Unsupervised Learning with Graph Neural Networks

Petar Veličković

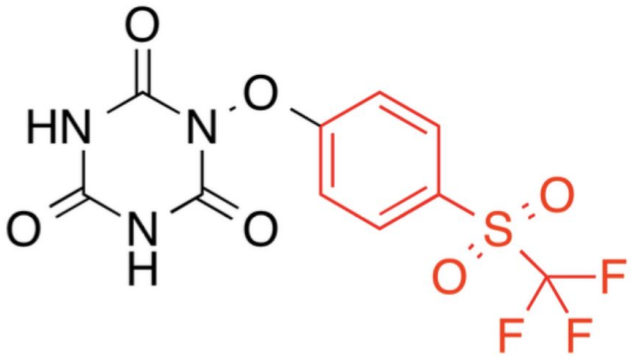
ACDL 2019 Satellite Workshop on Graph Neural Networks
22 July 2019

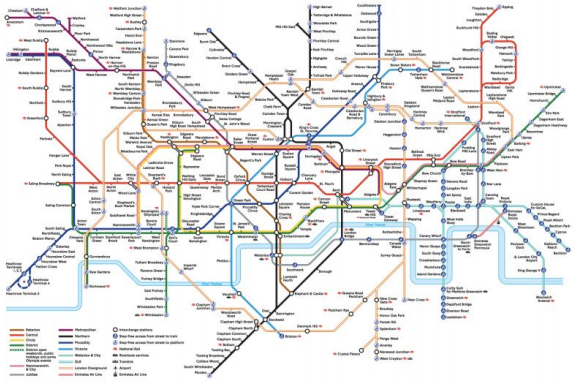
Motivation

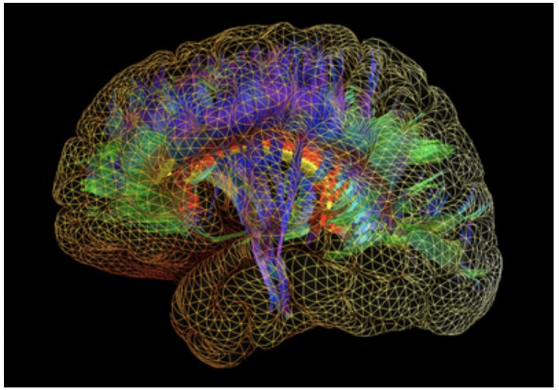
- In this talk: using **graph neural networks** for **unsupervised learning**.
- Very important direction: *most graphs in the wild are unlabeled!*
  - However, still largely understudied.
- Outline:
  - Graph neural networks recap (**GCN**/**GAT**/**MPNN**);
  - Random-walk objectives (**VGAE**);
  - Mutual information maximisation (**DGI**);
  - Latent graph inference (**NRI**).

Graph Neural Networks

Graphs are **everywhere!**



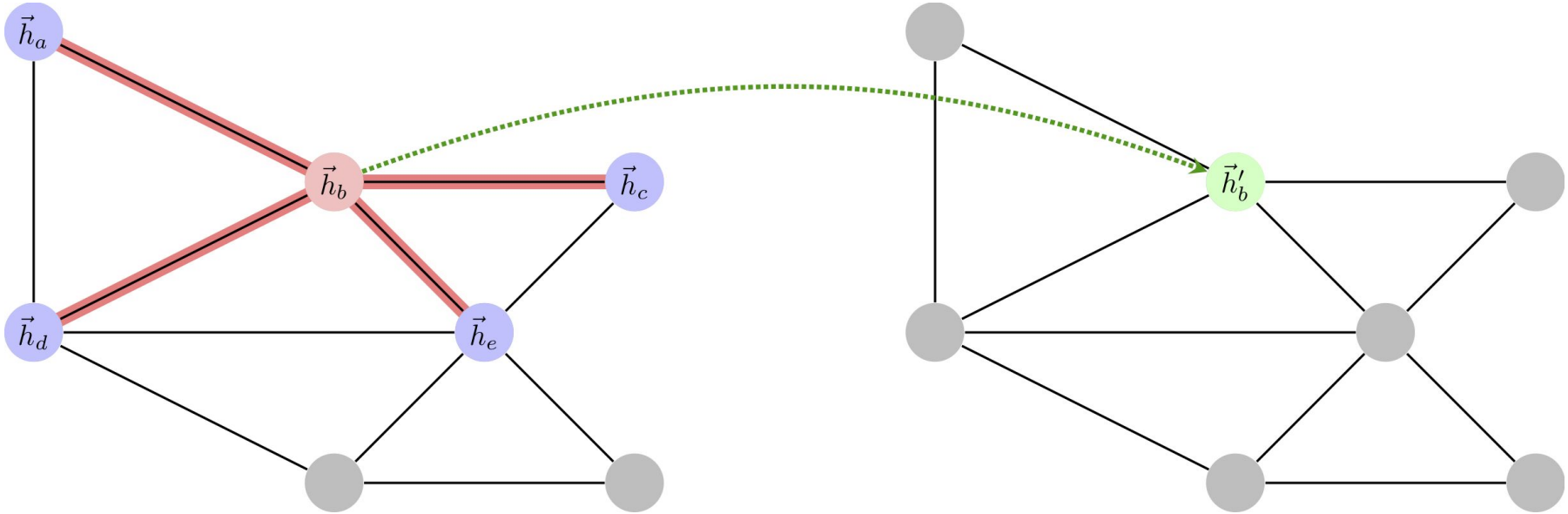




Mathematical setup

- **Graph**: $G = (V, E)$
- **Node features**: $\mathbf{H} = \{\vec{h}_1, \vec{h}_2, \ldots, \vec{h}_N\}; \qquad \vec{h}_i \in \mathbb{R}^F$
- **Adjacency matrix**: $\mathbf{A} \in \mathbb{R}^{N \times N}$
- **Neighbourhoods**: $\mathcal{N}_i = \{j \mid i = j \vee \mathbf{A}_{ij} \neq 0\}$
- **(Edge features)**: $\vec{e}_{ij} \in \mathbb{R}^{F'}; \qquad (i, j) \in E$

Graph *Convolutional* Network



$$\vec{h}'_i = g(\vec{h}_a, \vec{h}_b, \vec{h}_c, \dots) \qquad (a, b, c, \dots \in \mathcal{N}_i)$$

Message-passing neural networks

- Want **permutation invariance** => *aggregate* (e.g. sum) across $\mathcal{N}_i$.
- Generic graph neural network (GNN) layer usually expressed as:

$$\vec{h}'_i = \sigma\left(\sum_{i \in \mathcal{N}_j} f(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})\right)$$

  where *f* is a (learnable) function, computing **message** from j to i.

- Message-passing neural network (**MPNN**; Gilmer *et al*., ICML 2017).

Simpler GNN variants

- Graph convolutional network (**GCN**; Kipf and Welling, ICLR 2017):

$$\vec{h}'_i = \sigma\left(\sum_{i \in \mathcal{N}_j} \frac{1}{\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}} \mathbf{W}\vec{h}_j\right)$$

- Graph attention network (**GAT**; Veličković *et al.*, ICLR 2018):

$$\vec{h}'_i = \sigma\left(\sum_{i \in \mathcal{N}_j} a(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})\mathbf{W}\vec{h}_j\right)$$

Random-walk objectives

Unsupervised learning setup

- The GNN parameters can be optimised by a **supervision** signal.
- But what if we have no **labels** / task is **unknown**?
- Want: ***"good"*** node* features that adapt to new tasks well.

*if we have good node features, can easily derive good edge/graph features if required.

Structure preservation

- What is in a “good” representation?
- Graphs carry interesting **structure**!
  - Good node representations should preserve it.
- Simplest notion of graph structure is an *edge*.
  - Features of node *i* and *j* should be predictive of existence of edge *(i,j)*!
- Yields a straightforward unsupervised objective...

Link prediction

- Hold out some edges from the graph.
- Use a GNN to obtain node representations, $\vec{h}_i$.
- Optimise the GNN such that $\vec{h}_i$ and $\vec{h}_j$ are predictive of whether $(i, j) \in E$:

$$\sum_{(i,j)\in E} \log \sigma(\vec{h}_i^T \vec{h}_j) + \sum_{(i,j)\notin E} \log\left(1 - \sigma(\vec{h}_i^T \vec{h}_j)\right)$$

- (Variational) graph auto-encoders (**VGAE**; Kipf and Welling, NIPS BDL 2016)

Random-walk objectives

- The link-prediction objective is a special case of **random-walk objectives.**
- Predictive of whether *i* and *j* co-occur in a (short) random walk.
- **Dominated** unsupervised graph representation learning prior to GNNs!
  - DeepWalk, node2vec, LINE, ...
- Combined with GNNs in **GraphSAGE** (Hamilton et al., NIPS 2017).

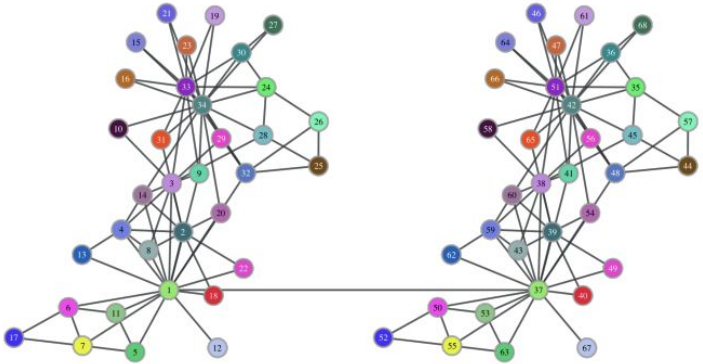
Mutual information maximisation

Combining GNNs and local objectives

- Random-walk objectives inherently capture **local** similarities.
- But a GNN *summarises local patches* of the graph!
  - Neighbouring nodes tend to highly overlap in n-step neighbourhoods;
  - Therefore, a GNN *enforces* similar features for neighbouring nodes by **design**.
- Random walk-objectives can fail to provide useful signal to GNNs!
  - At times, matched by a **random-init** GNN! (Veličković et al., ICLR 2019)
- We require a move to *global objectives*...

Global unsupervised objectives

- Want the model to discover interesting structural similarities *anywhere*!



- **Hypothesis**: similar *structural roles* => likely to be related
  - regardless of path distance!

Mutual information maximisation

- A natural way to capture similarities is optimising **mutual information**.
- Attempt to simultaneously optimise $MI(\vec{h}_i, \vec{h}_j)$ for all pairs of nodes...
  - The model will “discover” which pairs can be best compressed.
- Scalable proxy: optimise $MI(\vec{h}_i, \vec{s})$, where $\vec{s}$ is a graph summary vector.
- Yielded great returns on **images** (**Deep InfoMax**; Hjelm et al., ICLR 2019).

How to optimise MI?

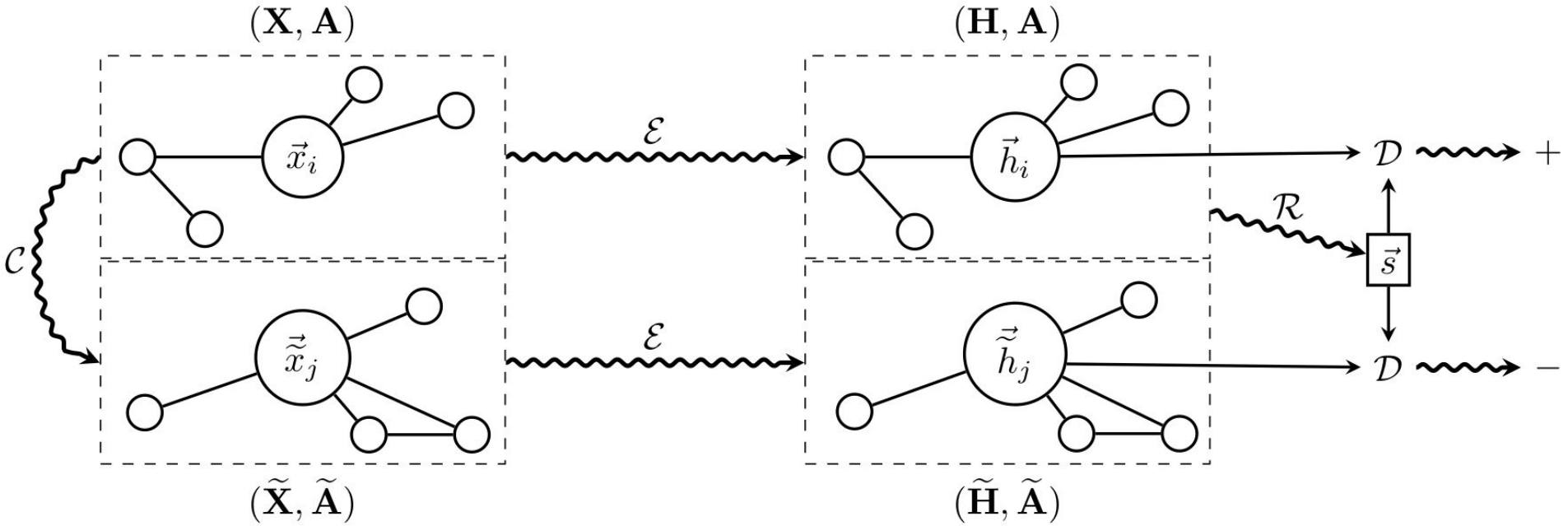
- In neural networks, MI typically optimised using a *discriminator*.
  - Binary classifier, determining whether $\vec{h}_i$ and $\vec{s}$ are related.
  - e.g. “Does node *i* belong in the graph summarised by $\vec{s}$?”

- Requires *negative examples*: nodes not belonging to the graph.
  - But often only have one graph!

- Solution: explicit **corruption function** to produce *negative graphs*.

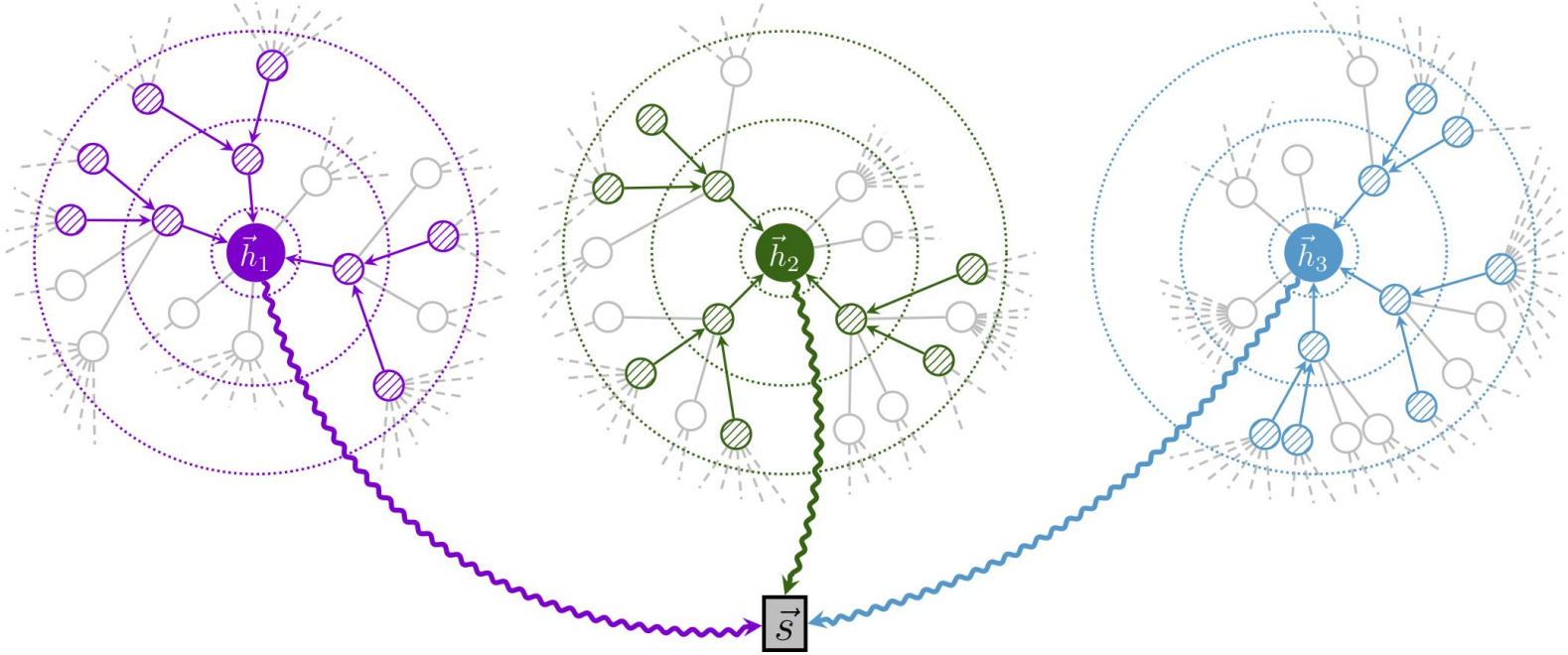
$$\frac{1}{N+M}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\sum_{i=1}^{N}\log\mathcal{D}\left(\vec{h}_i,\vec{s}\right)+\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\sum_{j=1}^{M}\log\left(1-\mathcal{D}\left(\vec{\widetilde{h}}_j,\vec{s}\right)\right)\right]\right]$$

Deep Graph Infomax (DGI)



Veličković *et al.*, ICLR 2019

Scaling to large graphs

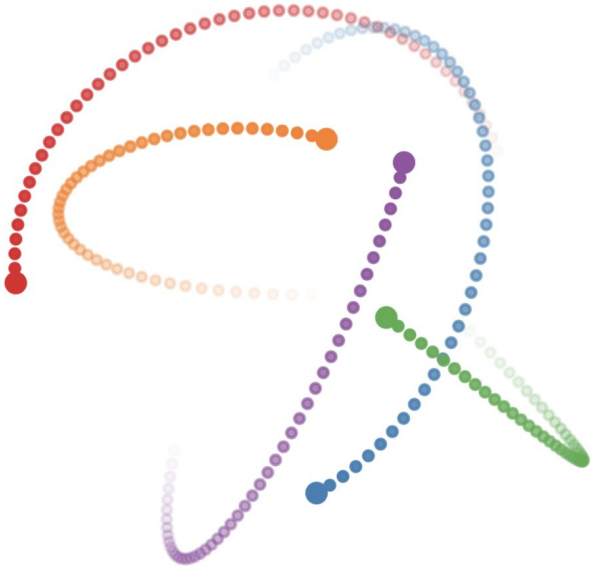


Competitive results when combined with GraphSAGE-style subsampling

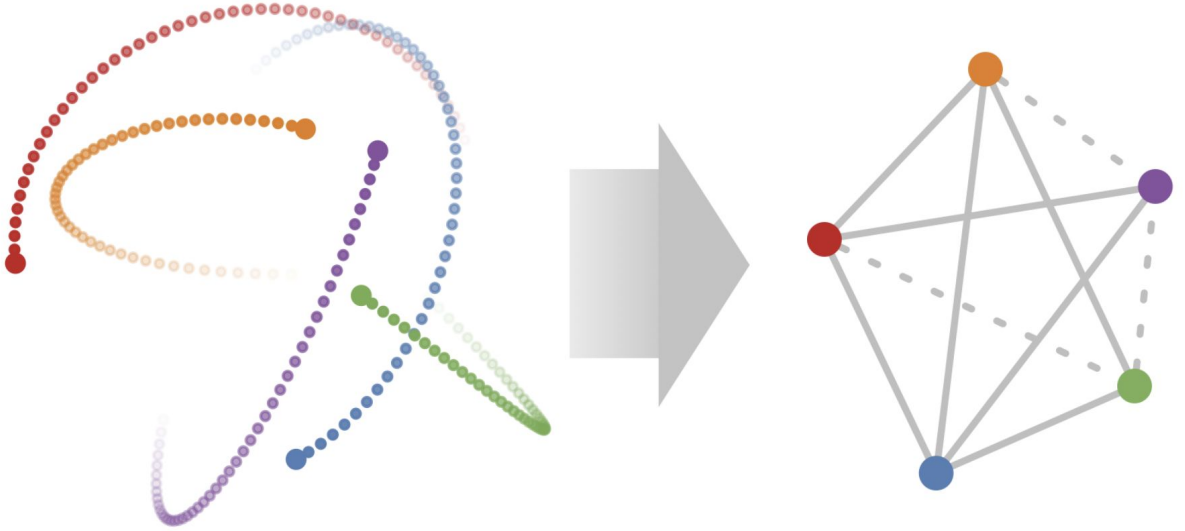
Latent graph inference

Latent graph inference

- All techniques so far require a graph to be **provided** as input…
- But can we always *see* it?
- Could use a fully-connected graph, but…
  - not representative of interaction “modes”;
  - does not scale!
- **Latent graph inference** represents a key challenge for graph neural networks to tackle.



Problem setup

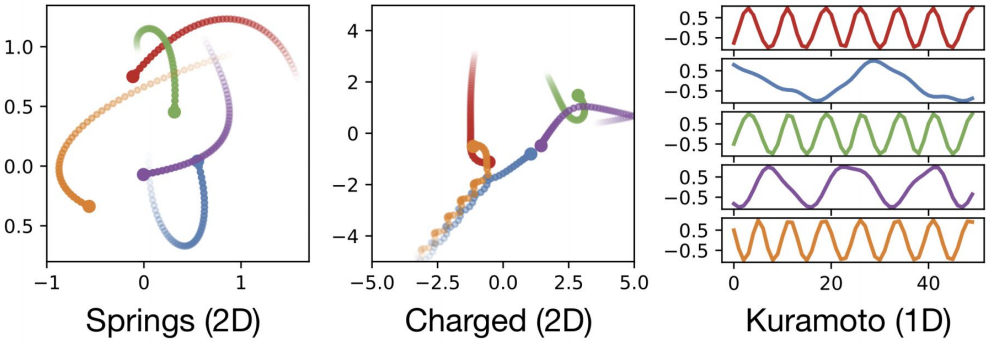


Neural Relational Inference (**NRI**; Kipf, Fetaya *et al.*, ICML 2018)

Learning the interaction graph

- Use a **downstream task** to drive graph construction:
  - First, infer the graph (**encoder** network);
  - Then, use the graph to make predictions (**decoder** network).
  - Optimise the entire system end-to-end through predictive loss.

- In the NRI paper:
  - Inputs are particle **trajectories**;
  - Want to predict **future positions**.
  - Infer **physics interactions**.



The NRI encoder

- Assume upfront **K** different interaction types
  - (e.g. spring, charged, none…)

- Run an MPNN over the complete graph
  - This will yield node as well as **edge** features, $\vec{m}_{ij} = f(\vec{h}_i, \vec{h}_j)$

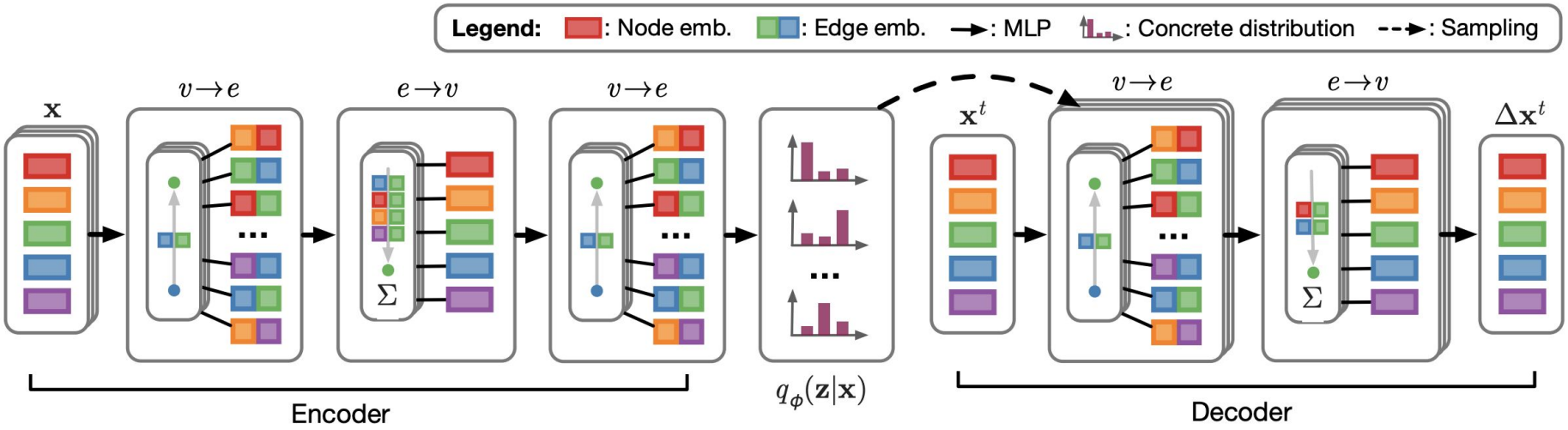
- Learn a K-way classifier over each edge to infer interaction type
  - Sample from its predictions to obtain the latent graph

$$e_{ij} \sim \text{Categorical}(\text{softmax}(\mathbf{W}\vec{m}_{ij}))$$

The NRI decoder

- Run another MPNN over the latent graph
- Predict the desired output using the node representations
- Optimise the entire system through gradient descent on the output error
  - (e.g. MSE in the case of particle trajectories)
- Problem: cannot backpropagate through the sampling operation!
  - Use the **Gumbel softmax** trick at training time.

The NRI architecture



Concluding remarks

- NRI is an important step towards latent graph inference
- Main limitation: does not scale to large graphs!
  - Still requires message passing on a complete graph…
  - In practice, should not be required!
- Plentiful improvements possible, as well as necessary

Thank you!

Questions?