Graph generation methods

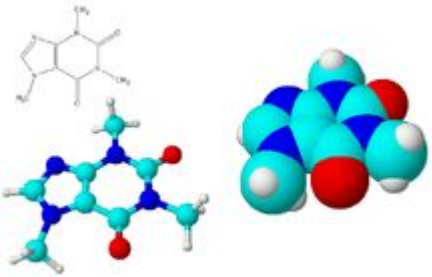
Cătălina Cangea

R250 MPhil ACS/CST Part III, 17th January 2020

UNIVERSITY OF CAMBRIDGE

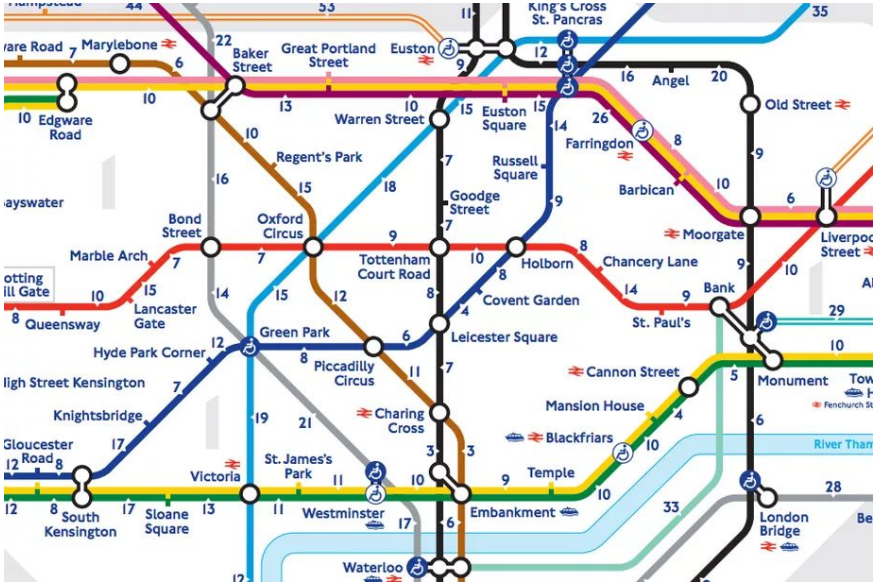
Why learn to generate graphs?

- Get (more) data!
- Learn about properties of existing data
  - E.g. train encoder-decoder
  - Use encoder to get features
  - Predict graph property
- Existing data → world!

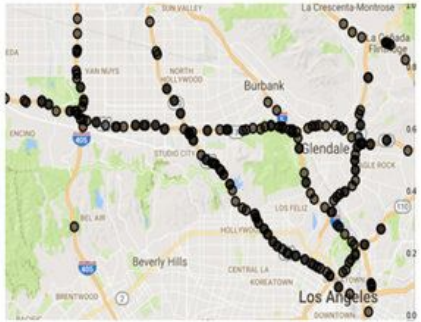


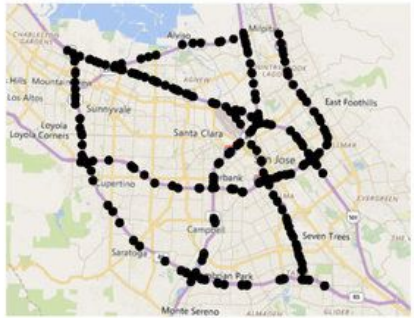
en.wiktionary.org/wiki/molecule

medium.com/analytics-vidhya/social-network-analytics-f082f4e21b16



theverge.com/2015/11/11/9712376/london-walk-tube-underground-map





researchgate.net/publication/318316069

https://arxiv.org/abs/1907.03950

Challenges

- Complexity of the output space
  - n^2 values to specify graph of size n
- Permutation- and size-invariant representation
  - Don't assume # or ordering of nodes
- Structural dependencies
  - Don't want to model edges independently

Approaches

- Traditional models
  - e.g. stochastic block (SBM), Erdos-Renyi (ER), Barabasi-Albert (BA)
- Independent generation of graph components
  - e.g. VGAE, GraphVAE
- Auto-regressive
  - e.g. GraphRNN, GRAN
- Flow-based
  - GNF

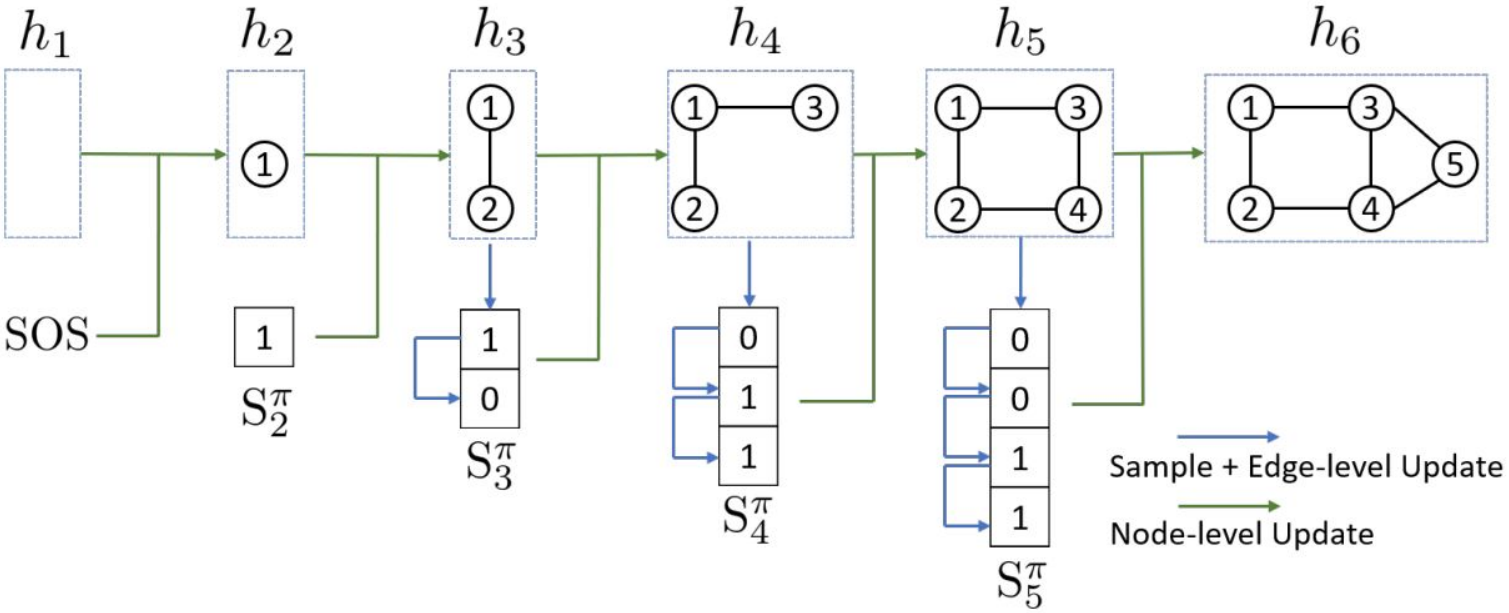
GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

You and Ying et al., 2016

GraphRNN

- Hierarchical model
  - Graph-level RNN
  - Edge-level RNN
- BFS node ordering scheme
- Worst-case $O(n^2)$ operations

GraphRNN


h_1
h_2
h_3
h_4
h_5
h_6
SOS
1
S_2^π
1
0
S_3^π
0
1
1
S_4^π
0
0
1
1
S_5^π
Sample + Edge-level Update
Node-level Update

Modelling graph distributions

- Learning a distribution $p_{model}(G)$
- ...based on observed graphs from $p_{data}(G)$

Modelling graph distributions

- Can think of graphs as sequences:

$$S^\pi = f_S(G, \pi) = (S_1^\pi, ..., S_n^\pi)$$

$$S_i^\pi = (A_{1,i}^\pi, ..., A_{i-1,i}^\pi)^T, \forall i \in \{2, ..., n\}$$

Modelling graph distributions

- Rewrite distribution by marginalising:

$$p(G) = \sum_{S^\pi} p(S^\pi)\ \mathbb{1}[f_G(S^\pi) = G]$$

Modelling graph distributions

- Decompose:

$$p(S^\pi) = \prod_{i=1}^{n+1} p(S_i^\pi | S_1^\pi, ..., S_{i-1}^\pi)$$

$$p(S_i^\pi | S_{<i}^\pi) = \prod_{j=1}^{i-1} p(S_{i,j}^\pi | S_{i,<j}^\pi, S_{<i}^\pi)$$

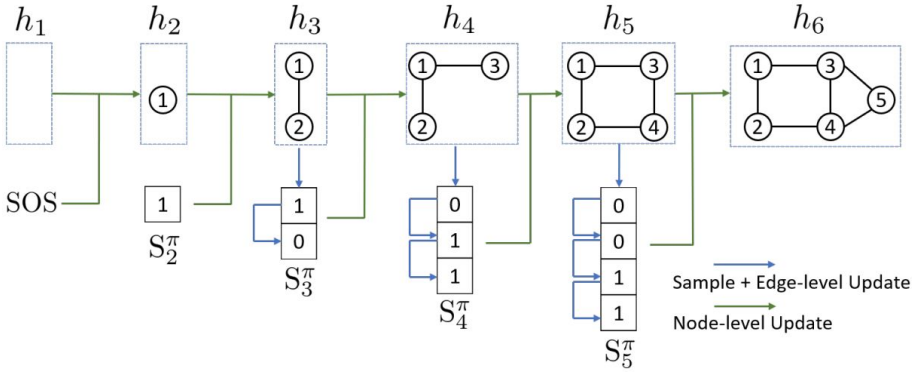
Modelling graph distributions

- Restrict # of sequences using BFS ordering:

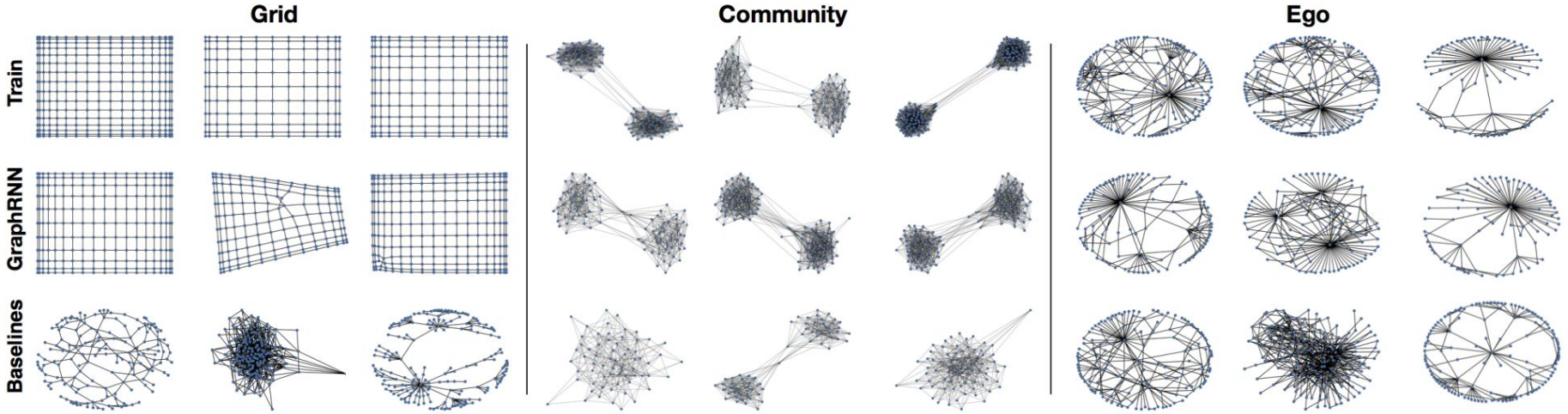
$$S^{\pi} = f_S(G, \mathbf{BFS}(G, \pi))$$

GraphRNN inference

Input: RNN-based transition module f_{trans}, output module f_{out}, probability distribution $\mathcal{P}_{\theta_i}$ parameterized by θ_i, start token SOS, end token EOS, empty graph state h'
Output: Graph sequence S^π
$S_1^\pi = \text{SOS}, h_1 = h', i = 1$
repeat
$i = i + 1$
$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$ {update graph state}
$\theta_i = f_{\text{out}}(h_i)$
$S_i^\pi \sim \mathcal{P}_{\theta_i}$ {sample node i's edge connections}
until S_i^π is EOS
Return $S^\pi = (S_1^\pi, ..., S_i^\pi)$



Samples



GraphRNN limitations

- Scale
  - # of generation steps $O(n^2)$
  - Permutation invariance harder to achieve
- Long-term RNN dependencies
  - 2 nearby nodes can be far apart in the sequential generation process

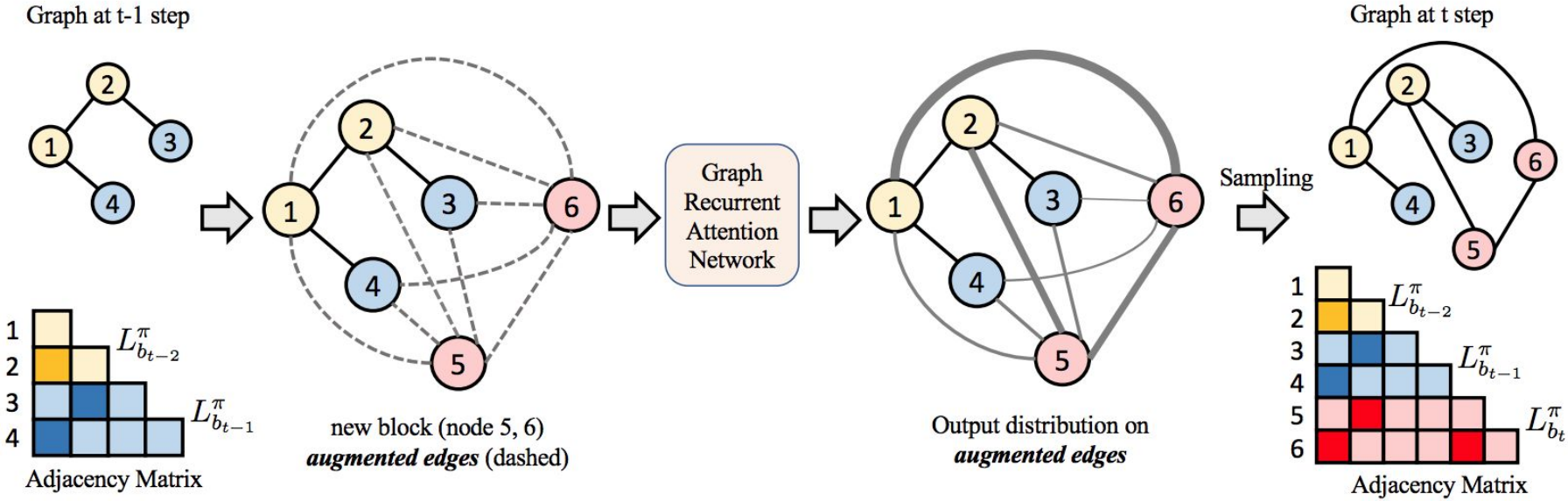
Efficient Graph Generation with Graph Recurrent Attention Networks (GRANs)

Liao et al., 2019

GRANs

- O(*N*) auto-regressive generation steps
  - 1 step generates a block of nodes
- Attention-based GNN for linking new nodes to existing ones
- Correlated edge modelling
- Choose optimal node ordering from “canonical” set
- Scales up to **5k** nodes

GRAN


Graph at t-1 step
1
2
3
4
$L^{\pi}_{b_{t-2}}$
$L^{\pi}_{b_{t-1}}$
Adjacency Matrix
new block (node 5, 6)
augmented edges (dashed)
Graph Recurrent Attention Network
Output distribution on augmented edges
Sampling
Graph at t step
5
6
$L^{\pi}_{b_t}$
Adjacency Matrix

Modelling graph distributions

- Think of graphs as matrices:

$$p(G) = \sum_{\pi} p(G, \pi) = \sum_{\pi} p(A^{\pi}) \qquad A^{\pi} = L^{\pi} + L^{\pi\top}$$

Modelling graph distributions

- 1 step: generate block of B rows in L^{π}; indices $\boldsymbol{b_t} = \{B(t-1)+1, ..., Bt\}$

$$p(L^{\pi}) = \prod_{t=1}^{T} p(L^{\pi}_{\boldsymbol{b_t}} | L^{\pi}_{\boldsymbol{b_1}}, \cdots, L^{\pi}_{\boldsymbol{b_{t-1}}})$$

Block generation

- h^R_i = final embedding of node *i* after *R* message passing steps
- Mixture of Bernoulli distributions:

$$p(L^{\pi}_{\boldsymbol{b_t}} | L^{\pi}_{\boldsymbol{b_1}}, ..., L^{\pi}_{\boldsymbol{b_{t-1}}}) = \sum_{k=1}^{K} \alpha_k \prod_{i \in \boldsymbol{b_t}} \prod_{1 \le j \le i} \theta_{k,i,j},$$

$$\alpha_1, \ldots, \alpha_K = \text{Softmax}\left(\sum\nolimits_{i \in \boldsymbol{b_t}, 1 \le j \le i} \text{MLP}_{\alpha}(h_i^R - h_j^R)\right)$$

$$\theta_{1,i,j}, \ldots, \theta_{K,i,j} = \text{Sigmoid}\left(\text{MLP}_{\theta}(h_i^R - h_j^R)\right)$$

- Set $K > 1 \rightarrow$ correlated edges due to latent mixture components

Node orderings

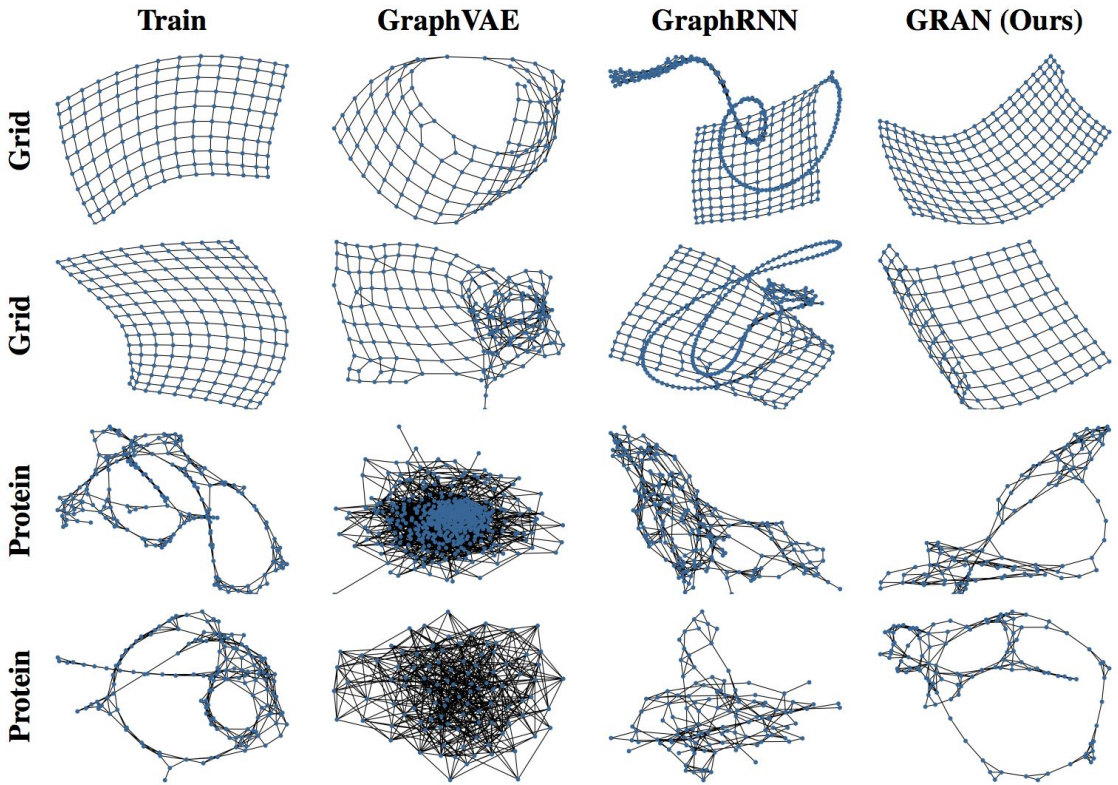
- Train under family of "canonical" orderings $\mathcal{Q} = \{\pi_1, ..., \pi_M\}$
  - Based on universal graph properties
  - i.e. descending node degree, BFS, DFS, k-core (novel)

- Learn to maximize the lower bound:

$$\log p(G) \geq \log \sum_{\pi \in \mathcal{Q}} p(G, \pi)$$

- Trade-off: **tightness of bound ←→ computational cost**

Samples


Train
GraphVAE
GraphRNN
GRAN (Ours)
Grid
Grid
Protein
Protein

Graph Normalizing Flows

Liu and Kumar et al., 2019

GNFs

- Building blocks
  - Normalizing flows adapted to graph-structured data
  - Graph auto-encoders
- Competes with GraphRNN
- Still $O(N^2)$ for message passing
- ...**BUT** can be ||-ised

Normalizing Flows - crash course

- NFs = generative models
- Use invertible map to transform between observed **x** and latent **z**:

$$\mathbf{z} = f(\mathbf{x}) \qquad \mathbf{x} = f^{-1}(f(\mathbf{x}))$$

- Probability density functions are related via the Jacobian:

$$P(\mathbf{z}) = P(\mathbf{x}) \left| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right|^{-1}$$

- Key idea: **map complex distribution to Gaussian** while keeping the Jacobian (efficiently) **computable**

NFs

- This work uses non-volume preserving flows (RealNVPs)
- Idea: partition x into $x^{(0)}$ and $x^{(1)}$ such that:

$$\mathbf{z}^{(0)} = \mathbf{x}^{(0)}$$
$$\mathbf{z}^{(1)} = \mathbf{x}^{(1)} \odot \exp(s(\mathbf{x}^{(0)})) + t(\mathbf{x}^{(0)})$$

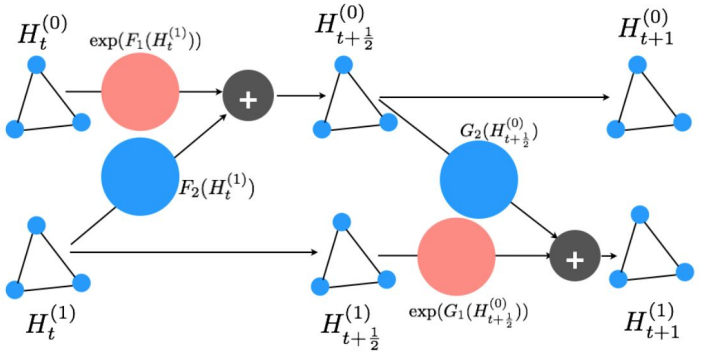
- The Jacobian is lower-triangular → efficiently computable :-)

Graph NFs

- Message passing:

$$H_{t+\frac{1}{2}}^{(0)} = H_t^{(0)} \odot \exp\left(F_1\left(H_t^{(1)}\right)\right) + F_2(H_t^{(1)})$$
$$H_{t+\frac{1}{2}}^{(1)} = H_t^{(1)}$$

$$H_{t+1}^{(0)} = H_{t+\frac{1}{2}}^{(0)}$$
$$H_{t+1}^{(1)} = H_{t+\frac{1}{2}}^{(1)} \odot \exp\left(G_1\left(H_{t+\frac{1}{2}}^{(0)}\right)\right) + G_2\left(H_{t+\frac{1}{2}}^{(0)}\right)$$



- Density transformation:

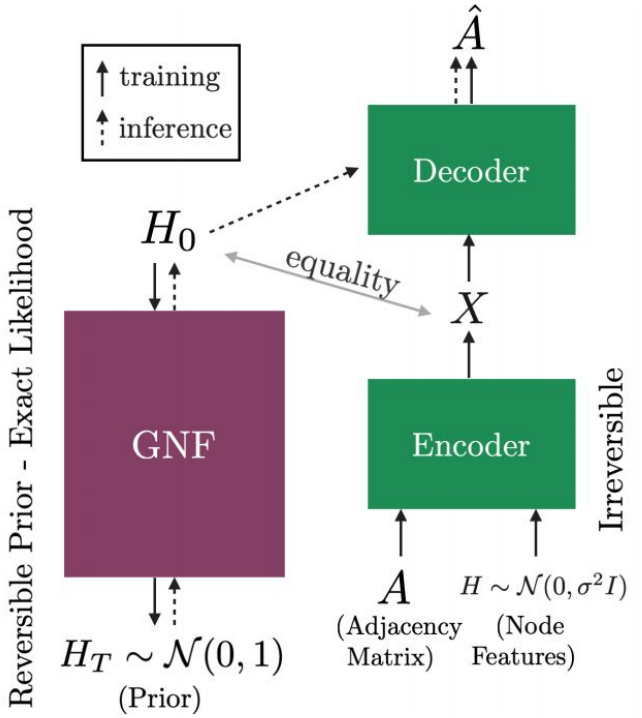
$$P(\mathcal{G}) = \det\left|\frac{\partial H_T}{\partial H_0}\right| P(H_T) = P(H_T) \prod_{t=1}^{T} \det\left|\frac{\partial H_t}{\partial H_{t-1}}\right|$$

Graph generation pipeline

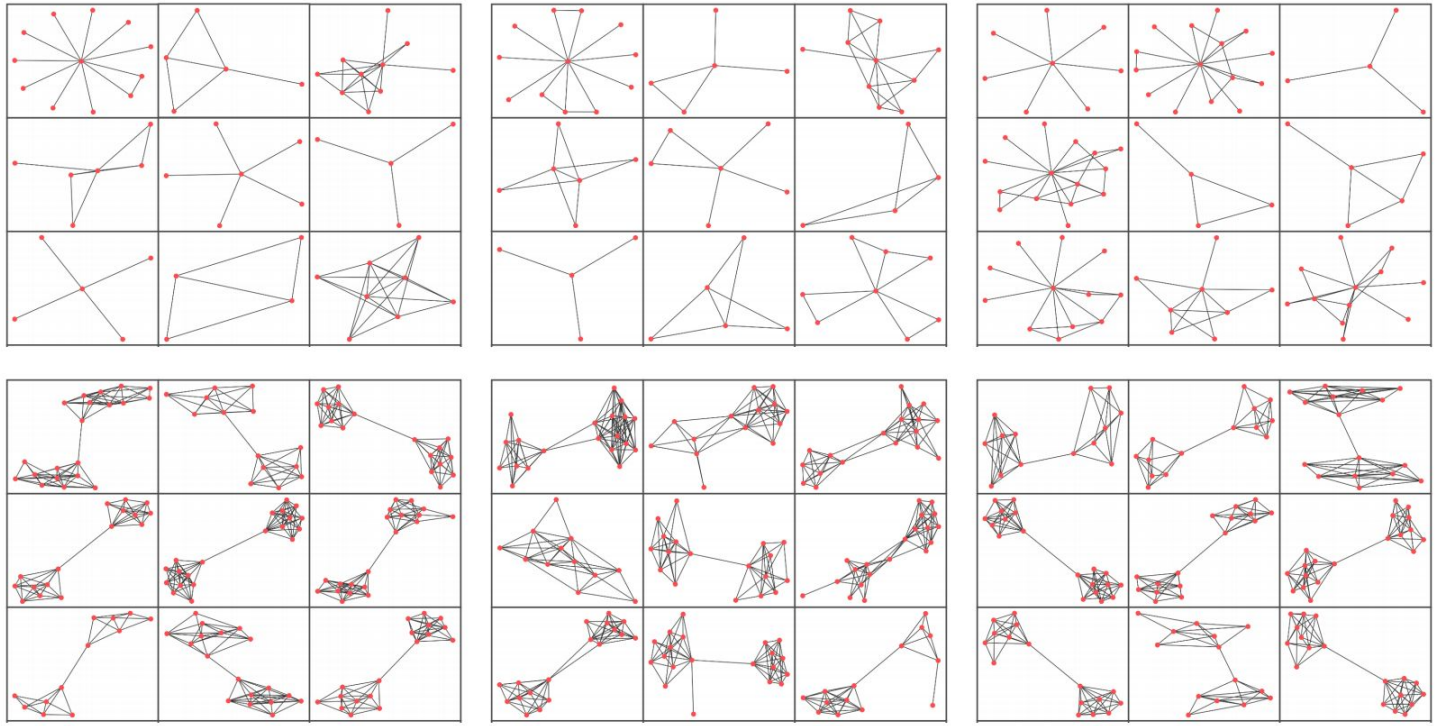
- Encoder:
  - GNN
  - Multi-head dot-product attention

- Decoder:

$$\hat{A}_{ij} = \frac{1}{1 + \exp(C(\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 - 1))}$$



Samples



(a) Training data (b) GNF samples (c) GRAPHRNN samples

What’s next?

Future directions

- Even larger graphs
- Spatio-temporal graphs!
- Getting rid of orderings / scaling GNFs
- Leveraging application-specific priors

Thank you!