Carnegie Mellon

School of Computer Science

Deep Reinforcement Learning and Control

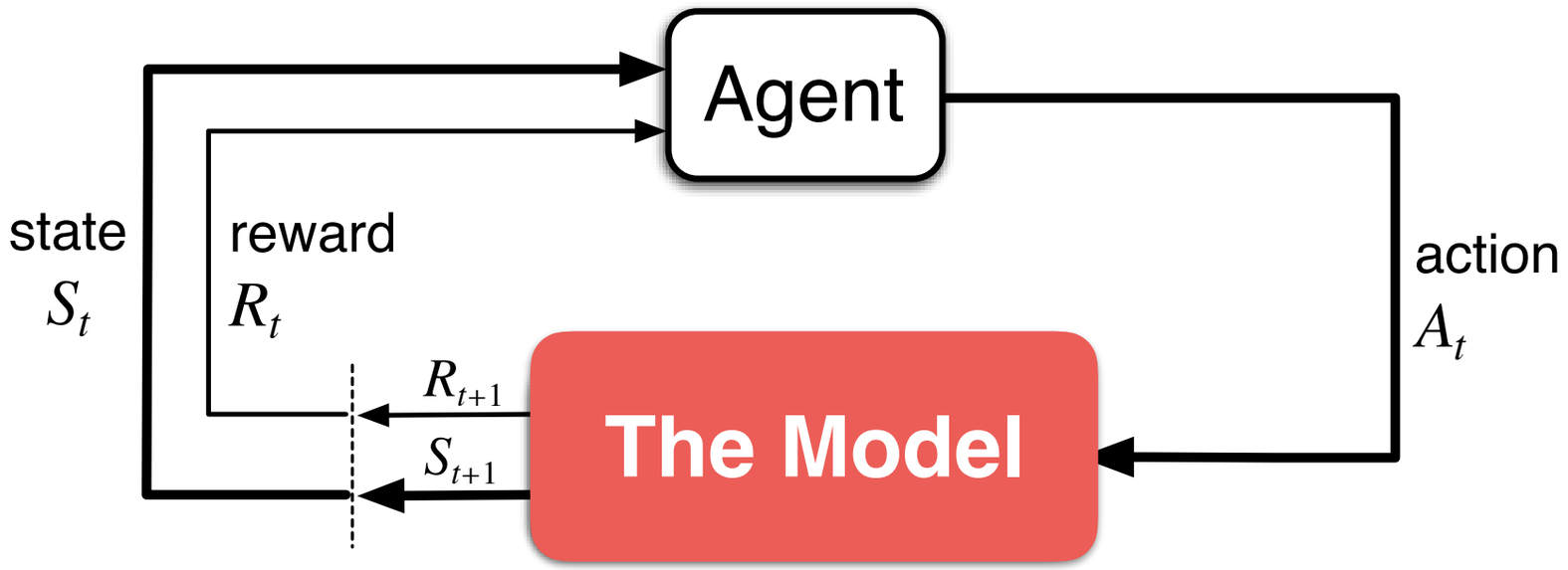
Model Based Reinforcement Learning in the sensory space II

Spring 2020, CMU 10-403

Katerina Fragkiadaki

Model based RL

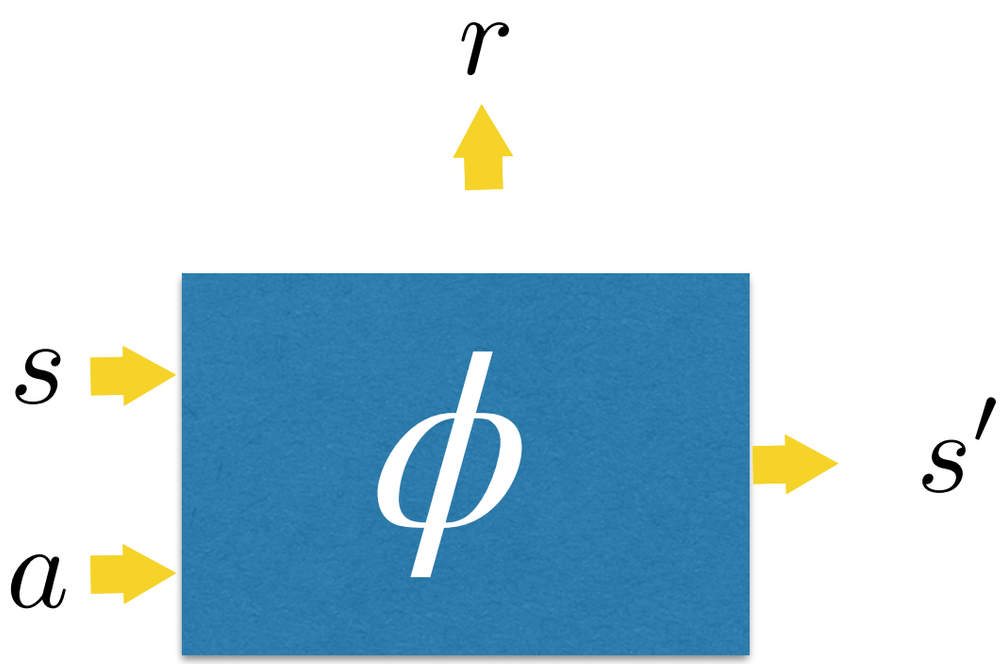
We want to learn to look ahead into the future and predict consequences of our actions. We want to use such look-ahead ability to select actions.



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

Model learning

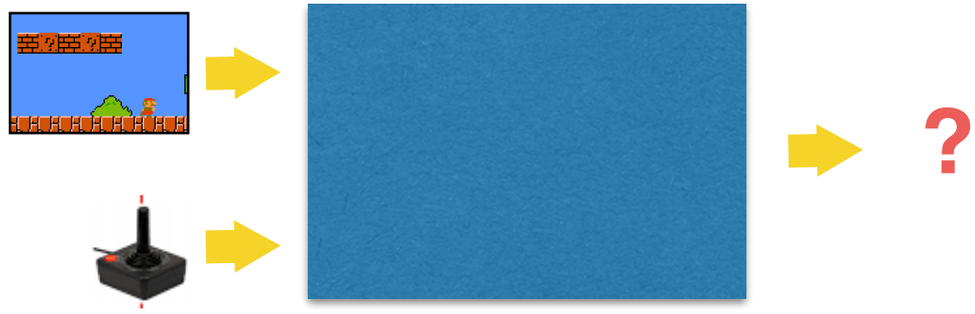
We will be learning the model using experience tuples. A supervised learning problem.



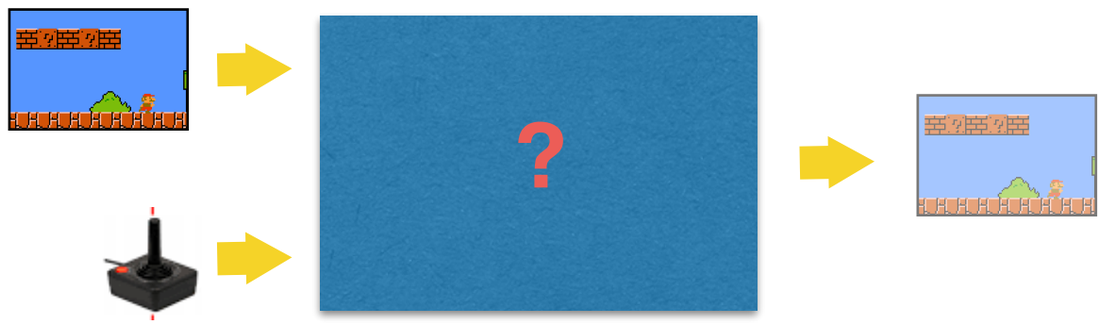
gaussian process,
random forest, deep
neural network,
linear function

Model Learning - 3 Qs always in mind

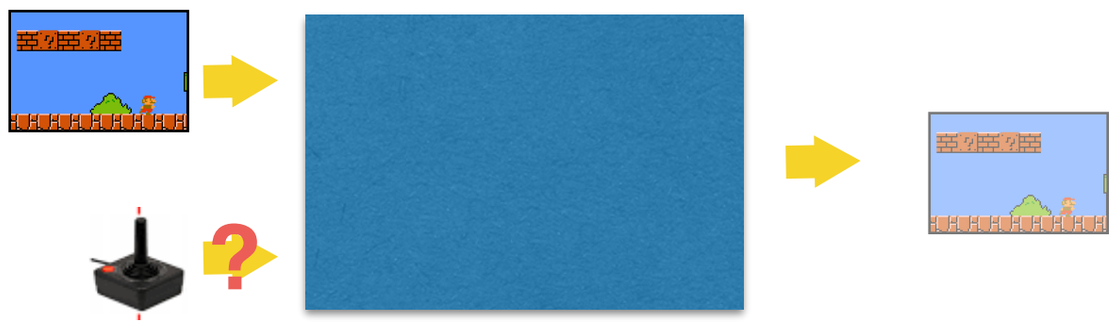
- What shall we be predicting?



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

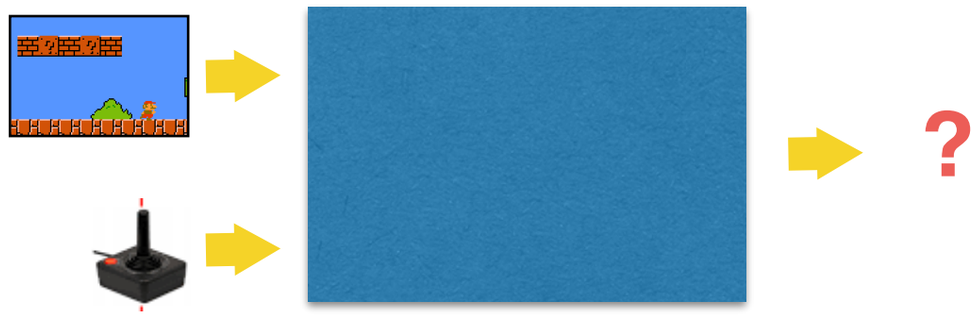


- What is the action representation?



Model Learning - 3 Qs always in mind

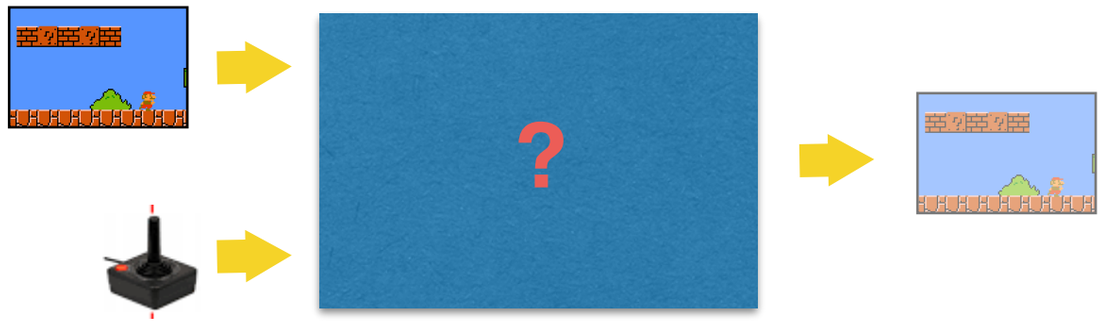
- What shall we be predicting?



Answers:
1.Future images (WXHX3 matrices)
2.Future images + Rewards
3.Object motions (in 2D or 3D)
4.Image abstractions
5.Object abstractions

Model Learning - 3 Qs always in mind

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



Answers:
1.CNNs
2.Object-centric CNNs
3.Graph neural nets

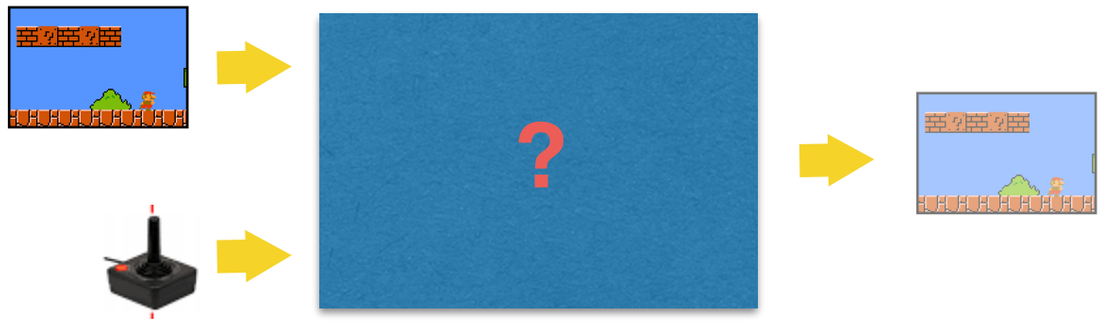
Challenges in model learning in sensory space

- Predicting images is hard:
  - Distributions over images are very multimodal images are high dimensional, in practice it is a very hard problem.
  - How things look like (image regression loss) may be quite irrelevant to the task at hand

- CNNs cannot effectively generalize under varying number of objects and configurations. What architecture we should be using for combinatorial generalization? (more objects, different arrangements)

Model Learning - 3 Qs always in mind

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



Answers:

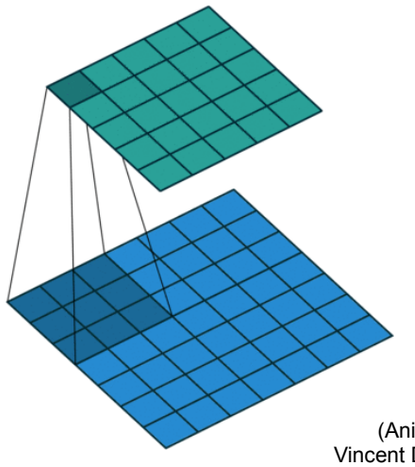
1.CNNs

2.Object-centric CNNs

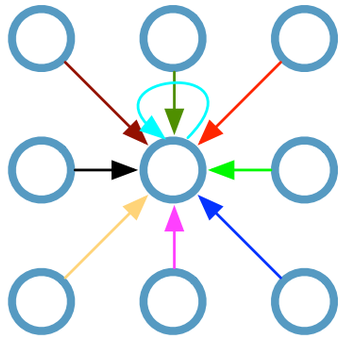
3.Graph neural nets

From CNNs to Graph neural networks (GNNs)

Single CNN layer with 3x3 filter:

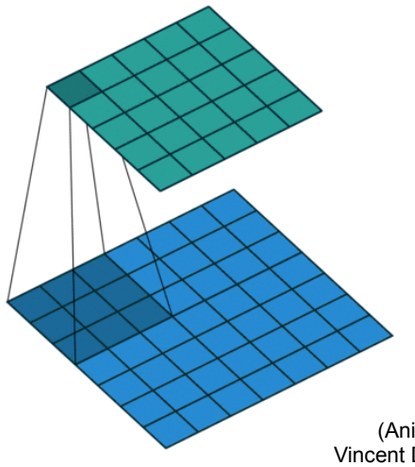


(Animation by Vincent Dumoulin)

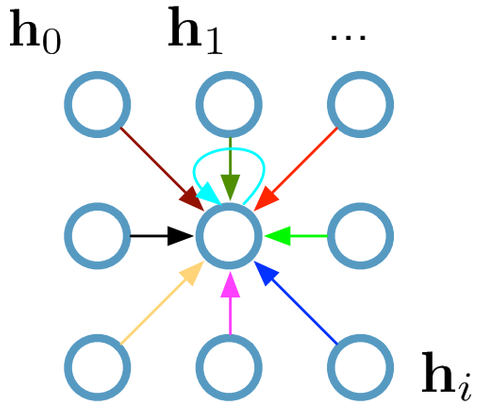


From CNNs to GNNs

Single CNN layer with 3x3 filter:

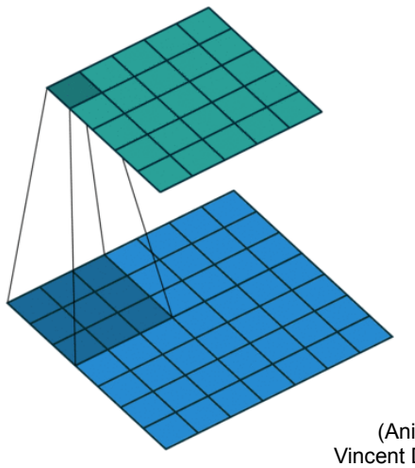


(Animation by Vincent Dumoulin)

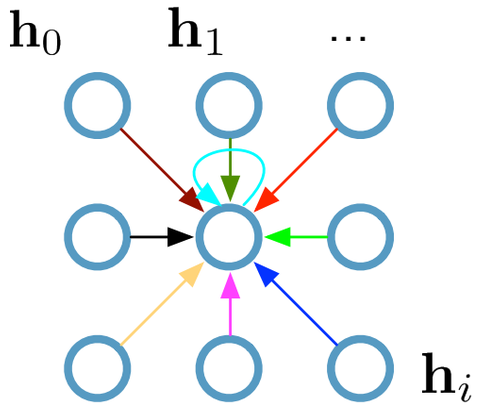


From CNNs to GNNs

Single CNN layer with 3x3 filter:



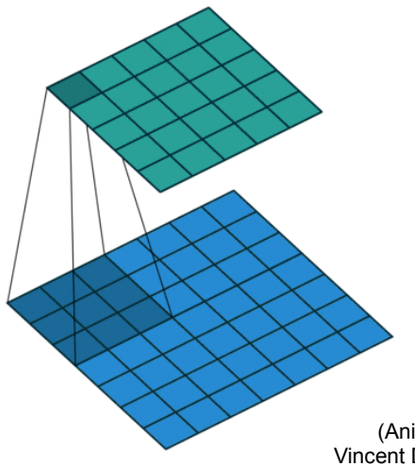
(Animation by Vincent Dumoulin)



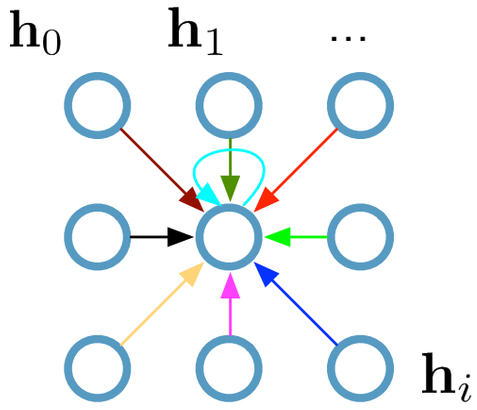
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

From CNNs to GNNs

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



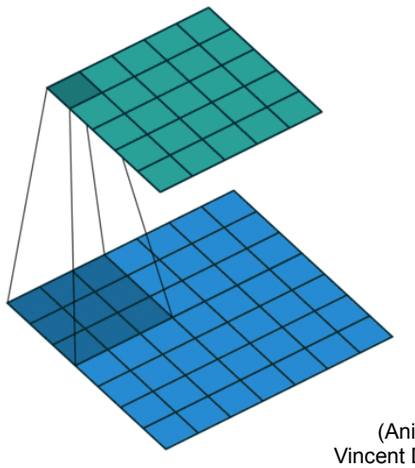
Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

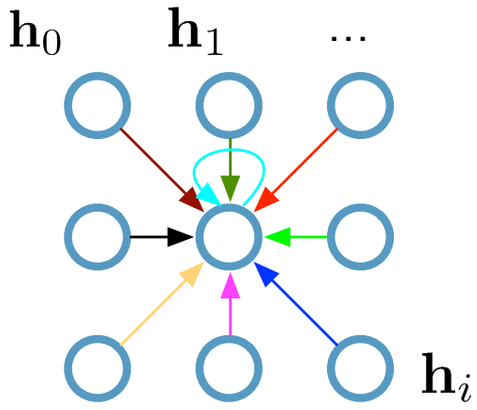
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

From CNNs to GNNs

Single CNN layer with 3x3 filter:



(Animation by Vincent Dumoulin)



Update for a single pixel:

- Transform messages individually $\mathbf{W}_i\mathbf{h}_i$
- Add everything up $\sum_i \mathbf{W}_i\mathbf{h}_i$

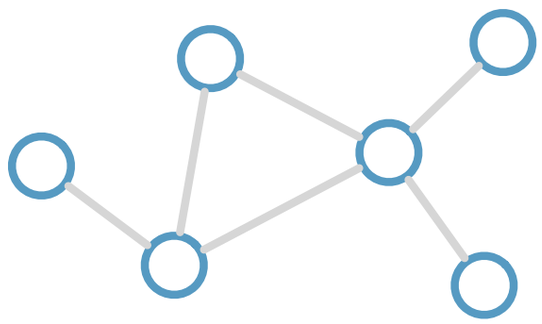
$\mathbf{h}_i \in \mathbb{R}^F$ are (hidden layer) activations of a pixel/node

Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

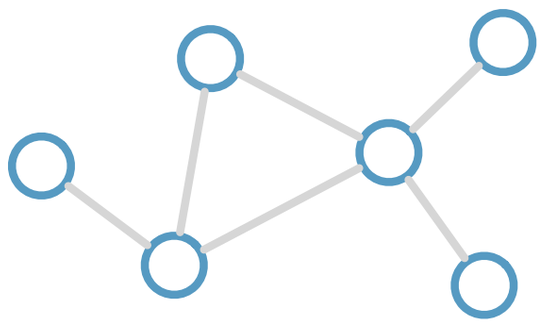
From CNNs to GNNs

Consider this
undirected graph:

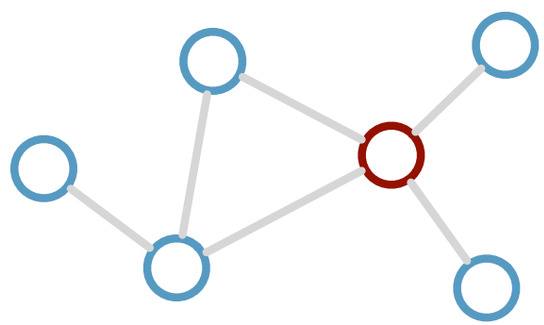


From CNNs to GNNs

Consider this undirected graph:

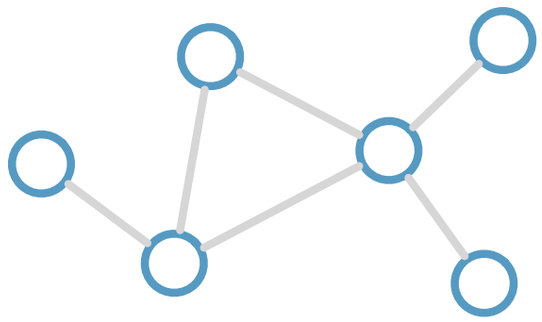


Calculate update for node in red:

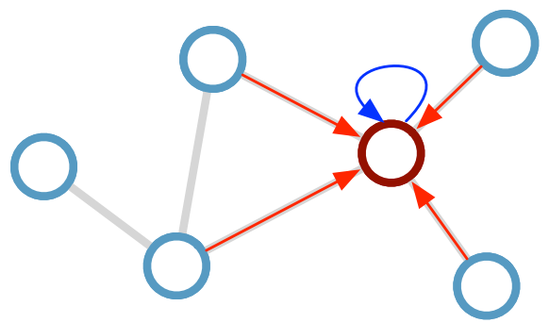


From CNNs to GNNs

Consider this undirected graph:

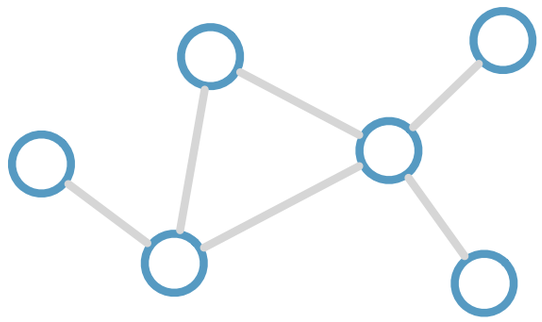


Calculate update for node in red:

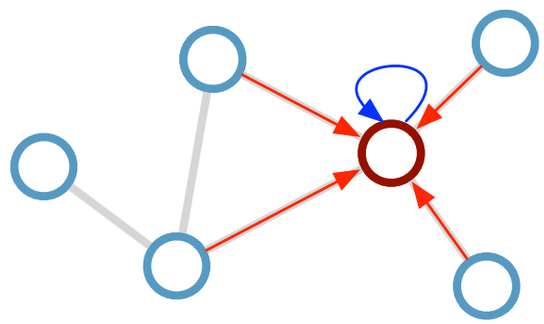


From CNNs to GNNs

Consider this undirected graph:



Calculate update for node in red:



Update rule:

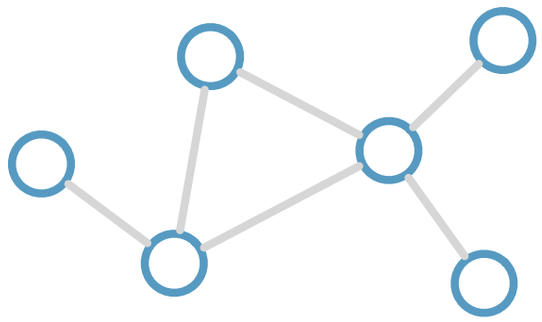
$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

$\mathcal{N}_i$: neighbor indices

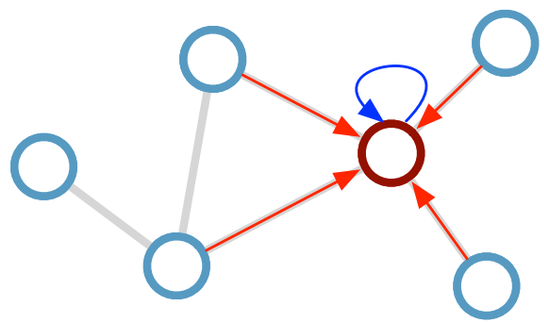
c_{ij} : norm. constant (fixed/trainable)

From CNNs to GNNs

Consider this undirected graph:



Calculate update for node in red:



Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

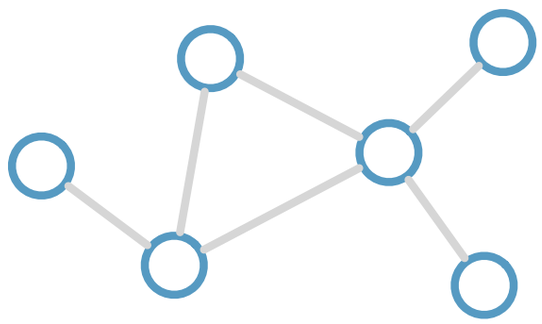
Scalability: subsample messages [Hamilton et al., NIPS 2017]

$\mathcal{N}_i$: neighbor indices

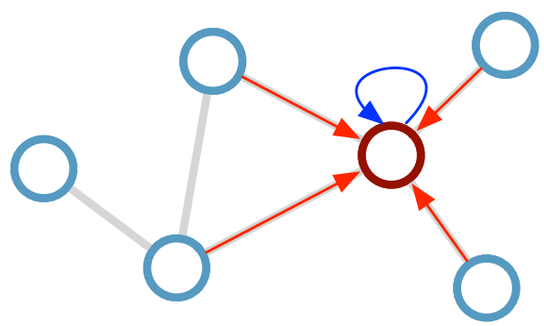
c_{ij} : norm. constant (fixed/trainable)

From CNNs to GNNs

Consider this undirected graph:



Calculate update for node in red:



Desirable properties:

- Weight sharing over all locations
- Invariance to permutations
- Linear complexity O(E)

Update rule:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(l)}\mathbf{W}_1^{(l)}\right)$$

Scalability: subsample messages [Hamilton et al., NIPS 2017]

$\mathcal{N}_i$: neighbor indices

c_{ij} : norm. constant (fixed/trainable)

Graph Neural Networks

To represent:

- Scenes of objects or object parts
- Robot bodies
- Scenes of objects or object parts and robot bodies

Interaction Networks for Learning about Objects, Relations and Physics, Battaglia et al., 2016

Interaction Networks for Learning about Objects, Relations and Physics

Peter W. Battaglia
Google DeepMind
London, UK N1C 4AG
peterbattaglia@google.com

Razvan Pascanu
Google DeepMind
London, UK N1C 4AG
razp@google.com

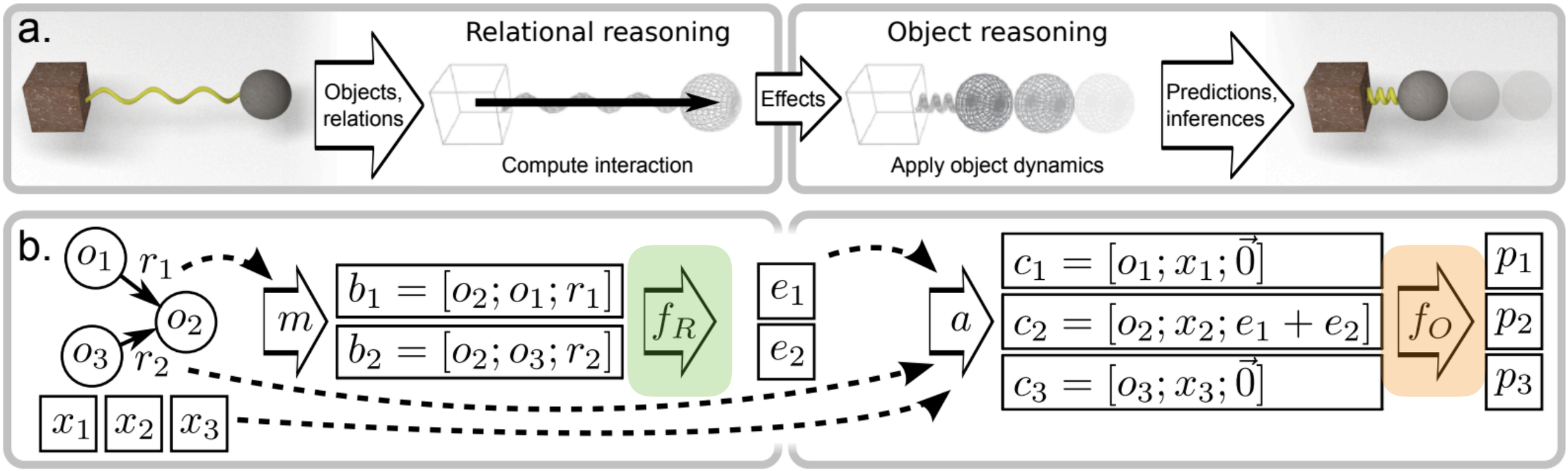
Matthew Lai
Google DeepMind
London, UK N1C 4AG
matthewlai@google.com

Danilo Rezende
Google DeepMind
London, UK N1C 4AG
danilor@google.com

Koray Kavukcuoglu
Google DeepMind
London, UK N1C 4AG
korayk@google.com

Main idea: Given a set of objects or object parts, use graph neural networks to predict their future velocities, given their physical properties and current positions and velocities

Interaction Networks for Learning about Objects, Relations and Physics, Battaglia et al., 2016



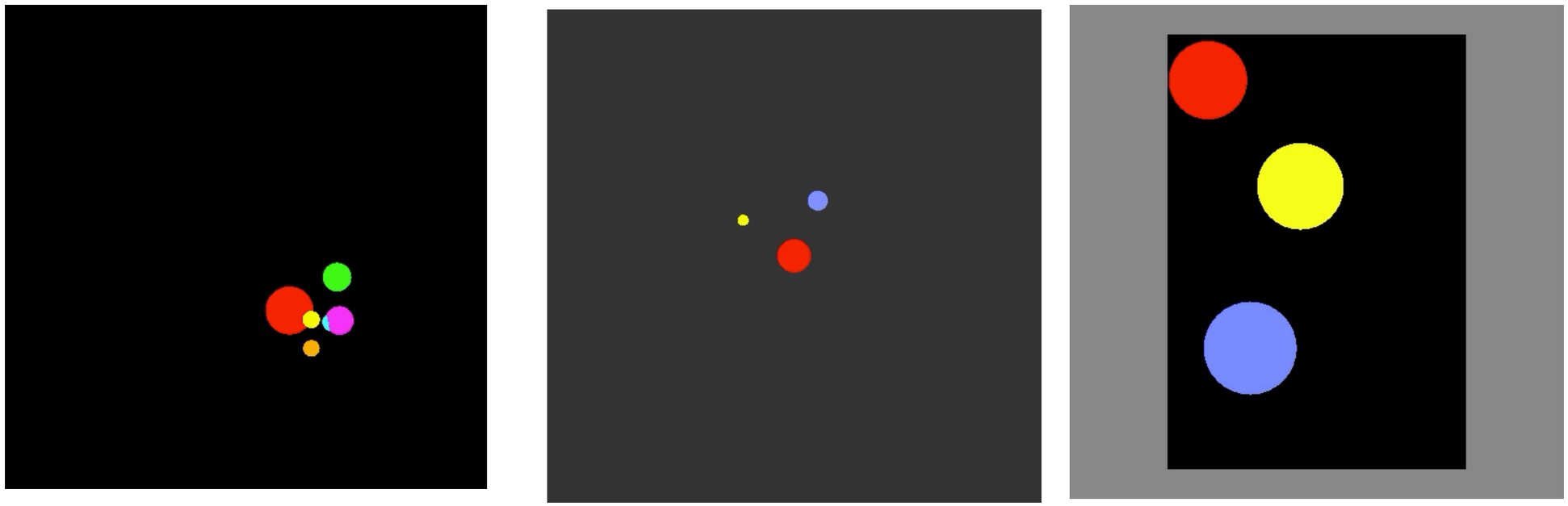
- Input:
  - Object state: dynamic (position/velocity), static(mass, size, shape)-> assumed given
  - Relation attributes: coefficient of restitution, spring constant
- Output: the velocities of the objects in the next time step.

- Relational (edge) neural net: takes two object states as input and relational attributes and predicts a feature vector
- Object (node) neural net: takes object states and summation of incoming edge messages and predicts future object velocity

- Can be used for unrolling by feeding the predictions back as input
- It does not work with high sensory input directly :-(

Interaction Networks for Learning about Objects, Relations and Physics, Battaglia et al., 2016

Unrolling results



Interaction Networks for Learning about Objects, Relations and Physics, Battaglia et al., 2016

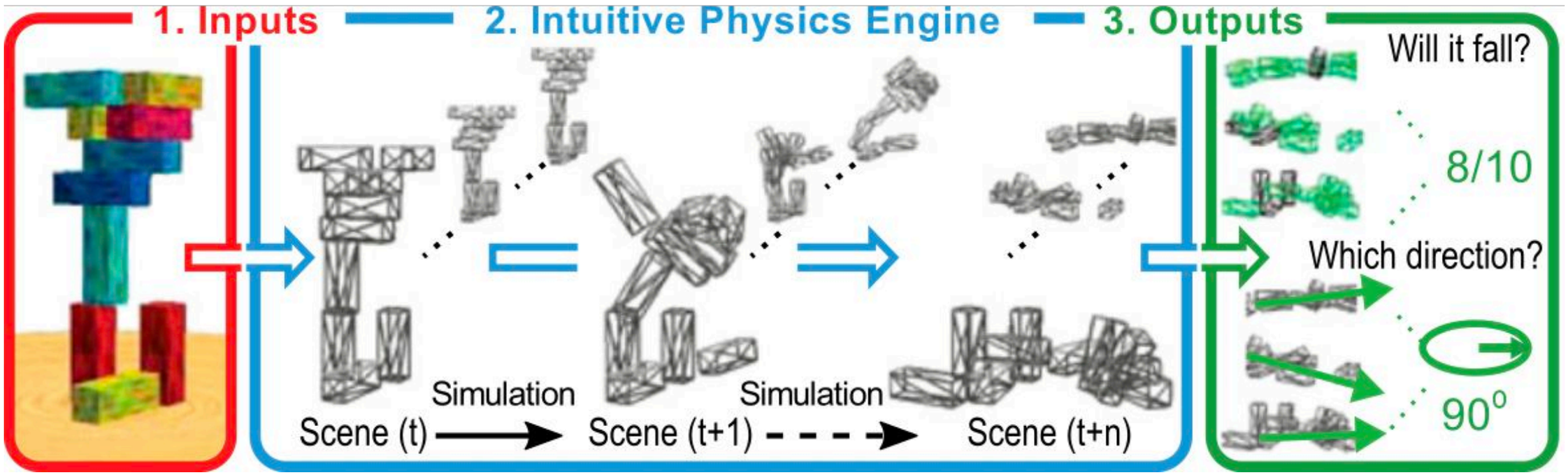
From CNNs to GNNs

To handle high-dimensional sensory input, an object detector needs to detect the objects and and instantiate the nodes, and infer their static and dynamic properties

object detection

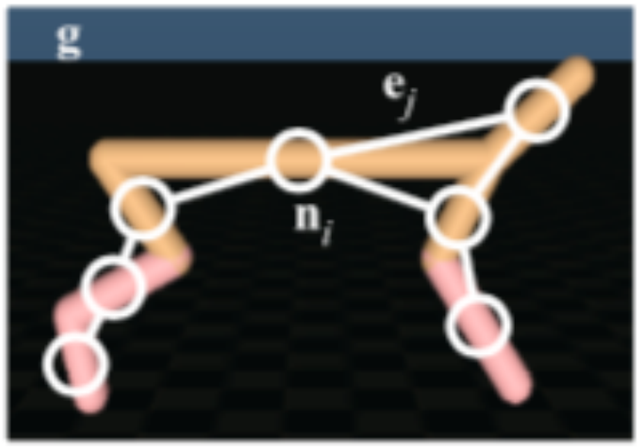
object GNN

motion predictions



Robots as graphs

A physical system's bodies and joints can be represented by a graph's nodes and edges.



Node features

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

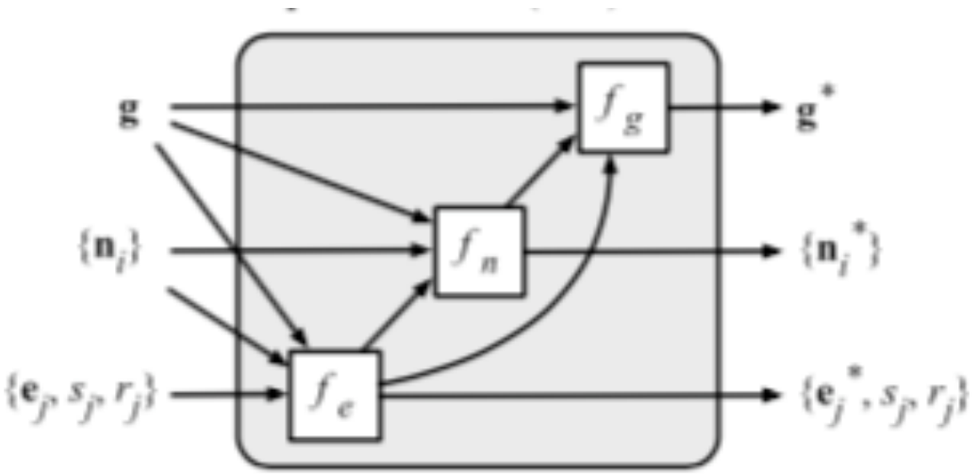
Predictions: I predict only the dynamic features, their temporal difference.
Train with regression.

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

Robots as graphs

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, $G^* = (\mathbf{g}^*, \{\mathbf{n}_i^*\}, \{\mathbf{e}_j^*, s_j, r_j\})$

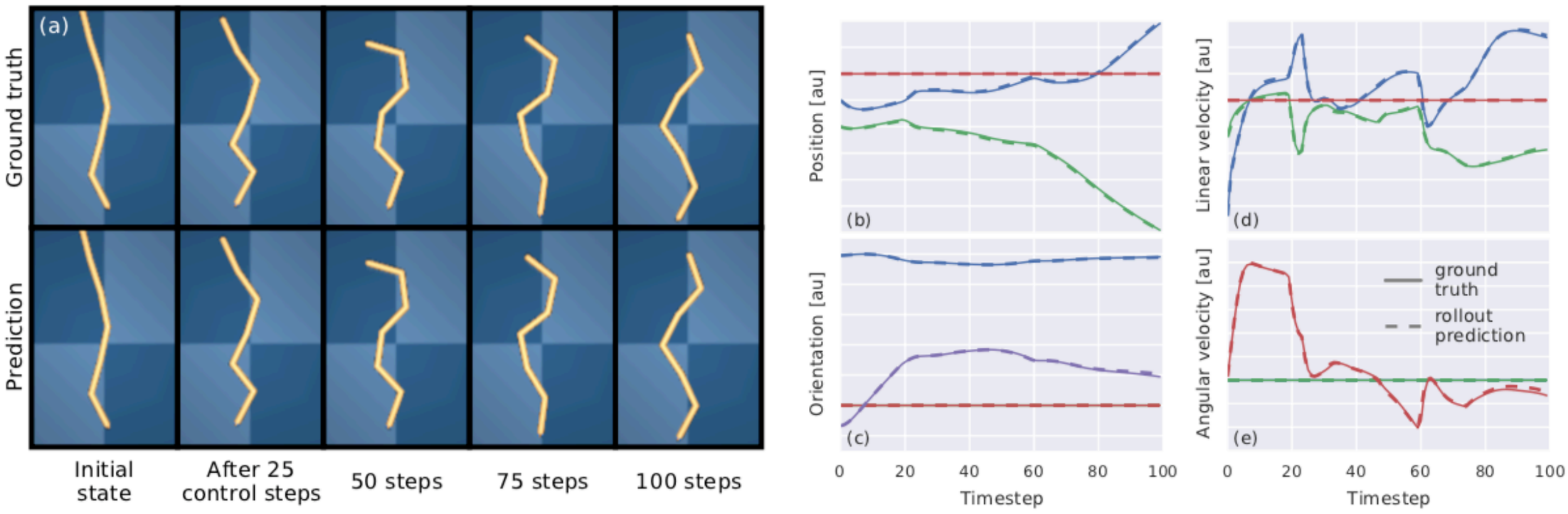
Predictions: I predict only the dynamic features, their temporal difference.
Train with regression.

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

Robots as graphs

Node features

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

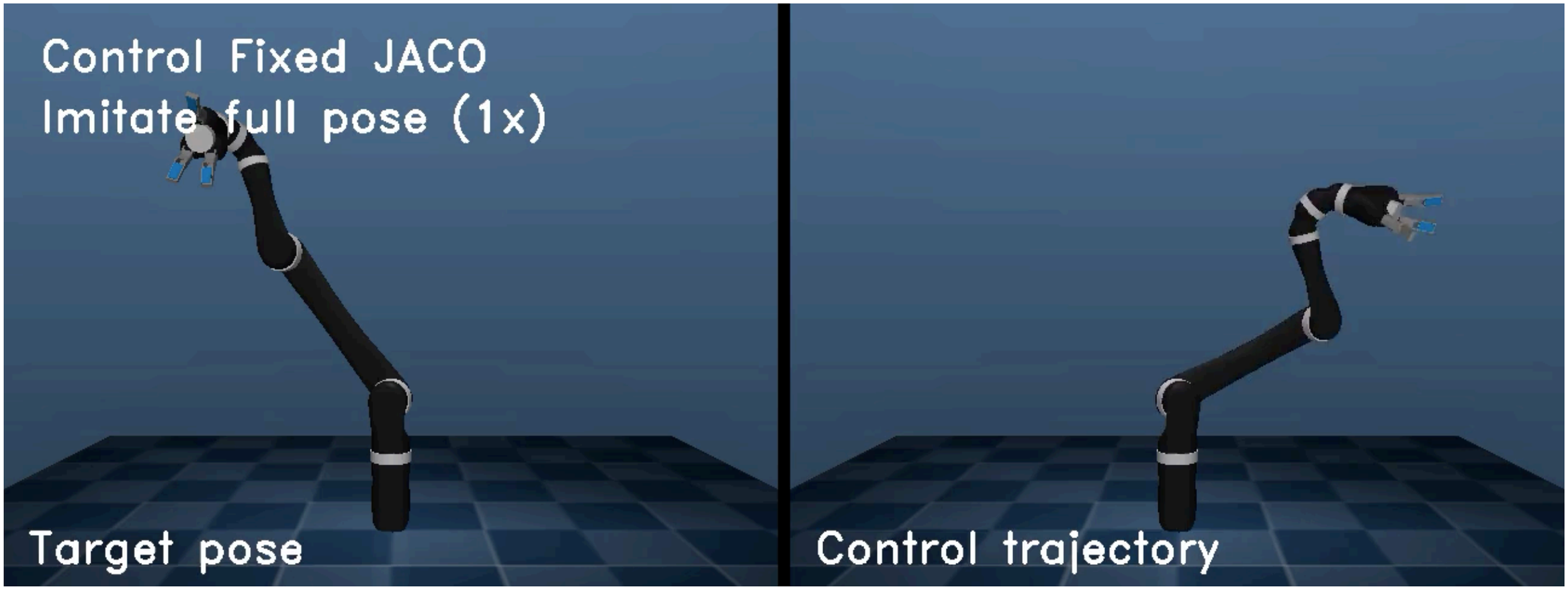


Predictions: I predict only the dynamic features, their temporal difference.

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

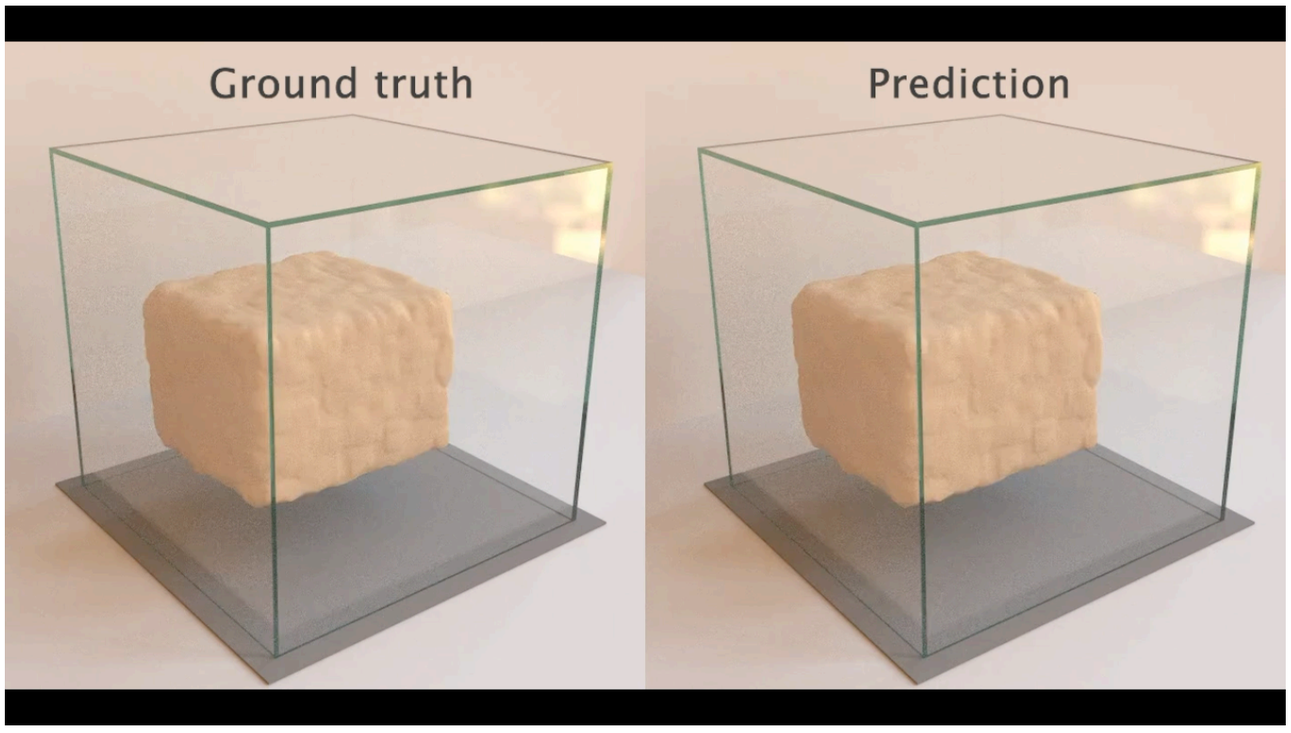
Model predictive control (MPC)

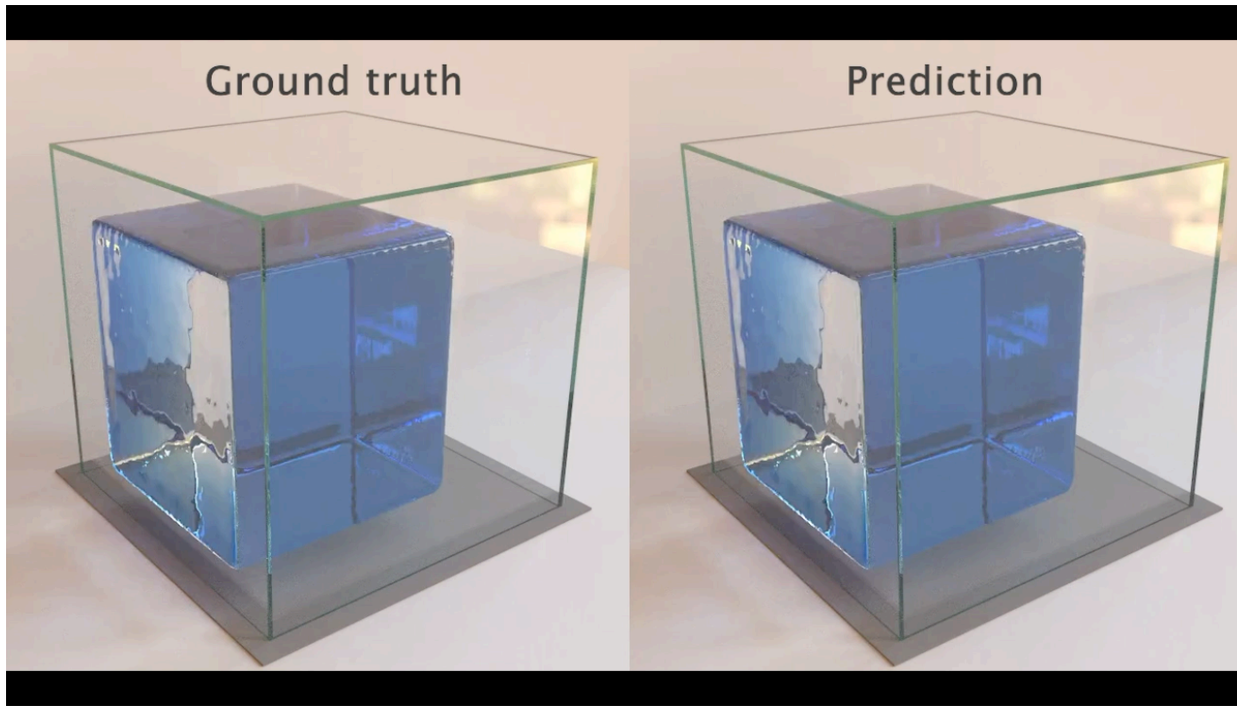
MPC to reach a target configuration

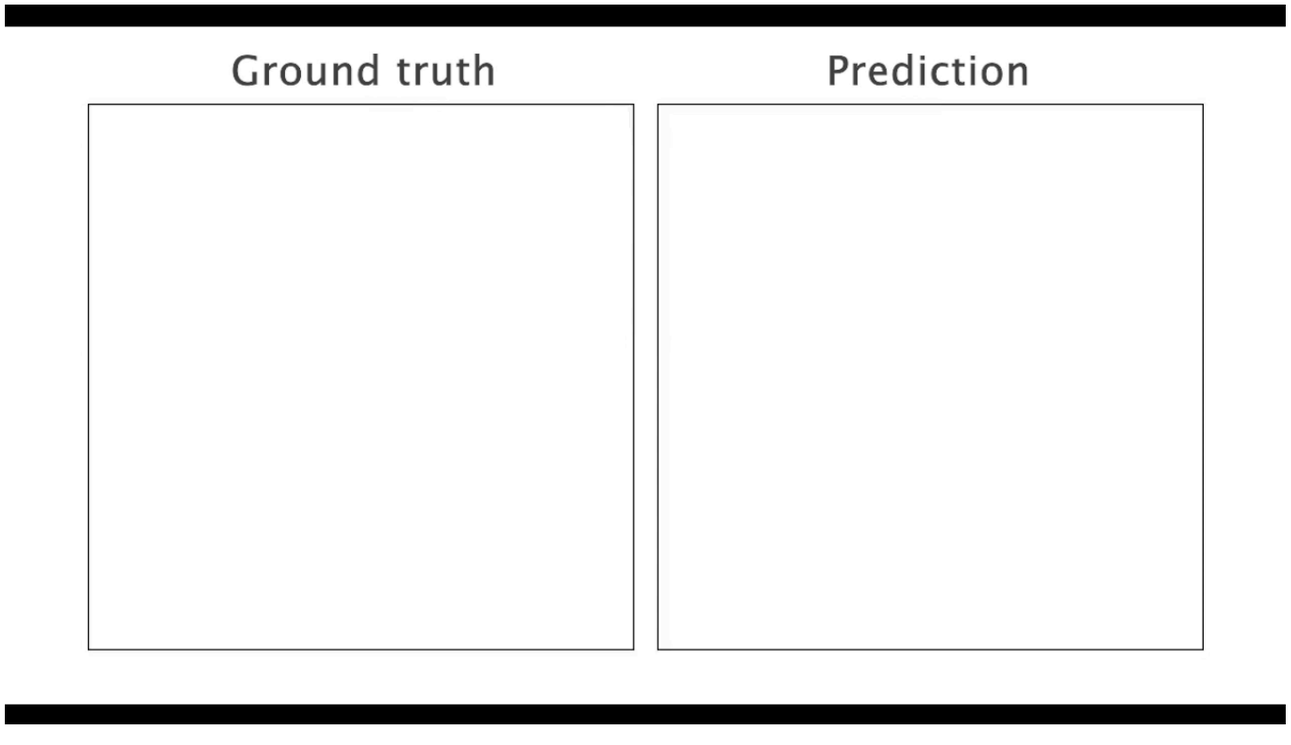


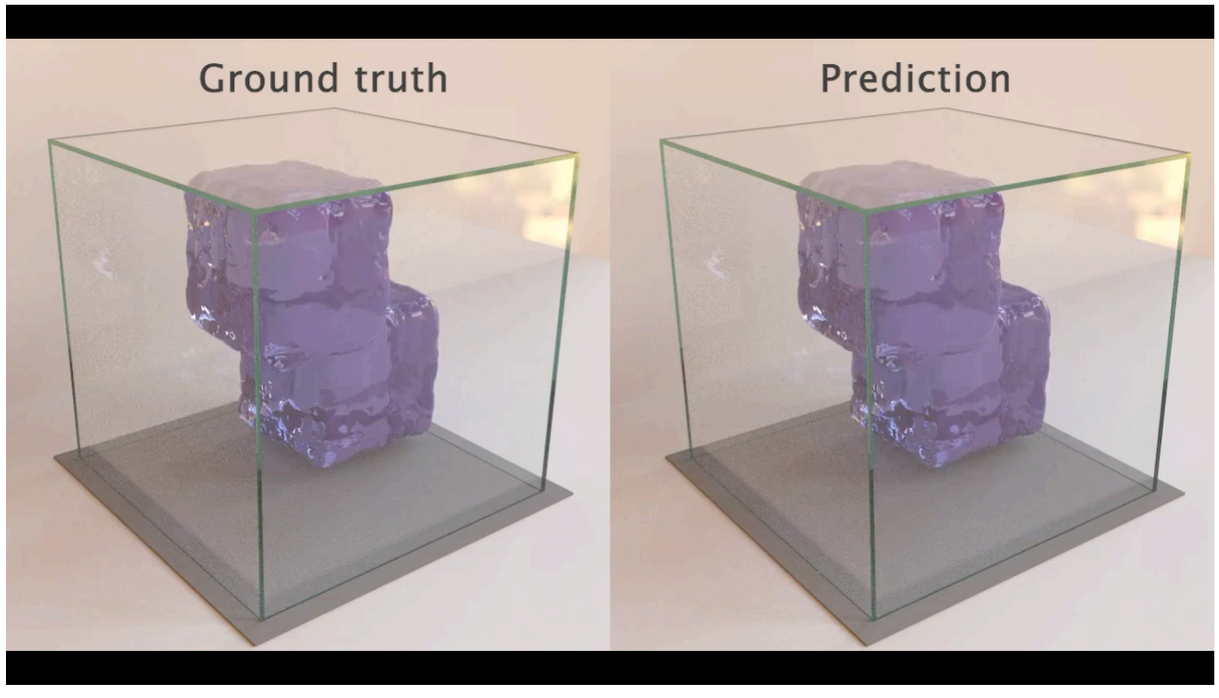
Graph Networks as Learnable Physics Engines for Inference and Control, Gonzalez et al.

GNNs over particles



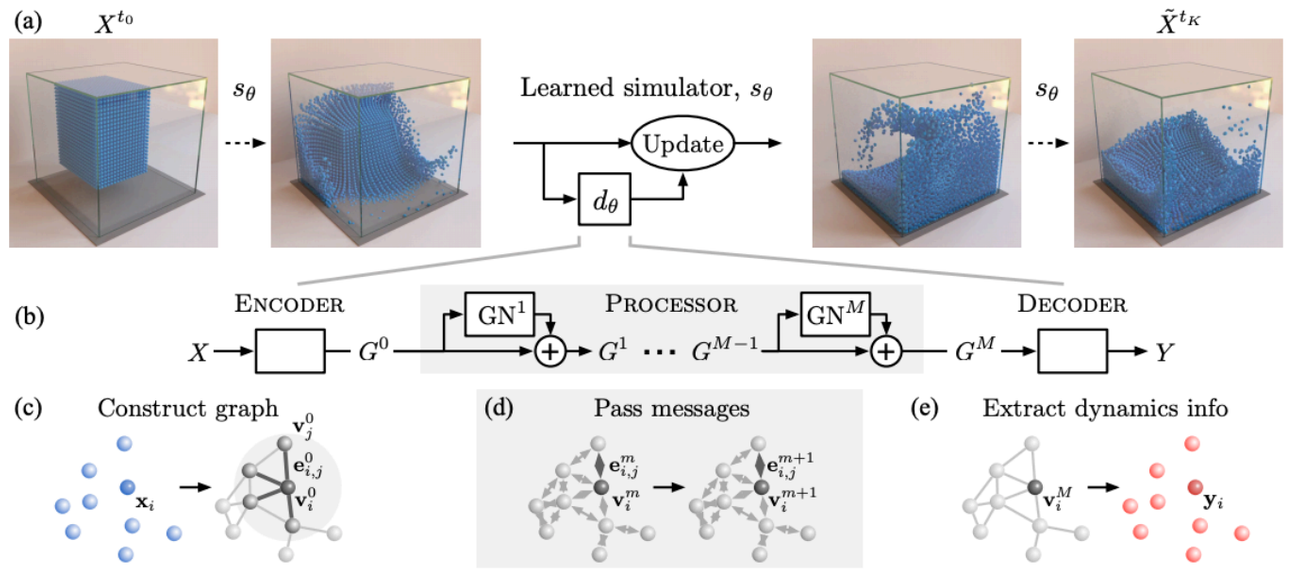






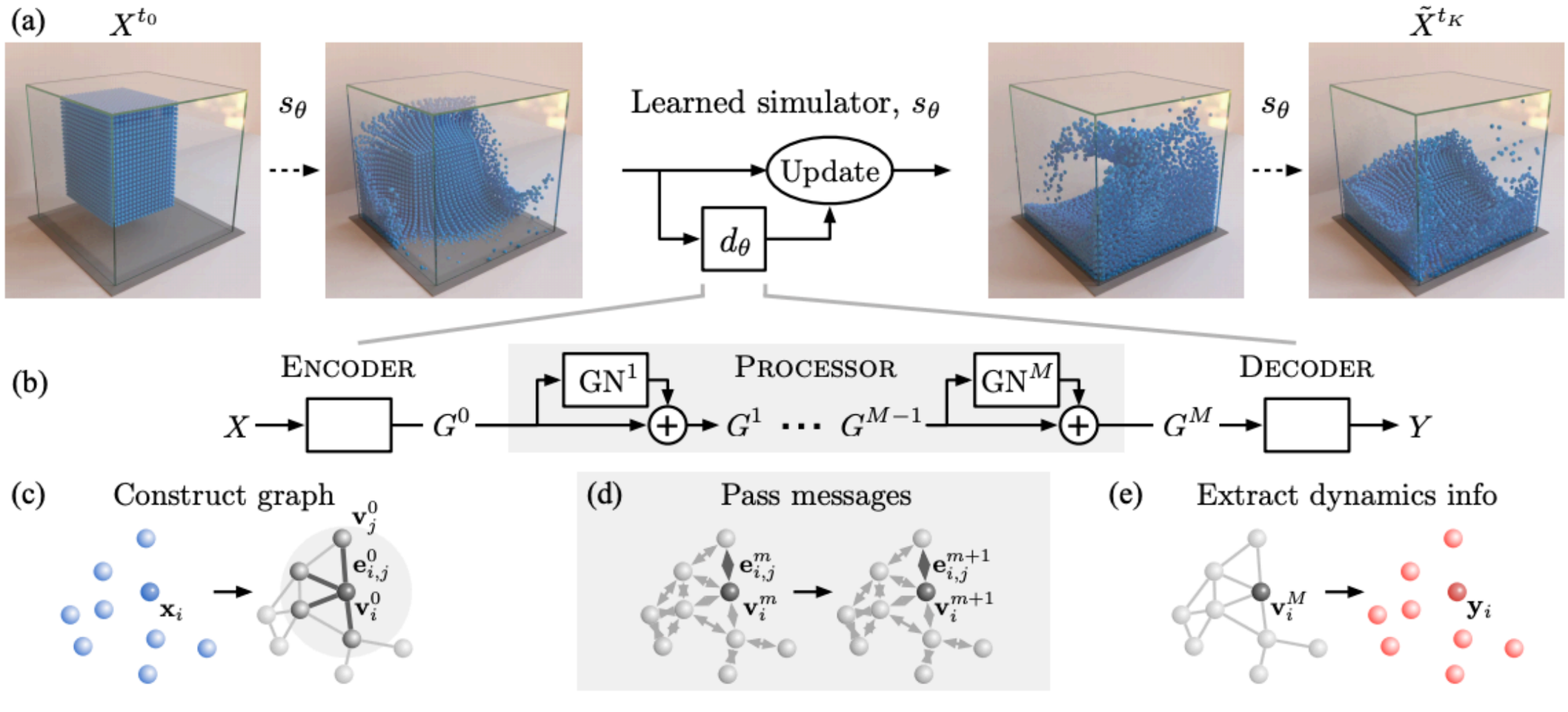
Learning to Simulate Complex Physics with Graph Networks

Alvaro Sanchez-Gonzalez [*1] **Jonathan Godwin** [*1] **Tobias Pfaff** [*1] **Rex Ying** [*2] **Jure Leskovec** [2]
Peter W. Battaglia [1]



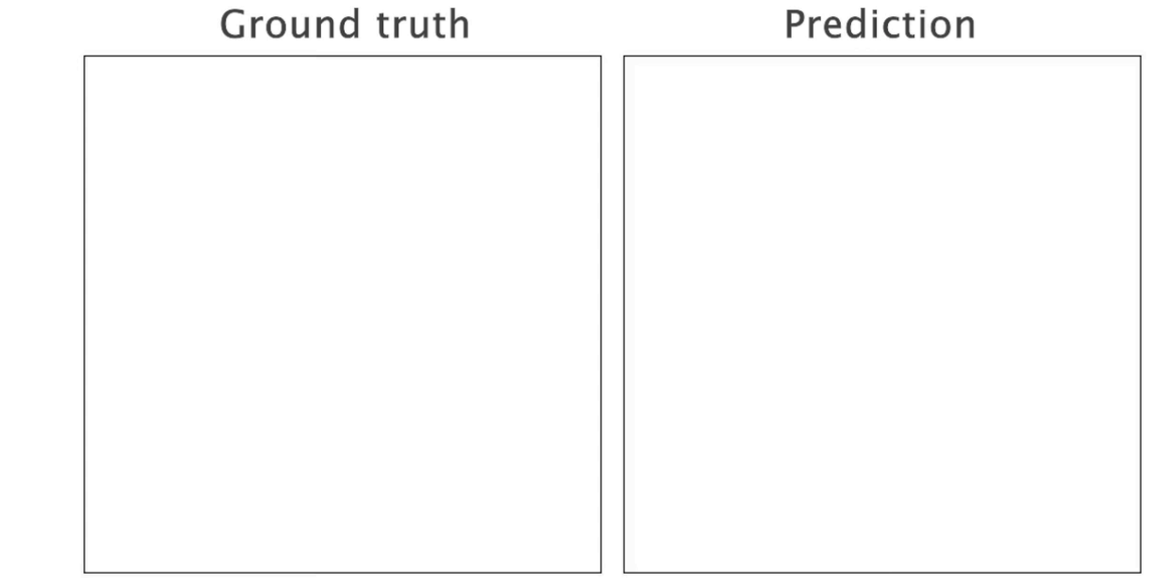
Represent objects as graphs of particles and scenes as graph of all the particles from all objects.

- Q: Why? How are particle nodes different than object nodes?
- A: They do not need to capture appearance information only particles displacement! Appearances of particles stays constant over time, while appearance of objects changes: the appearance of the water changes, but the appearance of each of its particles did not. The shape of the object/material is captured simply by the particle graph (the location of its nodes).



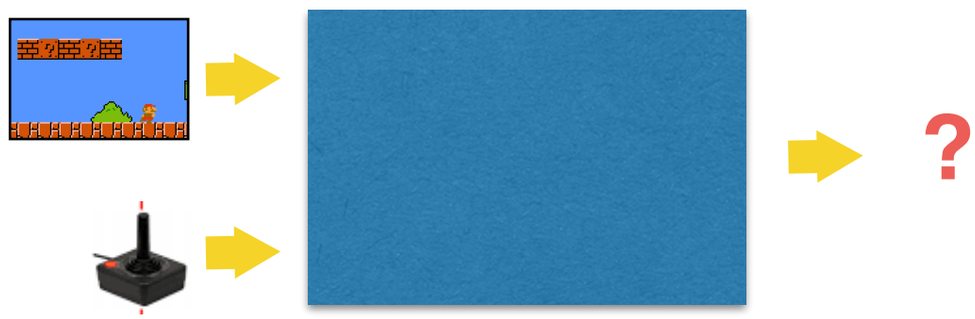
- Input: particle velocities of the last 5 time steps, output: particle acceleration.
- Train for single step prediction.
- Handle error accumulation during unrolling by injecting noise in particle velocities during training.
- Q: how can we encode particle locations?
- A:Edges encode relative distances between two particles, no absolute position, else the neural net would not be translation invariant
- Multiple rounds of message passing: necessary to transmit the interaction across the graph. Each round has node and edge weights that are different.

Generalization



Model Learning - 3 Qs always in mind

- What shall we be predicting?



Answers:

1. Future images (WXHX3 matrices): very high dimensional and multimodal, and image unrolling potentially irrelevant/too hard for the end task
2. Future images +Rewards: very high dimensional and multimodal, and image unrolling potentially irrelevant/too hard for the end task
3. Object motions (in 2D): cannot model appearance changes over time, hard to handle object occlusion (that affect object appearances), hard to handle
4. Image abstractions!
5. Object abstractions

Unrolling in the image space

- What shall we be predicting?

Unrolling in object displacement space

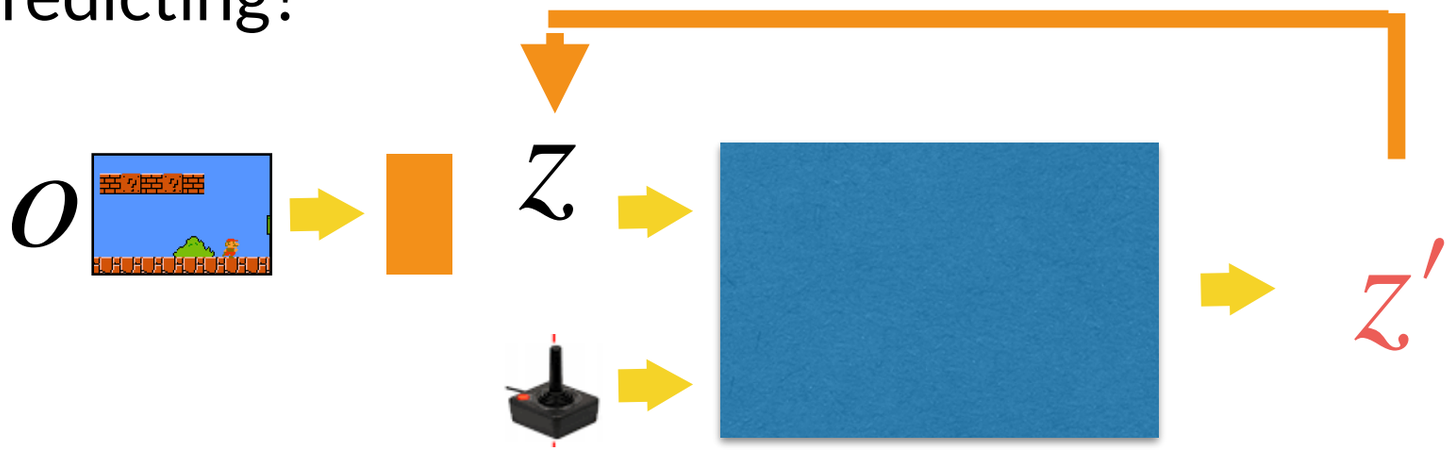
- What shall we be predicting?

Our answers so far:

1. Images: very high dimensional, very hard to model their distribution
2. 2D or 3D object/particle velocities: engineered, would not work beyond simple rigid worlds for objects, and it requires unrealistic supervision for particles.

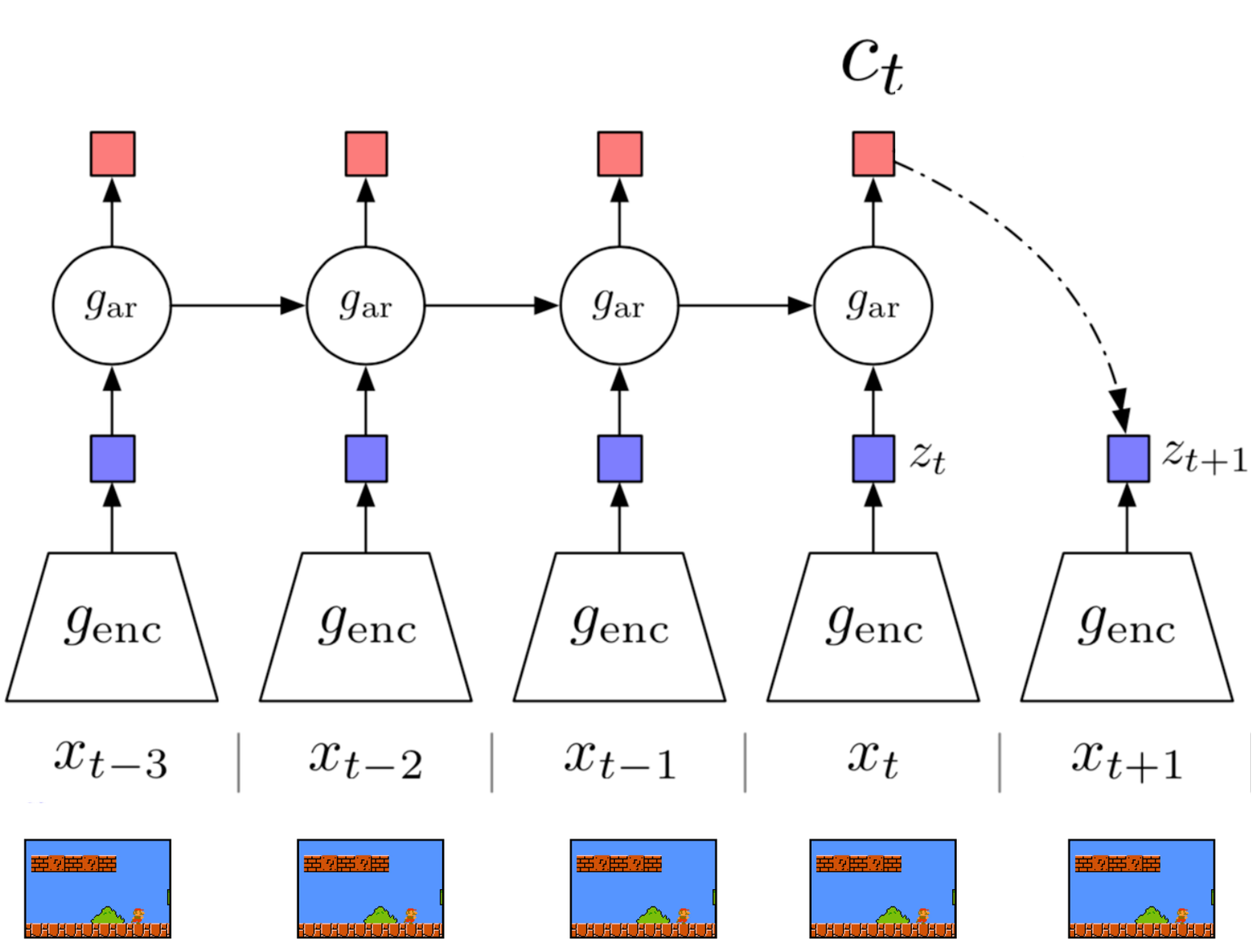
Unrolling in a latent image space

- What shall we be predicting?

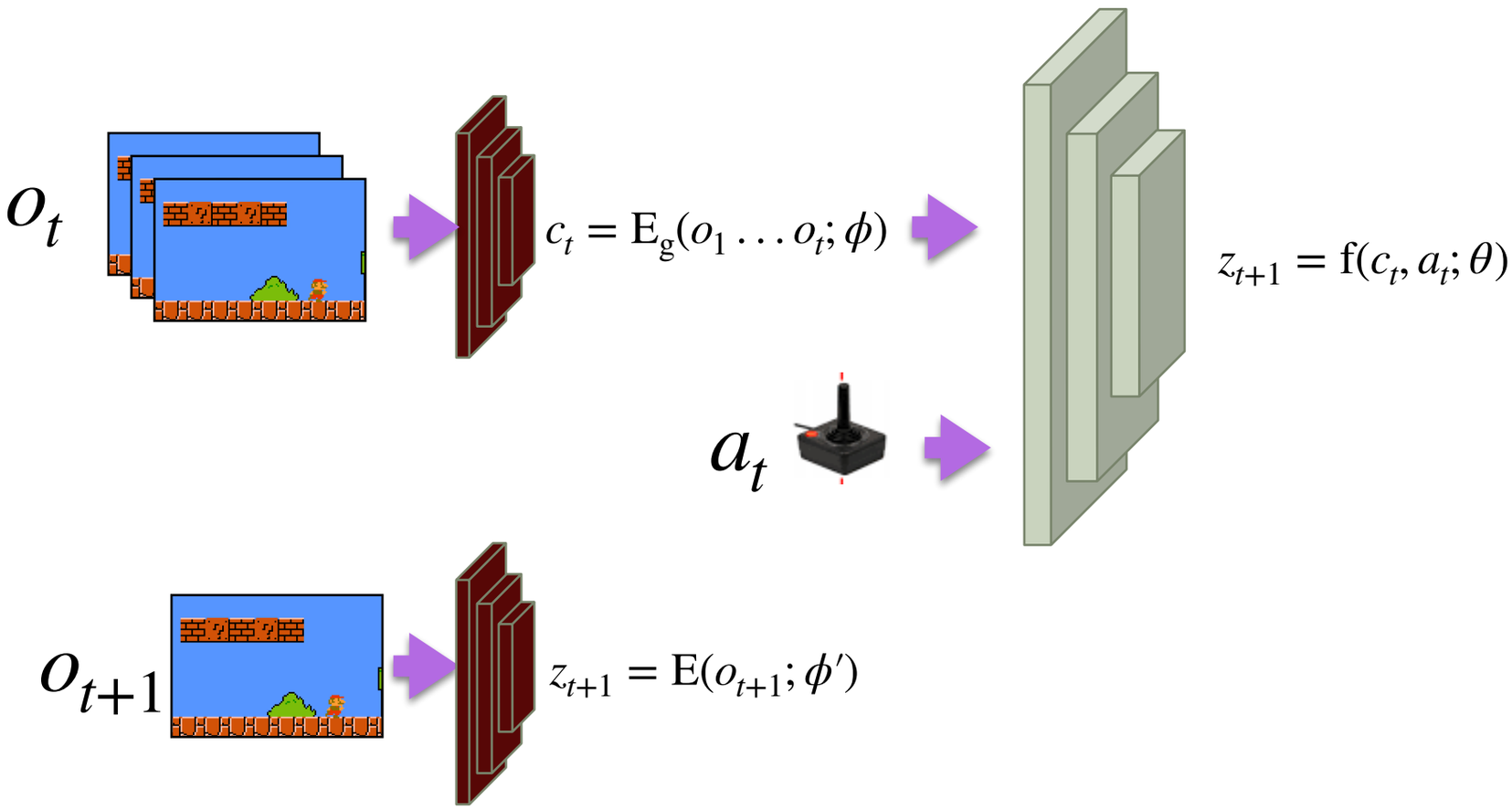


Prediction in a latent image space

- c_t : current context vector, uses information from the past up till time t
- z_t : latent image embedding vector, uses only information of time
- We want to predict z_{t+1} from $c_t, \forall t$



Prediction in a latent space



$$\min_{\theta,\phi,\phi'} . \quad \|\mathrm{f}(c_t, a_t; \theta) - z_{t+1}\|$$

Q:What is the problem with this optimization problem?

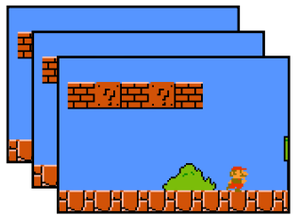
A:There is a trivial solution :-(

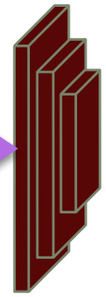
Q:Would the problem go away if instead we had: $z_{t+1} = z_t + \mathrm{f}(c_t, a_t; \theta)$

A:No, it's exactly the same problem.

Prediction in a latent space

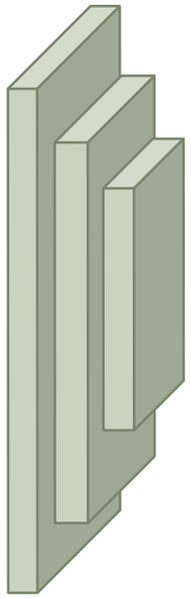
o_t





$c_t = \mathrm{E_g}(o_1 \ldots o_t; \phi)$

a_t

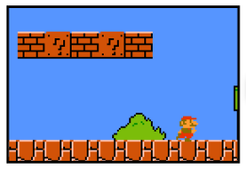


$z_{t+1} = \mathrm{f}(c_t, a_t; \theta)$

$$\min_{\theta,\phi,\phi'} . \quad \|\mathrm{f}(c_t, a_t; \theta) - z_{t+1}\|$$

Autoencoding: $\min_{\phi',\omega} . \quad \|D(\mathrm{E}(o_t; \phi'), \omega) - o_t\|$

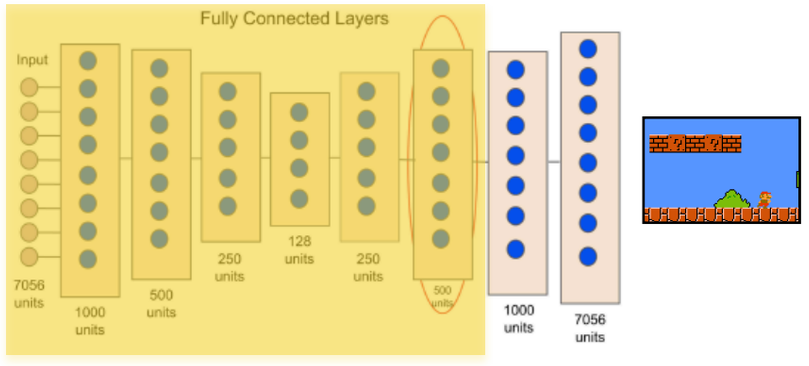
o_{t+1}



$z_{t+1} = \mathrm{E}(o_{t+1}; \phi')$

$\mathrm{E}(o_t; \phi')$

o_t



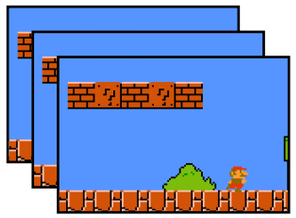
$\hat{o}_t = D(\mathrm{E}(o_t; \phi'), \omega)$

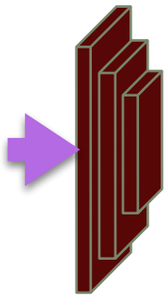
- Let's learn image encoding using autoencoders (to avoid the trivial solution)
- …and suffer the problems of autoencoding reconstruction loss that has little to do with our task

Incentivizing exploration in RL with deep predictive models, Stadie et al.

Prediction in a latent space

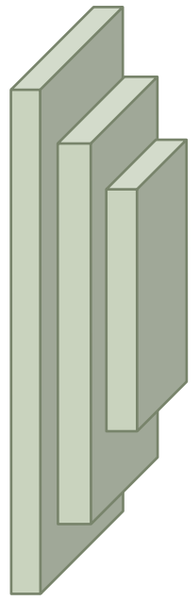
o_t





$c_t = \mathrm{E_g}(o_1 \ldots o_t; \phi)$

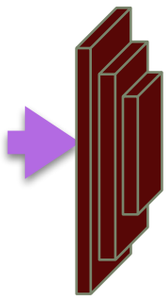
a_t



$z_{t+1} = \mathrm{f}(c_t, a_t; \theta)$

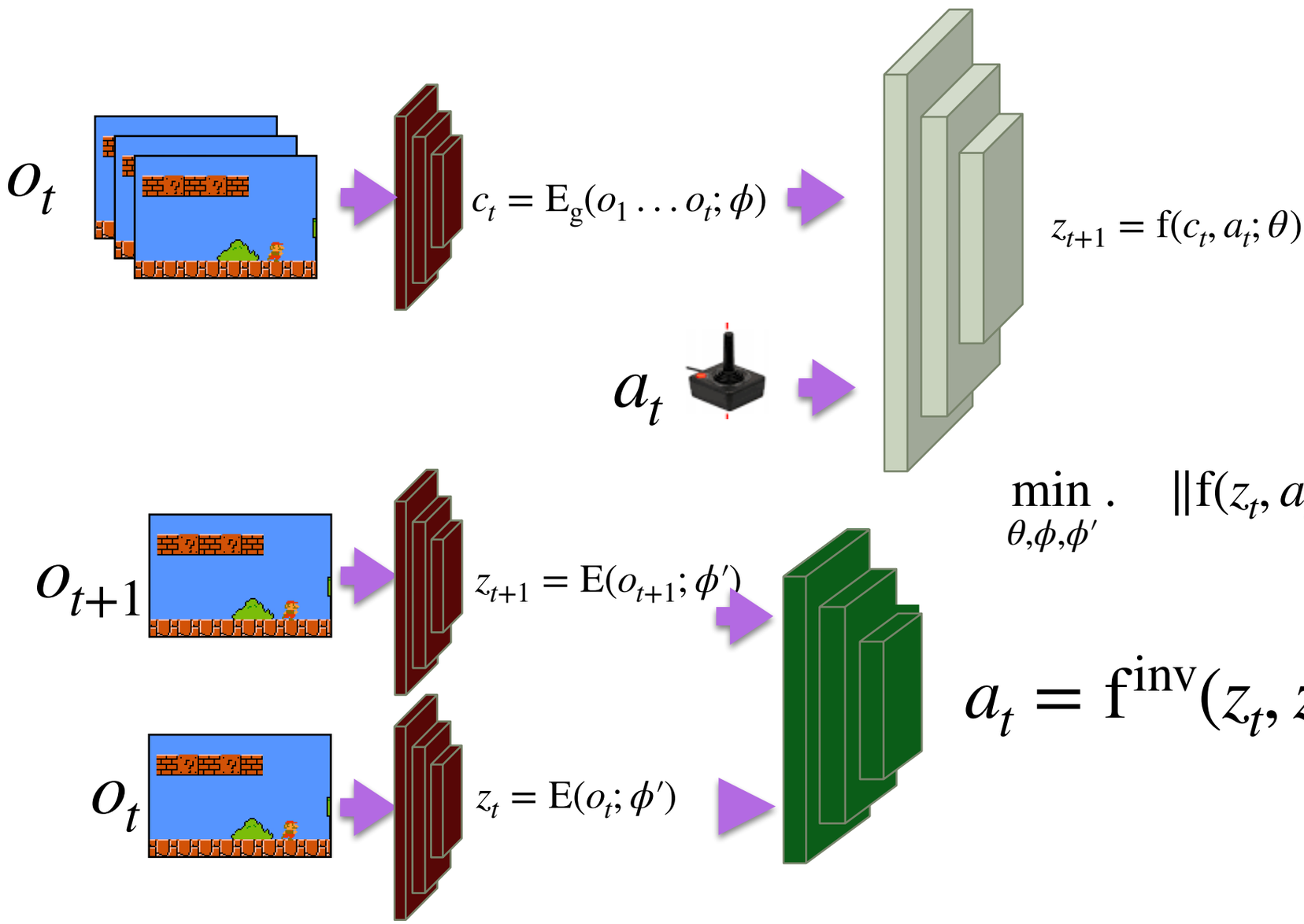
$$\min_{\theta,\phi,\phi'} . \quad \|\mathrm{f}(c_t, a_t; \theta) - z_{t+1}\|$$

o_{t+1}



$z_{t+1} = \mathrm{E}(o_{t+1}; \phi')$

Prediction in a latent space



$$\min_{\theta,\phi,\phi'} . \quad \|\mathrm{f}(z_t, a_t; \theta) - z_{t+1}\| + \|\mathrm{f}^{\mathrm{inv}}(z_t, z_{t+1}; \psi) - a_t\|$$

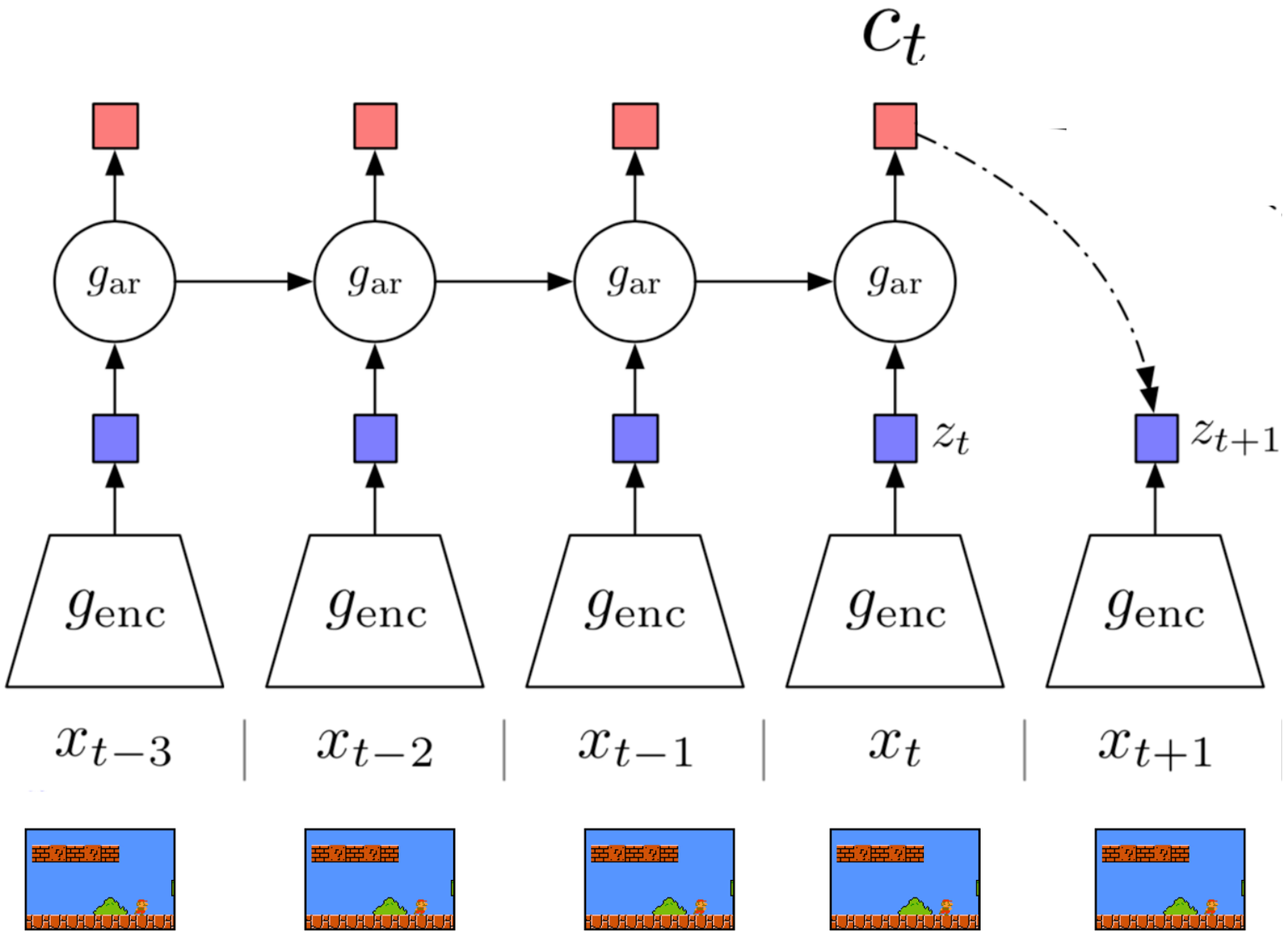
$$a_t = \mathrm{f}^{\mathrm{inv}}(z_t, z_{t+1}, \psi)$$

- Let's couple forward and inverse models (to avoid the trivial solution)
- ...then we will only predict things that the agent can control

Learning to poke by poking, *Agrawal et al. 2016*

Contrastive prediction

- Generative: model the distribution of future observations/embeddings
- Discriminative: model how much better are the future observations than other alternatives.
- Imagine we could discretize all the possible future: then we would just need to predict the right probability distribution over all (discrete set of) possibilities. Then, we want to maximize the probability of the correct outcome.



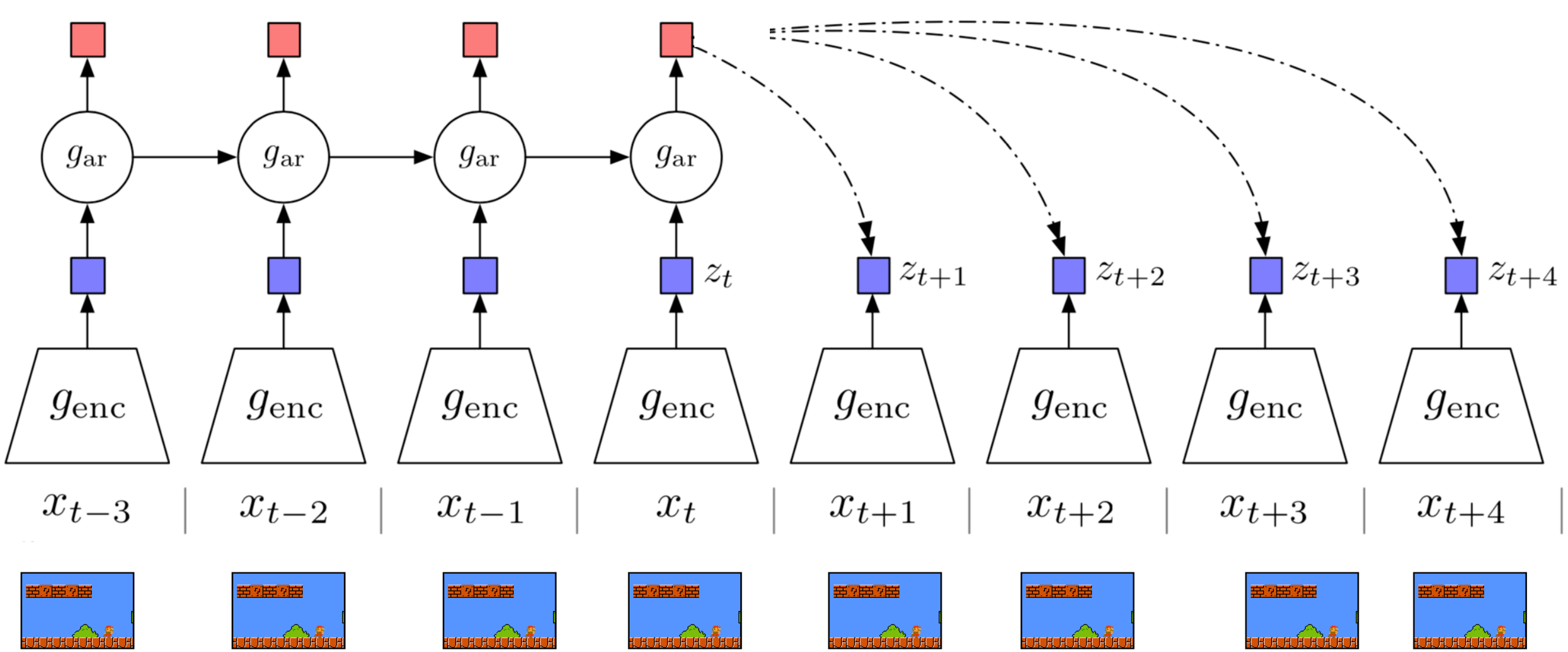
$$P(d = i \mid c) = \frac{\exp(c^\top W z^i)}{\sum_j \exp(c^\top W z^j)}$$

$$\mathscr{L} = -\frac{1}{TN} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \frac{\exp(c_t^{i\top} W z_{t+1}^i)}{\sum_{j \in Neg} \exp(c_t^{i\top} W z^j)}$$

Contrastive prediction

- Generative: model the distribution of future observations
- Discriminative: model how much better are the future observations than other alternatives.
- Imagine we could discretize all the possible future: then we would just need to predict the right probability distribution over all (discrete set of) possibilities.

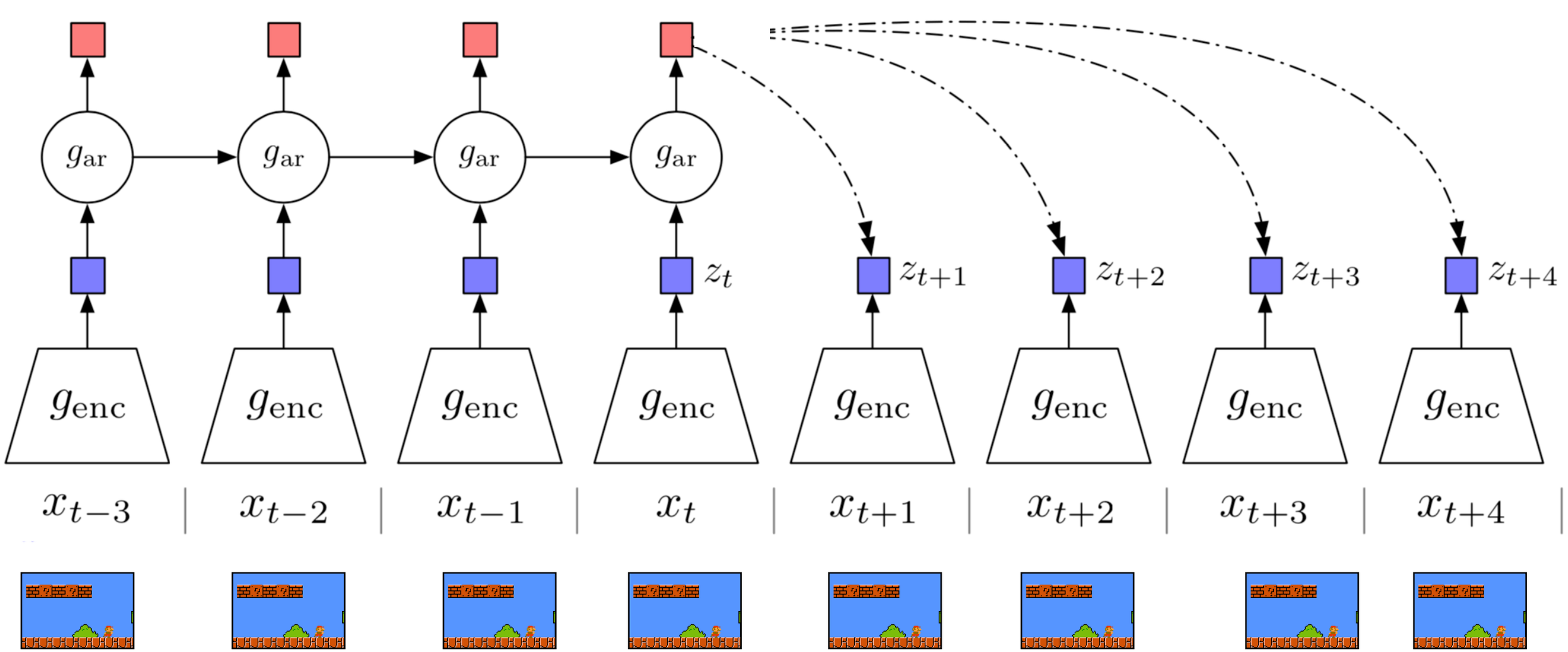
$$\mathscr{L} = -\frac{1}{TNK}\sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K}\log\frac{\exp(c_t^{i\top}W^k z_{t+k}^i)}{\sum_{j\in Neg}\exp(c_t^{i\top}Wz^j)}$$



Contrastive prediction-what about forward unrolling?

- Q:Since we do not directly predict the future z_{t+1}, how can we unroll this model forward in time?
- A1: Through ranking! Consider a set of possibilities and rank them
- A2: Through gradient descent?

$$\mathscr{L} = -\frac{1}{TNK}\sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K}\log\frac{\exp(c_t^{i\top}W^k z_{t+k}^i)}{\sum_{j\in Neg}\exp(c_t^{i\top}Wz^j)}$$



Contrastive structured prediction

We have seen two good ideas:

- Scenes as graphs
- Contrastive prediction

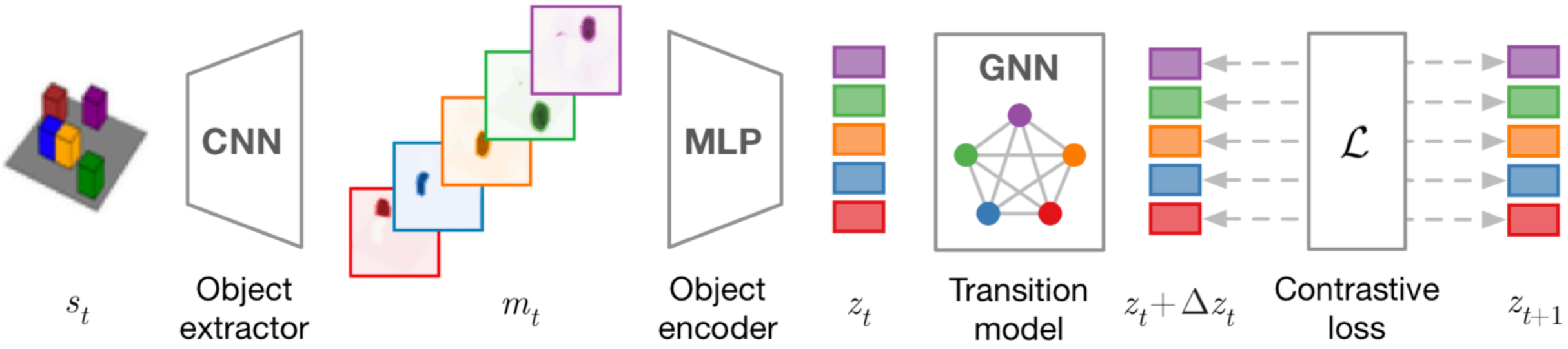
Can we put them together?

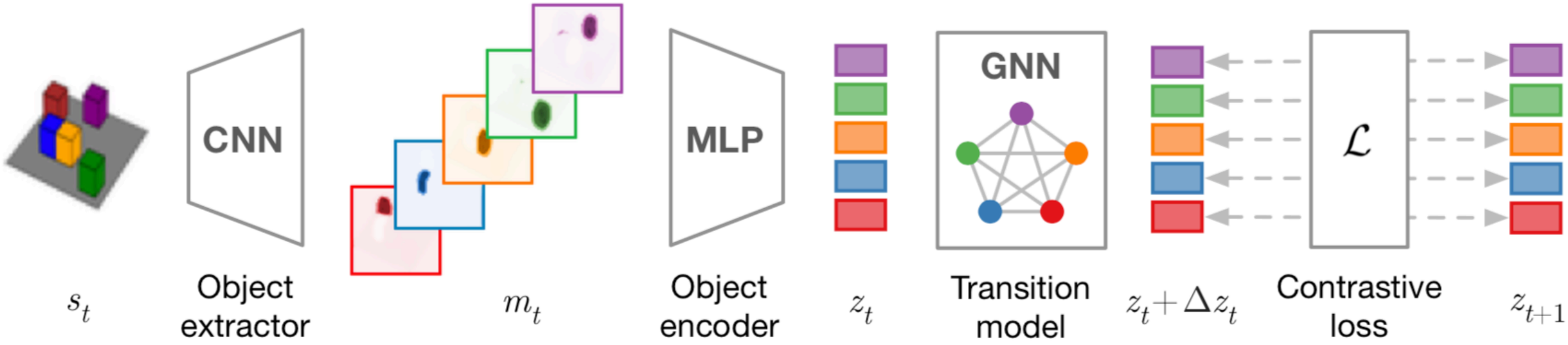
CONTRASTIVE LEARNING OF STRUCTURED WORLD MODELS

Thomas Kipf
University of Amsterdam
t.n.kipf@uva.nl

Elise van der Pol
University of Amsterdam
UvA-Bosch Delta Lab
e.e.vanderpol@uva.nl

Max Welling
University of Amsterdam
CIFAR
m.welling@uva.nl





$$e_t^{(i,j)} = f_{\text{edge}}([z_t^i, z_t^j])$$

$$\Delta z_t^j = f_{\text{node}}([z_t^j, a_t^j, \sum_{i \neq j} e_t^{(i,j)}])$$

- The object extractor spits out k feature maps $f_k, k = 1...K$, one for each object, $(W \times H \times K)$
- Each f_k is flattened and passes through a fully connected net, to produce the object embedding
- A GNN combines the object embedding and actions acted on each object and predicts Δz_t through edge and node functions
- Then a contrastive loss is taken between those predictions and the ground truth $z_{t+1}^k, k = 1...K$
- Q: How is the z_{t+1}^k computed?
- A: We run the object extractor and object encoder in the next frame!

Learning World Models from Videos

- Predicting future images and forward unrolling in the image space seems to be both too hard and unnecessary for MBRL.
- Graph neural networks as architectural choice for world models seem like a good idea: they permit generalization across number of objects/fixations in the scene. Nodes are something like object entities.
- Graph neural networks over particles though show impressive simulations it is unclear how to infer or train from real world input.
- Predicting future image embeddings trained with contrastive prediction seems like a better idea. Yet, unrolling through ranking is still unclear how well it does.

Learning World Models from Videos-Challenges

Problems remain:
1) Objects change appearance in the 2D image domain. Why?
Models that predict object motion assume appearance does not change over time (they use toy worlds).
2) It is hard to model deformations / articulations
3) Learning world models in the real world would benefit from a 3D object tracker that currently does not exist.
4) Temporal prediction over a fixed fine-grained temporal granularity may be a bad idea.