# EECE 571F: Deep Learning with Structures

## Lecture 12: Learning Latent Graph Structures

Renjie Liao

University of British Columbia

Winter, Term 1, 2022

# Course Scope

- Brief Intro to Deep Learning

- Geometric Deep Learning

    - Deep Learning Models for Sets and Sequences: Deep Sets & Transformers
    - Deep Learning Models for Graphs: Graph Convolution & Message Passing GNNs
    - Expressiveness & Generalizations of GNNs
    - Unsupervised/Self-supervised Graph Representation Learning

- Probabilistic Deep Learning

    - Deep Generative Models:
      Auto-regressive models, GANs, VAEs, Diffusion/Score based models
    - Discrete/Hybrid Latent Variable Models: RBMs, Latent Graph Models
    - Stochastic Gradient Estimation

# Course Scope

- Brief Intro to Deep Learning

- Geometric Deep Learning

    - Deep Learning Models for Sets and Sequences: Deep Sets & Transformers
    - Deep Learning Models for Graphs: Graph Convolution & Message Passing GNNs
    - Expressiveness & Generalizations of GNNs
    - Unsupervised/Self-supervised Graph Representation Learning

- Probabilistic Deep Learning

    - Deep Generative Models:
      Auto-regressive models, GANs, VAEs, Diffusion/Score based models
    - Discrete/Hybrid Latent Variable Models: RBMs, **Latent Graph Models**
    - Stochastic Gradient Estimation

# Contents

Learning Latent Graphs for Deep Probabilistic Models

- Neural Relational Inference
- Learning Latent Graphs via Bi-level Optimization

# Neural Relational Inference

Suppose we observe dynamics of particles, we are interested in inferring the latent interaction graph

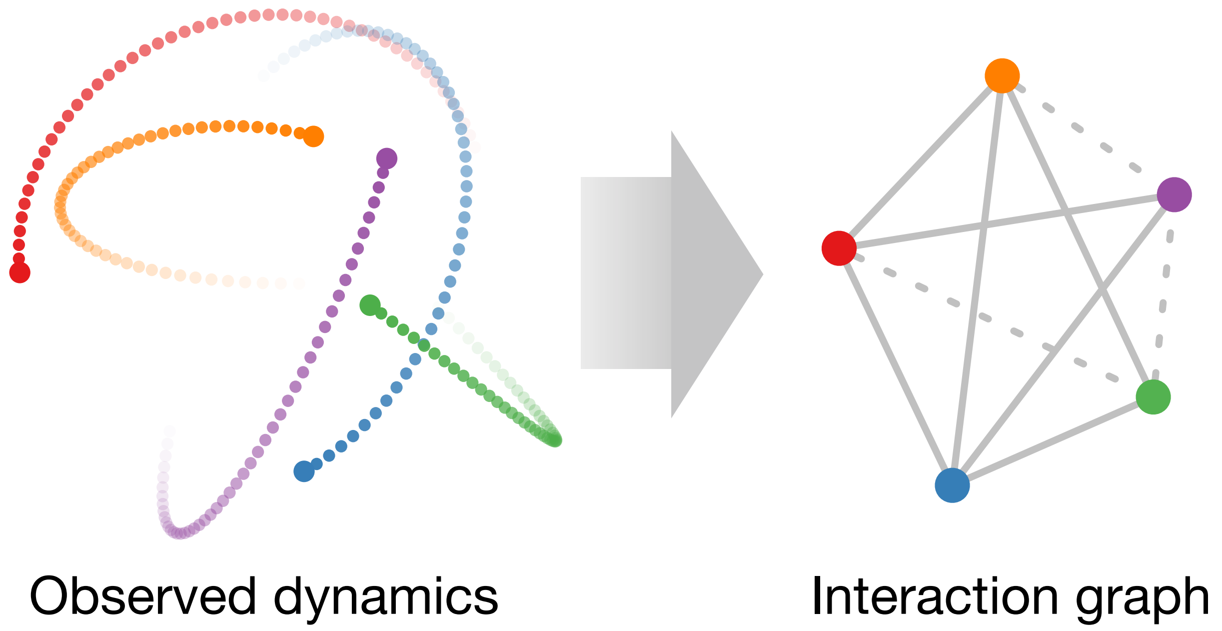



Image Credit: [1]

# Neural Relational Inference

Suppose we observe dynamics of particles, we are interested in inferring the latent interaction graph

It arises in dynamic systems from physics, biology, sports, transportation, etc.

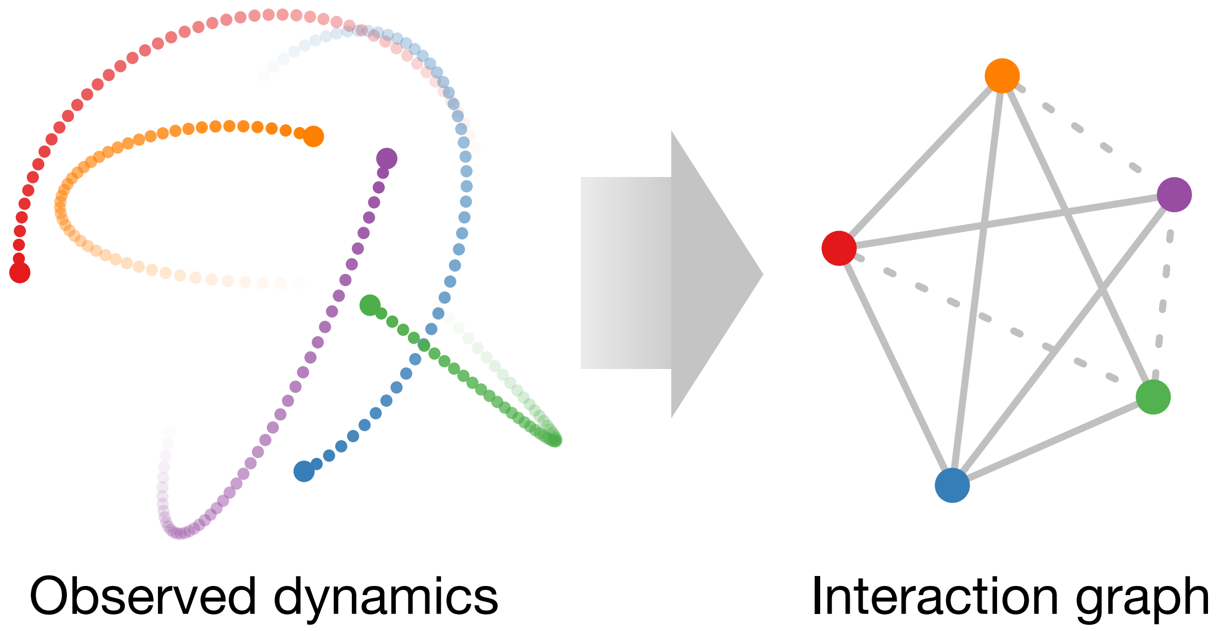



Image Credit: [1]

# Neural Relational Inference

Let us formalize the problem:

We have N particles (nodes) $\mathcal{V} = \{v_1, ..., v_N\}$

At time t, the feature is $\mathbf{x}^t = \{\mathbf{x}_1^t, ..., \mathbf{x}_N^t\}$

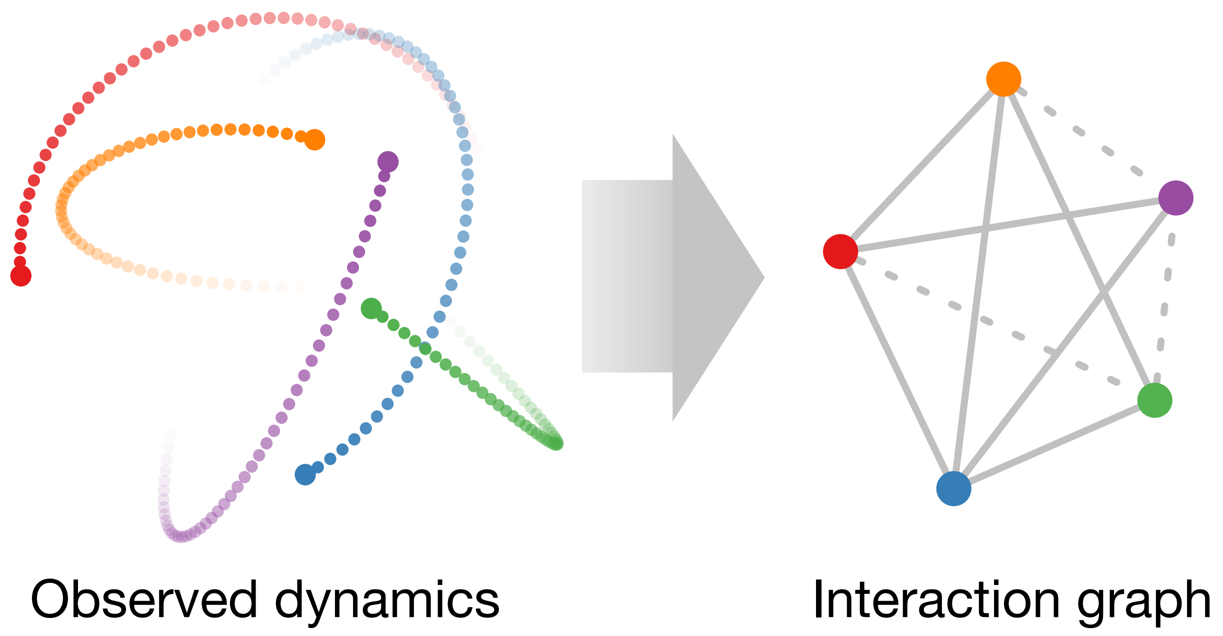



Image Credit: [1]

# Neural Relational Inference

Let us formalize the problem:

We have N particles (nodes) $\mathcal{V} = \{v_1, ..., v_N\}$

At time t, the feature is $\mathbf{x}^t = \{\mathbf{x}_1^t, ..., \mathbf{x}_N^t\}$

For all nodes, we have N trajectories

$$\mathbf{x} = (\mathbf{x}^1, ..., \mathbf{x}^T)$$

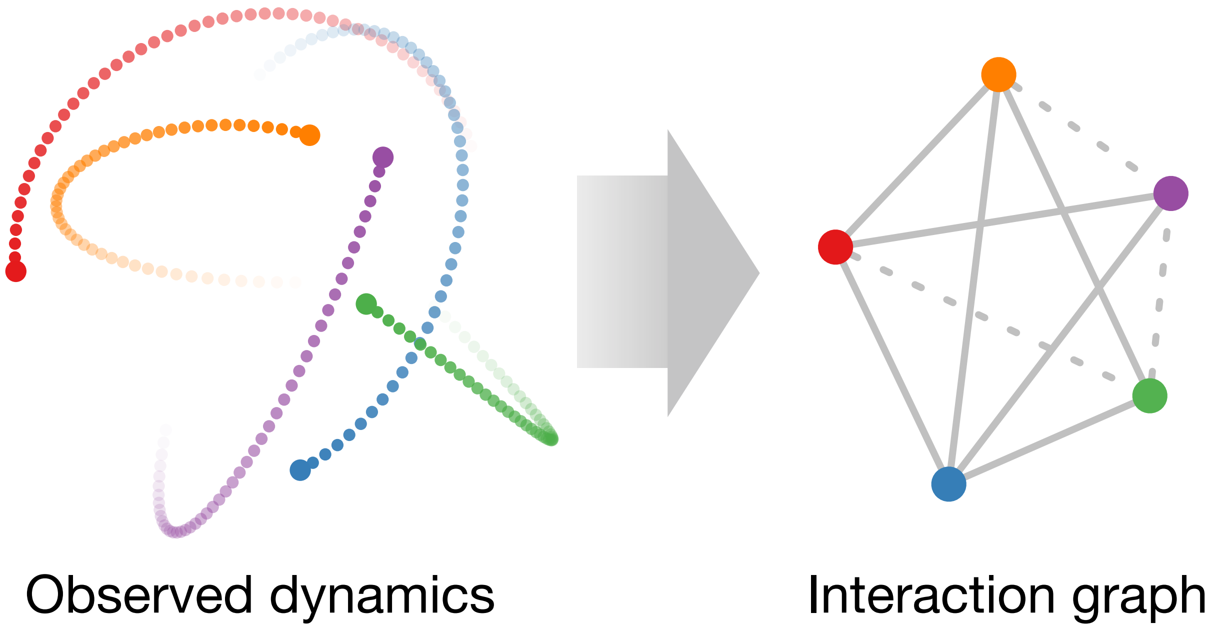



Image Credit: [1]

# Neural Relational Inference

Let us formalize the problem:

We have N particles (nodes) $\mathcal{V} = \{v_1, ..., v_N\}$

At time t, the feature is $\mathbf{x}^t = \{\mathbf{x}_1^t, ..., \mathbf{x}_N^t\}$

For all nodes, we have N trajectories

$$\mathbf{x} = (\mathbf{x}^1, ..., \mathbf{x}^T)$$

For any pair of node $(v_i, v_j)$ , we introduce

a discrete latent variable $\mathbf{z}_{ij}$ to model interaction

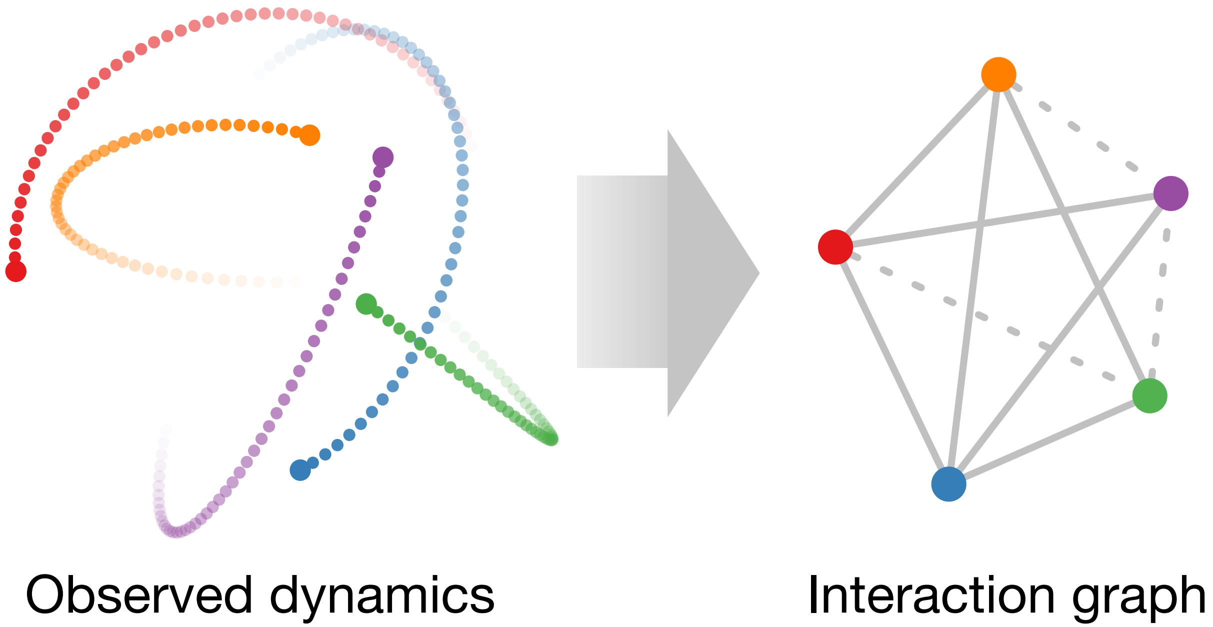



Image Credit: [1]

# Neural Relational Inference

Let us formalize the problem:

We have N particles (nodes) $\mathcal{V} = \{v_1, ..., v_N\}$

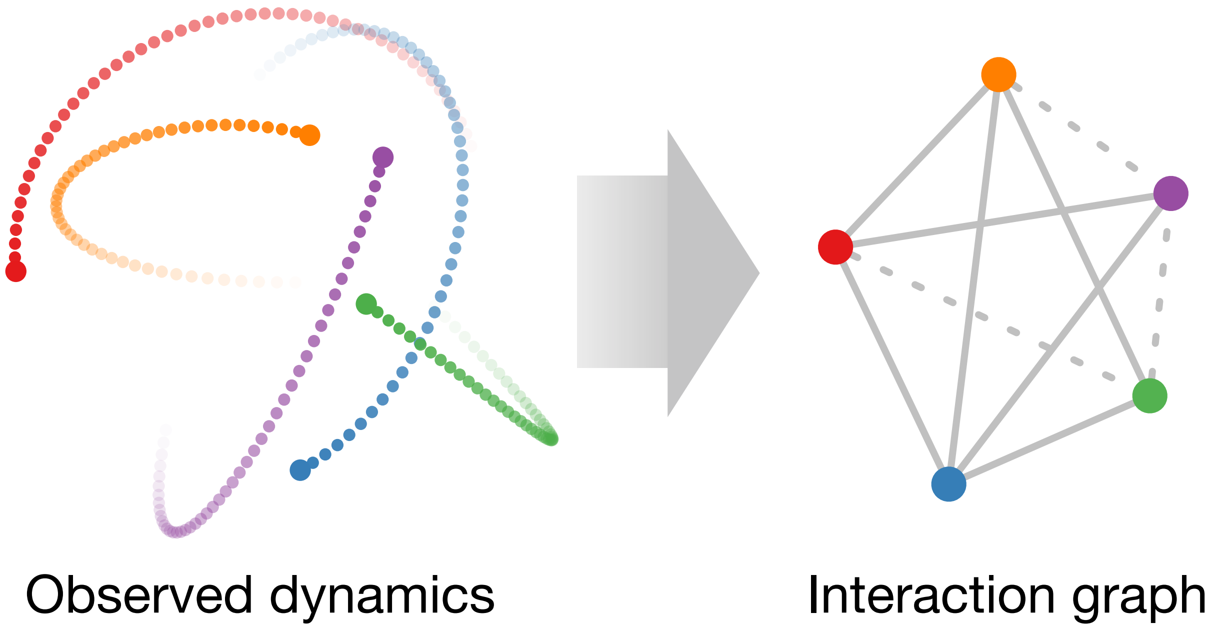
At time t, the feature is $\mathbf{x}^t = \{\mathbf{x}_1^t, ..., \mathbf{x}_N^t\}$

For all nodes, we have N trajectories

$$\mathbf{x} = (\mathbf{x}^1, ..., \mathbf{x}^T)$$

For any pair of node $(v_i, v_j)$ , we introduce

a discrete latent variable $\mathbf{z}_{ij}$ to model interaction




Our goal is to infer the set of all latent variables , which forms the latent graph!

Image Credit: [1]

# Neural Relational Inference

We use VAEs as the probabilistic framework

# Neural Relational Inference

We use VAEs as the probabilistic framework

- Encoder $q_\phi(\mathbf{z}|\mathbf{x})$
- Decoder $p_\theta(\mathbf{x}|\mathbf{z})$
- Prior $p(\mathbf{z})$

# Neural Relational Inference

We use VAEs as the probabilistic framework

- Encoder $q_\phi(\mathbf{z}|\mathbf{x})$
- Decoder $p_\theta(\mathbf{x}|\mathbf{z})$
- Prior $p(\mathbf{z})$
- Learning the model by maximizing the ELBO

$$\mathcal{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})]$$

# Neural Relational Inference

We use VAEs as the probabilistic framework

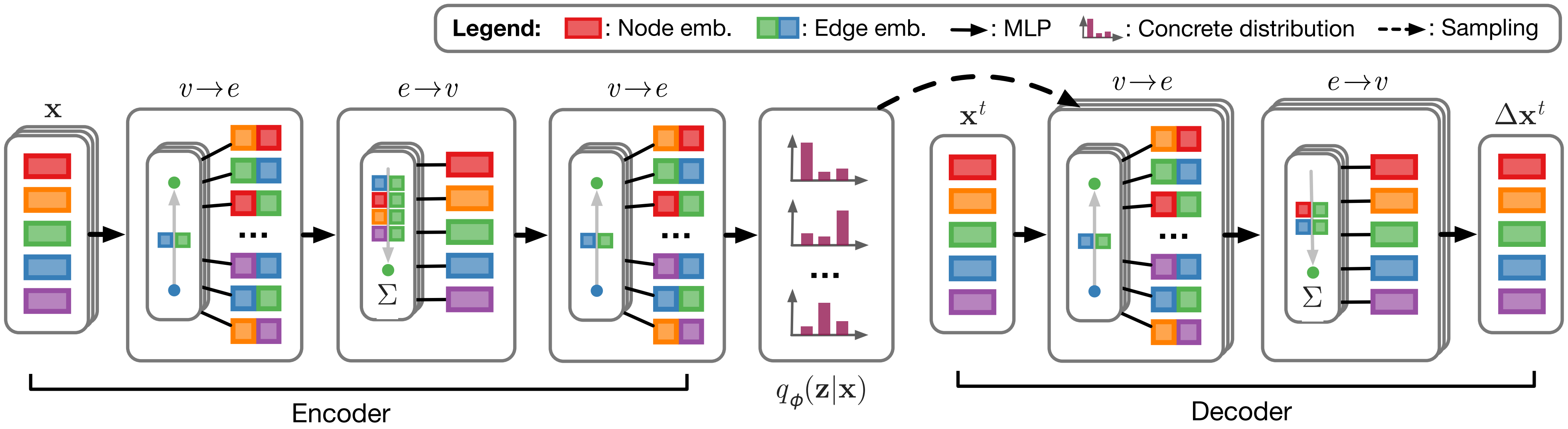



Image Credit: [1]

# Neural Relational Inference

Encoder: A GNN applied to a fully connected graph

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{ij} q_\phi(\mathbf{z}_{ij}|\mathbf{x})$$

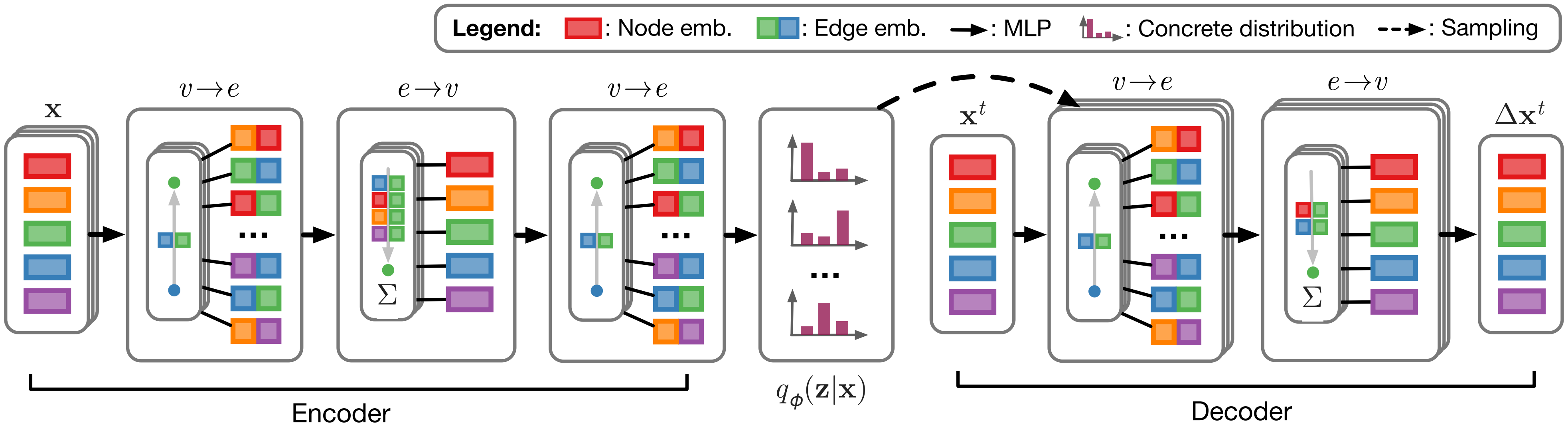
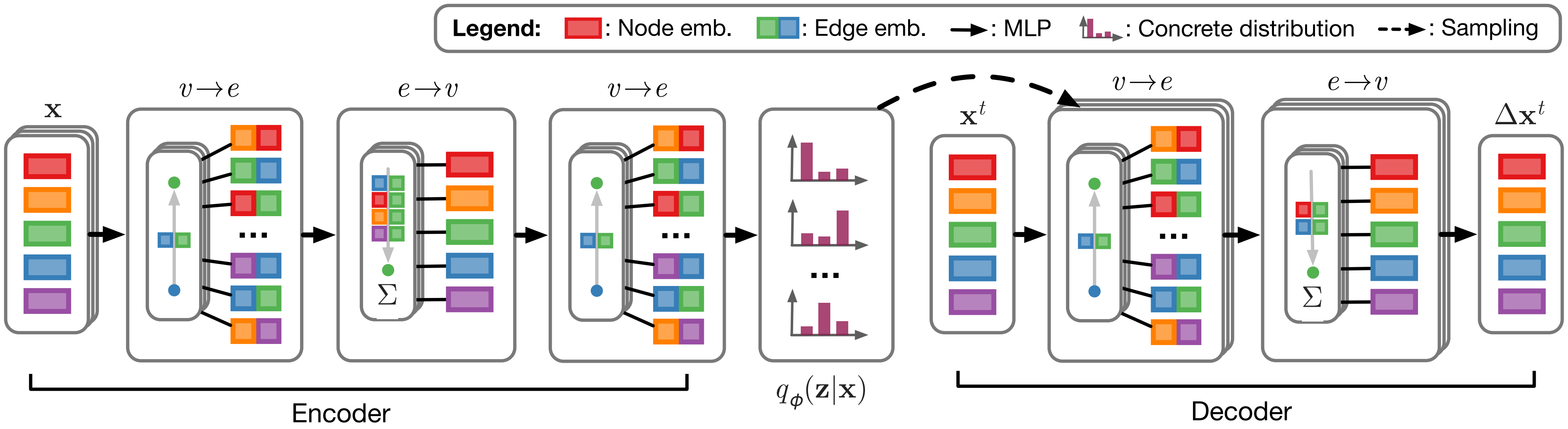



Image Credit: [1]

# Neural Relational Inference

Encoder: A GNN applied to a fully connected graph

$$q_\phi(\mathbf{z}|\mathbf{x}) = \prod_{ij} q_\phi(\mathbf{z}_{ij}|\mathbf{x})$$




$$\mathbf{h}_j^1 = f_{\text{emb}}(\mathbf{x}_j)$$

Node to Edge $v{\to}e$ : $\mathbf{h}^1_{(i,j)} = f_e^1([\mathbf{h}_i^1, \mathbf{h}_j^1])$

Edge to Node $e{\to}v$ : $\mathbf{h}_j^2 = f_v^1(\sum_{i \neq j} \mathbf{h}^1_{(i,j)})$

Node to Edge $v{\to}e$ : $\mathbf{h}^2_{(i,j)} = f_e^2([\mathbf{h}_i^2, \mathbf{h}_j^2])$

Readout $q_\phi(\mathbf{z}_{ij}|\mathbf{x}) = \text{softmax}(\mathbf{h}^2_{(i,j)})$

Image Credit: [1]

# Neural Relational Inference

Decoder: A GNN applied to the sampled graph

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z})$$

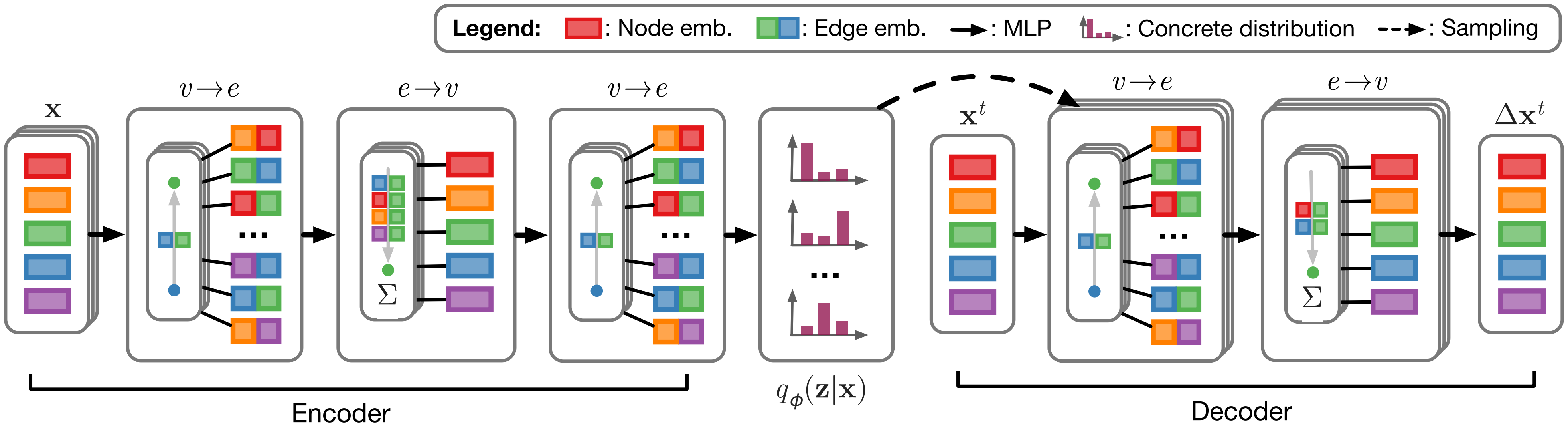
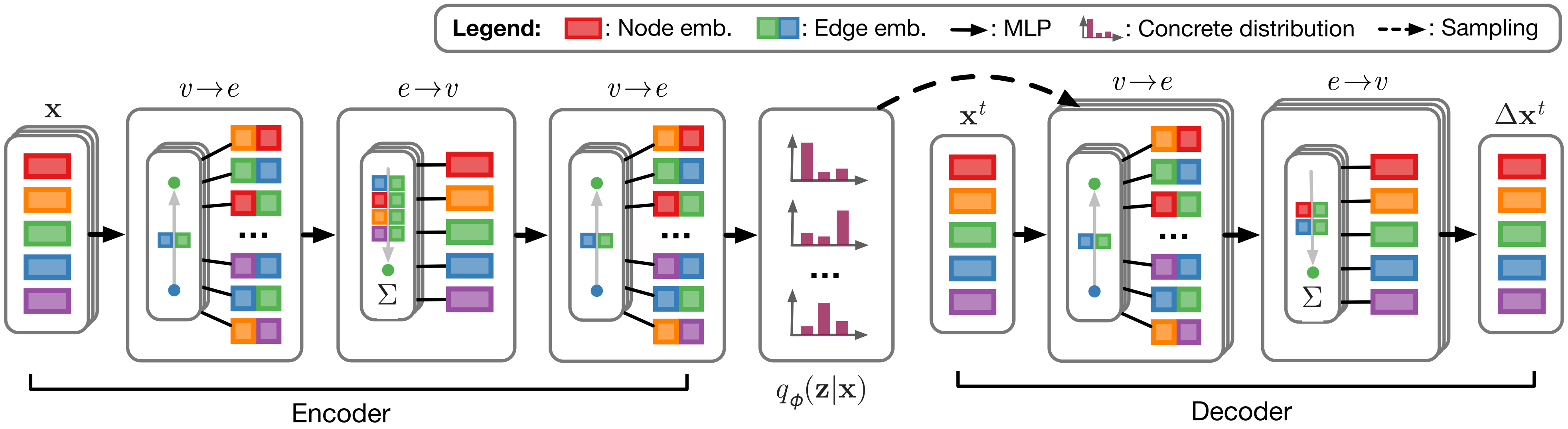



Image Credit: [1]

# Neural Relational Inference

Decoder: A GNN applied to the sampled graph

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z})$$




Option I: Markovian

$$p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z}) = p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, \mathbf{z})$$

Node to Edge $\quad v \to e: \quad \tilde{\mathbf{h}}^t_{(i,j)} = \sum_k z_{ij,k} \tilde{f}^k_e([\mathbf{x}^t_i, \mathbf{x}^t_j])$

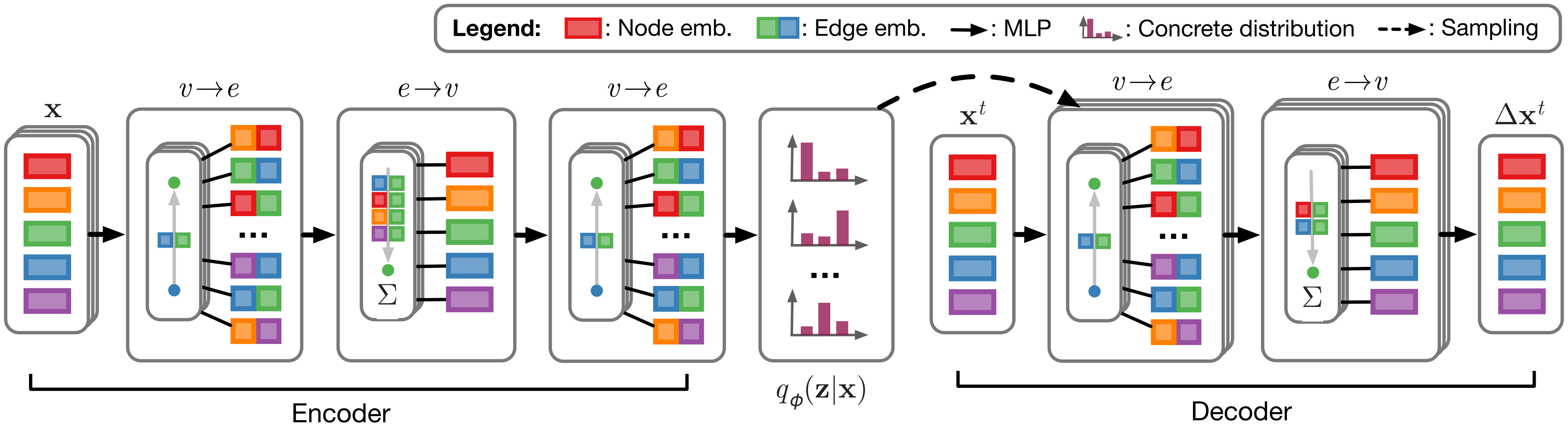
Edge to Node $\quad e \to v: \quad \boldsymbol{\mu}^{t+1}_j = \mathbf{x}^t_j + \tilde{f}_v(\sum_{i \neq j} \tilde{\mathbf{h}}^t_{(i,j)})$

$$p(\mathbf{x}^{t+1}_j|\mathbf{x}^t, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}^{t+1}_j, \sigma^2\mathbf{I})$$

Image Credit: [1]

# Neural Relational Inference

Decoder: A GNN applied to the sampled graph $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z})$




Option II: Auto-Regressive

Node to Edge $v{\to}e: \quad \tilde{\mathbf{h}}^t_{(i,j)} = \sum_k z_{ij,k} \tilde{f}^k_e([\tilde{\mathbf{h}}^t_i, \tilde{\mathbf{h}}^t_j])$

Edge to Node $e{\to}v: \quad \mathrm{MSG}^t_j = \sum_{i\neq j} \tilde{\mathbf{h}}^t_{(i,j)}$

$$\tilde{\mathbf{h}}^{t+1}_j = \mathrm{GRU}([\mathrm{MSG}^t_j, \mathbf{x}^t_j], \tilde{\mathbf{h}}^t_j)$$

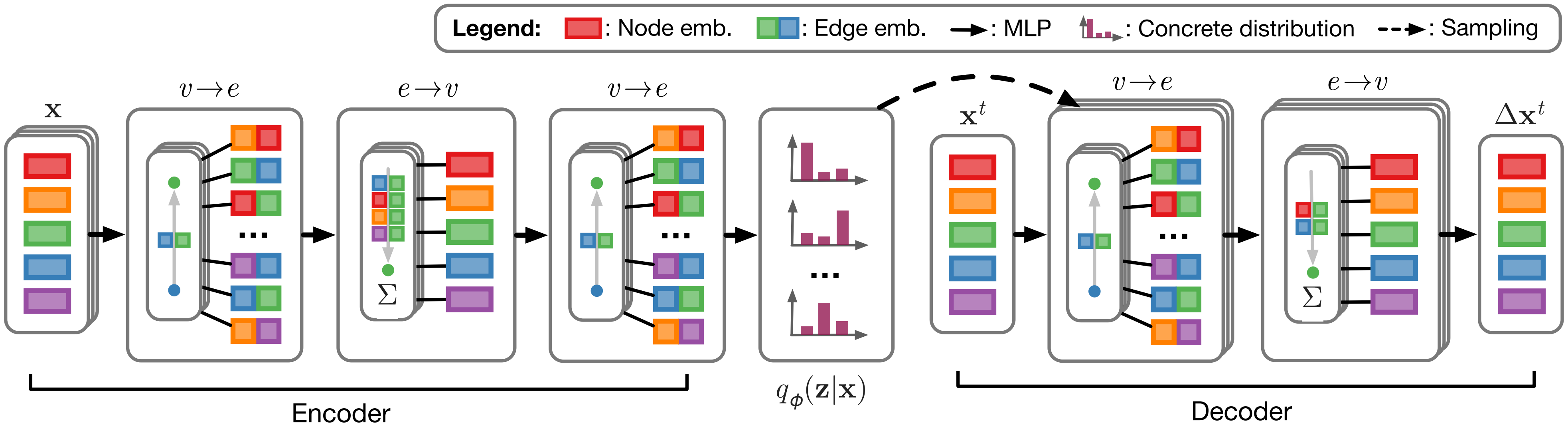
$$\boldsymbol{\mu}^{t+1}_j = \mathbf{x}^t_j + f_{\mathrm{out}}(\tilde{\mathbf{h}}^{t+1}_j)$$

$$p(\mathbf{x}^{t+1}|\mathbf{x}^t, ..., \mathbf{x}^1, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}^{t+1}, \sigma^2\mathbf{I})$$

Image Credit: [1]

# Neural Relational Inference

Decoder: A GNN applied to the sampled graph $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, \ldots, \mathbf{x}^1, \mathbf{z})$




Option II: Auto-Regressive

To avoid degenerated decoder:

- One message network per edge type
- Predict multiple futures

Node to Edge $v{\to}e: \quad \tilde{\mathbf{h}}^t_{(i,j)} = \sum_k z_{ij,k} \tilde{f}^k_e([\tilde{\mathbf{h}}^t_i, \tilde{\mathbf{h}}^t_j])$

Edge to Node $e{\to}v: \quad \mathrm{MSG}^t_j = \sum_{i \neq j} \tilde{\mathbf{h}}^t_{(i,j)}$

$$\tilde{\mathbf{h}}^{t+1}_j = \mathrm{GRU}([\mathrm{MSG}^t_j, \mathbf{x}^t_j], \tilde{\mathbf{h}}^t_j)$$

$$\boldsymbol{\mu}^{t+1}_j = \mathbf{x}^t_j + f_{\mathrm{out}}(\tilde{\mathbf{h}}^{t+1}_j)$$

$$p(\mathbf{x}^{t+1}|\mathbf{x}^t, \ldots, \mathbf{x}^1, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}^{t+1}, \sigma^2\mathbf{I})$$

Image Credit: [1]

# Neural Relational Inference

Prior: Independent uniform distributions over edge types

$$p_\theta(\mathbf{z}) = \prod_{i \neq j} p_\theta(\mathbf{z}_{ij})$$

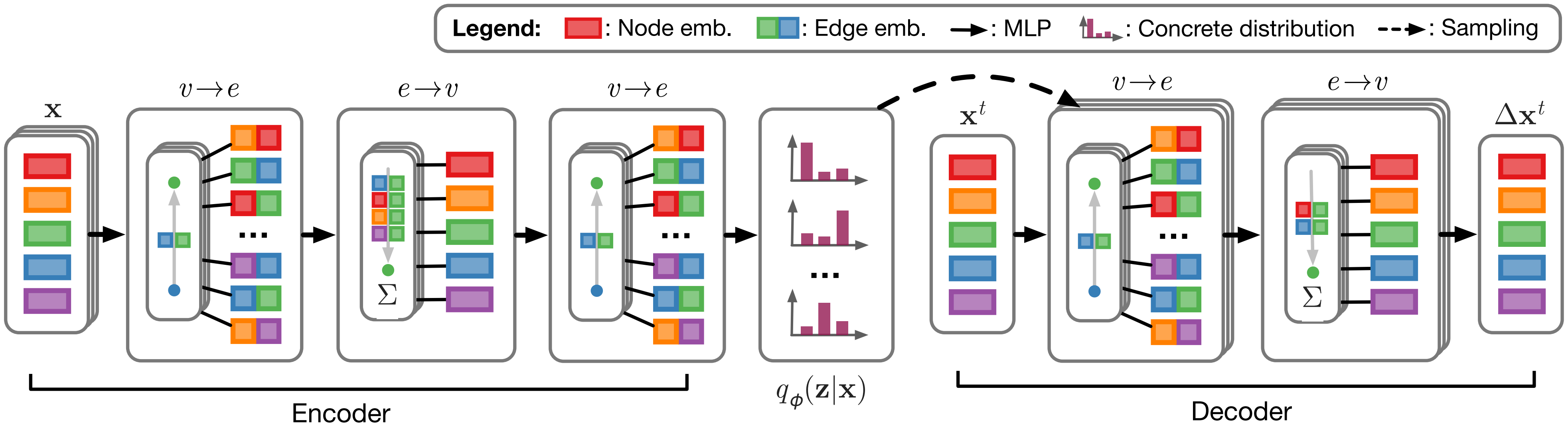
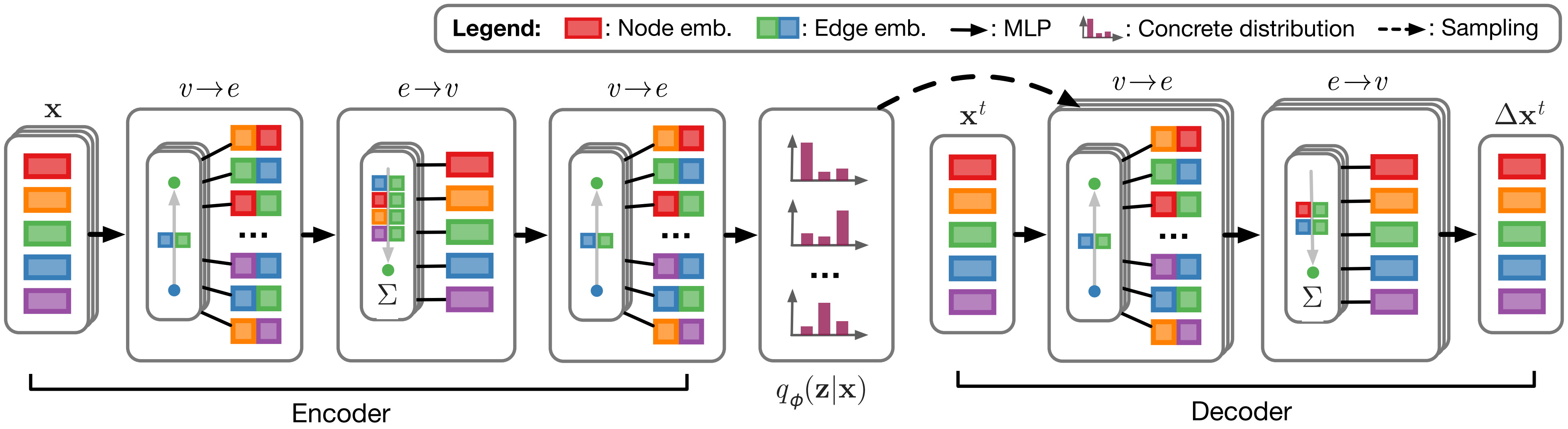



Image Credit: [1]

# Neural Relational Inference

Learning VAE

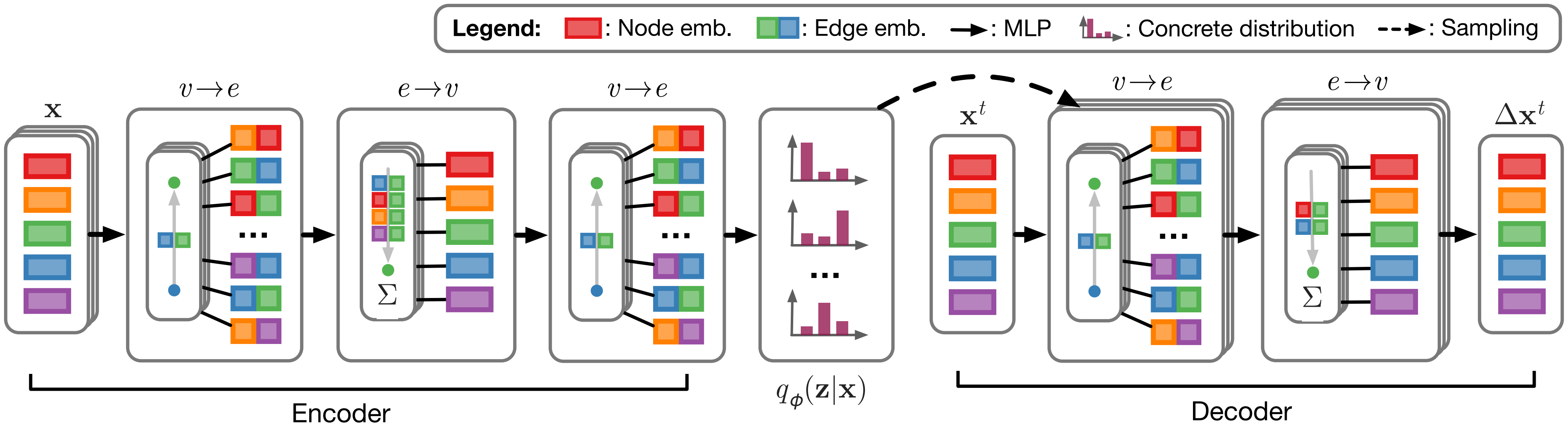



Sampling discrete latent variables

Image Credit: [1]

# Neural Relational Inference

Learning VAE



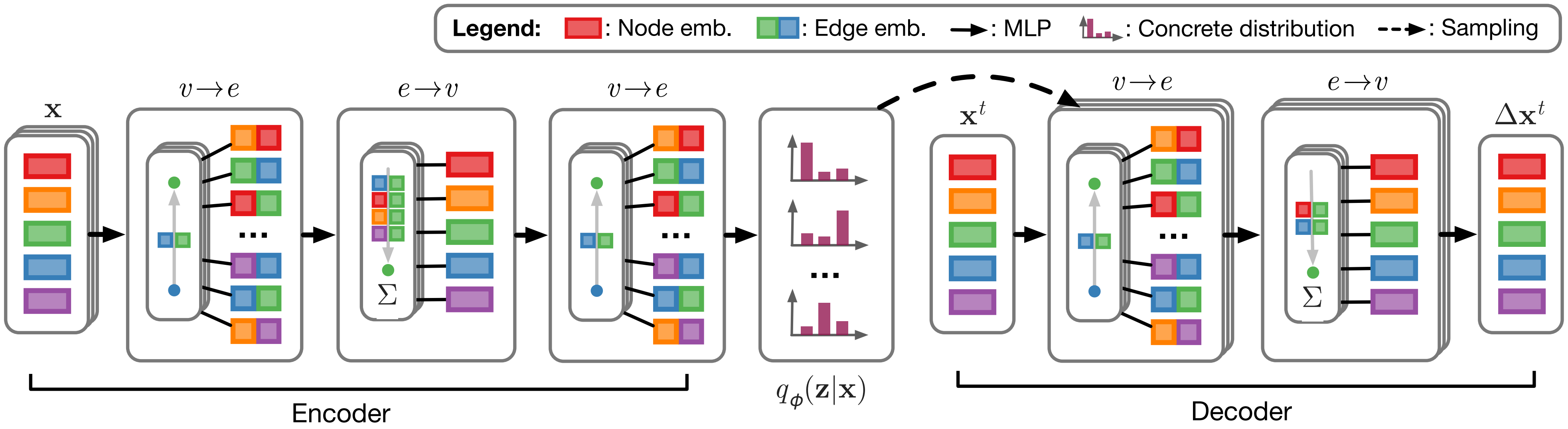

Sampling discrete latent variables

- Score function estimator (REINFORCE)
- Gumbel-Softmax / Concrete (Relaxation + Reparameterization)

Image Credit: [1]

# Neural Relational Inference

Learning VAE




Sampling discrete latent variables

- Score function estimator (REINFORCE)
- **Gumbel-Softmax / Concrete (Relaxation + Reparameterization)**

Image Credit: [1]

# Neural Relational Inference

Predicting the walking motion

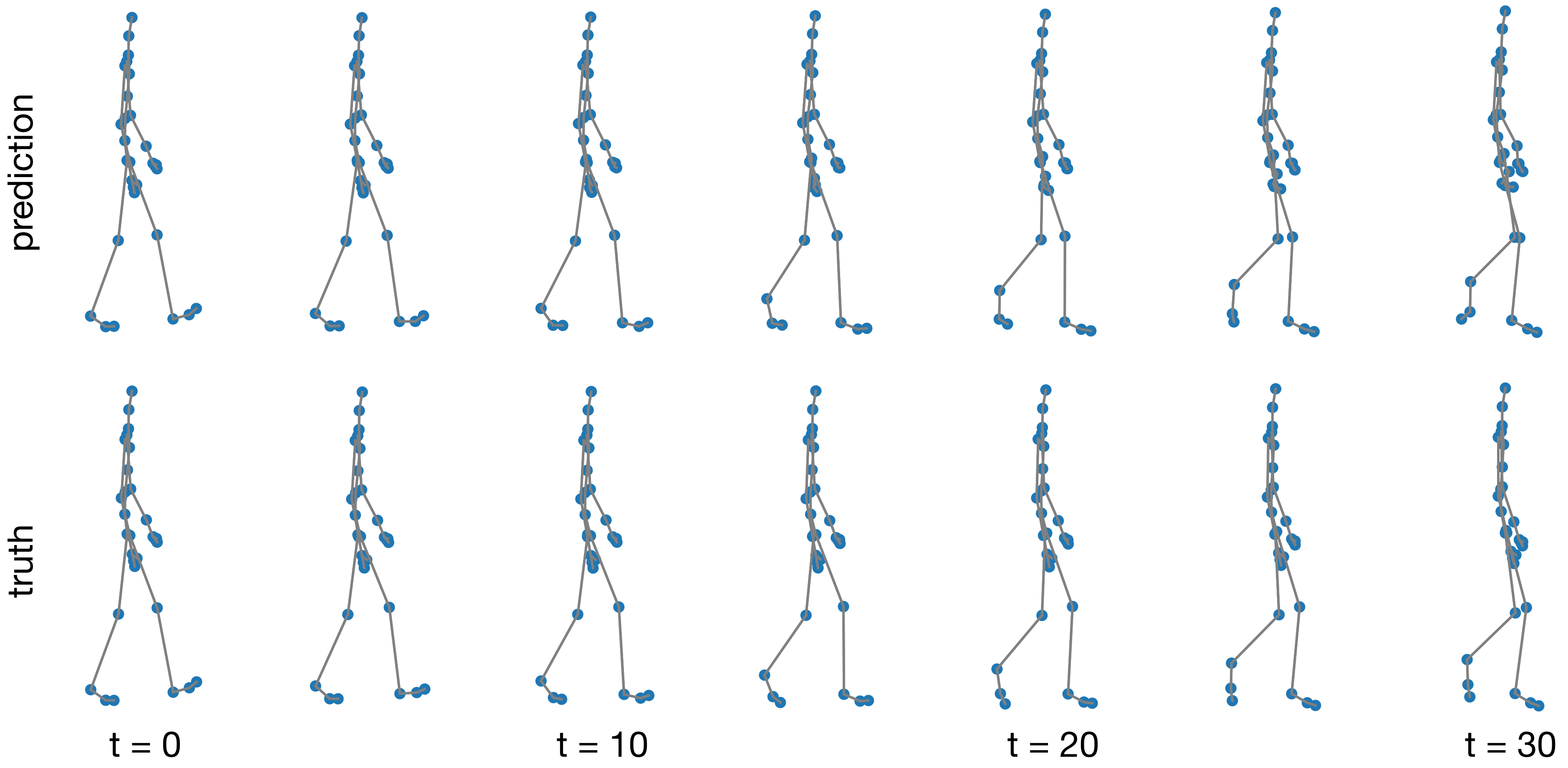
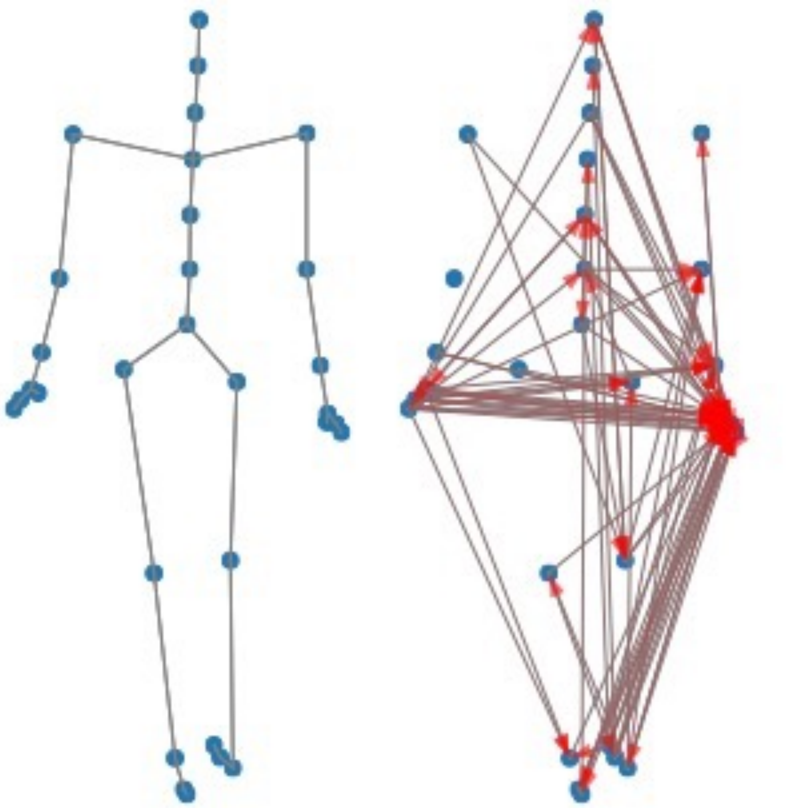


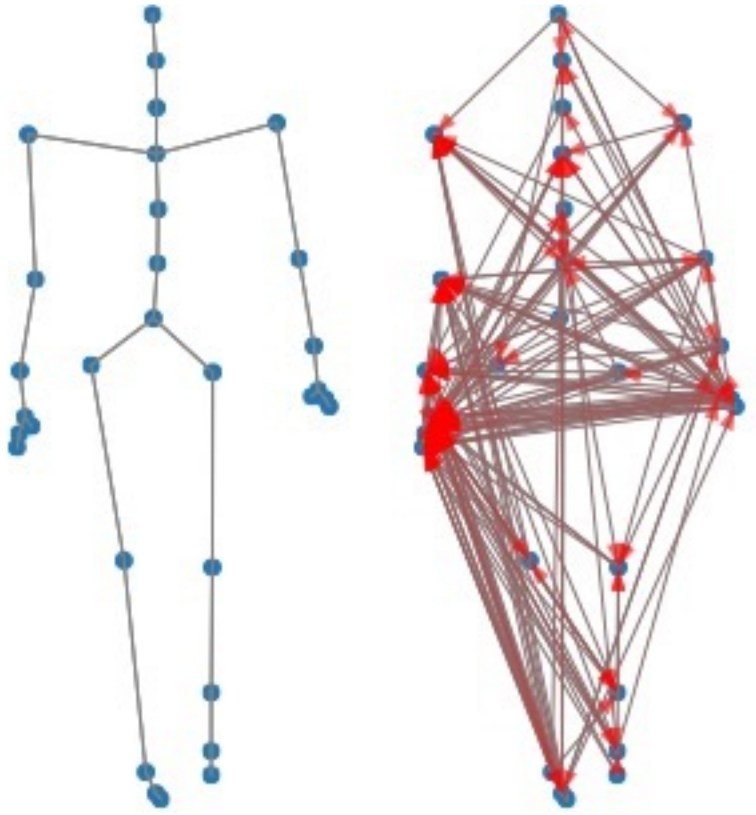

Image Credit: [1]

# Neural Relational Inference

Learned latent graphs



Right hand focus



Left hand focus

Image Credit: [1]

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

- Transductive vs. Inductive

**Transductive**: *reasoning from observed, specific (training) cases to specific (test) cases*

**Inductive**: *reasoning from observed training cases to general rules, which are then applied to the test cases*

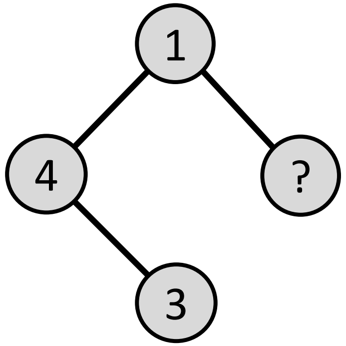
# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

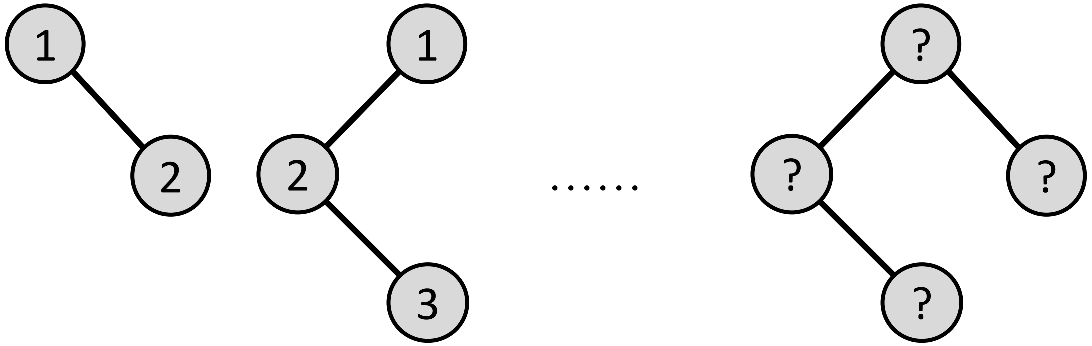
- Transductive vs. Inductive

**Transductive**: *reasoning from observed, specific (training) cases to specific (test) cases*

**Inductive**: *reasoning from observed training cases to general rules, which are then applied to the test cases*




Transductive




Inductive

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

Suppose the graph is incomplete or even completely missing

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

Suppose the graph is incomplete or even completely missing

Can we learn a latent graph and then apply GNNs?

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

Suppose the graph is incomplete or even completely missing

Can we learn a latent graph and then apply GNNs?

One can formulate a bi-level optimization problem [2]:

$$\begin{aligned} \min_{A} \quad & \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v) \\ \text{s.t.} \quad & w_A = \arg\min_{w} \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w) \end{aligned}$$

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

Suppose the graph is incomplete or even completely missing

Can we learn a latent graph and then apply GNNs?

One can formulate a bi-level optimization problem [2]:

$$\min_{A} \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v)$$

Outer level/loop

$$\text{s.t.} \quad w_A = \arg\min_{w} \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$$

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

Suppose the graph is incomplete or even completely missing

Can we learn a latent graph and then apply GNNs?

One can formulate a bi-level optimization problem [2]:

$$\min_{A} \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v)$$

Outer level/loop

$$\text{s.t.} \quad w_A = \arg\min_{w} \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$$

Inner level/loop

# Learning Discrete Structures for GNNs

Suppose we are given a single graph and want to perform (transductive) node classification

Suppose the graph is incomplete or even completely missing

Can we learn a latent graph and then apply GNNs?

One can formulate a bi-level optimization problem [2]:

$$\min_{A} \sum_{v \in V_{\mathtt{Val}}} \ell(f_{w_A}(X, A)_v, y_v)$$

Outer level/loop

$$\text{s.t.} \quad w_A = \arg\min_{w} \sum_{v \in V_{\mathtt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$$

Inner level/loop

$w_A$ is a function (parameterized by the optimization algorithm) of $A$!

# Learning Discrete Structures for GNNs

One can formulate a bi-level optimization problem [2]:

$$\begin{aligned} \min_{A} \quad & \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v) \\ \text{s.t.} \quad & w_A = \arg\min_{w} \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w) \end{aligned}$$

This problem is a hard mixed-integer programming problem!

# Learning Discrete Structures for GNNs

One can formulate a bi-level optimization problem [2]:

$$\begin{aligned} \min_{A} \quad & \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v) \\ \text{s.t.} \quad & w_A = \arg\min_{w} \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w) \end{aligned}$$

This problem is a hard mixed-integer programming problem!

We can relax it (introducing edge-independent Bernoulli distribution):

$$\begin{aligned} \min_{\theta} \quad & \mathbb{E}_{A \sim \text{Ber}(\theta)} \left[ \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v) \right] \\ \text{s.t.} \quad & w_\theta = \arg\min_{w} \; \mathbb{E}_{A \sim \text{Ber}(\theta)} \left[ \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w) \right] \end{aligned}$$

# Learning Discrete Structures for GNNs

One can formulate a bi-level optimization problem [2]:

$$\min_{A} \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v)$$

$$\text{s.t.} \quad w_A = \arg\min_{w} \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$$

This problem is a hard mixed-integer programming problem!

We can relax it (introducing edge-independent Bernoulli distribution):

We can unroll SGD for a few steps!

$$\min_{\theta} \quad \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[ \sum_{v \in V_{\texttt{Val}}} \ell(f_{w_A}(X, A)_v, y_v) \right]$$

$$\text{s.t.} \quad w_\theta = \arg\min_{w} \boxed{\mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[ \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w) \right]}$$

# Learning Discrete Structures for GNNs

Denoting the outer objective as $$F(w, A) = \sum_{v \in V_{\texttt{Val}}} \ell(f_w(X, A)_v, y_v)$$

Denoting the inner objective as $$L(w, A) = \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$$

# Learning Discrete Structures for GNNs

Denoting the outer objective as $F(w, A) = \sum_{v \in V_{\texttt{Val}}} \ell(f_w(X, A)_v, y_v)$

Denoting the inner objective as $L(w, A) = \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$

The bi-level optimization is simplified as

$$\min_{\theta} \quad \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[ F(w_\theta, A) \right]$$
$$\text{s.t.} \quad w_\theta = \arg\min_{w} \quad \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[ L(w, A) \right]$$

# Learning Discrete Structures for GNNs

Denoting the outer objective as $$F(w, A) = \sum_{v \in V_{\texttt{Val}}} \ell(f_w(X, A)_v, y_v)$$

Denoting the inner objective as $$L(w, A) = \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$$

The bi-level optimization is simplified as

$$\begin{aligned} \min_{\theta} \quad & \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[ F(w_\theta, A) \right] \\ \text{s.t.} \quad & w_\theta = \arg\min_{w} \quad \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[ L(w, A) \right] \end{aligned}$$

For Inner optimization, we unroll SGD as $$w_{\theta, t+1} = w_{\theta, t} - \gamma_t \nabla L(w_{\theta, t}, A_t)$$

# Learning Discrete Structures for GNNs

Denoting the outer objective as $$F(w, A) = \sum_{v \in V_{\texttt{Val}}} \ell(f_w(X, A)_v, y_v)$$

Denoting the inner objective as $$L(w, A) = \sum_{v \in V_{\texttt{Train}}} \ell(f_w(X, A)_v, y_v) + \Omega(w)$$

The bi-level optimization is simplified as

$$\begin{aligned} \min_{\theta} \quad & \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} [F(w_\theta, A)] \\ \text{s.t.} \quad & w_\theta = \arg\min_{w} \; \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} [L(w, A)] \end{aligned}$$

For Inner optimization, we unroll SGD as $$w_{\theta,t+1} = w_{\theta,t} - \gamma_t \nabla L(w_{\theta,t}, A_t)$$

The (hyper) gradient is

$$\nabla_\theta \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} [F(w_{\theta,T}, A)] = \mathbb{E}_{A \sim \mathrm{Ber}(\theta)} \left[ \frac{\partial F(w_{\theta,T}, A)}{\partial w_{\theta,T}} \frac{\partial w_{\theta,T}}{\partial \theta} + \frac{\partial F(w_{\theta,T}, A)}{\partial A} \frac{\partial A}{\partial \theta} \right]$$

# Learning Discrete Structures for GNNs

The (hyper) gradient is

$$\nabla_\theta \mathbb{E}_{A\sim\mathrm{Ber}(\theta)}\left[F(w_{\theta,T}, A)\right] = \mathbb{E}_{A\sim\mathrm{Ber}(\theta)}\left[\frac{\partial F(w_{\theta,T}, A)}{\partial w_{\theta,T}}\frac{\partial w_{\theta,T}}{\partial \theta} + \frac{\partial F(w_{\theta,T}, A)}{\partial A}\frac{\partial A}{\partial \theta}\right]$$

# Learning Discrete Structures for GNNs

The (hyper) gradient is

$$\nabla_\theta \mathbb{E}_{A\sim\mathrm{Ber}(\theta)}\left[F(w_{\theta,T}, A)\right] = \mathbb{E}_{A\sim\mathrm{Ber}(\theta)}\left[\frac{\partial F(w_{\theta,T}, A)}{\partial w_{\theta,T}}\frac{\partial w_{\theta,T}}{\partial \theta} + \frac{\partial F(w_{\theta,T}, A)}{\partial A}\frac{\partial A}{\partial \theta}\right]$$

A is discrete samples drawn from edge-independent Bernoulli distribution

Since it is non-differentiable, we use straight-through estimator [2]

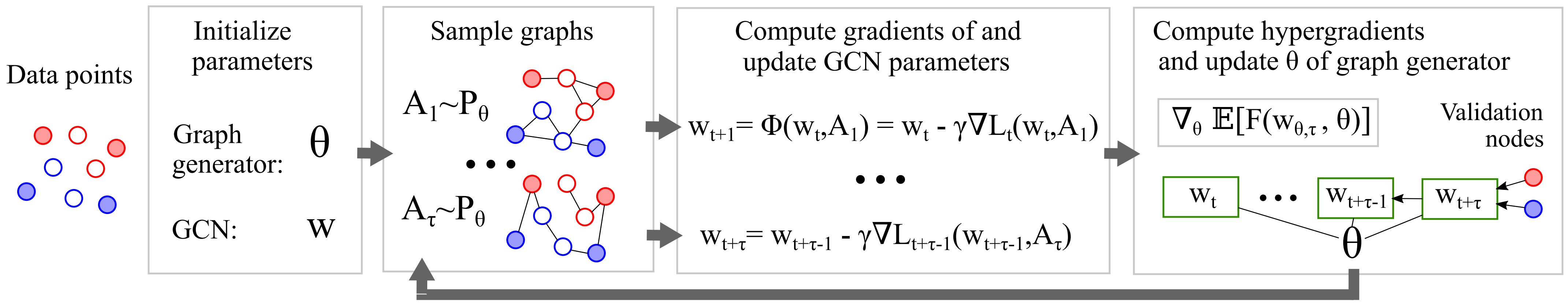
# Learning Discrete Structures for GNNs

The (hyper) gradient is

$$\nabla_\theta \mathbb{E}_{A\sim \mathrm{Ber}(\theta)}\left[F(w_{\theta,T}, A)\right] = \mathbb{E}_{A\sim \mathrm{Ber}(\theta)}\left[\frac{\partial F(w_{\theta,T}, A)}{\partial w_{\theta,T}}\frac{\partial w_{\theta,T}}{\partial \theta} + \frac{\partial F(w_{\theta,T}, A)}{\partial A}\boxed{\frac{\partial A}{\partial \theta}}\right]$$

A is discrete samples drawn from edge-independent Bernoulli distribution

Since it is non-differentiable, we use straight-through estimator [2]



# Learning Discrete Structures for GNNs

Transductive Classification:

| | Wine | Cancer | Digits | Citeseer | Cora | 20news | FMA |
|---|---|---|---|---|---|---|---|
| LogReg | 92.1 (1.3) | **93.3 (0.5)** | 85.5 (1.5) | 62.2 (0.0) | 60.8 (0.0) | 42.7 (1.7) | 37.3 (0.7) |
| Linear SVM | 93.9 (1.6) | **90.6 (4.5)** | 87.1 (1.8) | 58.3 (0.0) | 58.9 (0.0) | 40.3 (1.4) | 35.7 (1.5) |
| RBF SVM | **94.1 (2.9)** | **91.7 (3.1)** | 86.9 (3.2) | 60.2 (0.0) | 59.7 (0.0) | 41.0 (1.1) | **38.3 (1.0)** |
| RF | 93.7 (1.6) | **92.1 (1.7)** | 83.1 (2.6) | 60.7 (0.7) | 58.7 (0.4) | 40.0 (1.1) | **37.9 (0.6)** |
| FFNN | 89.7 (1.9) | **92.9 (1.2)** | 36.3 (10.3) | 56.7 (1.7) | 56.1 (1.6) | 38.6 (1.4) | 33.2 (1.3) |
| LP | 89.8 (3.7) | 76.6 (0.5) | **91.9 (3.1)** | 23.2 (6.7) | 37.8 (0.2) | 35.3 (0.9) | 14.1 (2.1) |
| ManiReg | 90.5 (0.1) | 81.8 (0.1) | 83.9 (0.1) | 67.7 (1.6) | 62.3 (0.9) | **46.6 (1.5)** | 34.2 (1.1) |
| SemiEmb | 91.9 (0.1) | 89.7 (0.1) | **90.9 (0.1)** | 68.1 (0.1) | 63.1 (0.1) | **46.9 (0.1)** | 34.1 (1.9) |
| Sparse-GCN | 63.5 (6.6) | 72.5 (2.9) | 13.4 (1.5) | 33.1 (0.9) | 30.6 (2.1) | 24.7 (1.2) | 23.4 (1.4) |
| Dense-GCN | 90.6 (2.8) | 90.5 (2.7) | 35.6 (21.8) | 58.4 (1.1) | 59.1 (0.6) | 40.1 (1.5) | 34.5 (0.9) |
| RBF-GCN | 90.6 (2.3) | **92.6 (2.2)** | 70.8 (5.5) | 58.1 (1.2) | 57.1 (1.9) | 39.3 (1.4) | 33.7 (1.4) |
| $k$NN-GCN | 93.2 (3.1) | **93.8 (1.4)** | **91.3 (0.5)** | 68.3 (1.3) | 66.5 (0.4) | 41.3 (0.6) | **37.8 (0.9)** |
| $k$NN-LDS (dense) | **97.5 (1.2)** | **94.9 (0.5)** | **92.1 (0.7)** | **70.9 (1.3)** | **70.9 (1.1)** | **45.6 (2.2)** | **38.6 (0.6)** |
| $k$NN-LDS | **97.3 (0.4)** | **94.4 (1.9)** | **92.5 (0.7)** | **71.5 (1.1)** | **71.5 (0.8)** | **46.4 (1.6)** | **39.7 (1.4)** |

# Conclusions

Summary:

- Neural Relational Inference

  Amortized inference, thus being applicable to inductive and transductive setting

  Use Edge-Independent Categorical and Gumbel-Softmax to learn latent graphs

# Conclusions

Summary:

- Neural Relational Inference

  Amortized inference, thus being applicable to inductive and transductive setting

  Use Edge-Independent Categorical and Gumbel-Softmax to learn latent graphs

- Learning Latent Graphs via Bi-level Optimization

  Learn a single latent graph, thus being inapplicable to inductive setting

  Use Edge-Independent Bernoulli and Straight-Through to learn latent graphs

# Open Questions

- Can we use more expressive generative models over graphs? E.g., deep auto-regressive models?

- For bi-level optimization, it may be beneficial to run inner SGD until convergence. Can we still efficiently learn the latent graph in this case?

  Yes, Implicit Differentiation [3, 4]!

# References

[1] Kipf, T., Fetaya, E., Wang, K.C., Welling, M. and Zemel, R., 2018, July. Neural relational inference for interacting systems. In International Conference on Machine Learning (pp. 2688-2697). PMLR.

[2] Franceschi, L., Niepert, M., Pontil, M. and He, X., 2019, May. Learning discrete structures for graph neural networks. In International conference on machine learning (pp. 1972-1982). PMLR.

[3] Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R. and Zemel, R., 2018, July. Reviving and improving recurrent back-propagation. In International Conference on Machine Learning (pp. 3082-3091). PMLR.

[4] Bai, S., Kolter, J.Z. and Koltun, V., 2019. Deep equilibrium models. Advances in Neural Information Processing Systems, 32.

Questions?