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
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?
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 $i$.

The interaction graph between particles will be a byproduct!
Simple baseline #1: RNN

- Let $\mathbf{x}^t$ denote the coordinates of all particles at time $t$:
  \[
  \mathbf{x}^t = [\mathbf{x}_1^t, \mathbf{x}_2^t, \ldots, \mathbf{x}_n^t]
  \]

- We can now define a recurrent neural network (e.g. LSTM or GRU) to operate on this sequential input:
  \[
  \tilde{h}^t = RNN(\tilde{h}^{t-1}, \mathbf{x}^t)
  \]

- From its hidden states, we can predict the future timesteps:
  \[
  \mathbf{x}^{t+1} = f(\tilde{h}^{t+1})
  \]

  where $f$ is an MLP.
Graph convolutional network

In a nutshell, obtain higher-level representations of a node $i$ by leveraging its neighbourhood $N_i$:

$$\vec{h}_{\ell+1}^i = g_\ell(\vec{h}_a^i, \vec{h}_b^i, \vec{h}_c^i, ..., a, b, c, ..., \in N_i)$$

where $g_\ell$ is the $\ell$-th graph convolutional layer.
Graph convolutional network

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

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

where $g^{\ell}$ is the $\ell$-th \textit{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*, $\vec{h}_i \rightarrow j$, for each edge $i \rightarrow j$ in the graph. Apply a simple MLP, $f^e_\ell$, over the features of $i$ and $j$:

$$\vec{h}^\ell_{i \rightarrow j} = f^e_\ell(\vec{h}_i^\ell, \vec{h}_j^\ell)$$
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*, $\vec{h}_{i \rightarrow j}^l$, for each edge $i \rightarrow j$ in the graph. Apply a simple MLP, $f_e^l$, over the features of $i$ and $j$:

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

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

$$
\vec{h}_j^{l+1} = f_v^l \left( \sum_{j \in \mathcal{N}_i} \vec{h}_{i \rightarrow j}^l \right)
$$
MPNN: initial setup

\[ \vec{h}_1, \vec{h}_2, \vec{h}_3, \vec{h}_4, \vec{h}_5, \vec{h}_6 \]
MPNN, computing messages

\[
\vec{h}_1^\ell \rightarrow \vec{h}_2^\ell \rightarrow \vec{h}_3^\ell \rightarrow \vec{h}_4^\ell \rightarrow \vec{h}_5^\ell \rightarrow \vec{h}_6^\ell
\]

\[f_e^\ell \rightarrow \vec{h}_3^\ell \rightarrow \vec{h}_4^\ell\]
MPNN, aggregating messages

\[\vec{h}_3^{\ell} \rightarrow h_4^{\ell} \quad \vec{h}_5^{\ell} \rightarrow h_4^{\ell} \]

\[\sum \vec{h}_2^{\ell} \rightarrow h_4^{\ell} \quad \vec{h}_6^{\ell} \rightarrow h_4^{\ell} \]

\[f_e^{\ell} \]

\[\vec{h}_1^{\ell} \quad \vec{h}_2^{\ell} \quad \vec{h}_3^{\ell} \quad \vec{h}_4^{\ell} \quad \vec{h}_5^{\ell} \quad \vec{h}_6^{\ell} \]
MPNN, computing node features

\[ \vec{h}_1^\ell \rightarrow \vec{h}_2^\ell \rightarrow \sum \rightarrow \vec{f}_v^\ell \rightarrow \vec{h}_{4}^\ell+1 \]

\[ \vec{h}_3^\ell \rightarrow \vec{h}_5^\ell \rightarrow \vec{h}_6^\ell \rightarrow \sum \rightarrow \vec{f}_e^\ell \rightarrow \vec{h}_3^\ell \rightarrow \vec{h}_4^\ell \]
MPNN, next-level features
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:

\[ \mathbf{h}_{i \rightarrow j}^{t} = f_e(\mathbf{x}_i^t, \mathbf{x}_j^t) \]

\[ \mathbf{x}_{j}^{t+1} = f_v \left( \sum_{i \neq j} \mathbf{h}_{i \rightarrow j}^{t} \right) \]

with a few kinks, specific to the trajectory predicting task...
First, to simplify the job of the network, have it only predict changes in position:

\[ \vec{h}_{i \rightarrow j}^t = f_e(\vec{x}_{i}^t, \vec{x}_{j}^t) \]

\[ \vec{x}_{j}^{t+1} = \vec{x}_{j}^t + f_v \left( \sum_{i \neq j} \vec{h}_{i \rightarrow j}^t \right) \]
Simple baseline #2: Complete graph

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

\[
\tilde{h}_{i \to j}^t = f_e(\tilde{x}_i^t, \tilde{x}_j^t)
\]

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

\[
\tilde{x}_j^{t+1} \sim \mathcal{N}(\tilde{\mu}_j^{t+1}, \sigma^2 I)
\]
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:

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

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

$$
\mu_{j}^{t+1} = \vec{x}_j^t + f_v \left( \vec{h}_{j}^{t+1} \right)
$$

$$
\vec{x}_{j}^{t+1} \sim \mathcal{N}(\mu_{j}^{t+1}, \sigma^2 I)
$$
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 $i \to j$ is of the $k$-th type.

- Assume an edge cannot have more than one type, i.e., $\tilde{Z}_{ij}$ is one-hot.
Now this graph can be exploited—define a separate MLP $f_e^k$ for each edge type. For the Markov decoder:

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

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

$$\tilde{x}_j^{t+1} \sim \mathcal{N}(\tilde{\mu}_{j}^{t+1}, \sigma^2 I)$$
Now this graph can be exploited—define a separate MLP $f^k_e$ for each edge type. For the Markov decoder:

\[
\tilde{h}^t_{i \rightarrow j} = \sum_k z_{ijk} f^k_e (\tilde{x}^t_i, \tilde{x}^t_j)
\]

\[
\tilde{\mu}^{t+1}_j = \tilde{x}^t_j + f_v \left( \sum_{i \neq j} \tilde{h}^t_{i \rightarrow j} \right)
\]

\[
\tilde{x}^{t+1}_j \sim \mathcal{N}(\tilde{\mu}^{t+1}_j, \sigma^2 I)
\]
The NRI decoder
The NRI decoder, computing messages...
The NRI decoder, computing messages...
The NRI *decoder*, computing messages...
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:

\[ \vec{h}_j^1 = f(\vec{x}_j) \]
\[ \vec{h}_{i \rightarrow j}^1 = f^1_e(\vec{h}_i^1, \vec{h}_j^1) \]
\[ \vec{h}_j^2 = f^1_v \left( \sum_{i \neq j} \vec{h}_{i \rightarrow j}^1 \right) \]
\[ \vec{h}_{i \rightarrow j}^2 = f^2_e(\vec{h}_i^2, \vec{h}_j^2) \]

\[ z_{ij} \sim \text{Categorical}(\text{softmax}(\vec{h}_{i \rightarrow j}^2)) \]

where \( f \) is an embedding, and \( f^1_e, f^1_v \) and \( f^2_e \) are MLPs.
The variational setup

- The encoder gives us the probability distribution $q(z|\vec{x})$, and the decoder gives us the probability distribution $p(\vec{x}|z)$.

- Combine learning the two in a VAE-style framework by maximising the evidence lower bound (ELBO):

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

- The prior $p(z)$ 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.
The operation of selecting $z_{ij}$ is a \textit{discrete} decision—therefore, we cannot directly propagate gradients through it.

Can use the \textbf{Gumbel softmax} trick to circumvent this:

$$\tilde{z}_{ij} = \text{softmax}((\tilde{h}_{i \rightarrow j} + \tilde{g})/\tau)$$

where $g_k \sim \text{Gumbel}(0, 1)$ and $\tau$ is a temperature parameter (converges to one-hot when $\tau \rightarrow 0$).

This is a \textit{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 $\mathbf{x}^{t+1}$ from $\mathbf{x}^t$). 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*

\[ \tilde{\mu}_j^2 = \text{decode}(\tilde{x}_j^1) \]
\[ \tilde{\mu}_j^3 = \text{decode}(\tilde{\mu}_j^2) \]
\[ \tilde{\mu}_j^4 = \text{decode}(\tilde{\mu}_j^3) \]
\[ \vdots \]
\[ \tilde{\mu}_j^{M+1} = \text{decode}(\tilde{\mu}_j^M) \]
\[ \tilde{\mu}_j^{M+2} = \text{decode}(\tilde{x}_j^{M+1}) \]
\[ \tilde{\mu}_j^{M+3} = \text{decode}(\tilde{\mu}_j^{M+2}) \]
\[ \vdots \]
Putting it all together

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

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

Legend:
- Node emb.
- Edge emb.
- MLP
- \( \mathcal{L}_{\phi} \): Concrete distribution
- Sampling

\[ \begin{align*}
    x & \to v \\
    v & \to e \\
    e & \to v \\
    v & \to e \\
    \vdots \\
    \sum & \\
    \vdots \\
    x^t & \\
    v & \to e \\
    e & \to v \\
    \Delta x^t &
\end{align*} \]
Physics simulations: latent graph discovery

<table>
<thead>
<tr>
<th>Model</th>
<th>Springs</th>
<th>Charged</th>
<th>Kuramoto</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5 objects</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Corr. (path)</td>
<td>52.4±0.0</td>
<td>55.8±0.0</td>
<td>62.8±0.0</td>
</tr>
<tr>
<td>Corr. (LSTM)</td>
<td>52.7±0.9</td>
<td>54.2±2.0</td>
<td>54.4±0.5</td>
</tr>
<tr>
<td>NRI (sim.)</td>
<td>99.8±0.0</td>
<td>59.6±0.8</td>
<td>̶</td>
</tr>
<tr>
<td>NRI (learned)</td>
<td>99.9±0.0</td>
<td>82.1±0.6</td>
<td>96.0±0.1</td>
</tr>
<tr>
<td><strong>Supervised</strong></td>
<td>99.9±0.0</td>
<td>95.0±0.3</td>
<td>99.7±0.0</td>
</tr>
<tr>
<td><strong>10 objects</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Corr. (path)</td>
<td>50.4±0.0</td>
<td>51.4±0.0</td>
<td>59.3±0.0</td>
</tr>
<tr>
<td>Corr. (LSTM)</td>
<td>54.9±1.0</td>
<td>52.7±0.2</td>
<td>56.2±0.7</td>
</tr>
<tr>
<td>NRI (sim.)</td>
<td>98.2±0.0</td>
<td>53.7±0.8</td>
<td>̶</td>
</tr>
<tr>
<td>NRI (learned)</td>
<td>98.4±0.0</td>
<td>70.8±0.4</td>
<td>75.7±0.3</td>
</tr>
<tr>
<td><strong>Supervised</strong></td>
<td>98.8±0.0</td>
<td>94.6±0.2</td>
<td>97.1±0.1</td>
</tr>
</tbody>
</table>
Physics simulations: trajectory prediction

<table>
<thead>
<tr>
<th>Prediction steps</th>
<th>Springs</th>
<th>Charged</th>
<th>Kuramoto</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Static</td>
<td>7.93e-5</td>
<td>7.59e-3</td>
<td>2.82e-2</td>
</tr>
<tr>
<td>LSTM (single)</td>
<td>2.27e-6</td>
<td>4.69e-4</td>
<td>4.90e-3</td>
</tr>
<tr>
<td>LSTM (joint)</td>
<td>4.13e-8</td>
<td>2.19e-5</td>
<td>7.02e-4</td>
</tr>
<tr>
<td>NRI (full graph)</td>
<td>1.66e-5</td>
<td>1.64e-3</td>
<td>6.31e-3</td>
</tr>
<tr>
<td>NRI (learned)</td>
<td>3.12e-8</td>
<td>3.29e-6</td>
<td>2.13e-5</td>
</tr>
<tr>
<td>NRI (true graph)</td>
<td>1.69e-11</td>
<td>1.32e-9</td>
<td>7.06e-6</td>
</tr>
</tbody>
</table>

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

![Graph showing mean squared error (MSE) for different methods over # frames predicted into future.](image)
Motion capture: qualitative results

prediction

truth

$\begin{align*}
t &= 0 \\
t &= 10 \\
t &= 20 \\
t &= 30
\end{align*}$
Motion capture: qualitative results

prediction vs truth at different time steps:

- **t = 0**
- **t = 10**
- **t = 20**
- **t = 30**
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/