UNIVERSITY OF CAMBRIDGE

Neural Relational Inference for Interacting Systems

Petar Veličković

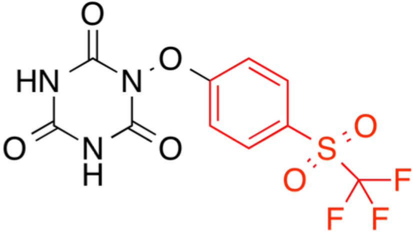
Artificial Intelligence Group
Department of Computer Science and Technology, University of Cambridge, UK

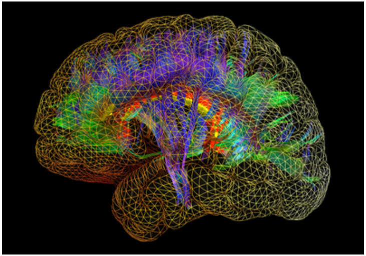
MILA Graph Representation Reading Group Meeting 25 July 2018

Introduction

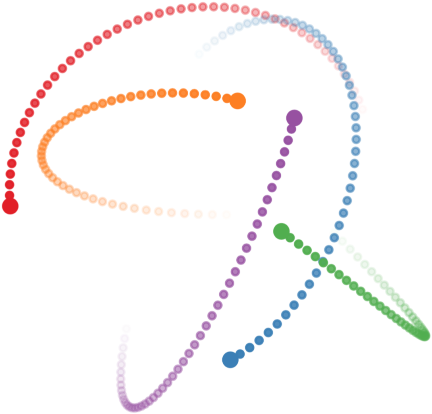
- In this talk, I will survey the recently published **Neural Relational Inference** model (Kipf, Fetaya *et al.*, ICML 2018).
- This model enables the **discovery and exploitation of latent interactions between objects**, through the synergy of *graph convolutional networks* and *variational autoencoders*.
- Exciting results + avenues for further work!

Graphs are **everywhere**!





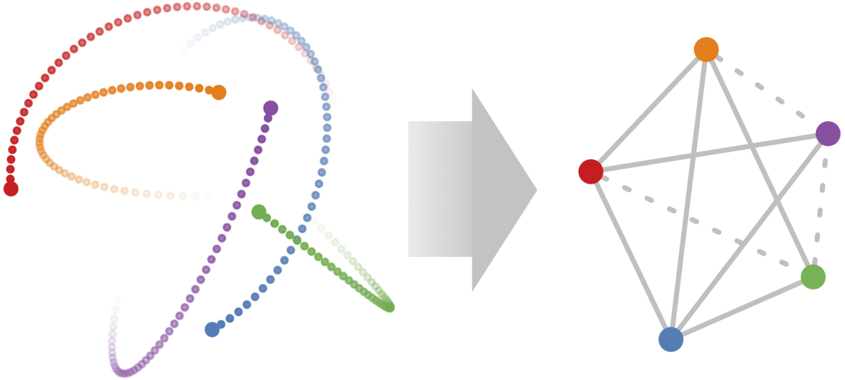
. . . but can we always *see* them?



Relational inference

- Virtually all graph convolutional techniques require a graph to be *provided* as input!
- However, often we will only have access to *node features*...
- Approaches such as *Relational Networks* (Santoro *et al.*, 2017), or *VAIN* (Hoshen, 2017) circumvent this by assuming a **complete graph** (i.e. all-pairs interactions).
- But most interaction graphs have properties (such as *sparsity*) that we may wish to explicitly demand!
- Furthermore, we may wish to *identify* and *decouple* different **types of interaction**.

Our task for today

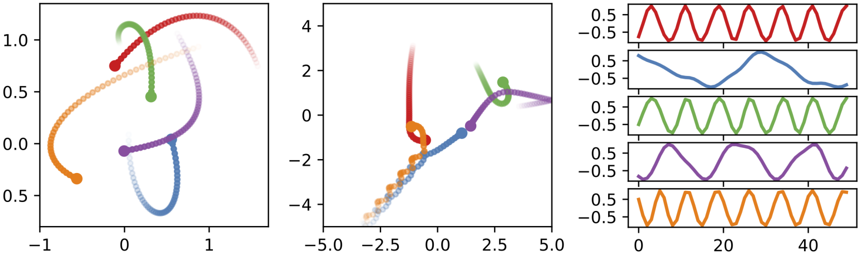


Observed dynamics

Interaction graph

Motivation: predicting *trajectories*

- **Input:** Trajectories (e.g. coordinates) $\vec{x}_i^{\leq t}$ for each particle i.
- **Output:** Future trajectories $\vec{x}_i^{>t}$ for each particle i.



Springs (2D) Charged (2D) Kuramoto (1D)

- The interaction graph between particles will be a *byproduct*!

Simple baseline #1: RNN

- Let $\vec{x}^t$ denote the coordinates of all particles at time t:

$$\vec{x}^t = \left[\vec{x}_1^t, \vec{x}_2^t, \ldots, \vec{x}_n^t\right]$$

- We can now define a recurrent neural network (e.g. LSTM or GRU) to operate on this sequential input:

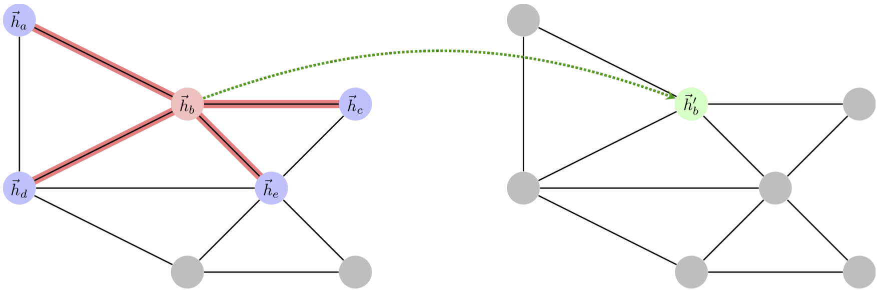
$$\vec{h}^t = RNN(\vec{h}^{t-1}, \vec{x}^t)$$

- From its hidden states, we can predict the future timesteps:

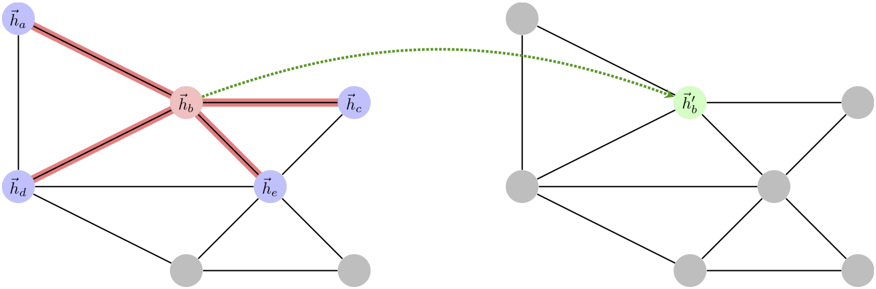
$$\vec{x}^{t+1} = f(\vec{h}^{t+1})$$

where f is an MLP.

Graph convolutional network



Graph convolutional network



In a nutshell, obtain higher-level representations of a node i by leveraging its *neighbourhood*, $\mathcal{N}_i$!

$$\vec{h}_i^{\ell+1} = g^\ell(\vec{h}_a^\ell, \vec{h}_b^\ell, \vec{h}_c^\ell, \dots) \ \ (a, b, c, \dots \in \mathcal{N}_i)$$

where g^ℓ is the ℓ-th *graph convolutional layer*.

The MPNN framework

- The NRI model leverages a graph convolutional layer inspired by *message-passing neural networks* (Gilmer *et al.*, 2017).

The MPNN framework

- The NRI model leverages a graph convolutional layer inspired by *message-passing neural networks* (Gilmer *et al.*, 2017).
- First, compute *edge messages*, $\vec{h}^{\ell}_{i\to j}$, for each edge $i \to j$ in the graph. Apply a simple MLP, f_e^{ℓ}, over the features of i and j:

$$\vec{h}^{\ell}_{i\to j} = f_e^{\ell}(\vec{h}^{\ell}_i, \vec{h}^{\ell}_j)$$

The MPNN framework

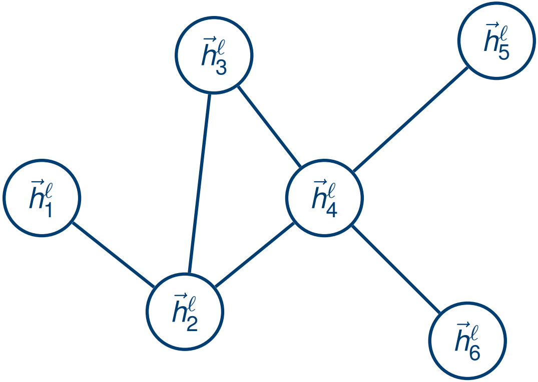
- The NRI model leverages a graph convolutional layer inspired by *message-passing neural networks* (Gilmer *et al.*, 2017).
- First, compute *edge messages*, $\vec{h}^{\ell}_{i\rightarrow j}$, for each edge $i \rightarrow j$ in the graph. Apply a simple MLP, f^{ℓ}_{e}, over the features of i and j:

$$\vec{h}^{\ell}_{i\rightarrow j} = f^{\ell}_{e}(\vec{h}^{\ell}_{i}, \vec{h}^{\ell}_{j})$$

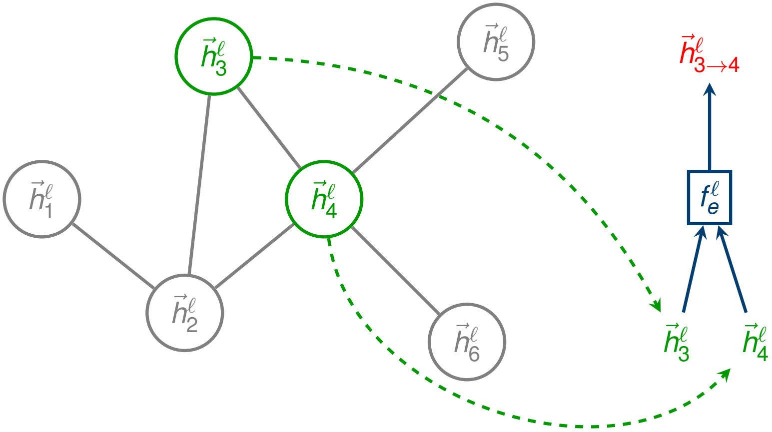
- Then, aggregate all messages entering a node j to obtain the next-level features, $\vec{h}^{\ell+1}_{j}$. Apply a simple MLP, f^{ℓ}_{v}, over the summed messages.

$$\vec{h}^{\ell+1}_{j} = f^{\ell}_{v}\left(\sum_{j\in\mathcal{N}_i} \vec{h}^{\ell}_{i\rightarrow j}\right)$$

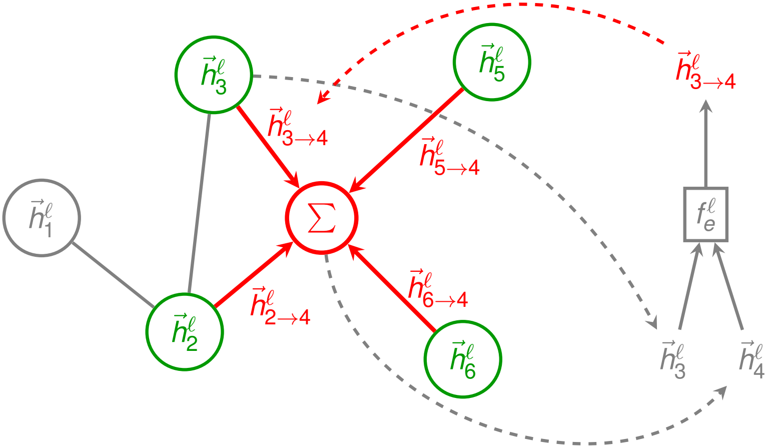
MPNN: initial setup



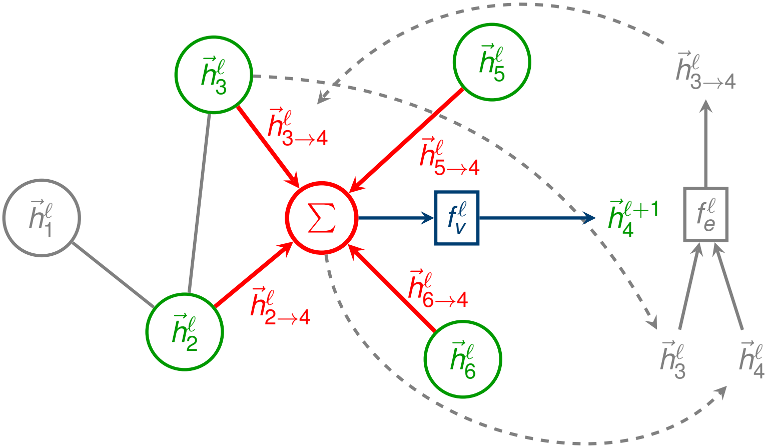
MPNN, computing messages



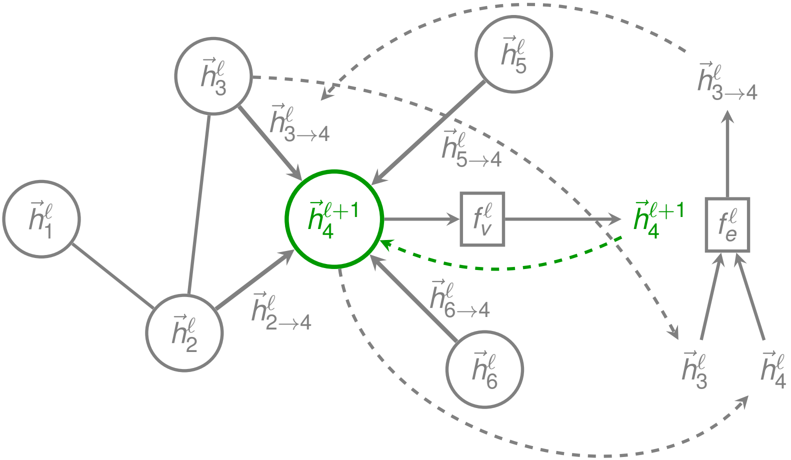
MPNN, aggregating messages



MPNN, computing node features



MPNN, next-level features



Simple baseline #2: Complete graph

- As another baseline approach, we may use this kind of layer to predict trajectories using a complete graph (assume all pairs of nodes interact).
- The equations of the baseline become equivalent to:

$$\vec{h}^t_{i\to j} = f_e(\vec{x}^t_i, \vec{x}^t_j)$$

$$\vec{x}^{t+1}_j = f_v\left(\sum_{i\neq j} \vec{h}^t_{i\to j}\right)$$

with a few kinks, specific to the trajectory predicting task…

Simple baseline #2: Complete graph

- First, to simplify the job of the network, have it only predict *changes in position*:

$$\vec{h}^t_{i\to j} = f_e(\vec{x}^t_i, \vec{x}^t_j)$$

$$\vec{x}^{t+1}_j = \boxed{\vec{x}^t_j} + f_v\left(\sum_{i\neq j} \vec{h}^t_{i\to j}\right)$$

Simple baseline #2: Complete graph

- Also, *explicitly model uncertainty*; will be useful for the variational framework later on.

$$\vec{h}^t_{i\to j} = f_e(\vec{x}^t_i, \vec{x}^t_j)$$

$$\boxed{\vec{\mu}^{t+1}_j} = \vec{x}^t_j + f_v\left(\sum_{i\neq j} \vec{h}^t_{i\to j}\right)$$

$$\boxed{\vec{x}^{t+1}_j \sim \mathcal{N}(\vec{\mu}^{t+1}_j, \sigma^2\mathbf{I})}$$

Simple baseline #2: Complete graph, *with GRU*

- The model thus far assumed the *Markov property* (i.e. that $\vec{x}^{t+1}$ depends fully on $\vec{x}^{t}$). This is OK for physics, but if necessary, we can alleviate the constraint by using a recurrent update:

$$\vec{h}^t_{i\rightarrow j} = f_e(\vec{x}^t_i, \vec{x}^t_j)$$

$$\boxed{\vec{h}^{t+1}_j = GRU\left(\left[\vec{x}^t_j, \sum_{i\neq j} \vec{h}^t_{i\rightarrow j}\right], \vec{h}^t_j\right)}$$

$$\vec{\mu}^{t+1}_j = \vec{x}^t_j + f_v \boxed{\left(\vec{h}^{t+1}_j\right)}$$

$$\vec{x}^{t+1}_j \sim \mathcal{N}(\vec{\mu}^{t+1}_j, \sigma^2\mathbf{I})$$

Interaction graph

- This baseline can be improved if we specify an explicit *interaction graph*. Initially, assume there are K edge types (with one type reserved for “no edge”).
- Then, define a binary tensor $\mathbf{z} \in \mathbb{R}^{V \times V \times K}$ such that z_{ijk} denotes whether the edge $i \rightarrow j$ is of the k-th type.
- Assume an edge cannot have more than one type, i.e., $\vec{z}_{ij}$ is one-hot.

Leveraging the interaction graph

- Now this graph can be exploited—define a separate MLP f_e^k for each edge type. For the Markov decoder:

$$\vec{h}_{i \to j}^t = f_e(\vec{x}_i^t, \vec{x}_j^t)$$

$$\vec{\mu}_j^{t+1} = \vec{x}_j^t + f_v\left(\sum_{i \neq j} \vec{h}_{i \to j}^t\right)$$

$$\vec{x}_j^{t+1} \sim \mathcal{N}(\vec{\mu}_j^{t+1}, \sigma^2 \mathbf{I})$$

The NRI *decoder*

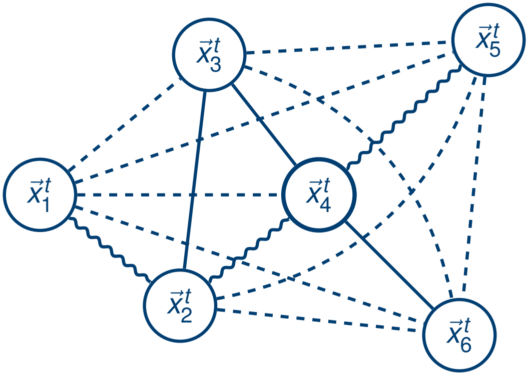
- Now this graph can be exploited—define a separate MLP f_e^k for each edge type. For the Markov decoder:

$$\vec{h}_{i\to j}^t = \boxed{\sum_k z_{ijk} f_e^k(\vec{x}_i^t, \vec{x}_j^t)}$$

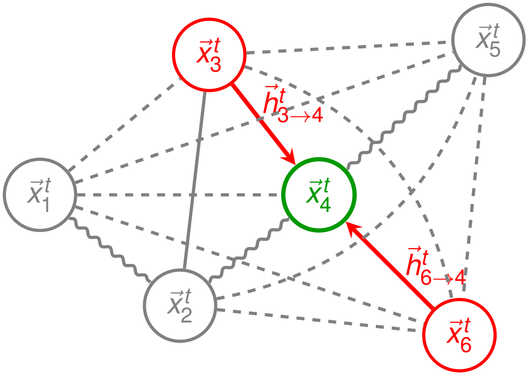
$$\vec{\mu}_j^{t+1} = \vec{x}_j^t + f_v\left(\sum_{i\neq j} \vec{h}_{i\to j}^t\right)$$

$$\vec{x}_j^{t+1} \sim \mathcal{N}(\vec{\mu}_j^{t+1}, \sigma^2\mathbf{I})$$

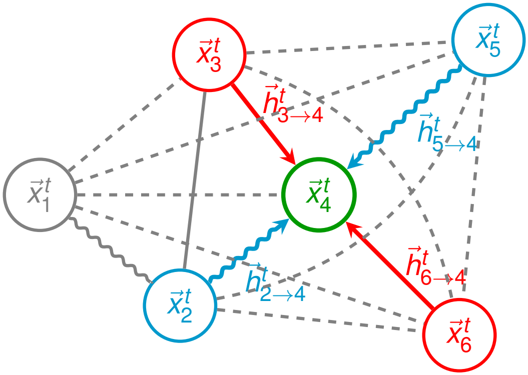
The NRI *decoder*



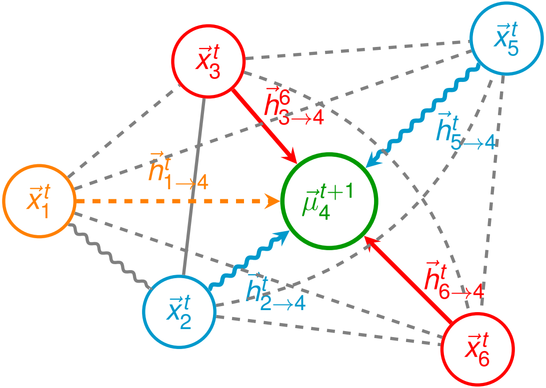
The NRI *decoder*, computing messages. . .



The NRI *decoder*, computing messages. . .



The NRI *decoder*, computing messages. . .



Latent graph inference

- We are still tasked with discovering the entries of the tensor **z.**
- **Idea:** Use MPNNs over a complete graph once more—then classify edge types based on the edge messages $\vec{h}_{i\rightarrow j}$.
- This time, *stack two layers*—so that edges can be derived based on global interactions!
  - $\vec{h}^1_{i\rightarrow j}$ will only depend on $\vec{x}_i$ and $\vec{x}_j$;
  - $\vec{h}^2_{i\rightarrow j}$ will depend on all the nodes in the graph.

The NRI *encoder*

- In equation form:

$$
\begin{aligned}
\vec{h}_j^1 &= f(\vec{x}_j) \\
\vec{h}_{i\to j}^1 &= f_e^1(\vec{h}_i^1, \vec{h}_j^1) \\
\vec{h}_j^2 &= f_v^1\left(\sum_{i\neq j} \vec{h}_{i\to j}^1\right) \\
\vec{h}_{i\to j}^2 &= f_e^2(\vec{h}_i^2, \vec{h}_j^2) \\
z_{ij} &\sim Categorical(softmax(\vec{h}_{i\to j}^2))
\end{aligned}
$$

where f is an embedding, and f_e^1, f_v^1 and f_e^2 are MLPs.

The variational setup

- The encoder gives us the probability distribution $q(\mathbf{z}|\vec{x})$, and the decoder gives us the probability distribution $p(\vec{x}|\mathbf{z})$.
- Combine learning the two in a VAE-style framework by maximising the evidence lower bound (ELBO):

$$\mathcal{L} = \underbrace{\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\vec{x})}[\log p(\vec{x}|\mathbf{z})]}_{\text{Reconstruction accuracy}} - \underbrace{D_{KL}(q(\mathbf{z}|\vec{x})\|p(\mathbf{z}))}_{\text{Regularisation}}$$

- The prior $p(\mathbf{z})$ can encode desirable properties of the latent graph. **Sparsity** is enforced by setting the probability of "no edge" to be higher than the other types.

Backpropagating through the sampling

- The operation of selecting z_{ij} is a *discrete* decision—therefore, we cannot directly propagate gradients through it.
- Can use the **Gumbel softmax** trick to circumvent this:

$$\vec{z}_{ij} = softmax((\vec{h}^2_{i\to j} + \vec{g})/\tau)$$

  where $g_k \sim Gumbel(0, 1)$ and τ is a temperature parameter (converges to one-hot when $\tau \to 0$).
- This is a *continuous approximation* to the discrete distribution—and gradients can be propagated through it.

Avoiding degenerate decoders

- Optimising the ELBO directly would involve only *single-step* predictions (predicting $\vec{x}^{t+1}$ from $\vec{x}^{t}$). This can often be nicely approximated by ignoring relational structure altogether!

- To enforce robust decoders, predict many steps at once! Every M steps, feed back the ground-truth input.

Avoiding degenerate decoders, *cont'd*

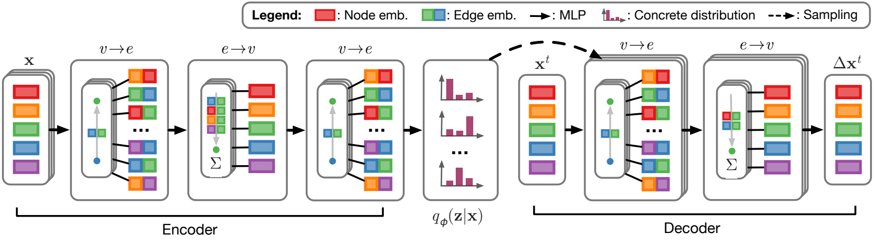
$$
\begin{aligned}
\vec{\mu}_j^2 &= decode(\vec{x}_j^1) \\
\vec{\mu}_j^3 &= decode(\vec{\mu}_j^2) \\
\vec{\mu}_j^4 &= decode(\vec{\mu}_j^3) \\
&\vdots \\
\vec{\mu}_j^{M+1} &= decode(\vec{\mu}_j^M) \\
\vec{\mu}_j^{M+2} &= decode(\vec{x}_j^{M+1}) \\
\vec{\mu}_j^{M+3} &= decode(\vec{\mu}_j^{M+2}) \\
&\vdots
\end{aligned}
$$

Putting it all together

For a given training trajectory, $\vec{x}$, of length T:

1. Compute $q(\mathbf{z}|\vec{x})$ using the encoder.
2. Sample $\vec{z}_{ij}$ from $q(\mathbf{z}|\vec{x})$ using the Gumbel softmax trick.
3. Execute the decoder to obtain $\vec{\mu}^t$ for $t \in \{2, 3, \ldots, T\}$.
4. Compute the reconstruction error (of $\vec{\mu}^t$ against $\vec{x}^t$) and KL-divergence (of $q(\mathbf{z}|\vec{x})$ against the prior $p(\mathbf{z})$).
5. Optimise the ELBO using gradient descent.

The NRI architecture



Physics simulations: latent graph discovery

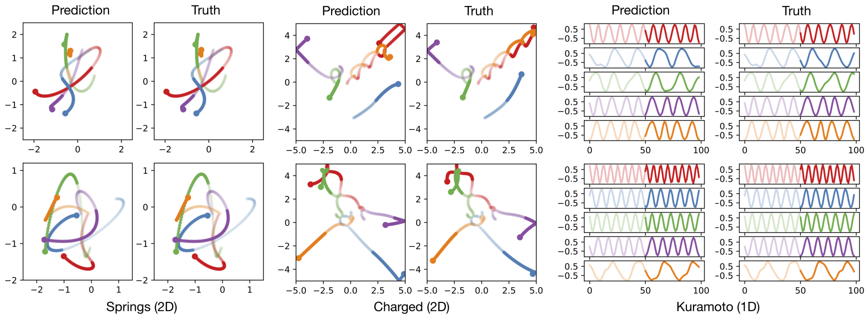
Model	Springs	Charged	Kuramoto
	5 objects		
Corr. (path)	52.4$\pm$0.0	55.8$\pm$0.0	62.8$\pm$0.0
Corr. (LSTM)	52.7$\pm$0.9	54.2$\pm$2.0	54.4$\pm$0.5
NRI (sim.)	**99.8**$\pm$0.0	59.6$\pm$0.8	–
NRI (learned)	**99.9**$\pm$0.0	**82.1**$\pm$0.6	**96.0**$\pm$0.1
Supervised	99.9$\pm$0.0	95.0$\pm$0.3	99.7$\pm$0.0
	10 objects		
Corr. (path)	50.4$\pm$0.0	51.4$\pm$0.0	59.3$\pm$0.0
Corr. (LSTM)	54.9$\pm$1.0	52.7$\pm$0.2	56.2$\pm$0.7
NRI (sim.)	**98.2**$\pm$0.0	53.7$\pm$0.8	–
NRI (learned)	**98.4**$\pm$0.0	**70.8**$\pm$0.4	**75.7**$\pm$0.3
Supervised	98.8$\pm$0.0	94.6$\pm$0.2	97.1$\pm$0.1

Physics simulations: trajectory prediction

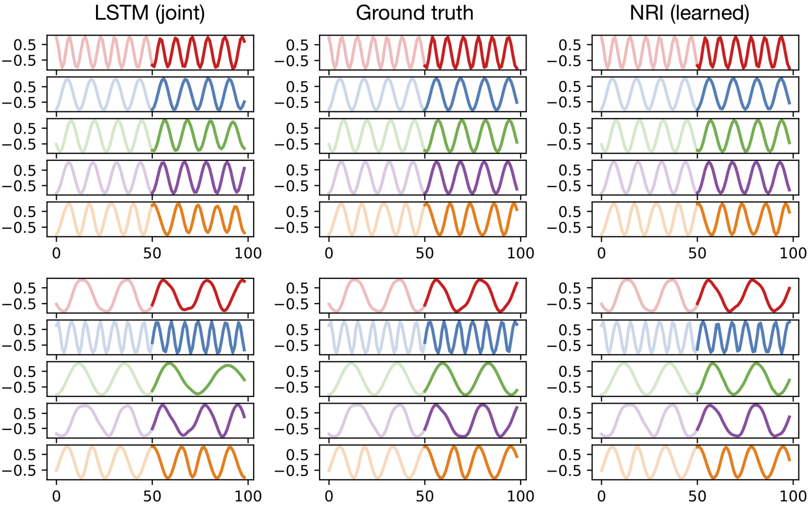
	Springs			Charged			Kuramoto		
Prediction steps	1	10	20	1	10	20	1	10	20
Static	7.93e-5	7.59e-3	2.82e-2	5.09e-3	2.26e-2	5.42e-2	5.75e-2	3.79e-1	3.39e-1
LSTM (single)	2.27e-6	4.69e-4	4.90e-3	2.71e-3	7.05e-3	1.65e-2	7.81e-4	3.80e-2	8.08e-2
LSTM (joint)	4.13e-8	2.19e-5	7.02e-4	1.68e-3	6.45e-3	1.49e-2	**3.44e-4**	**1.29e-2**	4.74e-2
NRI (full graph)	1.66e-5	1.64e-3	6.31e-3	**1.09e-3**	3.78e-3	9.24e-3	2.15e-2	5.19e-2	8.96e-2
NRI (learned)	**3.12e-8**	**3.29e-6**	**2.13e-5**	**1.05e-3**	**3.21e-3**	**7.06e-3**	1.40e-2	2.01e-2	**3.26e-2**
NRI (true graph)	1.69e-11	1.32e-9	7.06e-6	1.04e-3	3.03e-3	5.71e-3	1.35e-2	1.54e-2	2.19e-2

It might seem as if the LSTM outperforms the NRI on Kuramoto!
Qualitative analysis may show otherwise...

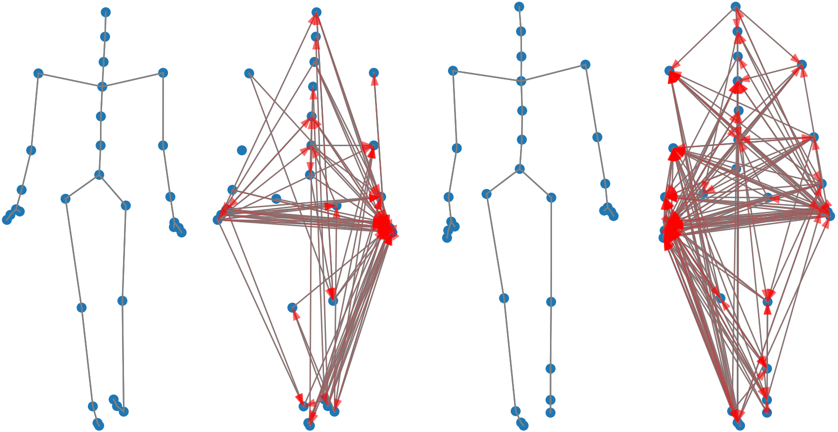
Physics simulations: qualitative results



Physics simulations: qualitative results

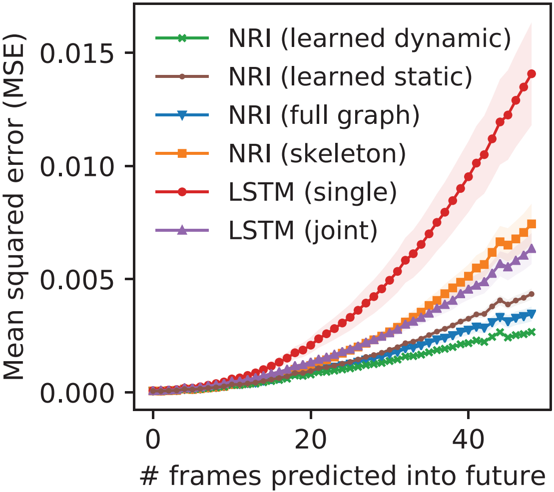


Motion capture

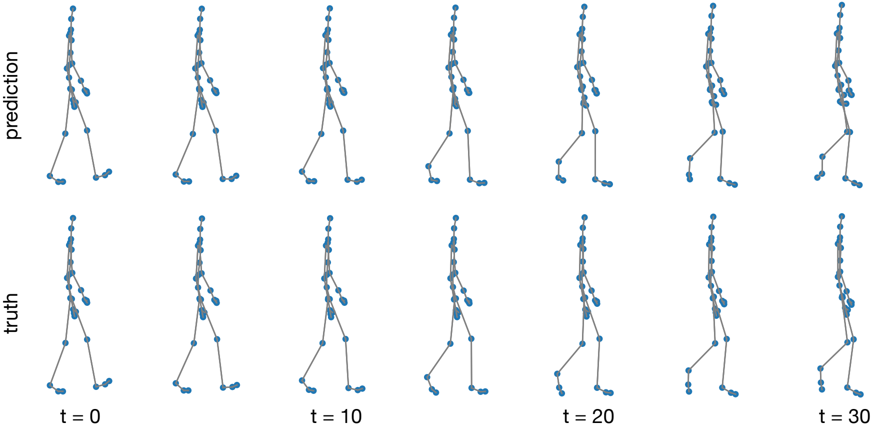


The graph is now **dynamic**! Re-evaluate at every decoding step.

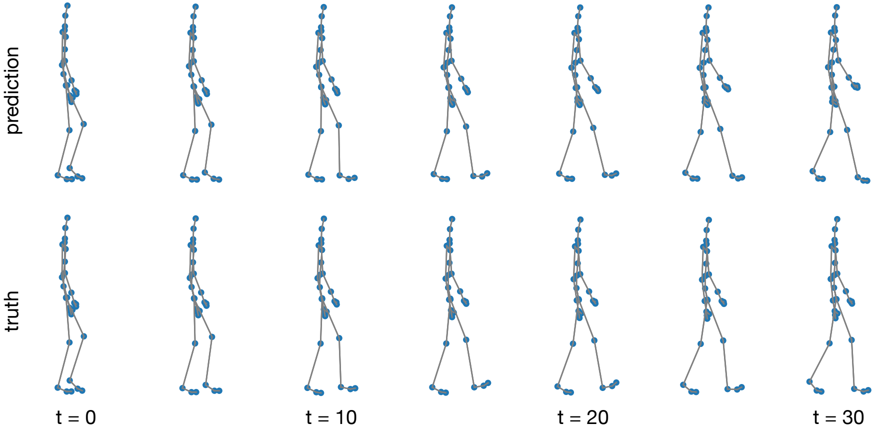
Motion capture: trajectory prediction



Motion capture: qualitative results



Motion capture: qualitative results



Concluding remarks

- The NRI is an extremely versatile model for inferring latent interaction graphs from pointwise trajectories.
- Latent graph discovery is still in its early phases of development—plentiful improvements possible!
- **Limitation:** *does not scale to large graphs!* $O(V^2)$ memory requirements, and computing edge messages makes subsampling cumbersome.
- Should not be required—most real-world graphs are sparse! But techniques we have thus far need to start with complete graph, and gradually discover sparsity...

Thank you!

Questions?

petar.velickovic@cst.cam.ac.uk

http://www.cst.cam.ac.uk/~pv273/