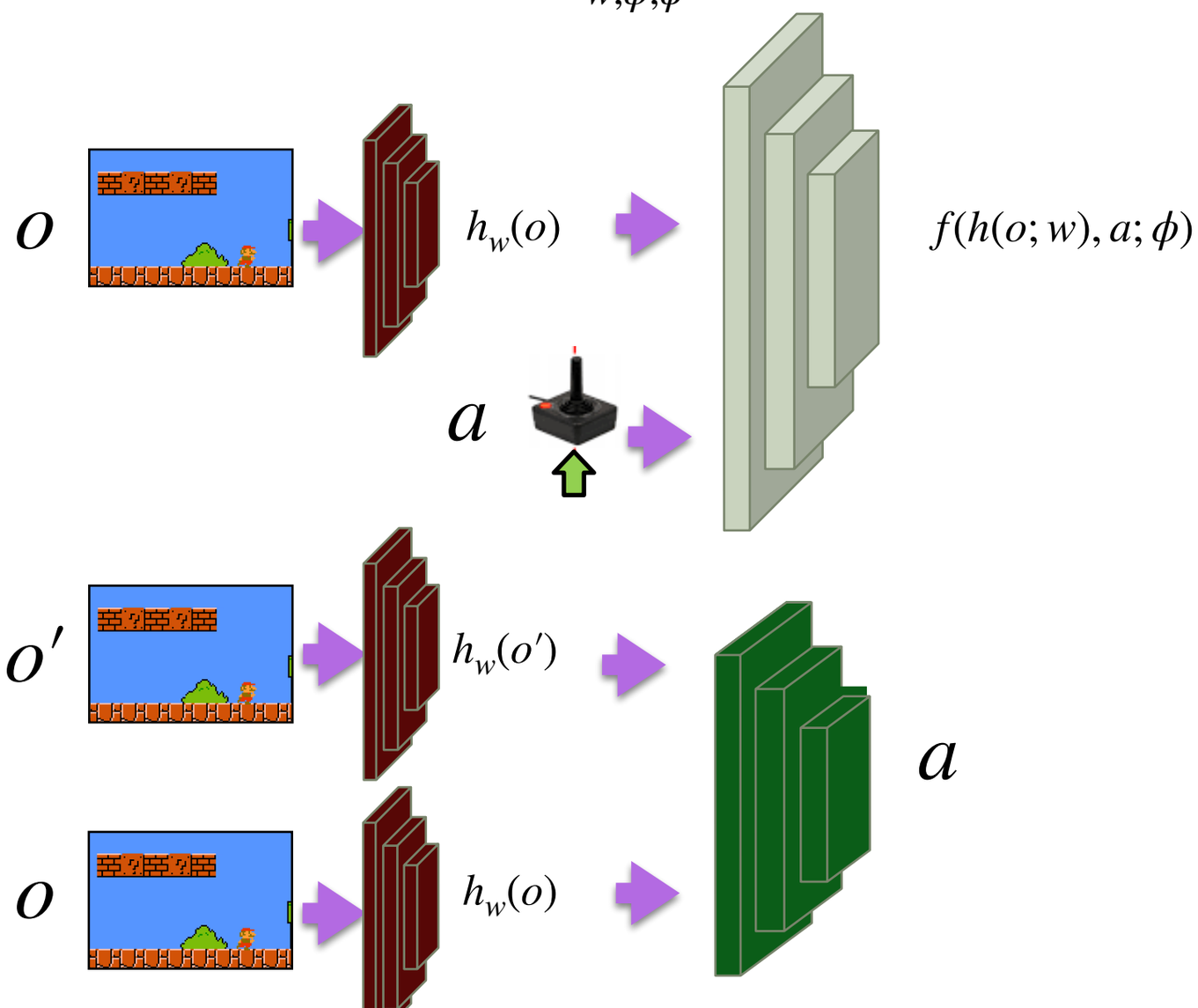
What is the problem with this optimization problem?

There is a trivial solution :-(

Prediction in a latent space

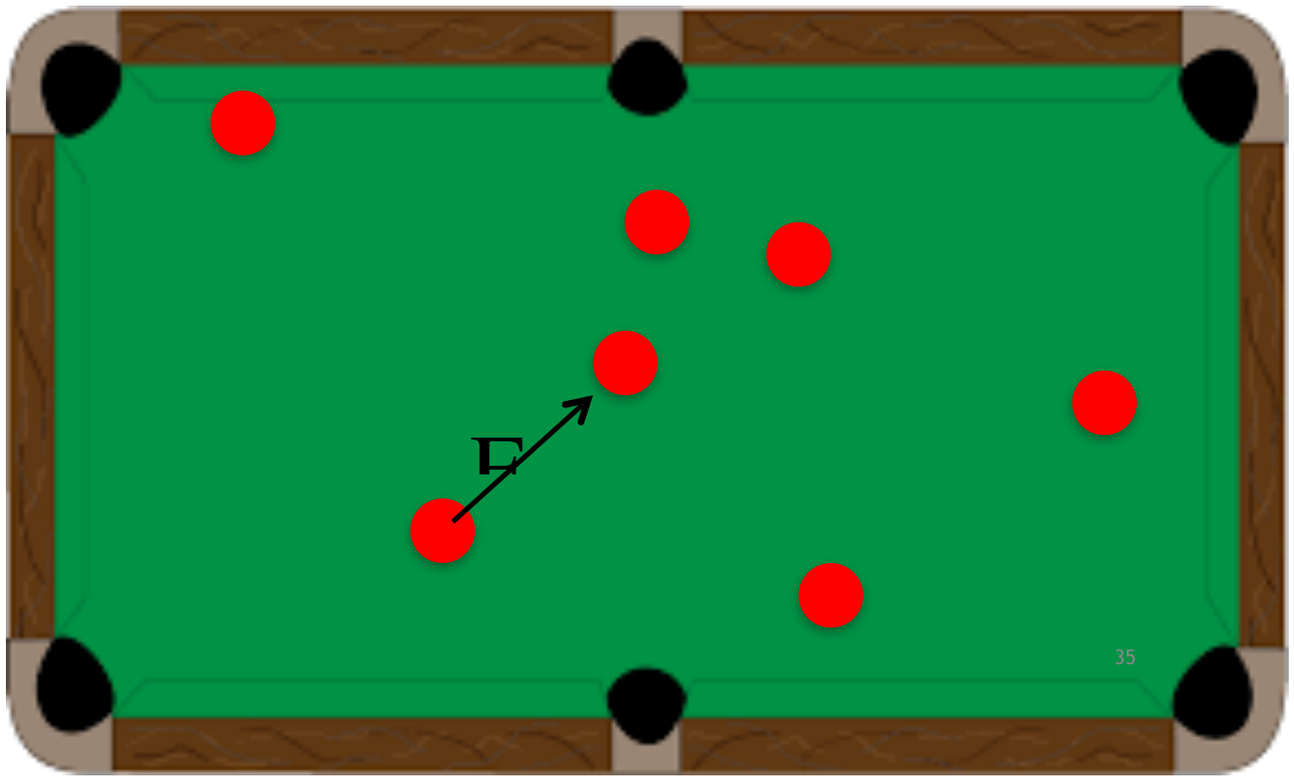
Our model tries to predict a (potentially latent) embedding, from which rewards can be computed, e.g., by matching the embedding from my desired goal image to the prediction.

$$\min_{w,\phi,\psi} . \quad \|f(h(o;w),a;\phi) - h(o';w)\| + \|\mathrm{Inv}(h(o;w),h(o');\psi) - a\|$$

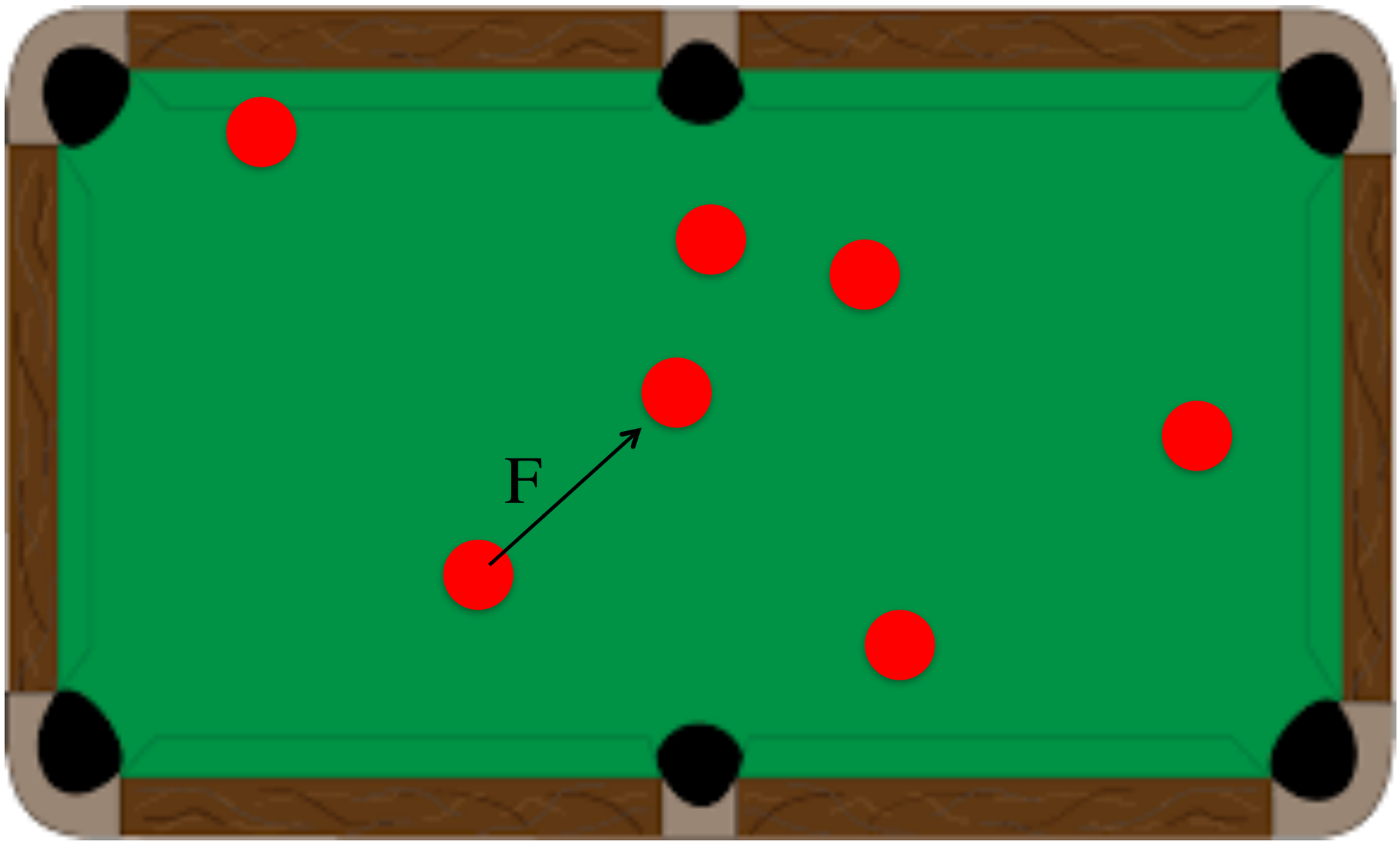


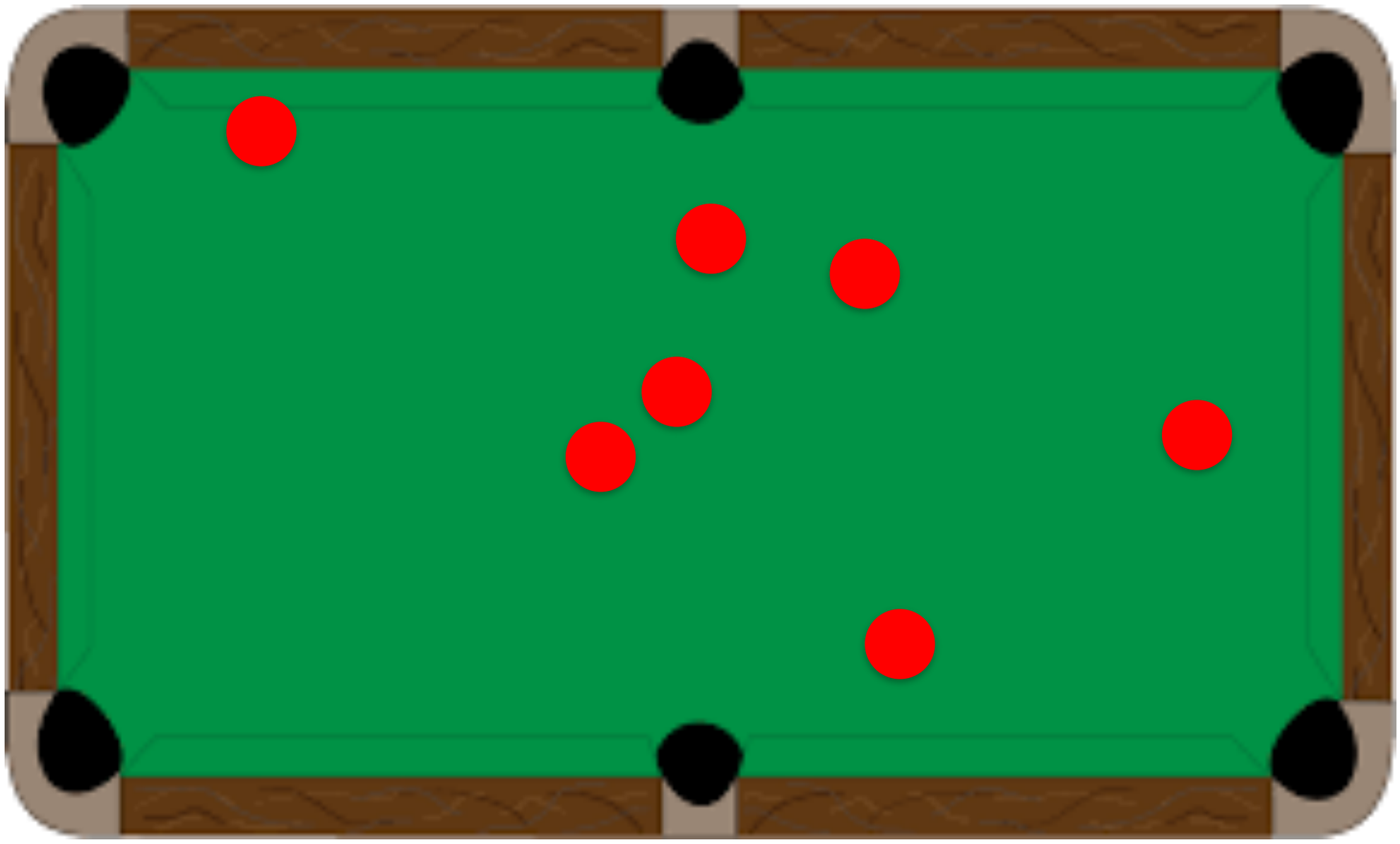
Learning visual dynamics

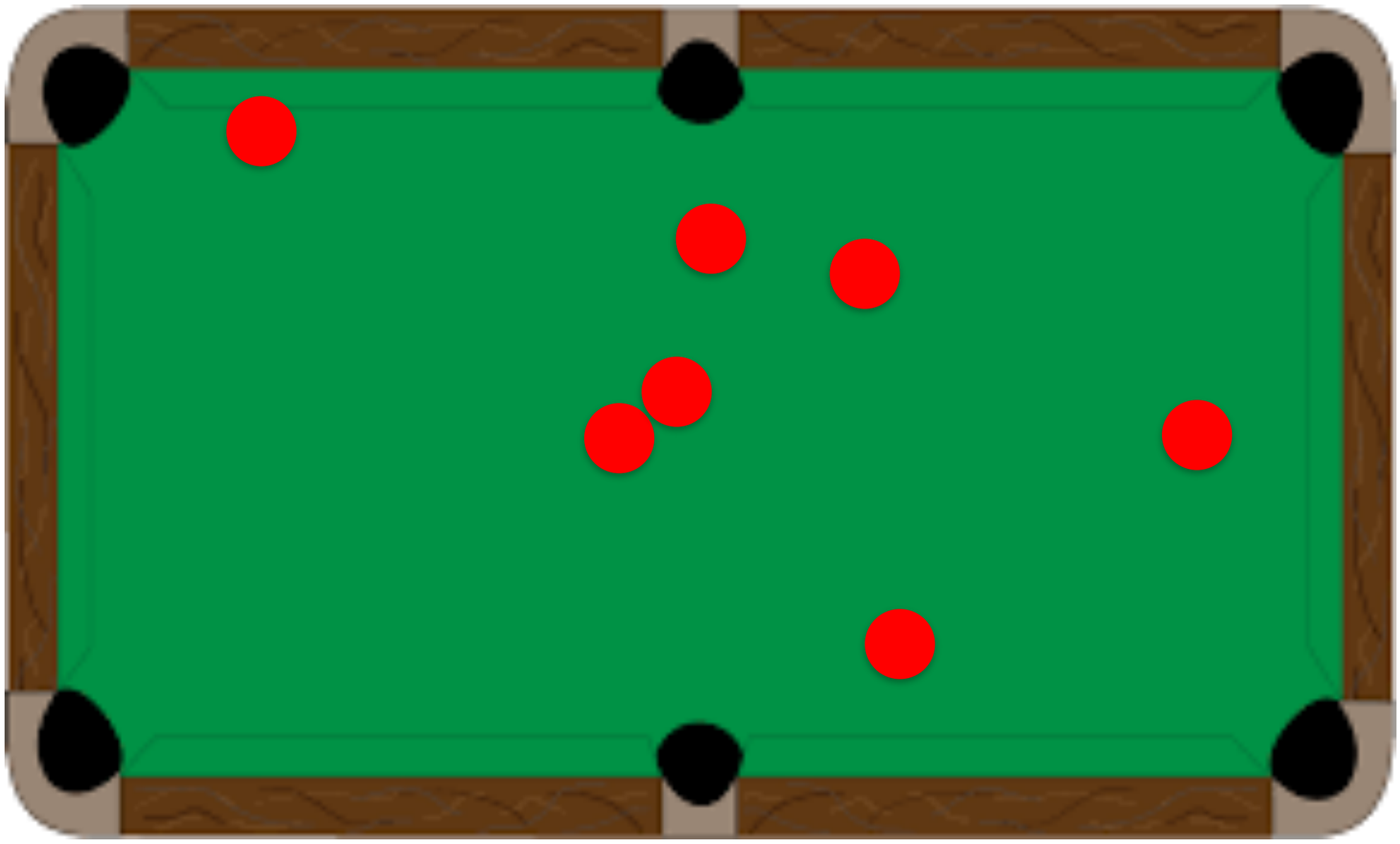
- Overall, the scene over time, content-wise does not change
- What if, instead of predicting appearance, or deep features that do not disentangle motion and appearance, we track objects and just **predict object motion?**

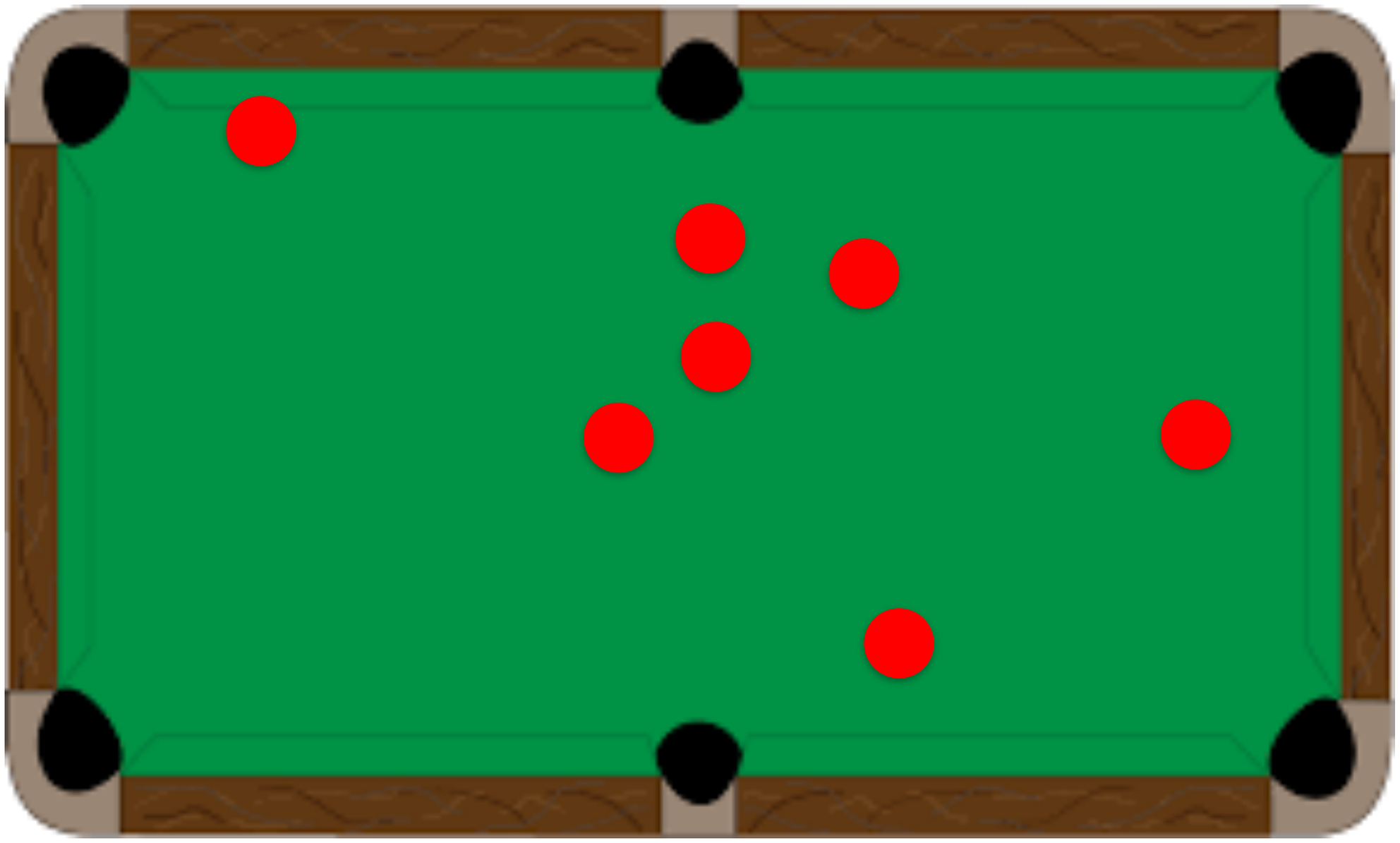


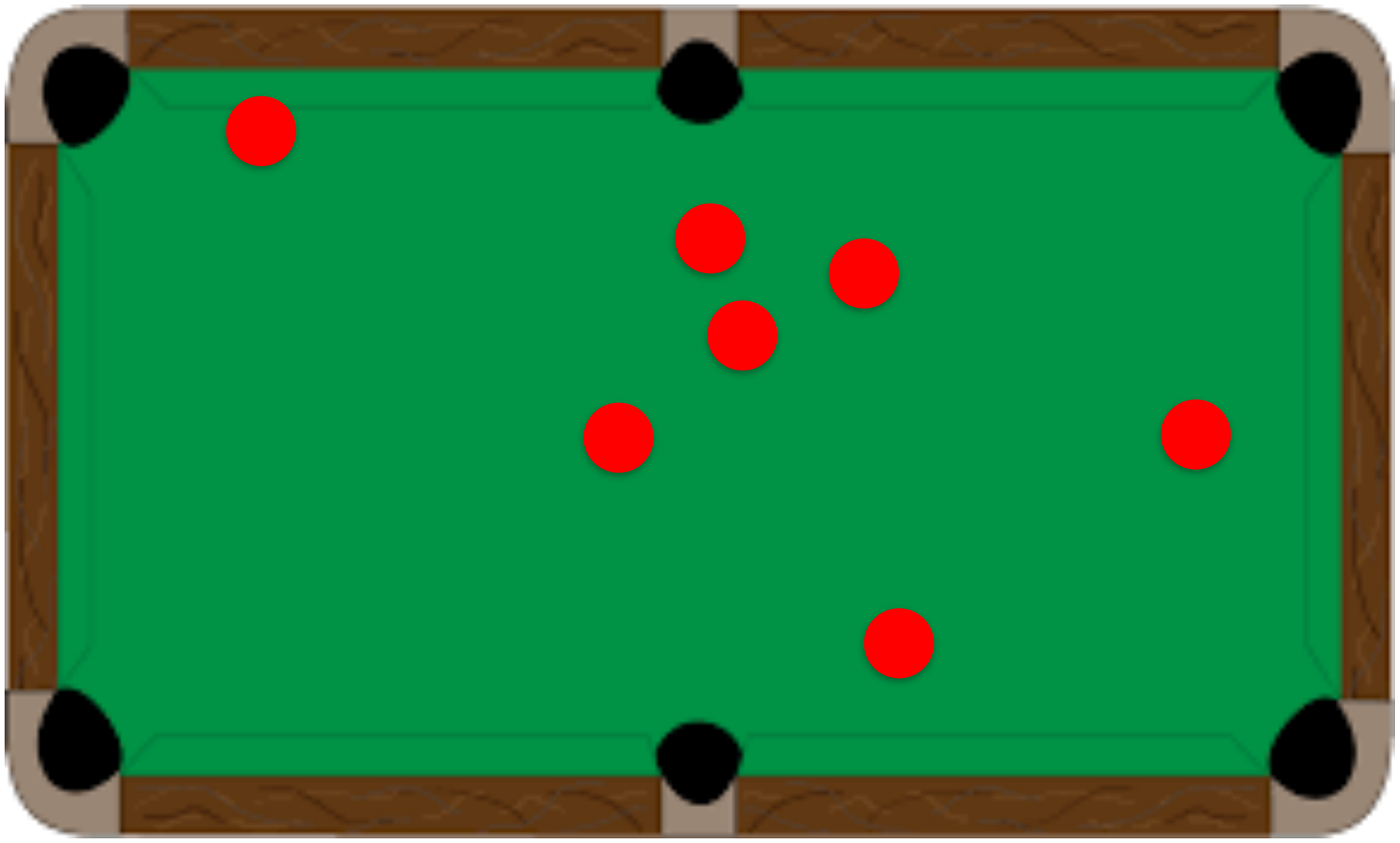
How do we learn to play Billiards?





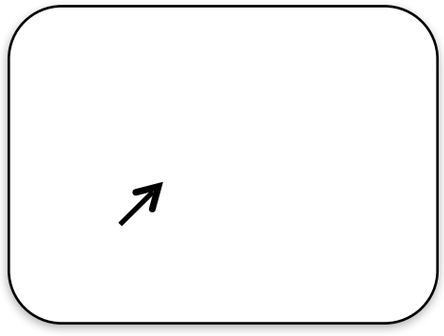


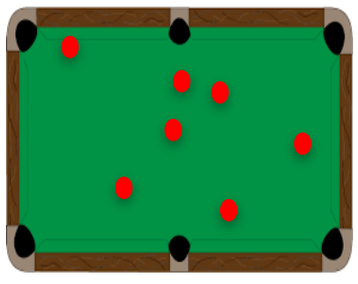


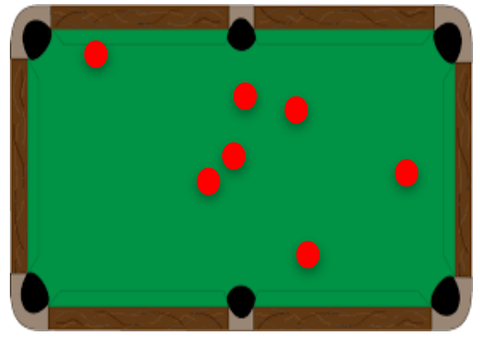


Learning Action-Conditioned Billiard Dynamics

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



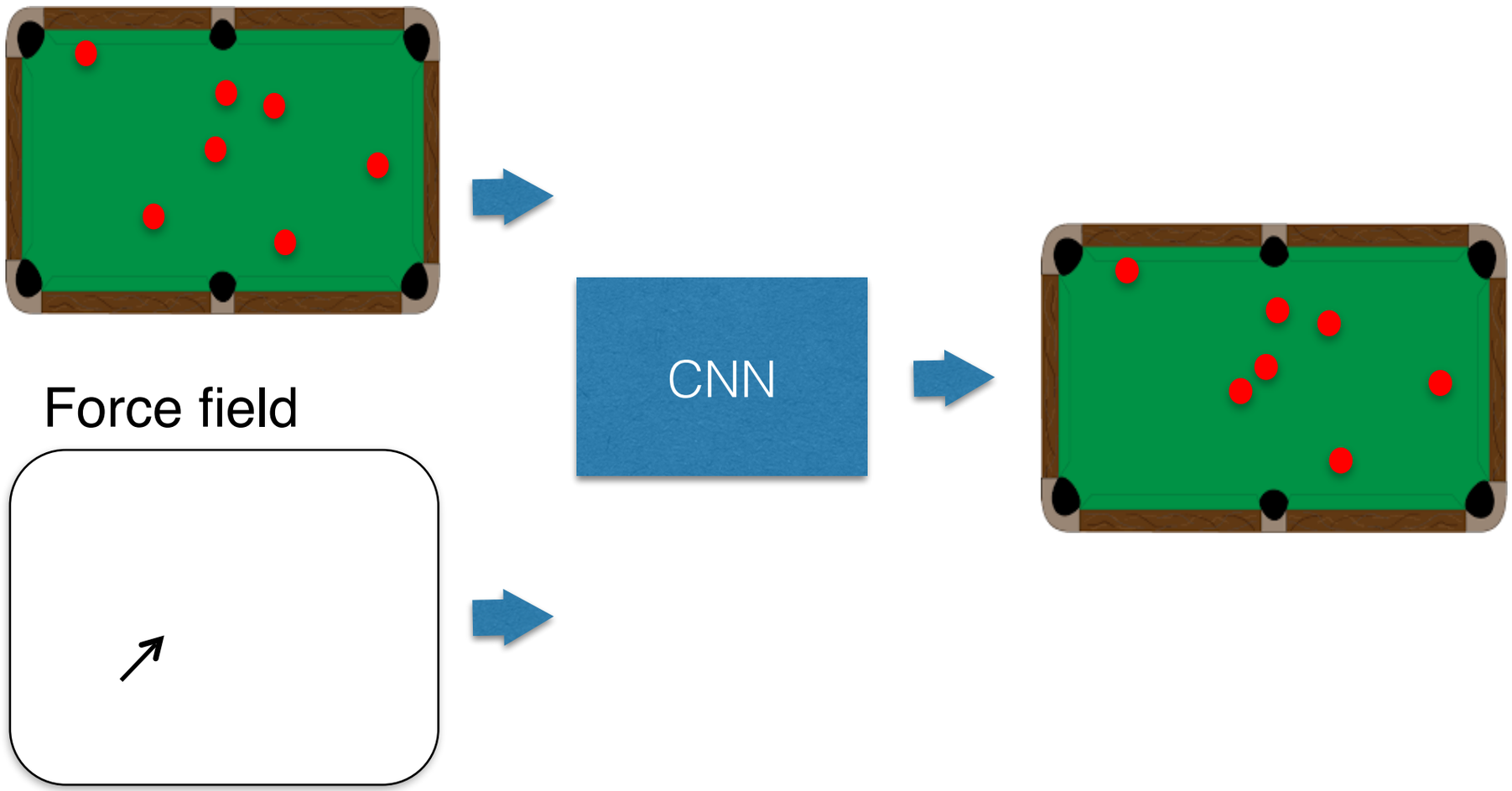




We simply predict ball motion trajectories

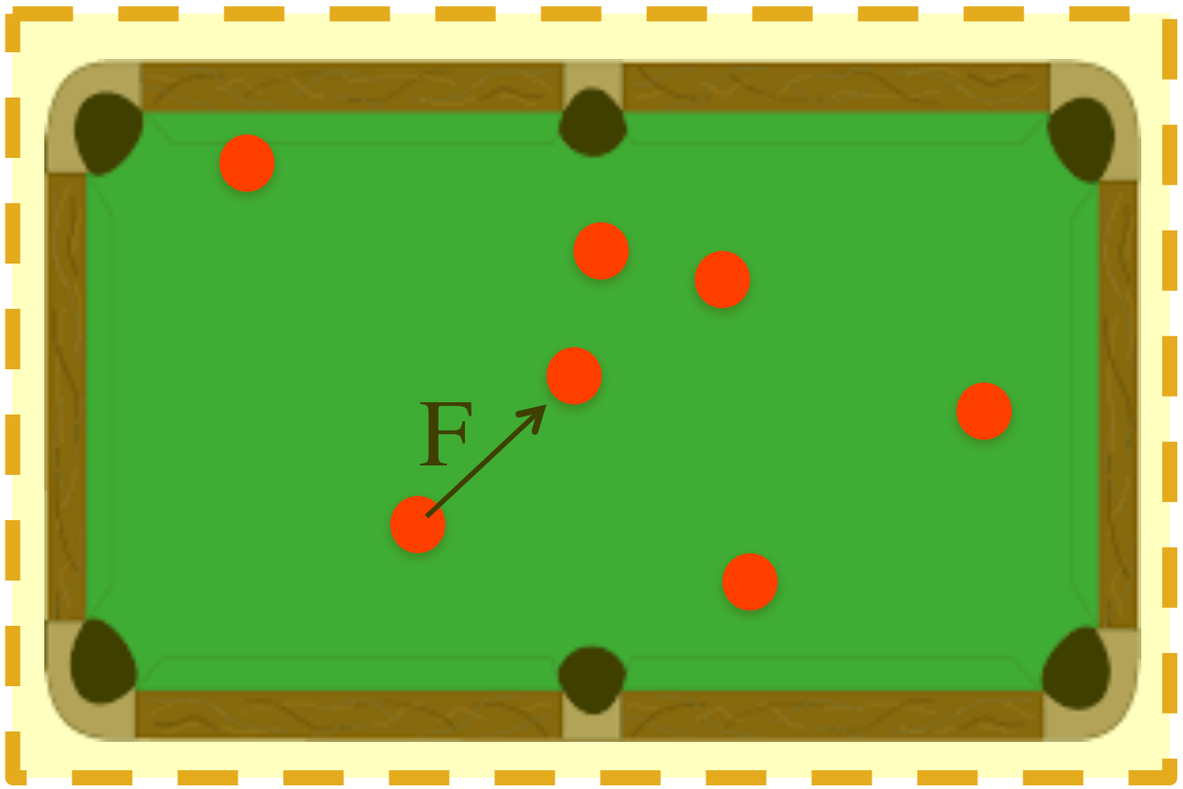
Predictive Visual Models of Physics for Playing Billiards, K.F. et al. ICLR 2016

Learning Action-Conditioned Billiard Dynamics

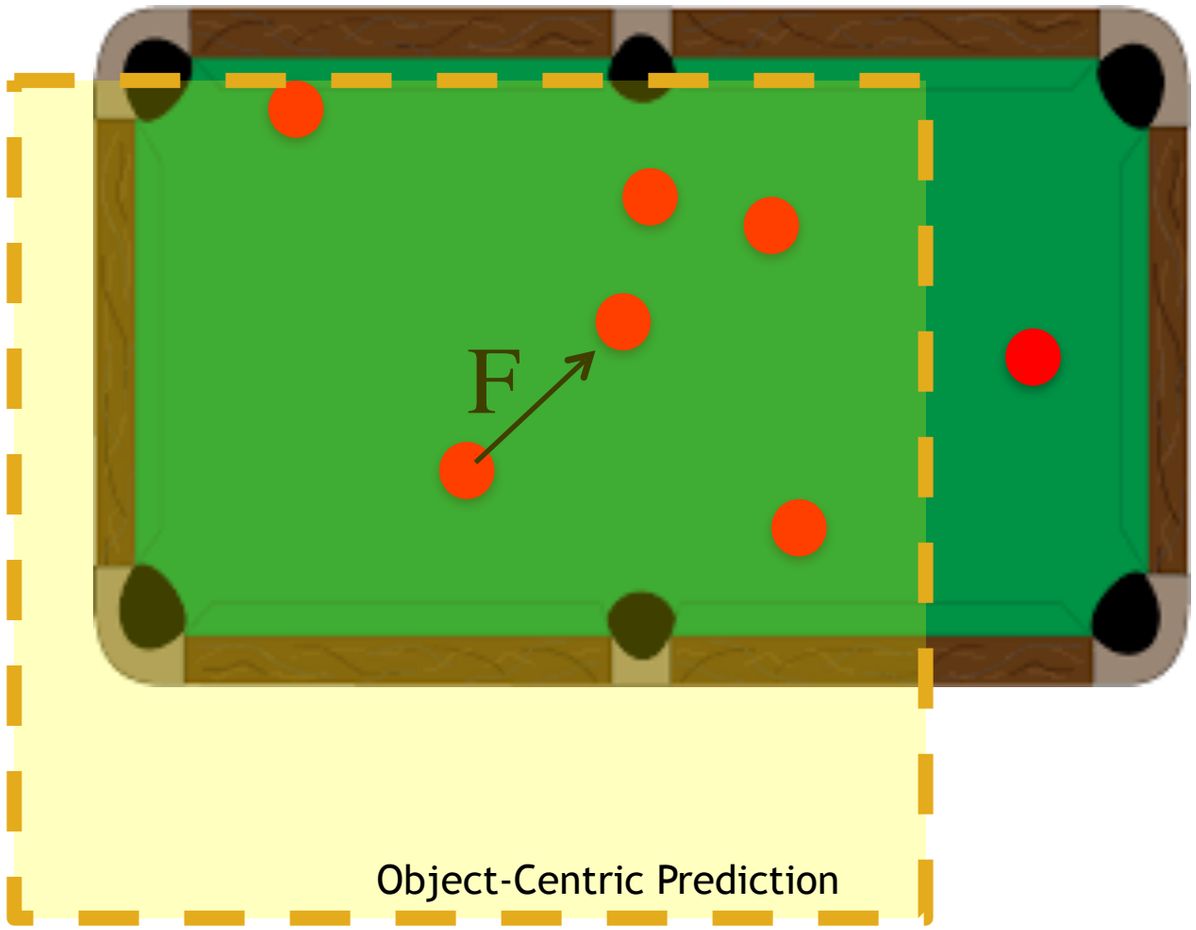


Q: will our model be able to generalize across different number of balls?

Learning Action-Conditioned Billiard Dynamics

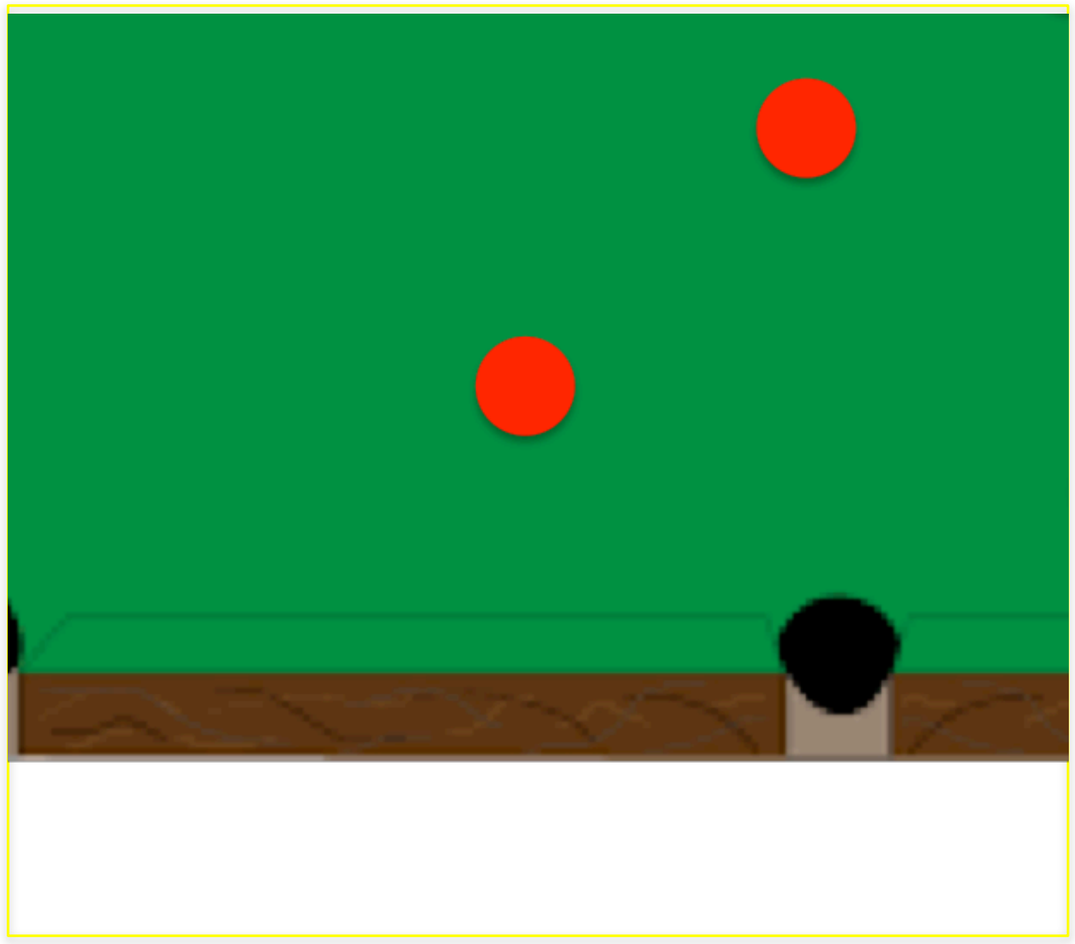


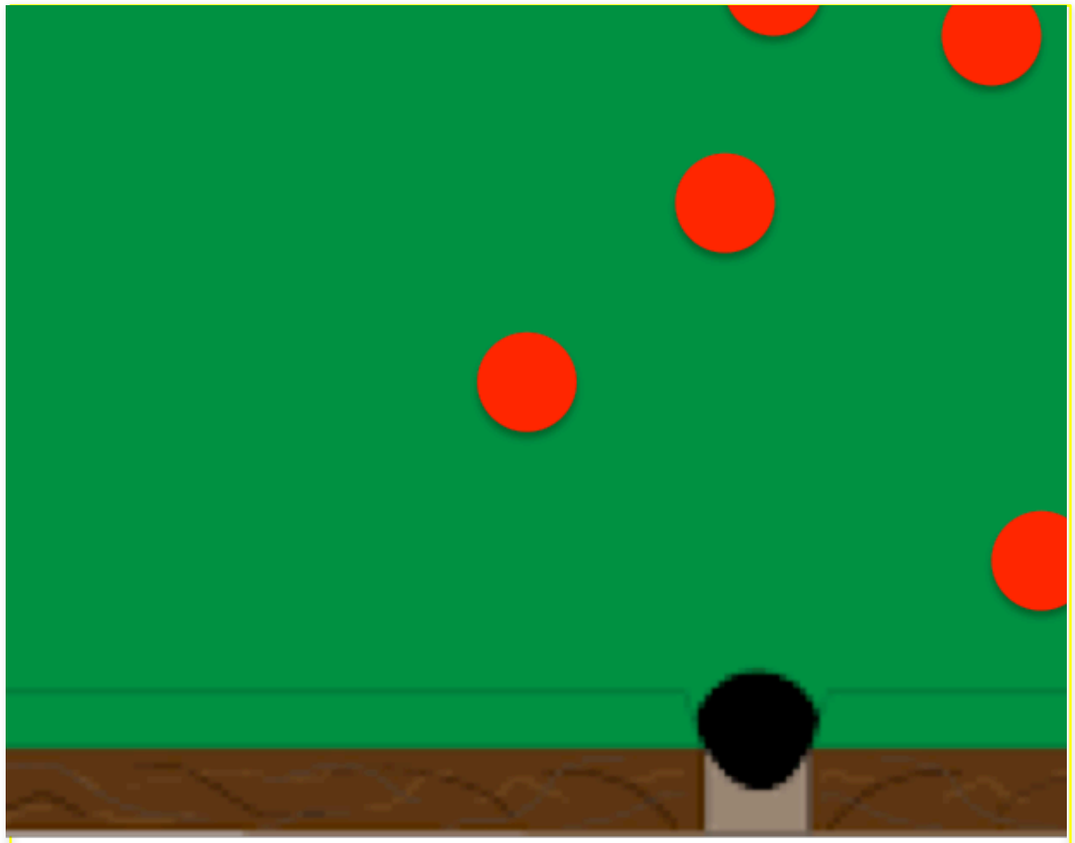
World-Centric Prediction

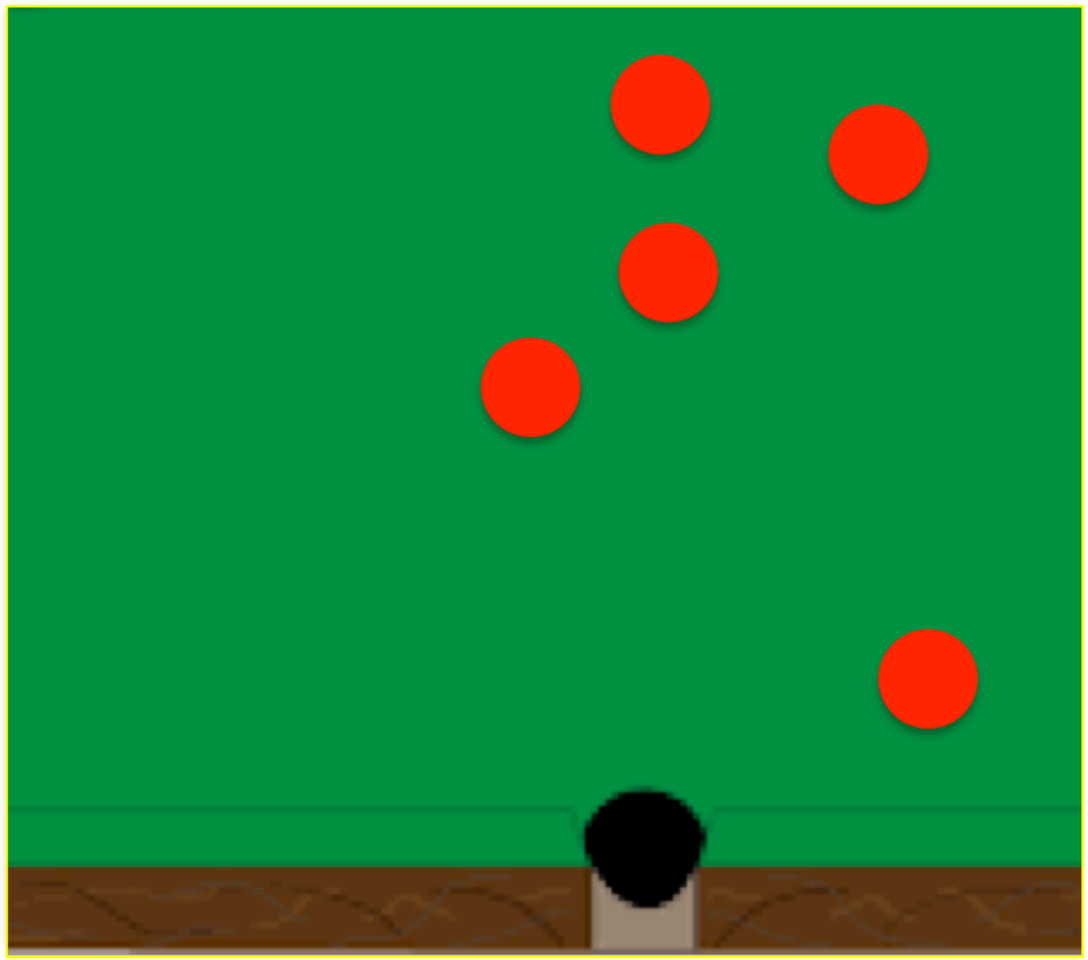


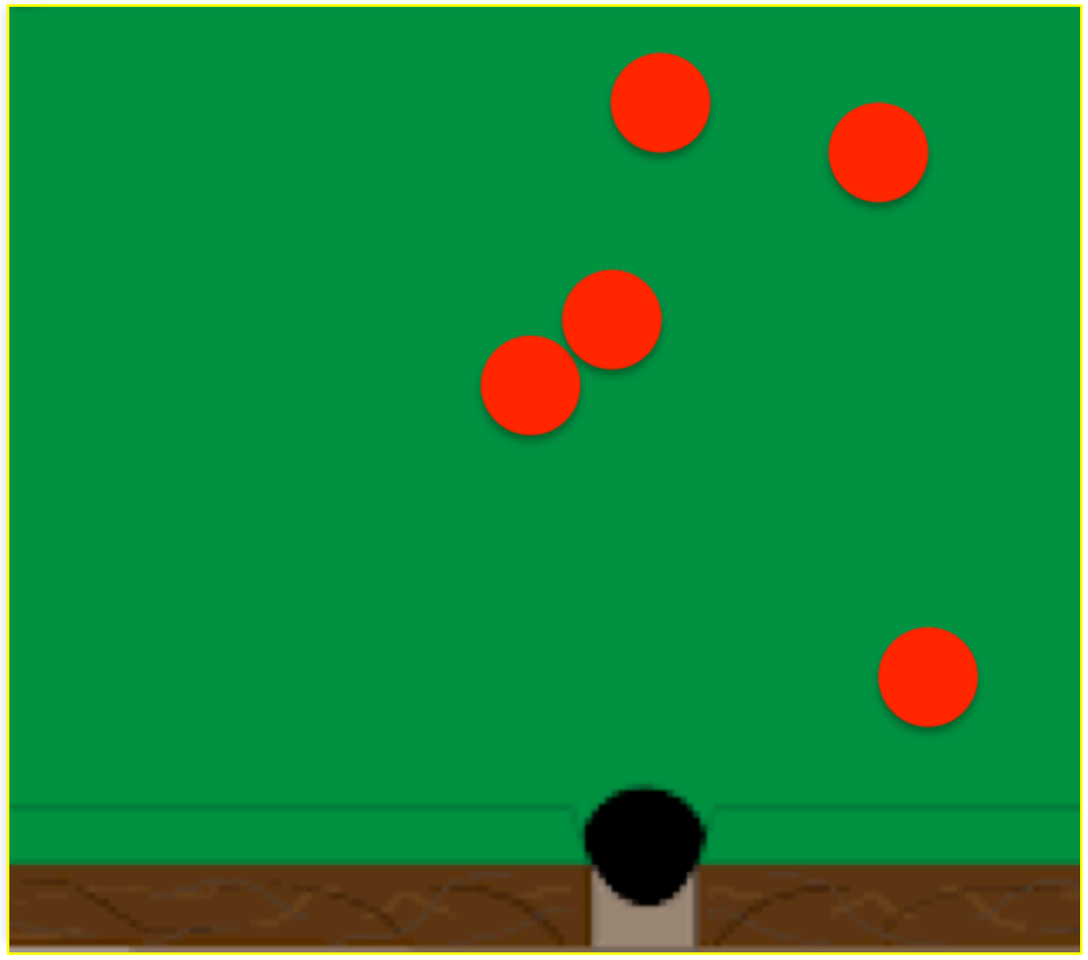
Object-Centric Prediction

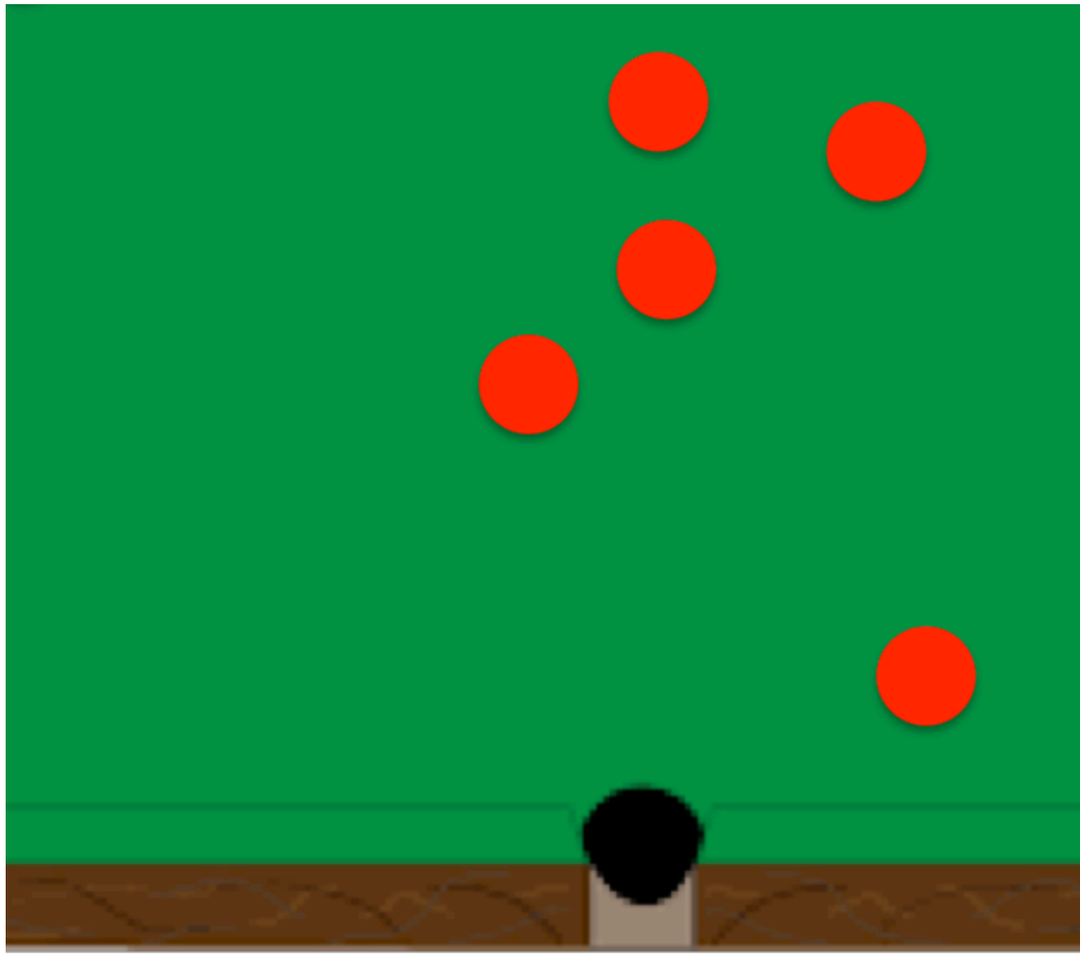
Q: will our model be able to generalize across different number of balls present?

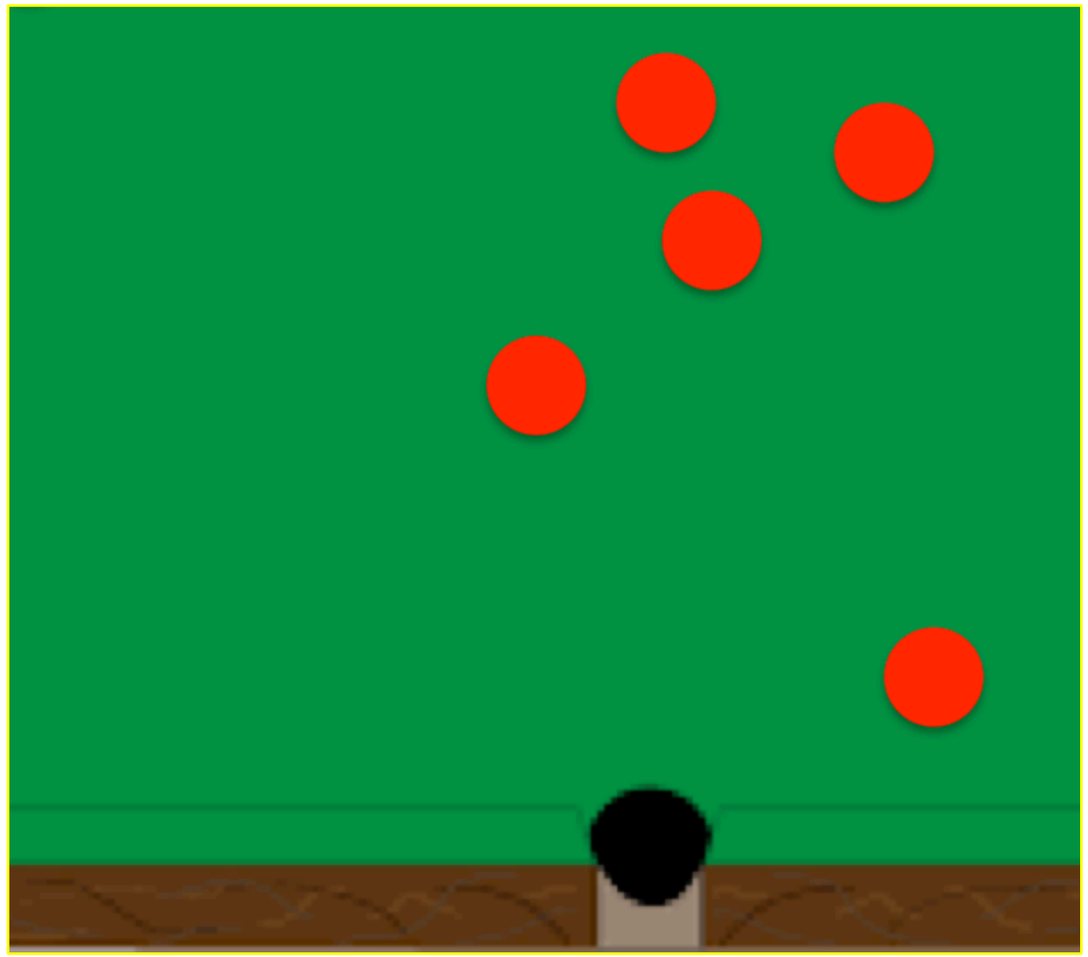




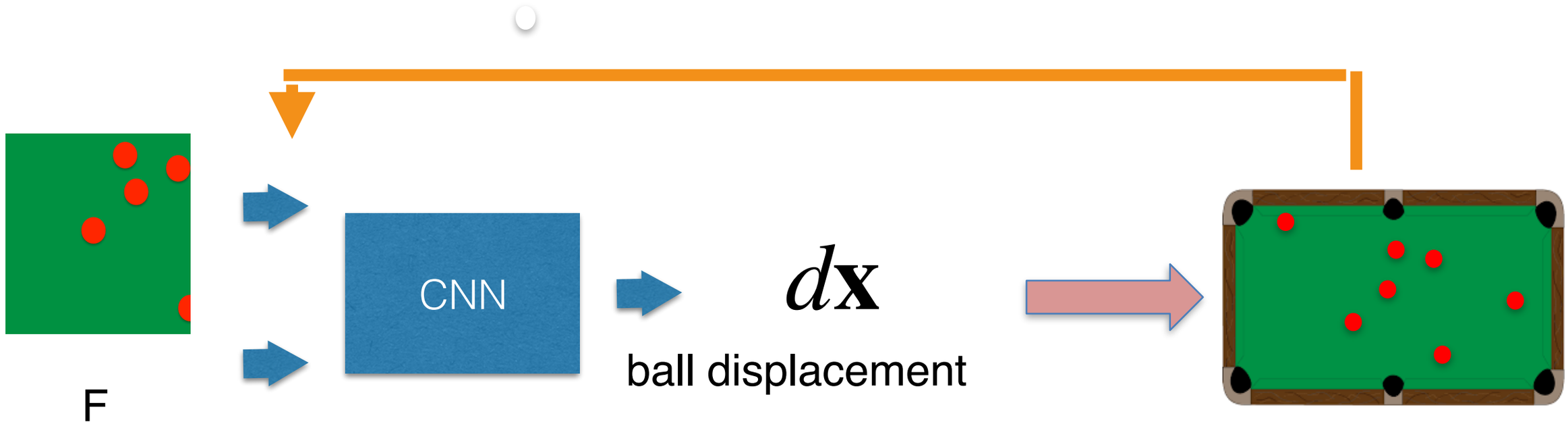




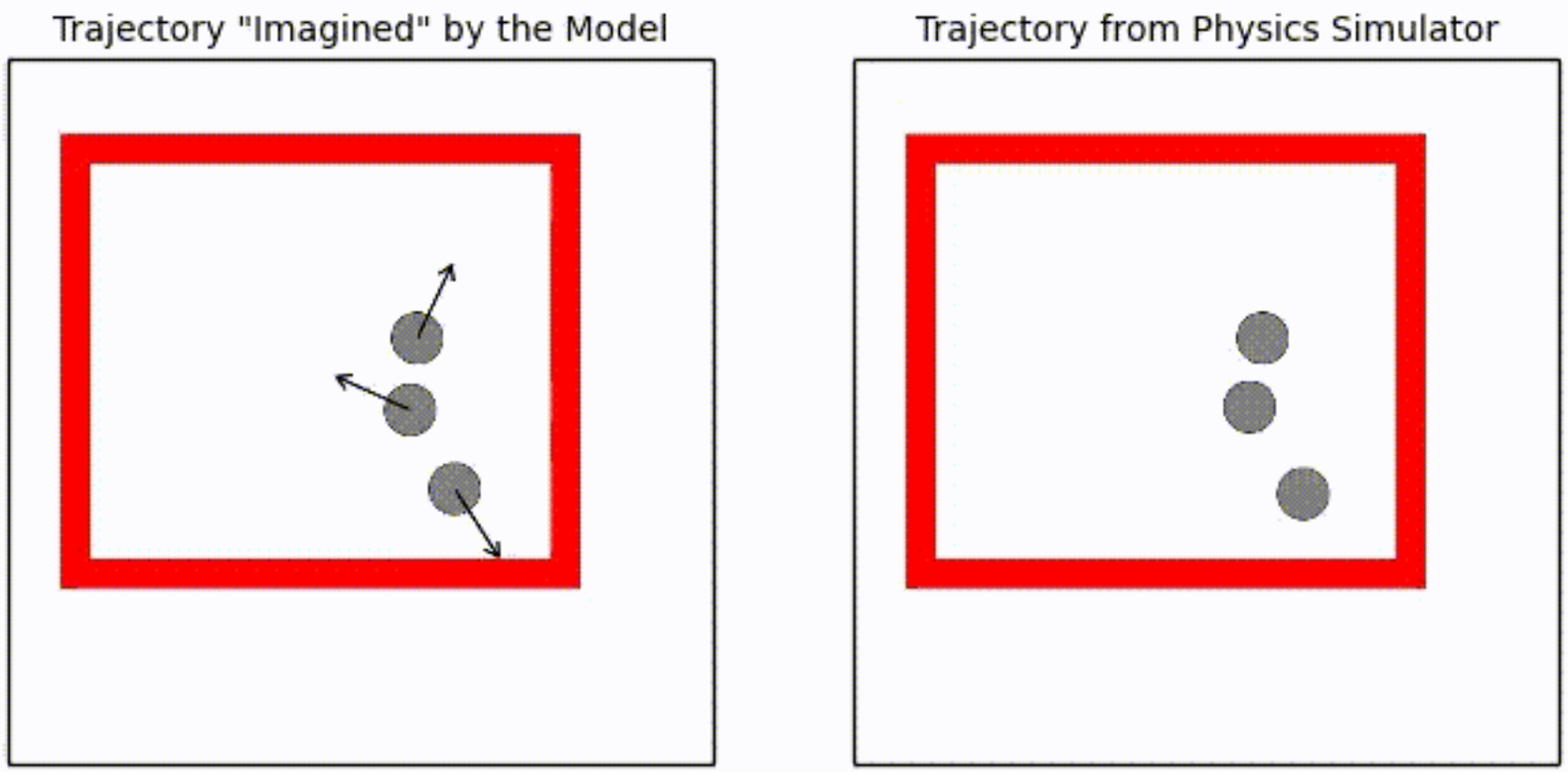


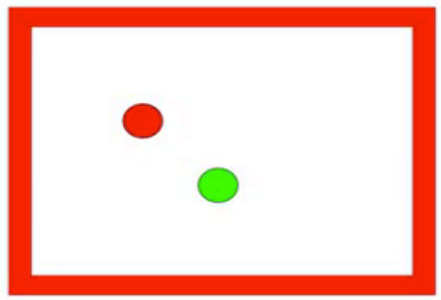


Object-centric Billiard Dynamics



- **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



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

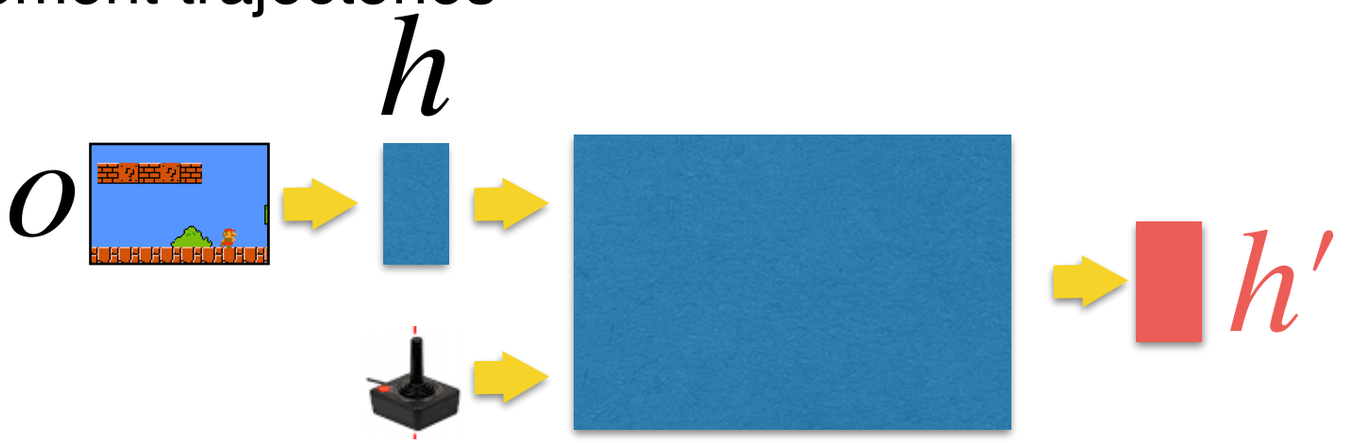
Learning Visual Dynamics

Two good ideas so far:

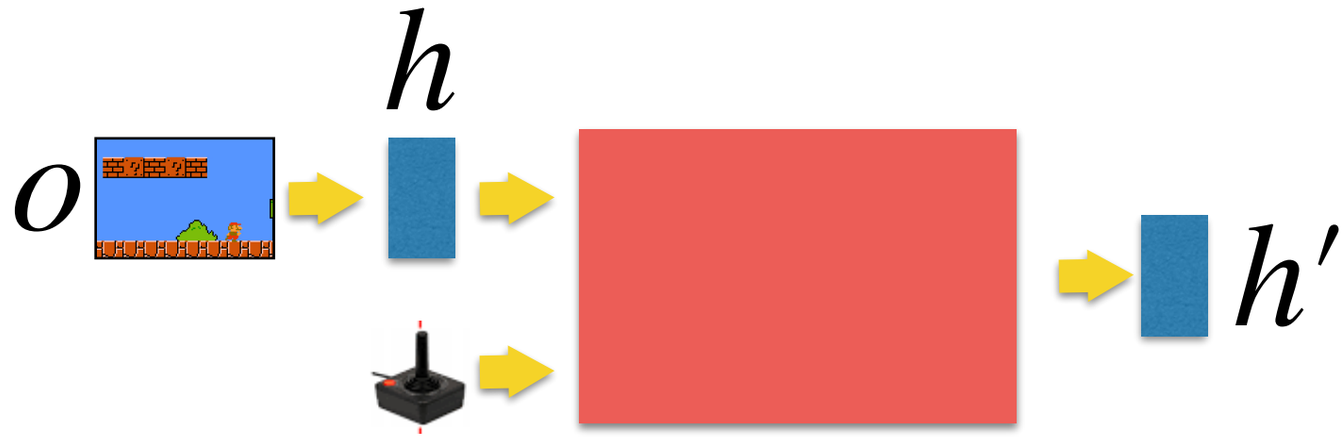
1) object graphs instead of images. Such encoding allows to generalize across different number of entities in the scene.
2) predict motion instead of appearance. Since appearance does not change, predicting motion suffices. Let's predict only the dynamic properties and keep the static one fixed.

Billiards

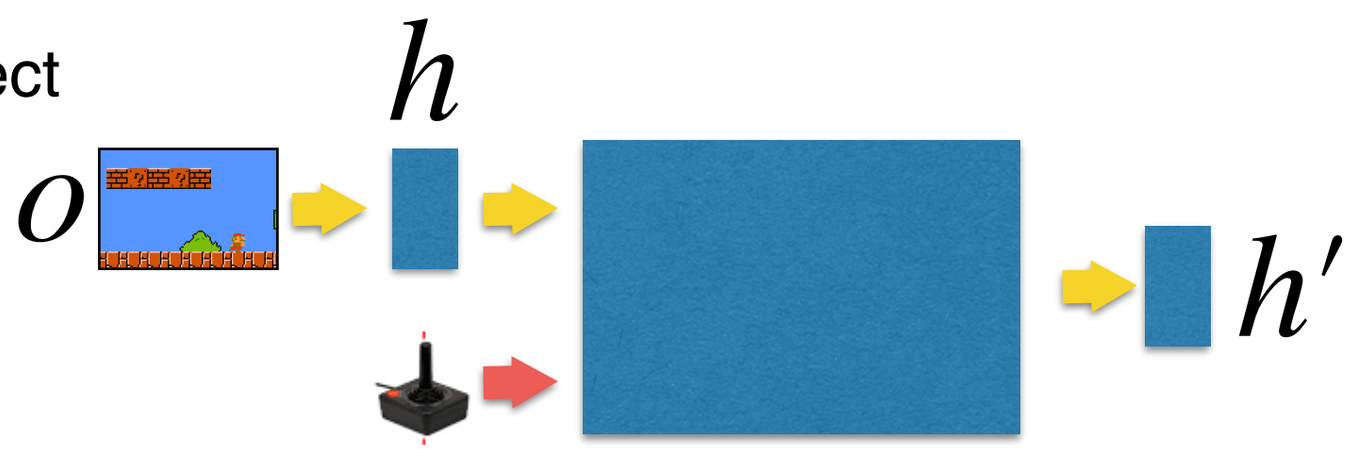
- We predicted object displacement trajectories



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

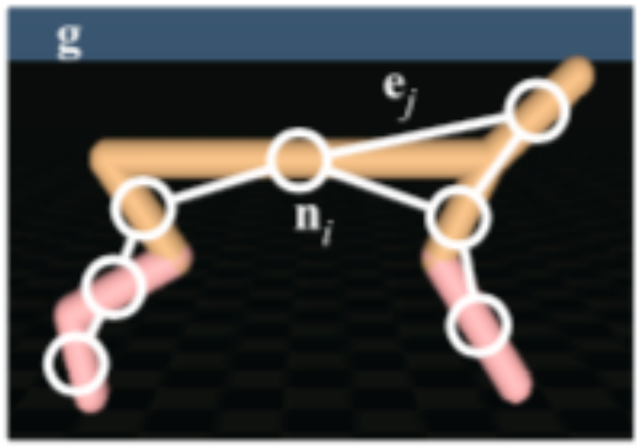


- A force applied to each object



Graph Encoding

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



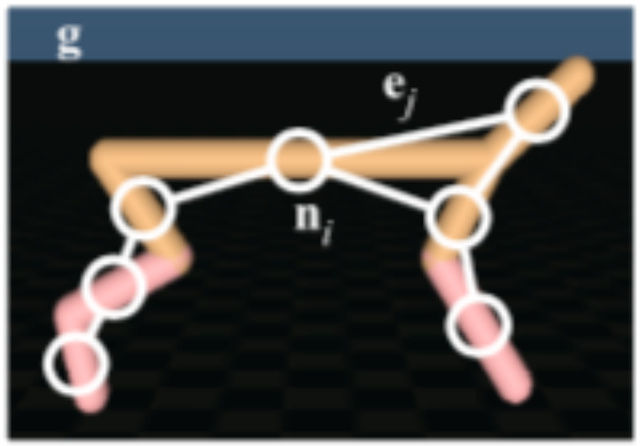
We will encode a robotic agent as a graph, where nodes are the different bodies of the agent and edges are the joints, links between the bodies

Graph neural networks!
They generalize convolutions to general graphs, as opposed to pixel grids.

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

Graph Encoding

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



Node features:

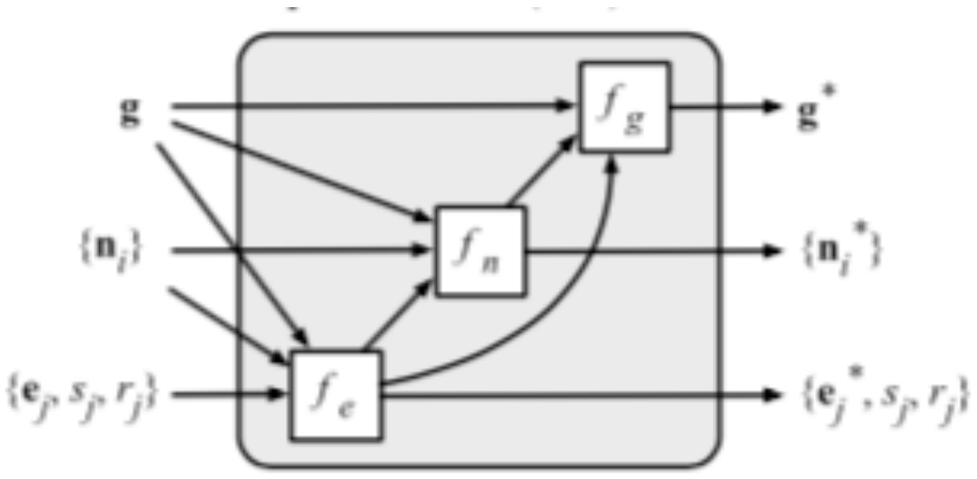
- Observable/dynamic: 3D position, 4D quaternion orientation, linear and angular velocities
- Unobservable/static: mass, inertia tensor
- Actions: forces applied on the joints

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

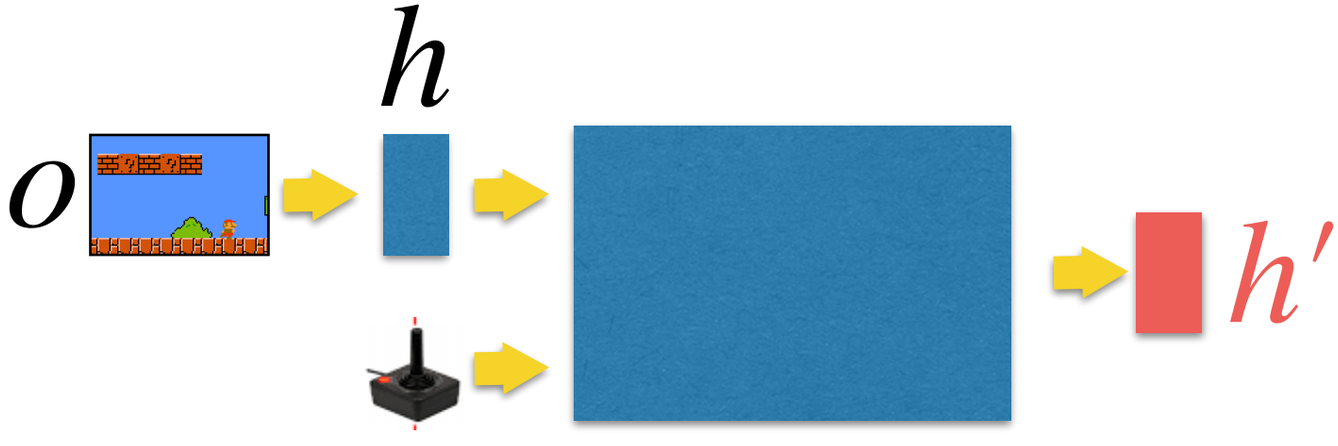
Input: Graph, $G = (\mathbf{g}, \{\mathbf{n}_i\}, \{\mathbf{e}_j, s_j, r_j\})$
for each edge $\{\mathbf{e}_j, s_j, r_j\}$ **do**
  Gather sender and receiver nodes $\mathbf{n}_{s_j}, \mathbf{n}_{r_j}$
  Compute output edges, $\mathbf{e}_j^* = f_e(\mathbf{g}, \mathbf{n}_{s_j}, \mathbf{n}_{r_j}, \mathbf{e}_j)$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

- I predict only the dynamic node features, their temporal difference.
- The node and edge computation functions are shared across all nodes and all edges! Graph convolution.
- Train with regression.

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

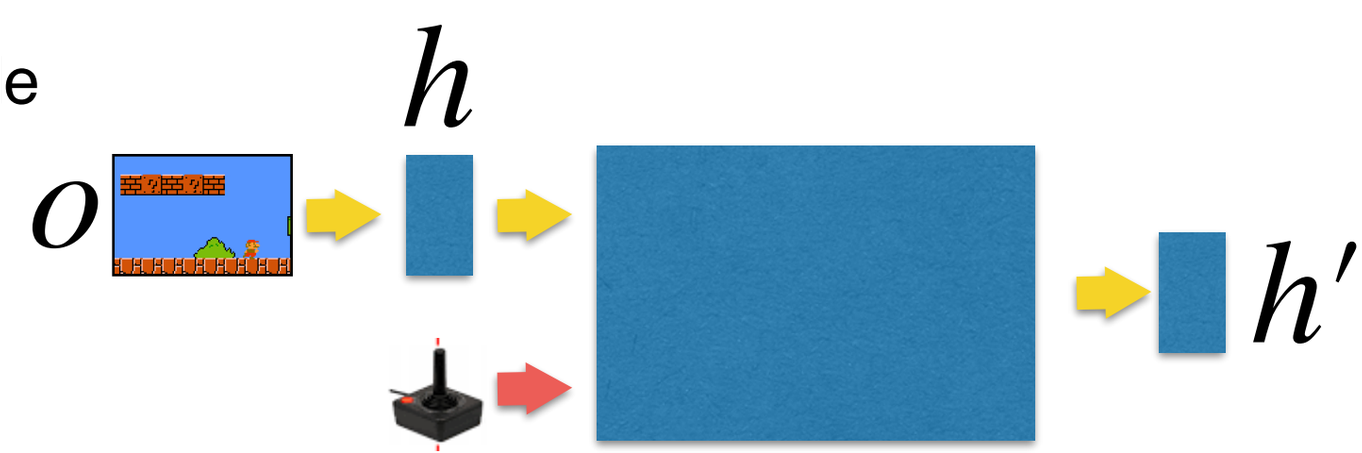
Robots as graphs

- We predicted dynamic node features



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

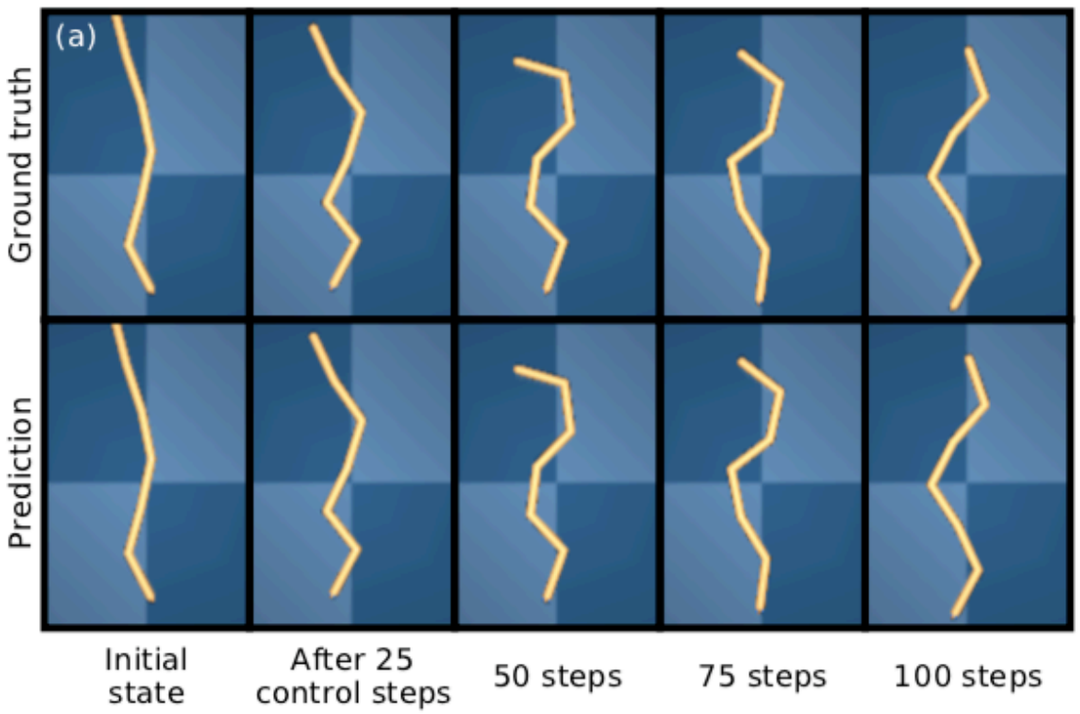
- Forces applied to each node

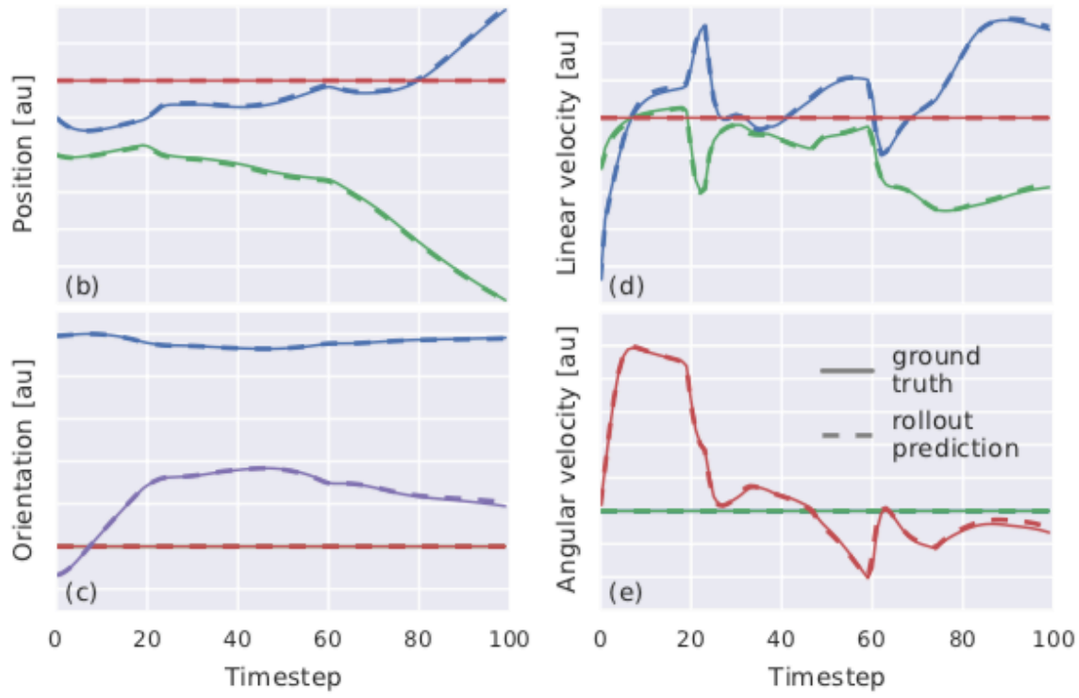


Graph Forward Dynamics

Node features

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

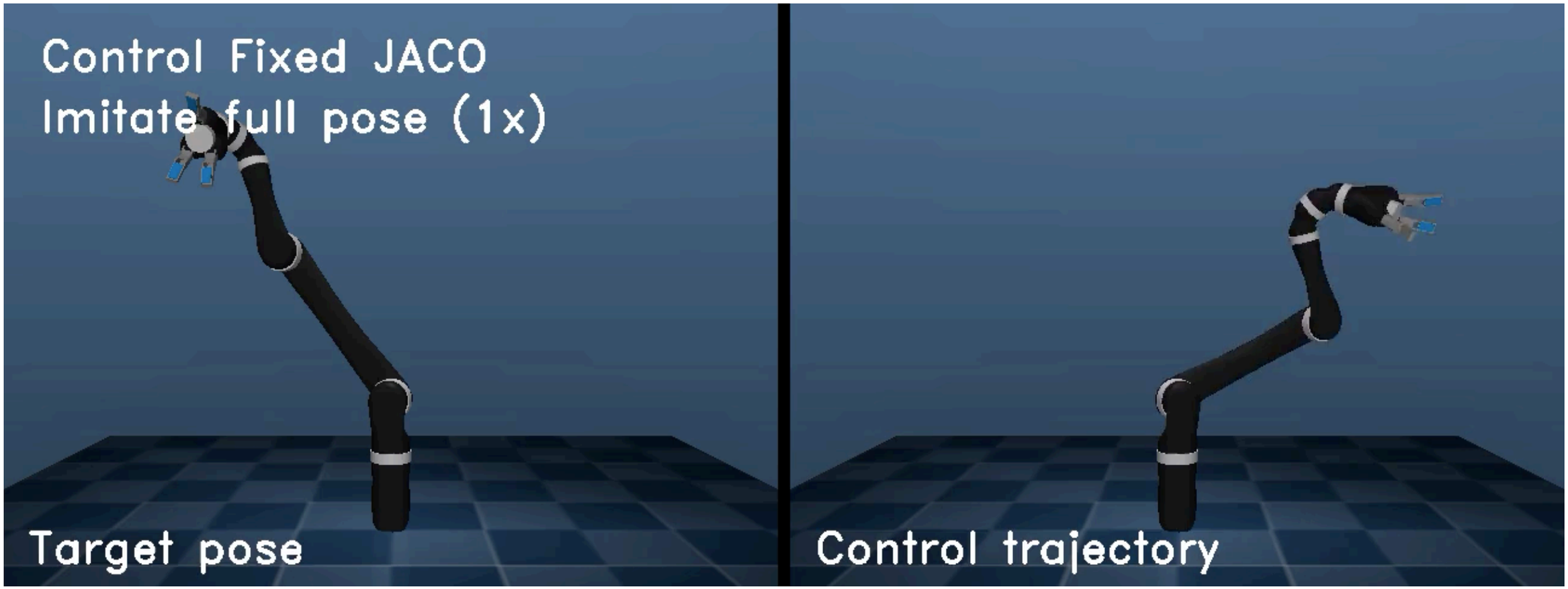




Predictions: I predict only the dynamic features, their temporal difference:

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.

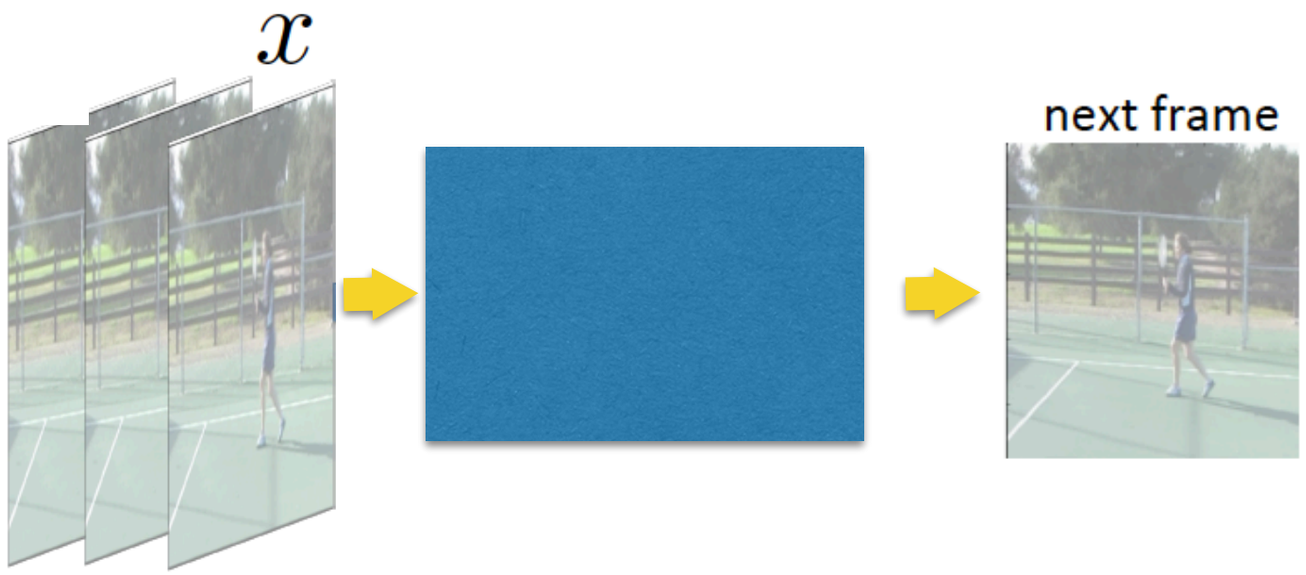
Learning Dynamics

Two good ideas so far:

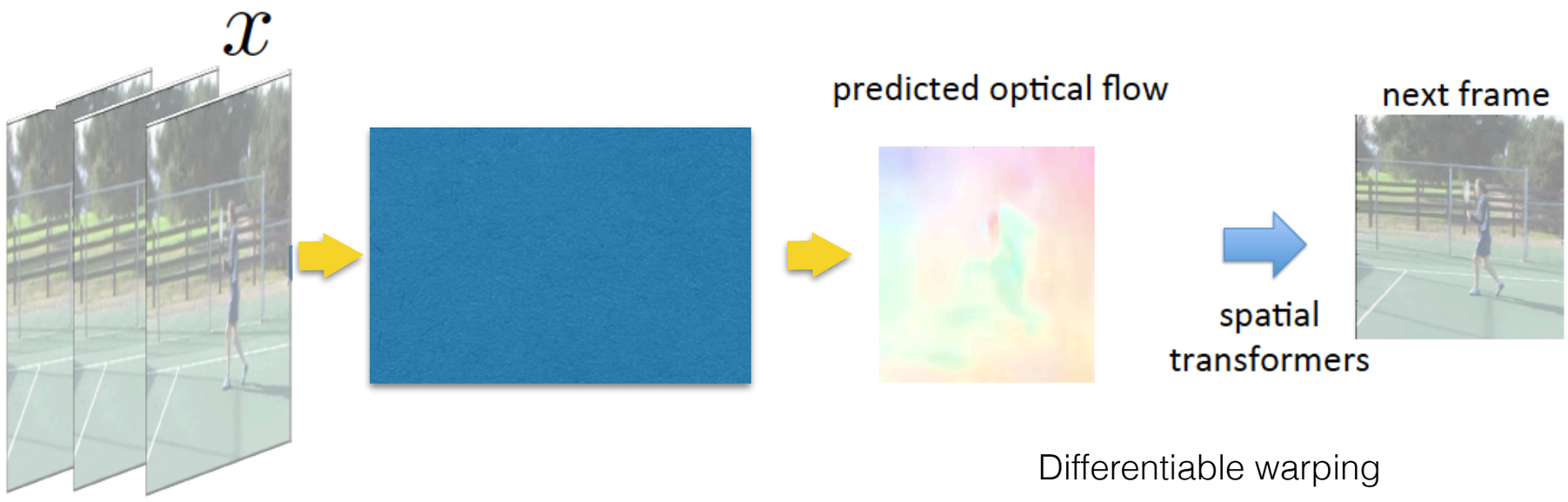
1) object graphs instead of images. Such encoding allows to generalize across different number of entities in the scene.
2) predict motion instead of appearance. Since appearance does not change, predicting motion suffices. Let's predict only the dynamic properties and keep the static one fixed.

Visual dynamics using motion transformation

Instead of predicting a next frame directly



We can predict a motion field and warp the last frame to produce the next



Differentiable warping

Q: Why this is beneficial?

Visual dynamics using motion transformation

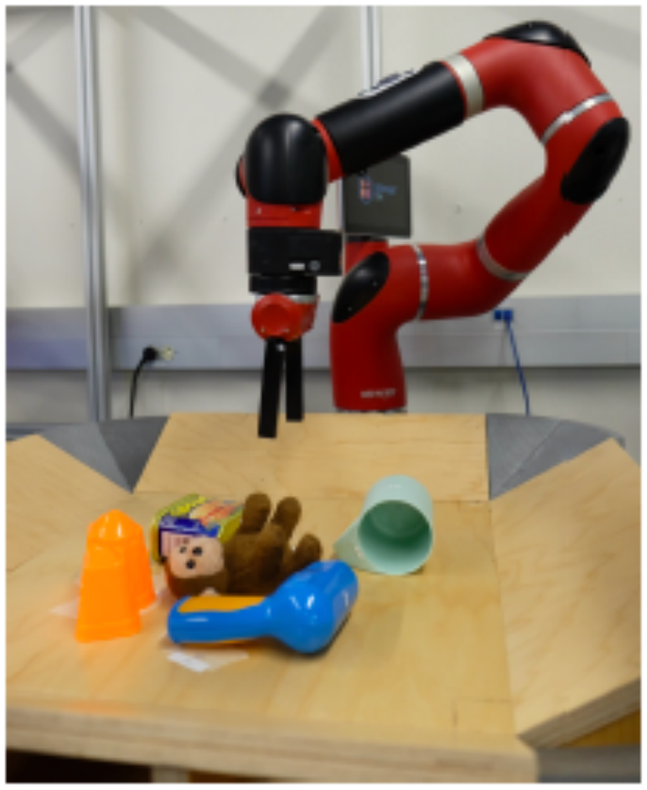


Figure 1: The robot learns to move new objects from self-supervised experience.

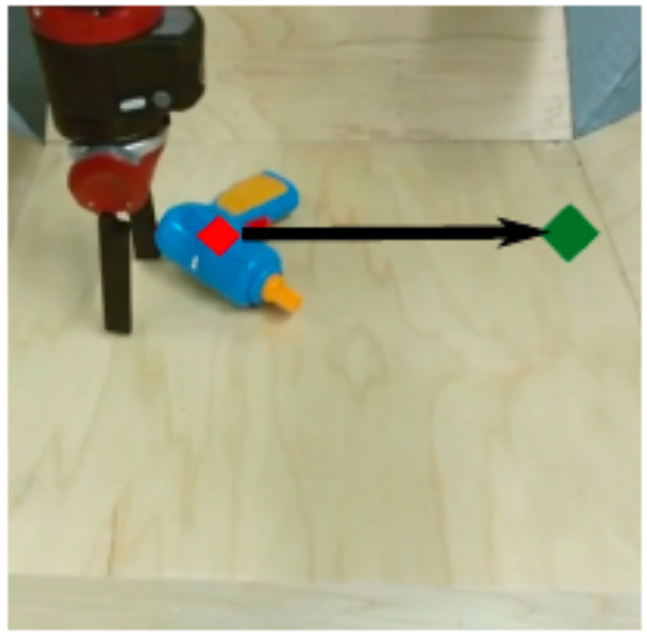


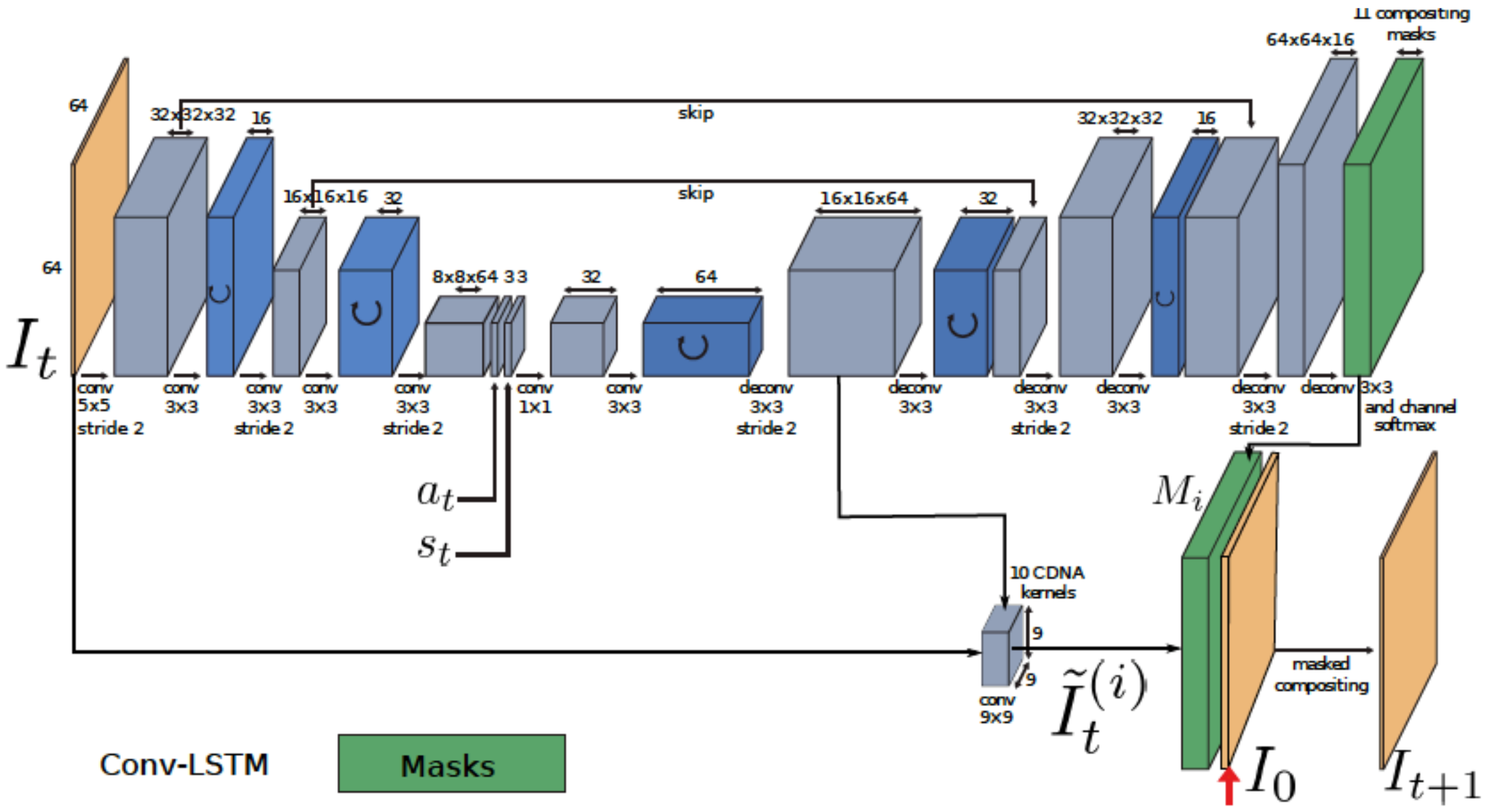
Figure 7: Pushing task. The designated pixel (red diamond) needs to be pushed to the green circle.

We will learn a model of pixel motion displacements

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

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

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

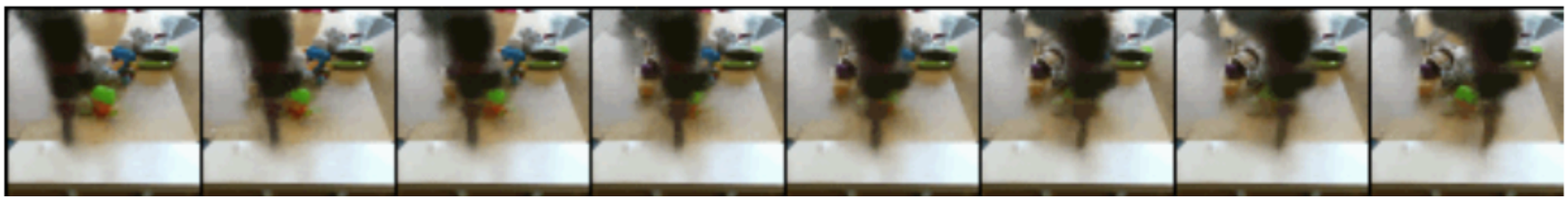
Goal representation: move certain pixels directly from the very initial image to desired locations

Temporal skip-connection! Handle occlusions by copying pixels from the very initial images, rather than the previous image.

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

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

Visual dynamics using motion transformation



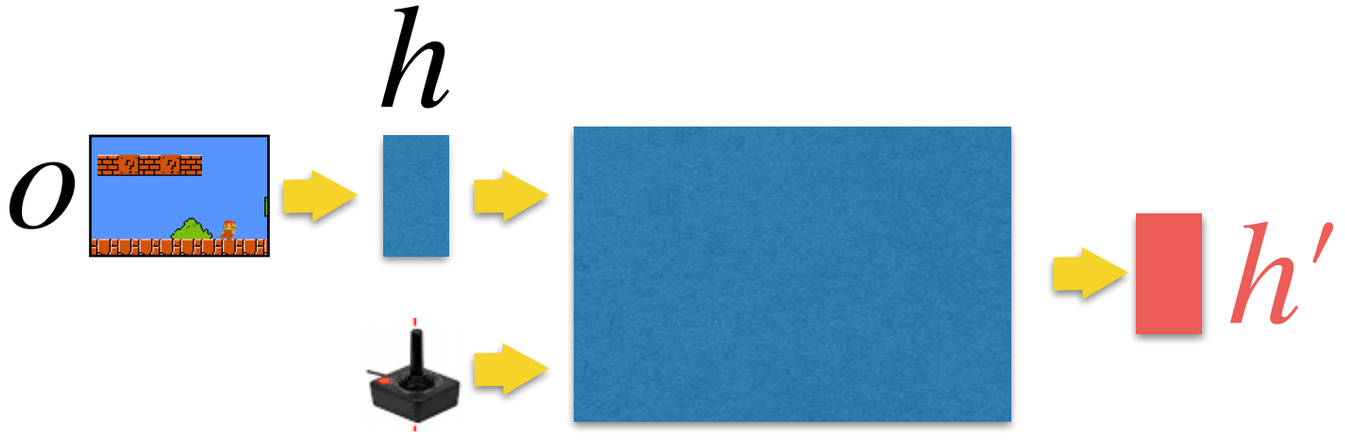
This model is used with Model predictive control to move pixels to desired locations

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

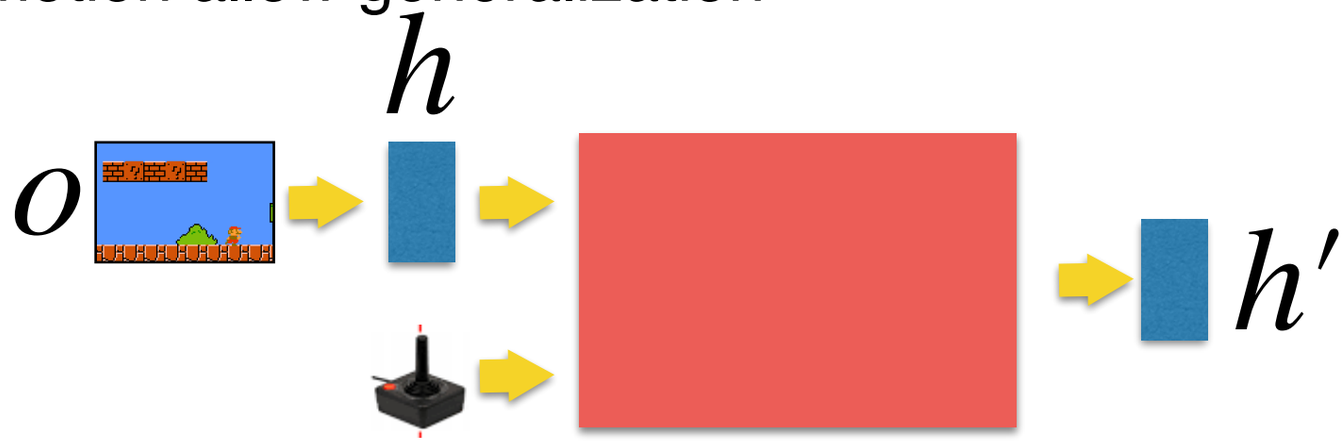
Self-Supervised Visual Planning with Temporal Skip Connections, Ebert et al.

Image prediction using temporal skip connections

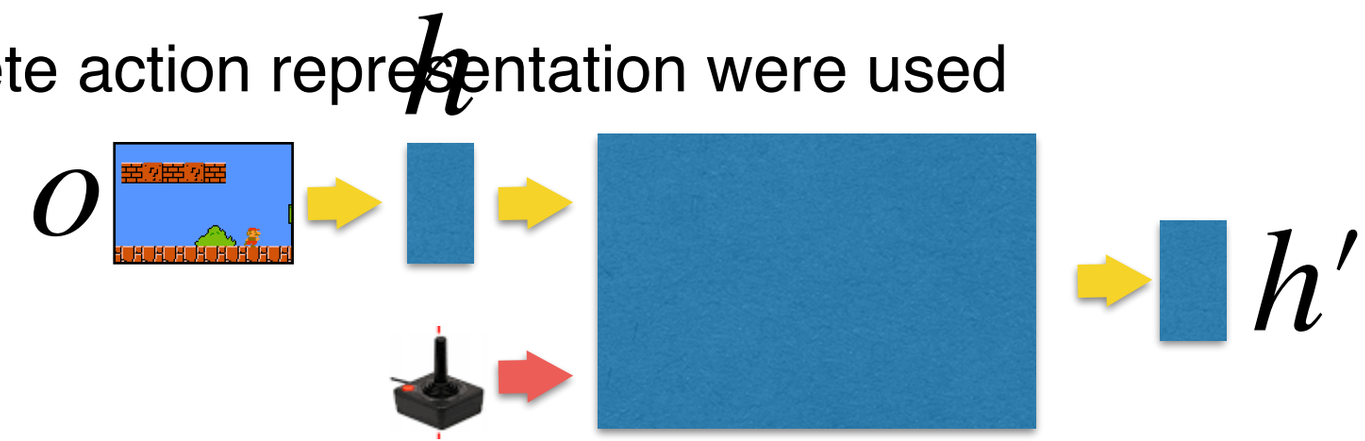
- We predicted future images



- Our model is a CNN that predicts motion masks for the initial image and right the previous image, Such motion allow generalization



- Both continuous and discrete action representation were used



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: $\mathbf{f} = (\mathbf{m}_{t+\tau_1} - \mathbf{m}_t, \cdots, \mathbf{m}_{t+\tau_n} - \mathbf{m}_t)$

What will be the future measurements for a set of future temporal instances (no unrolling)

(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

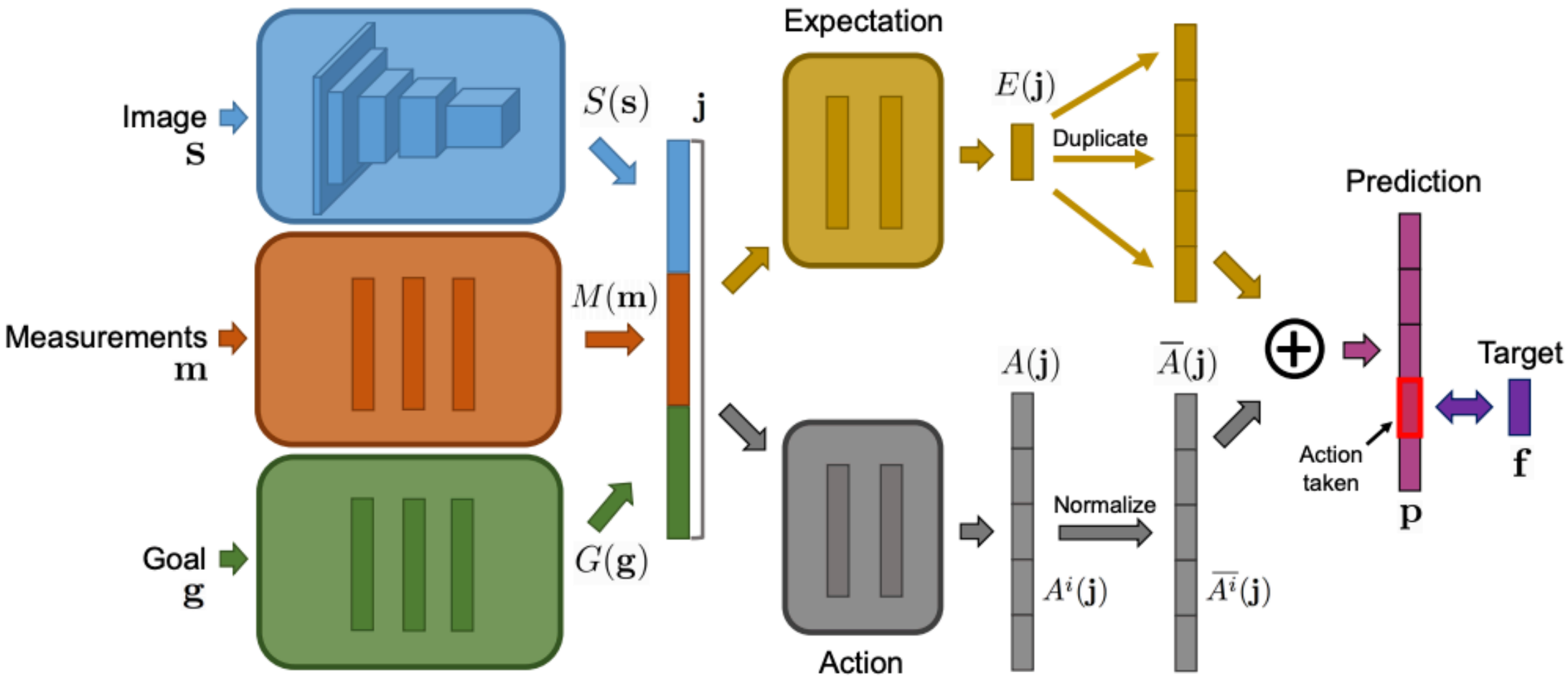
Train a deep predictor. No unrolling! One shot prediction of future values for all actions:

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

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

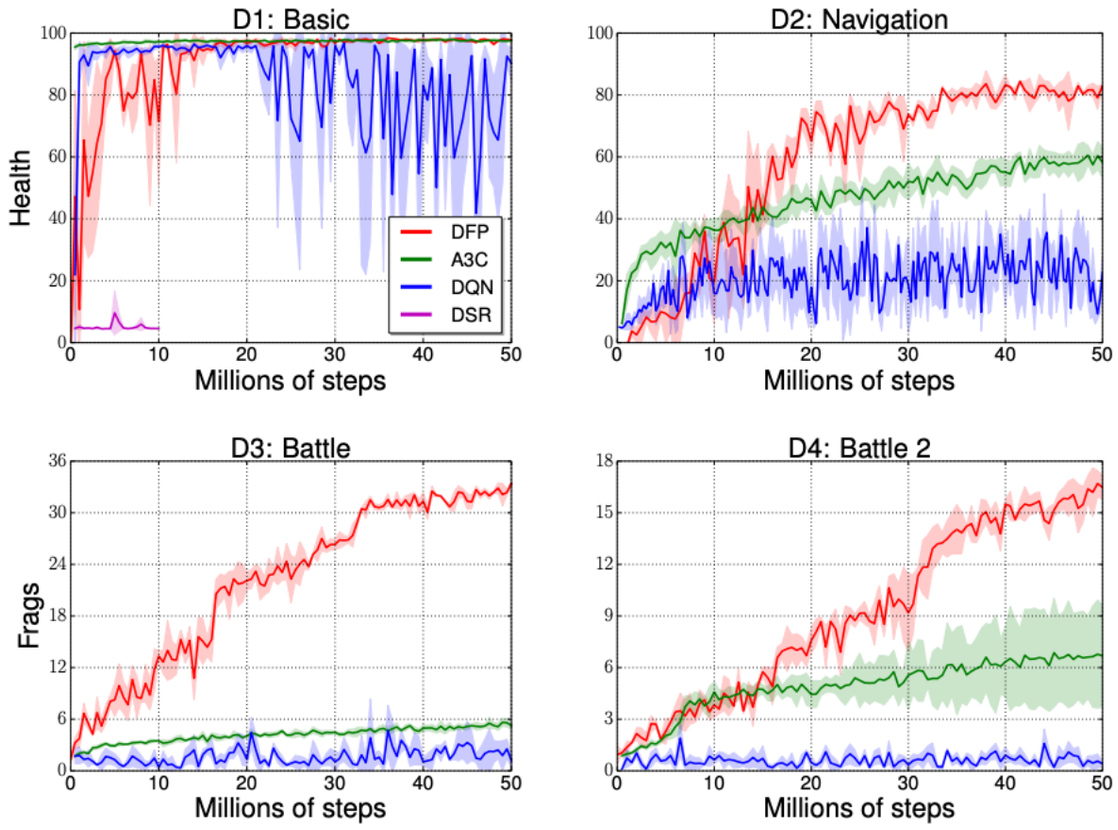


Action selection:

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

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