

# EECE 571F: Deep Learning with Structures

## Lecture 5: Deep Generative Models of Graphs Autoregressive Models

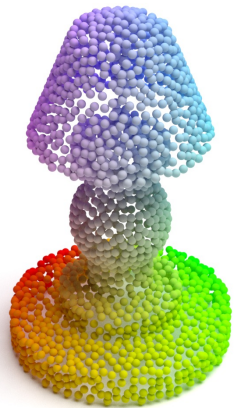
Renjie Liao

University of British Columbia

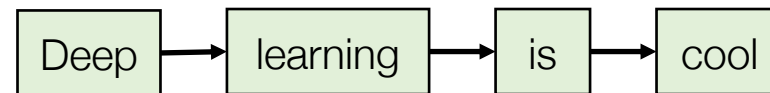
Winter, Term 2, 2021/22

# Course Scope

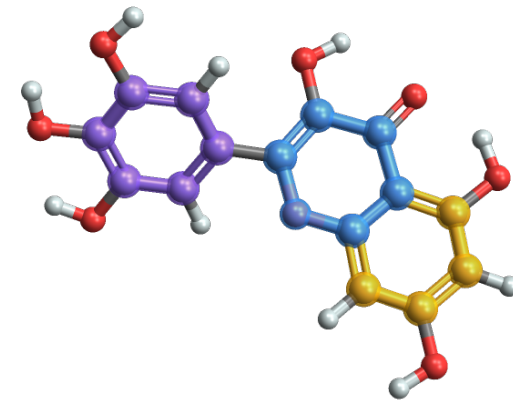
- Supervised Learning with Observable Structures
- **Unsupervised / Self-supervised Learning with Observable Structures**
- Supervised Learning with Latent Structures



Points/Sets



Lists/Sequences



Graphs

# Unsupervised / Self-Supervised Learning

Since only data is given, we need a learning criterion:

# Unsupervised / Self-Supervised Learning

Since only data is given, we need a learning criterion:

- Likelihood (Autoregressive models)
- Reconstruction Loss (Auto-encoders)
- Contrastive Loss (noise contrastive estimation, self-supervised learning)
- Min-max Loss (Generative adversarial networks)

.....



# Unsupervised / Self-Supervised Learning on Graphs

Since only data is given, we need a learning criterion:

- Likelihood (Autoregressive models)
- Reconstruction Loss (Auto-encoders)
- Contrastive Loss (noise contrastive estimation, self-supervised learning)
- Min-max Loss (Generative adversarial networks)

.....

For graphs, we need to learn connectivities as well as node/edge representations!

# Unsupervised / Self-Supervised Learning on Graphs

Since only data is given, we need a learning criterion:

- **Likelihood (Autoregressive models)**
- Reconstruction Loss (Auto-encoders)
- Contrastive Loss (noise contrastive estimation, self-supervised learning)
- **Min-max Loss (Generative adversarial networks)**

.....

For graphs, we need to learn connectivities as well as node/edge representations!

Learning could be *probabilistic*

# Unsupervised / Self-Supervised Learning on Graphs

Since only data is given, we need a learning criterion:

- Likelihood (Autoregressive models)
- **Reconstruction Loss (Auto-encoders)**
- **Contrastive Loss (noise contrastive estimation, self-supervised learning)**
- Min-max Loss (Generative adversarial networks)

.....

For graphs, we need to learn connectivities as well as node/edge representations!

Learning could be *probabilistic* or *deterministic*!

# Unsupervised / Self-Supervised Learning on Graphs

Since only data is given, we need a learning criterion:

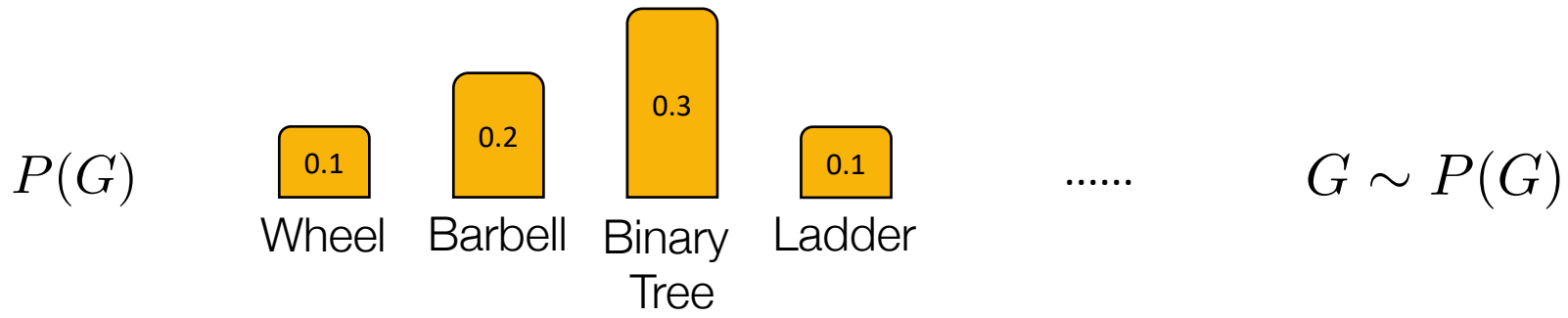
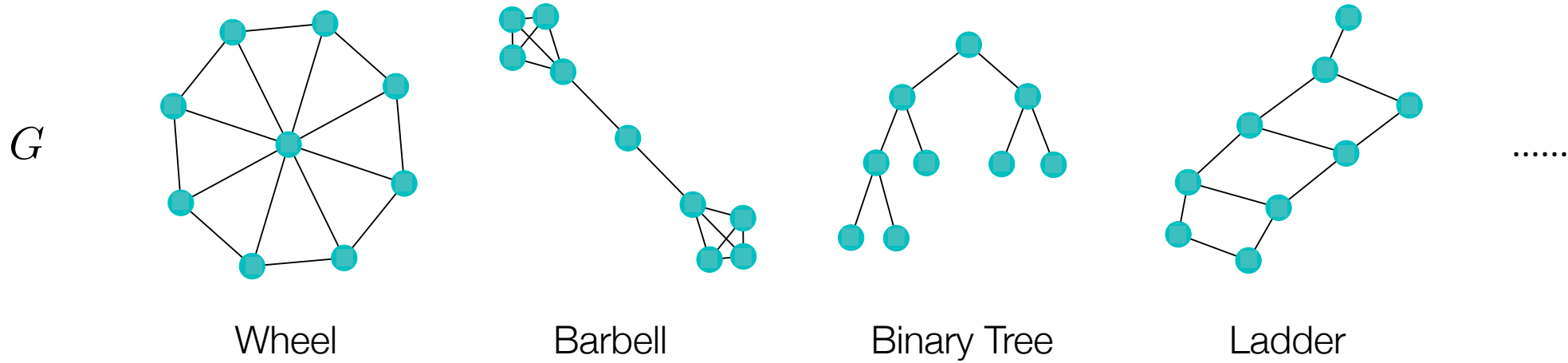
- Likelihood (Autoregressive models)
- Reconstruction Loss (Auto-encoders)
- Contrastive Loss (noise contrastive estimation, self-supervised learning)
- Min-max Loss (Generative adversarial networks)

.....

For graphs, we need to learn **connectivities** as well as node/edge representations!

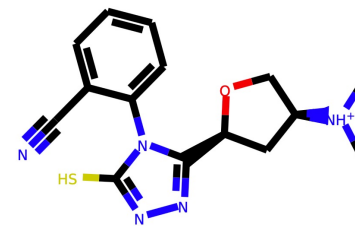
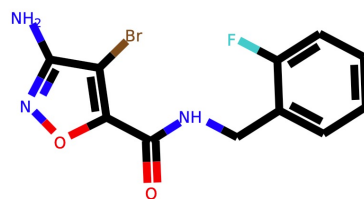
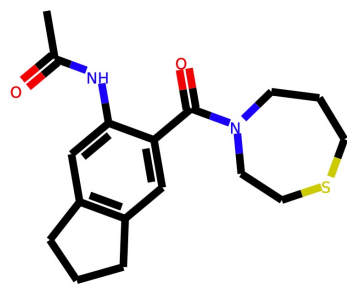
Learning could be *probabilistic* or *deterministic*!

# Deep Generative Models of Graphs



# Applications

Drug discovery, e.g. molecules [1]



# Applications

Urban planning, e.g. road layout [2]

NYC



SF

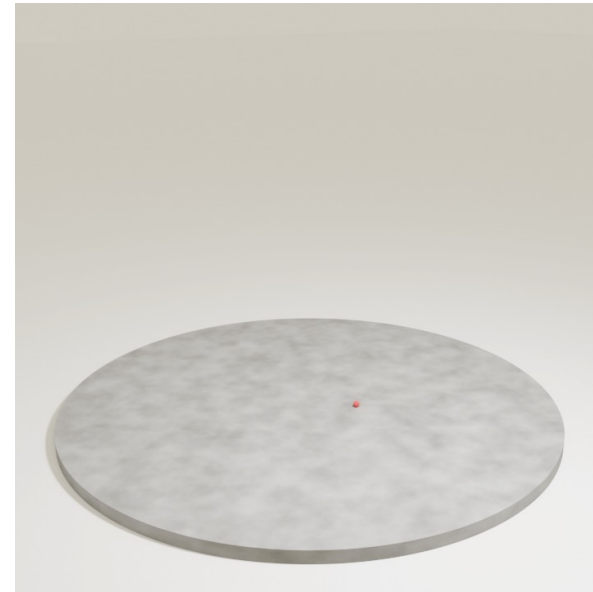
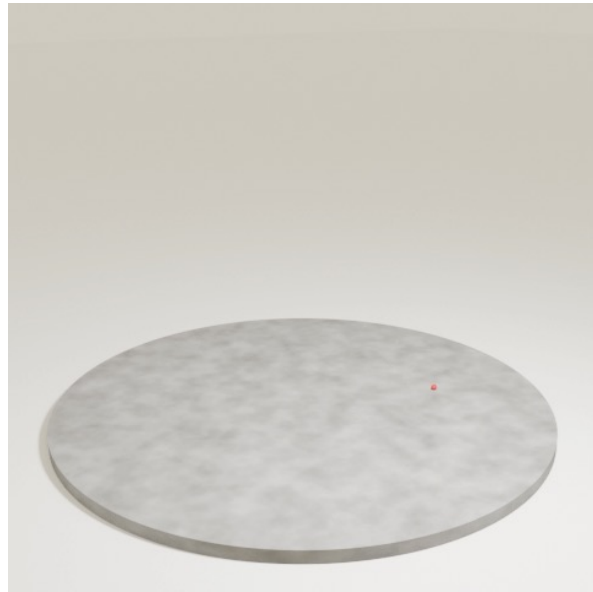
Berkeley



Istanbul

# Applications

Graphics, e.g., 3D objects [3]



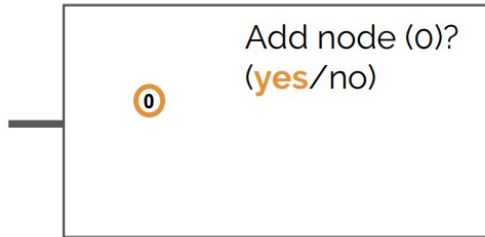


# Autoregressive Models for Graphs

Let us start with an intuitive generation process:

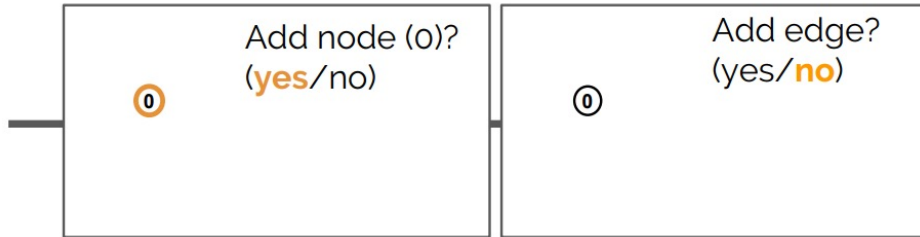
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



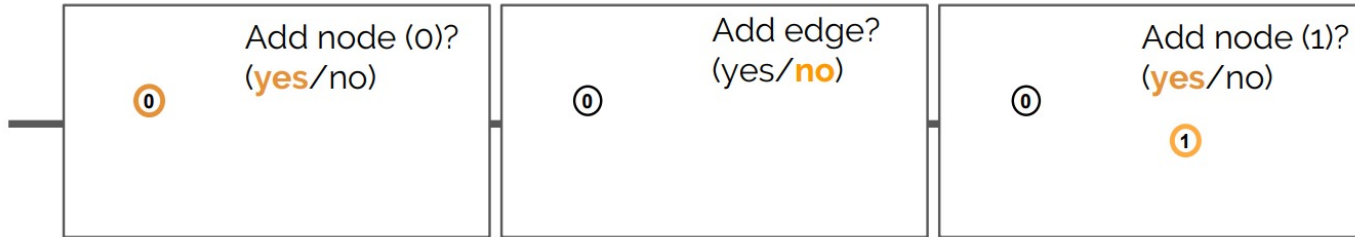
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



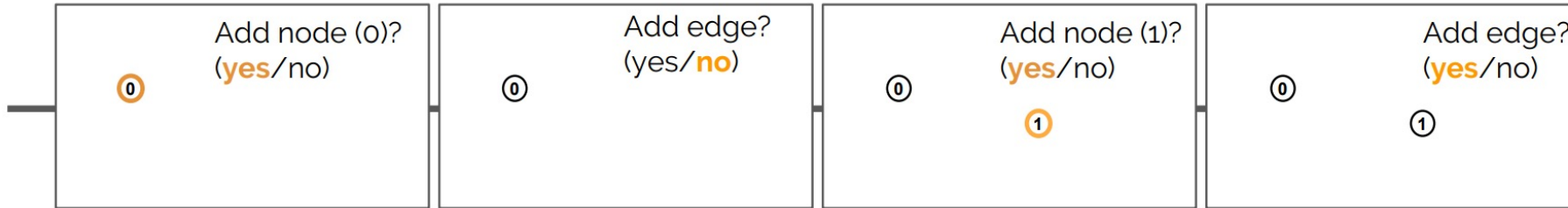
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



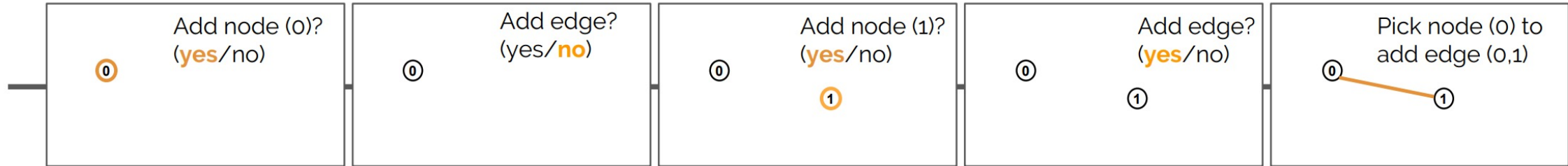
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



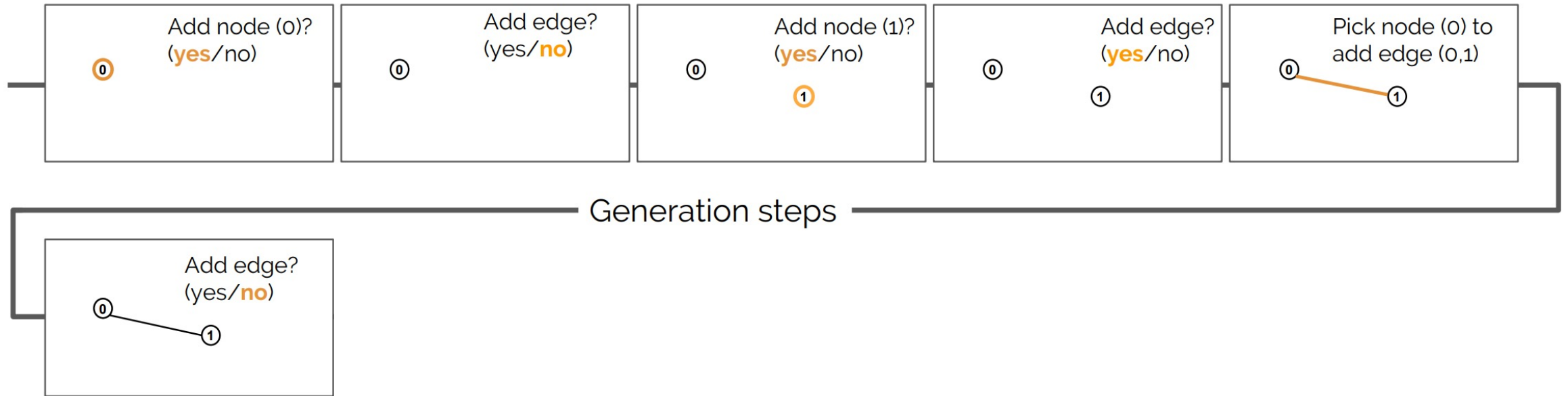
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



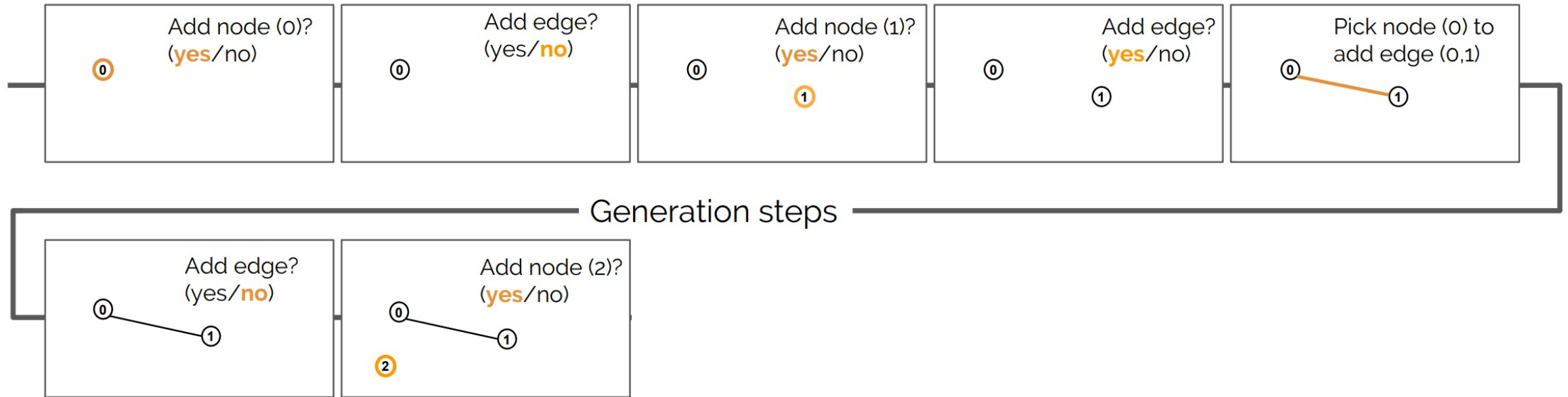
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



# Autoregressive Models for Graphs

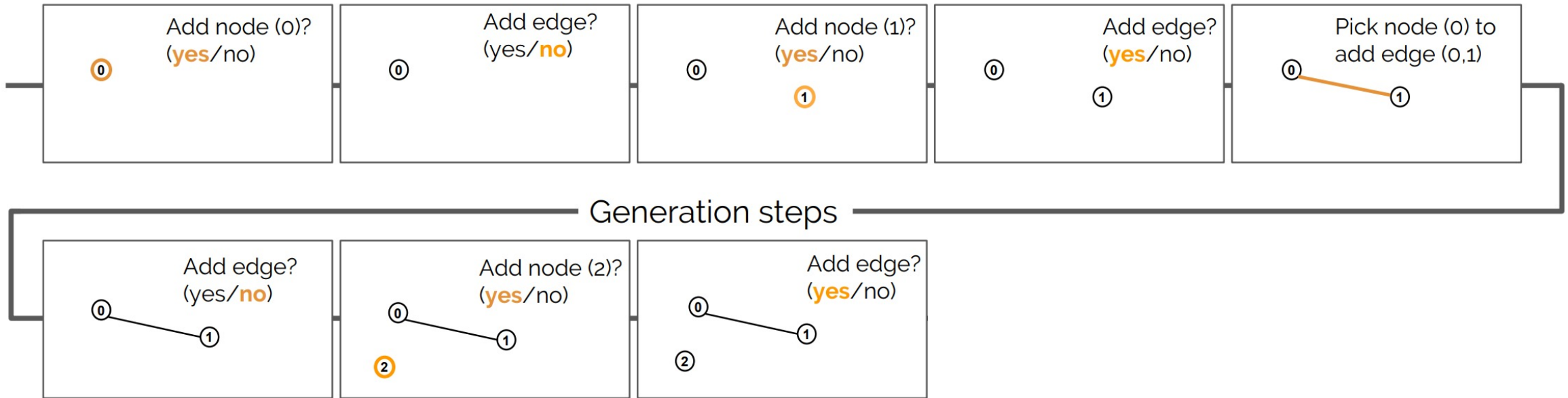
Let us start with an intuitive generation process:





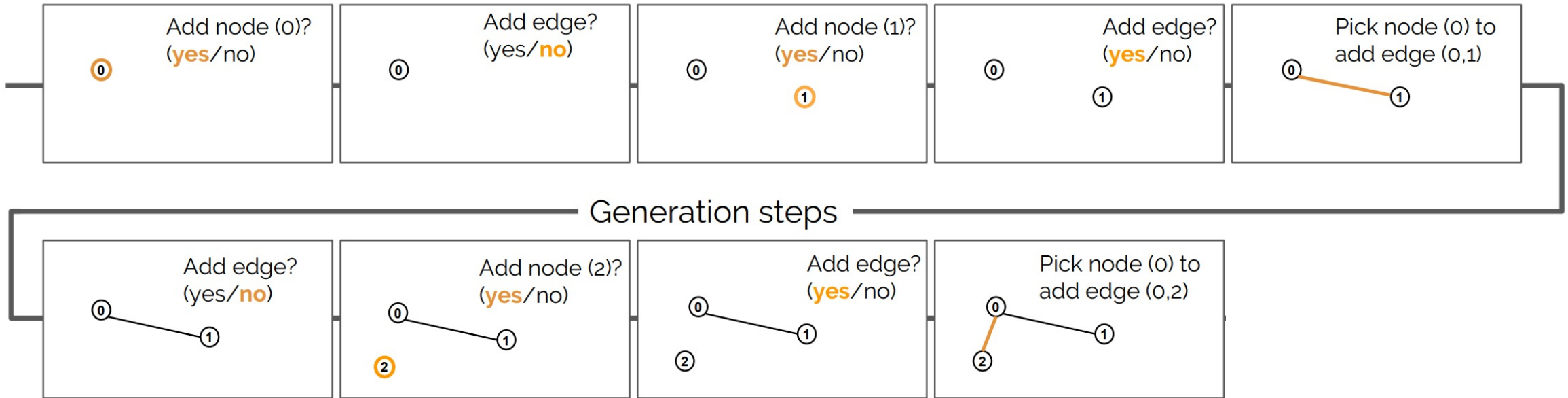
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



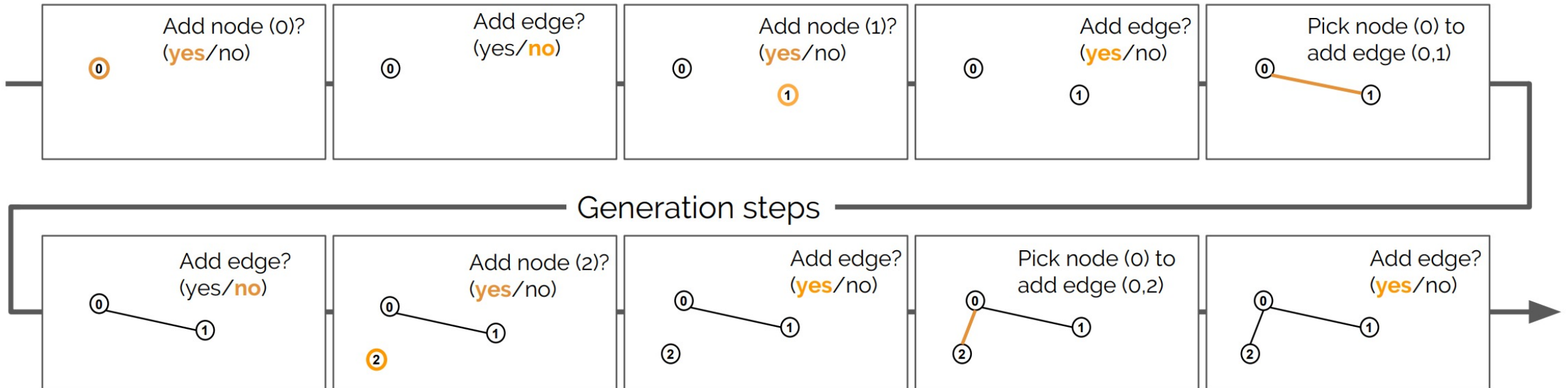
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



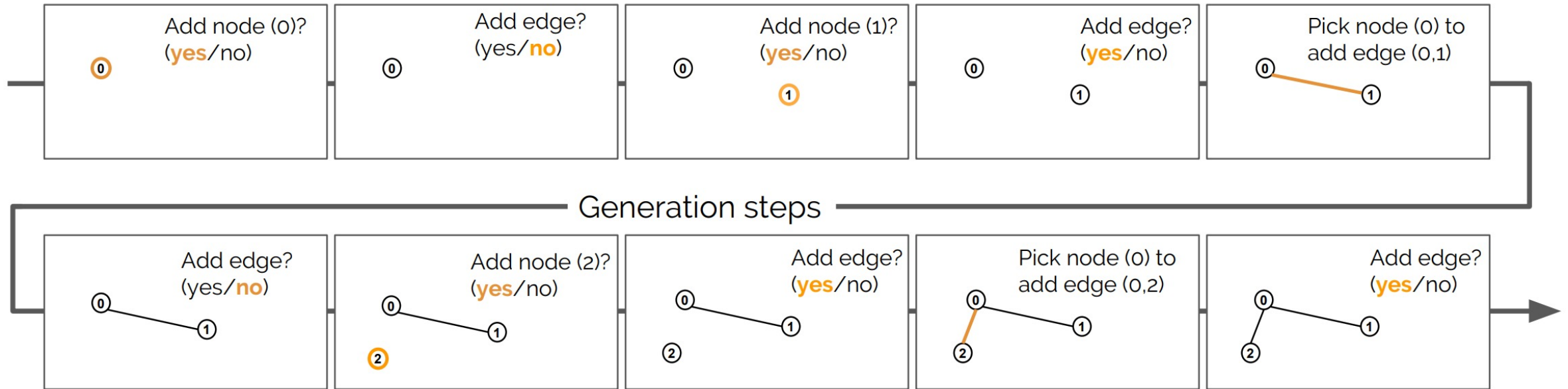
# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



# Autoregressive Models for Graphs

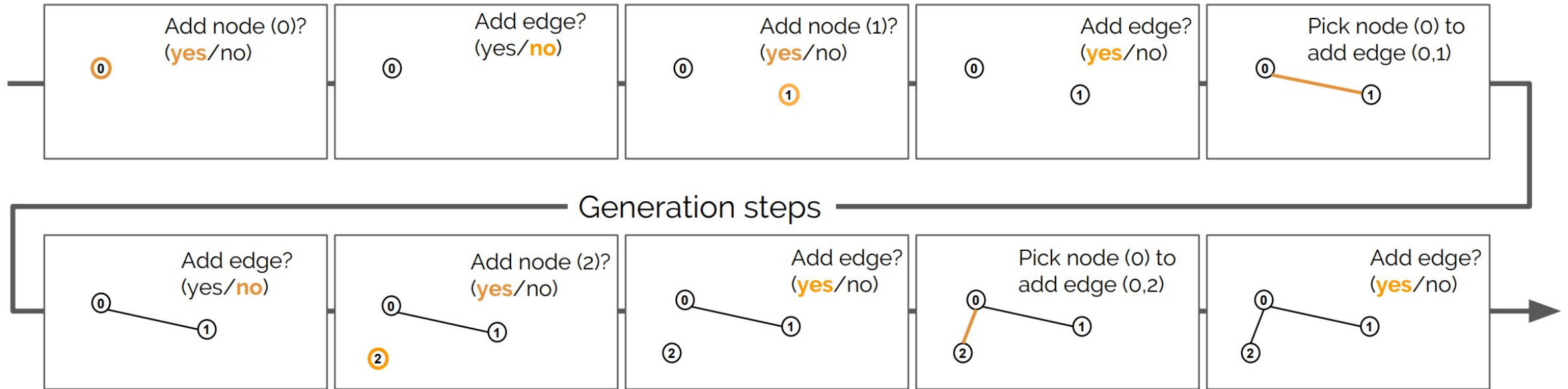
Let us start with an intuitive generation process:



- *Graph Generation = Sequential Decision Making!*

# Autoregressive Models for Graphs

Let us start with an intuitive generation process:



- *Graph Generation = Sequential Decision Making!*
- *We can sequentially generate a graph conditioning on the previously generated (sub)graph!*

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

At each step, decisions are:

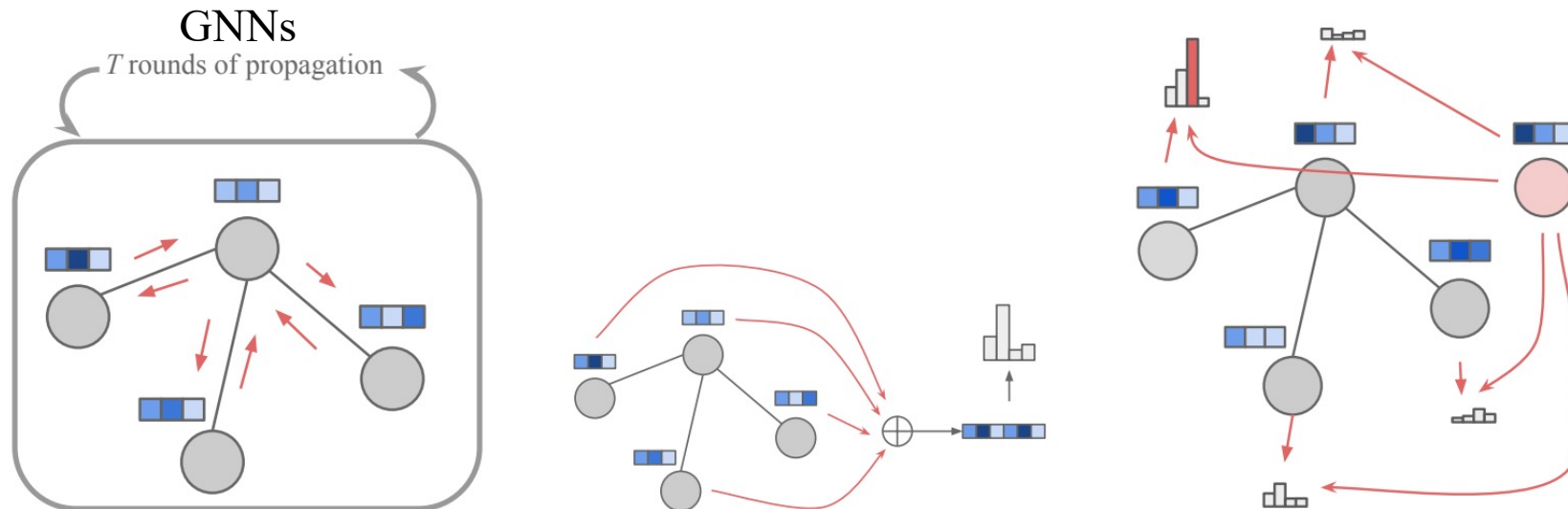
- Add Node: *stop or +1 node*
- Add Edge: *stop or +1 edge*
- Pick one node to connect: *categorical*

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

At each step, decisions are:

- Add Node: *stop or +1 node*
- Add Edge: *stop or +1 edge*
- Pick one node to connect: *categorical*





# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

At each step, decisions are:

- Add Node: *stop or +1 node*
- Add Edge: *stop or +1 edge*
- Pick one node to connect: *categorical*

$$\mathbf{h}_V^{(T)} = \text{prop}^{(T)}(\mathbf{h}_V, G)$$

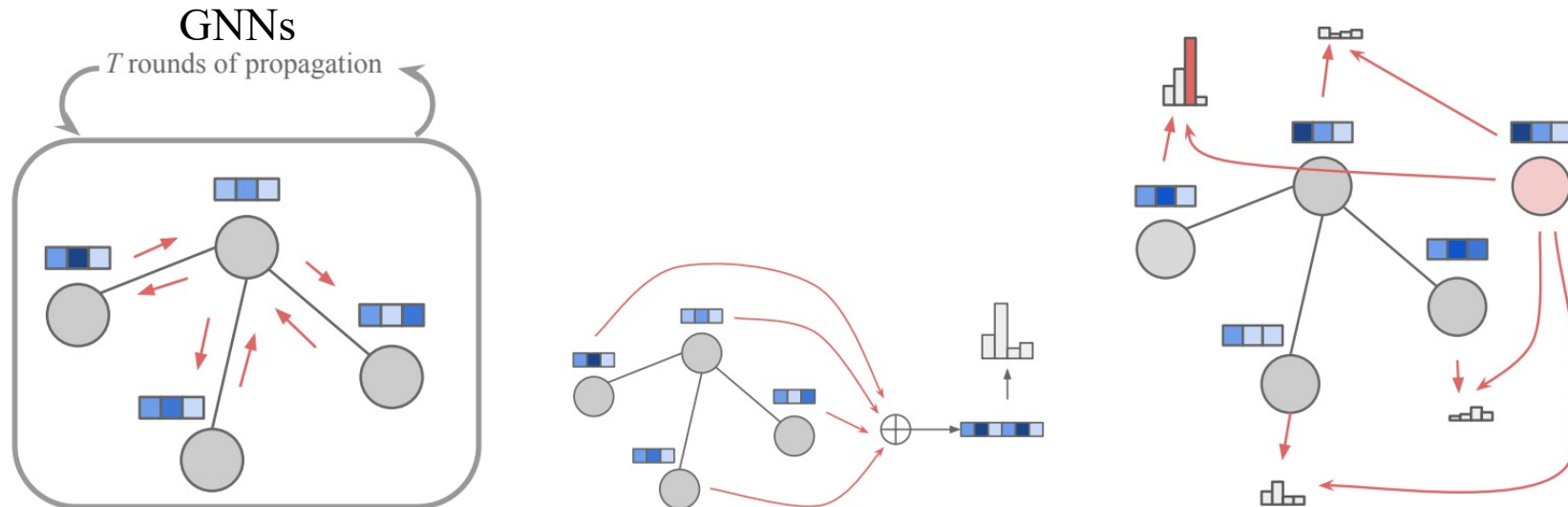
$$\mathbf{h}_G = R(\mathbf{h}_V^{(T)}, G)$$

$$f_{\text{addnode}}(G) = \text{softmax}(f_{\text{an}}(\mathbf{h}_G))$$

$$f_{\text{addedge}}(G, v) = \sigma(f_{\text{ae}}(\mathbf{h}_G, \mathbf{h}_v^{(T)}))$$

$$s_u = f_s(\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}), \quad \forall u \in V$$

$$f_{\text{nodes}}(G, v) = \text{softmax}(\mathbf{s})$$



# Autoregressive Models for Graphs

Recurrent Neural Network

GNN-RNN-based Autoregressive Model [4]

At each step, decisions are:

- Add Node: *stop or +1 node*
- Add Edge: *stop or +1 edge*
- Pick one node to connect: *categorical*

$$\mathbf{h}_V^{(T)} = \text{prop}^{(T)}(\mathbf{h}_V, G)$$

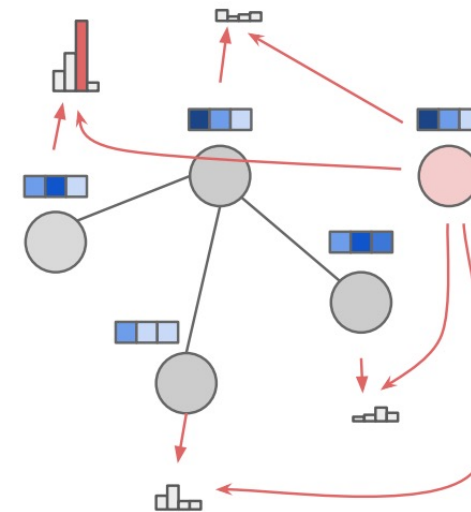
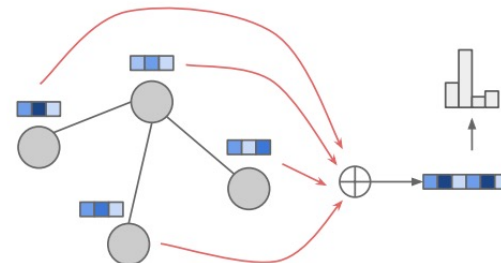
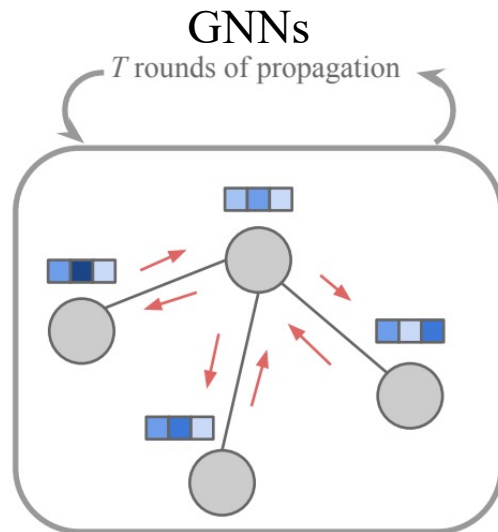
$$\mathbf{h}_G = R(\mathbf{h}_V^{(T)}, G)$$

$$f_{\text{addnode}}(G) = \text{softmax}(f_{\text{an}}(\mathbf{h}_G))$$

$$f_{\text{addedge}}(G, v) = \sigma(f_{\text{ae}}(\mathbf{h}_G, \mathbf{h}_v^{(T)}))$$

$$s_u = f_s(\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}), \quad \forall u \in V$$

$$f_{\text{nodes}}(G, v) = \text{softmax}(\mathbf{s})$$



# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

$$\max_{\theta} \log \left( \sum_{\pi \sim q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right] \right)$$

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

$$\max_{\theta} \log \left( \sum_{\pi \sim q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right] \right)$$

Learning via Back Propagation Through Time (BPTT)

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

Pros:

- Autoregressive models can capture the dependencies among nodes/edges
- Enable the usage of GNNs and RNNs for graph generation

# Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

Pros:

- Autoregressive models can capture the dependencies among nodes/edges
- Enable the usage of GNNs and RNNs for graph generation

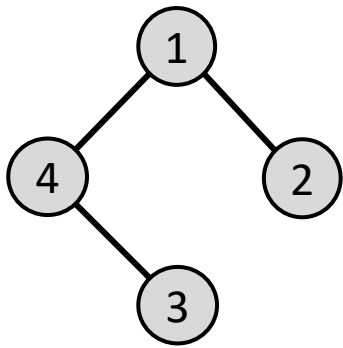
Cons:

- The likelihood is permutation-dependent
- Generating a medium-sized graph requires a lot of generation/unrolling steps
- Learning long-unrolled RNNs with BPTT is hard



# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

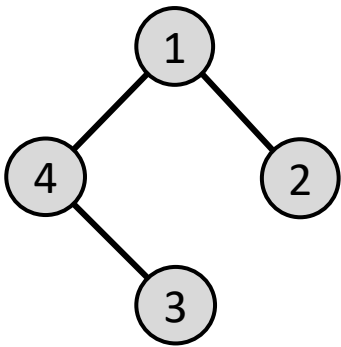


	1	2	3	4
1	0	1	0	1
2	1	0	0	0
3	0	0	0	1
4	1	0	1	0

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

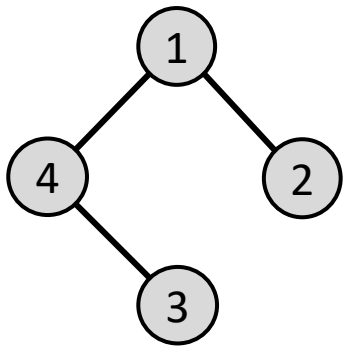
$$S_i^\pi = (A_{1,i}^\pi, \dots, A_{i-1,i}^\pi)^T, \forall i \in \{2, \dots, n\}$$



	1	2	3	4
1	0	1	0	1
2	1	0	0	0
3	0	0	0	1
4	1	0	1	0

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



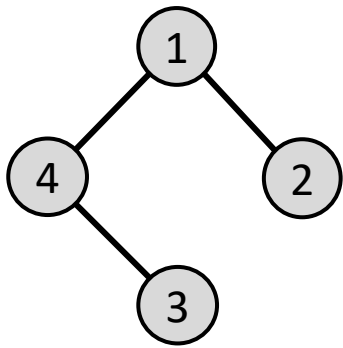
	1	2	3	4
1	0	1	0	1
2	1	0	0	0
3	0	0	0	1
4	1	0	1	0

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

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

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



	1	2	3	4
1	0	1	0	1
2	1	0	0	0
3	0	0	0	1
4	1	0	1	0

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

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

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

# Autoregressive Models for Graphs

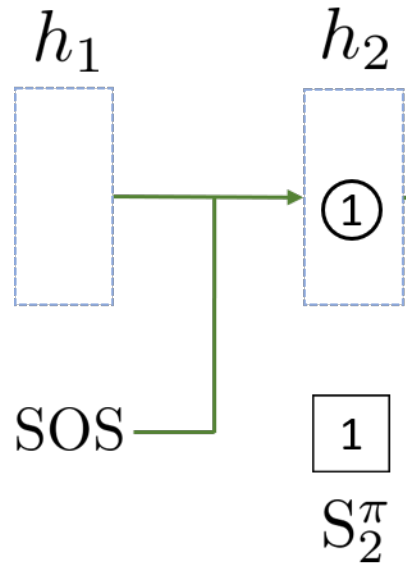
RNN-based Autoregressive Model, GraphRNN [5]



SOS

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



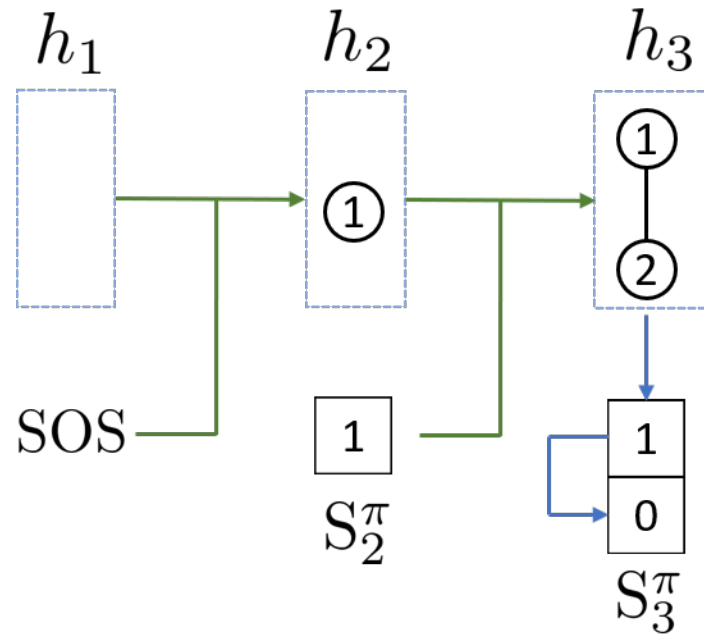
$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$

$$\theta_i = f_{\text{out}}(h_i)$$

$$S_i^\pi \sim p_{\theta_i}$$

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



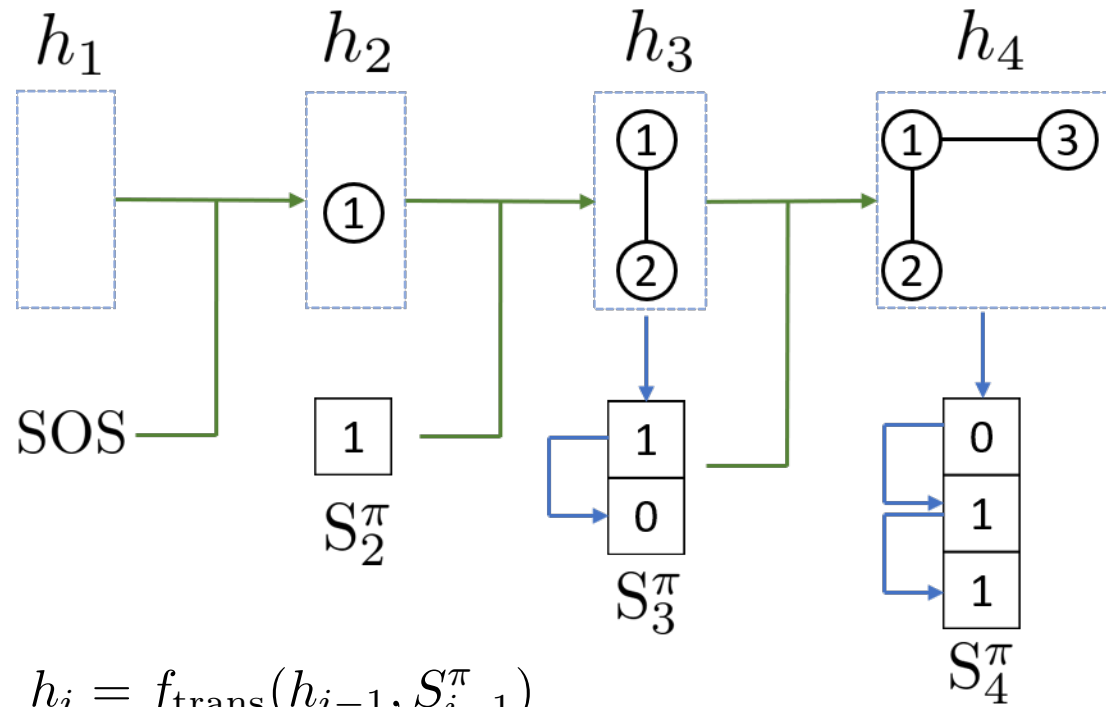
$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$

$$\theta_i = f_{\text{out}}(h_i)$$

$$S_i^\pi \sim p_{\theta_i}$$

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$

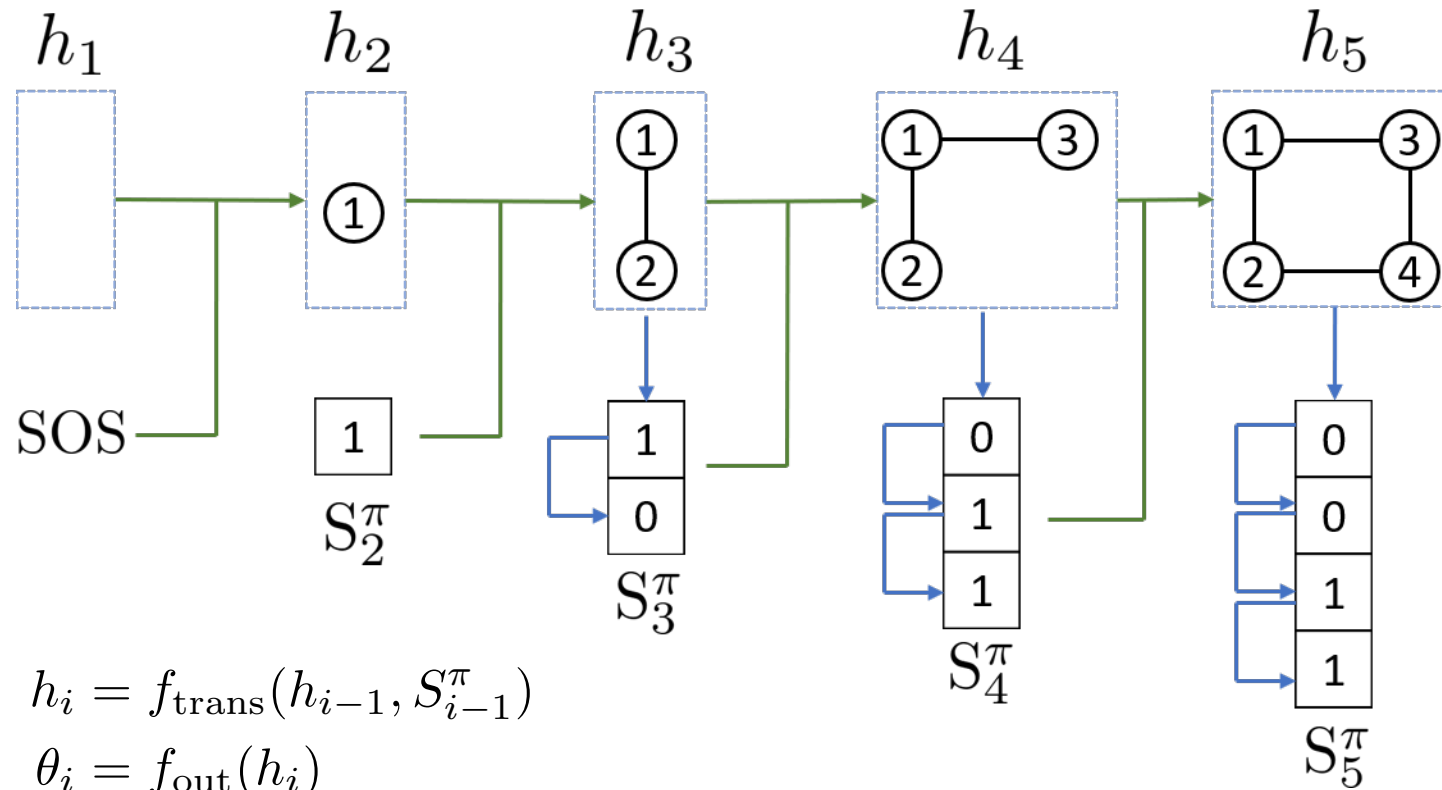
$$\theta_i = f_{\text{out}}(h_i)$$

$$S_i^\pi \sim p_{\theta_i}$$



# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



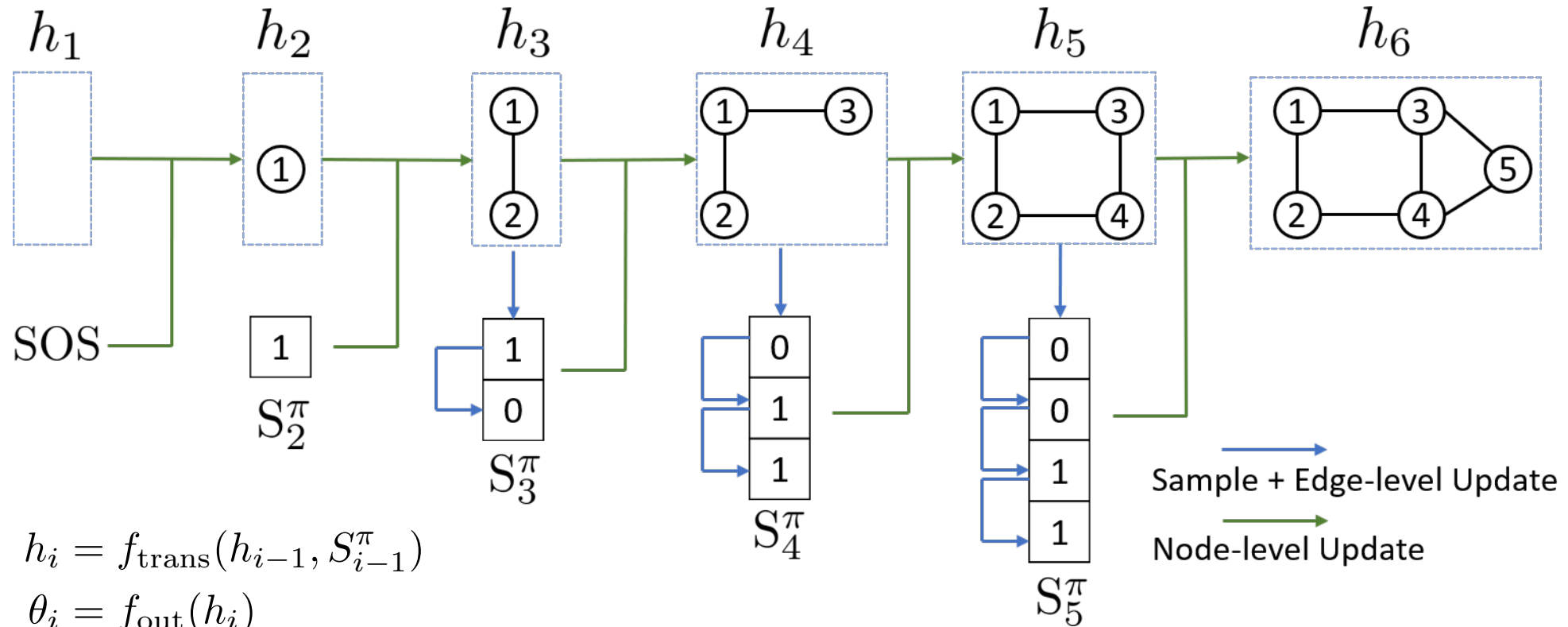
$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$

$$\theta_i = f_{\text{out}}(h_i)$$

$$S_i^\pi \sim p_{\theta_i}$$

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



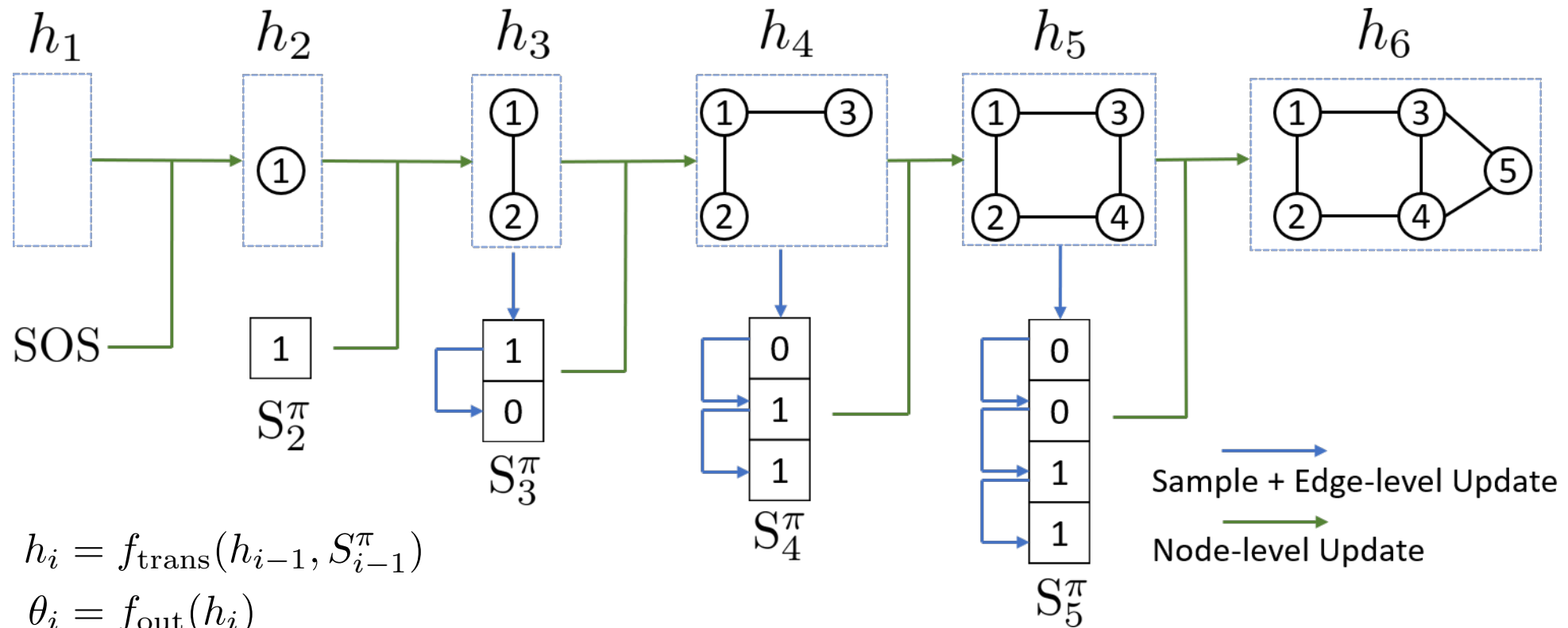
$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$

$$\theta_i = f_{\text{out}}(h_i)$$

$$S_i^\pi \sim p_{\theta_i}$$

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$

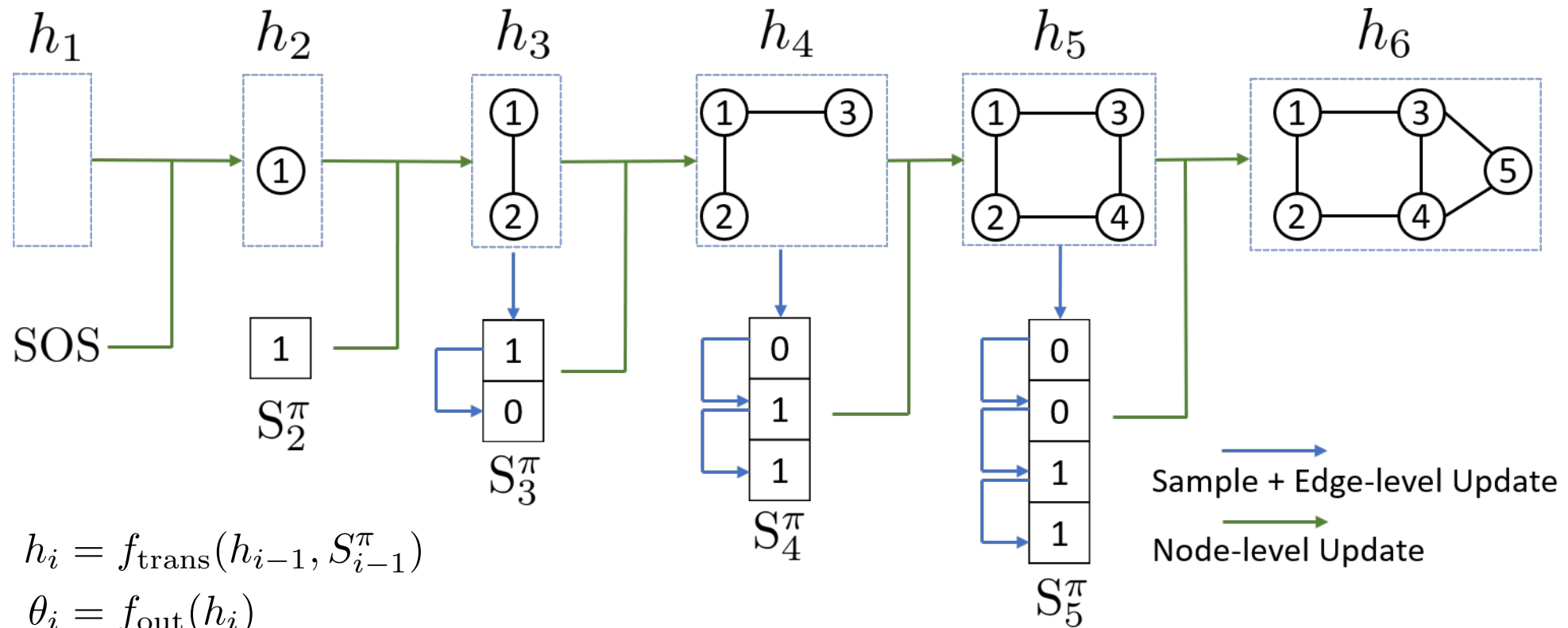
$$\theta_i = f_{\text{out}}(h_i)$$

$$S_i^\pi \sim p_{\theta_i}$$

GraphRNN-S:  $p_{\theta_i}$  is edge-independent Bernoulli

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$

$$\theta_i = f_{\text{out}}(h_i)$$

$$S_i^\pi \sim p_{\theta_i}$$

GraphRNN-S:  $p_{\theta_i}$  is edge-independent Bernoulli

$$\text{GraphRNN: } p_{\theta_i}(S_i^\pi | S_{<i}^\pi) = \prod_{j=1}^{i-1} p_{\theta_i}(S_{i,j}^\pi | S_{i,<j}^\pi, S_{<i}^\pi)$$

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

$$\max_{\theta} \log \left( \sum_{\pi \sim q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right] \right)$$

Learning via Back Propagation Through Time (BPTT)

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

What are their relationship?

$$G, \pi, S^{\pi}$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

$$\max_{\theta} \log \left( \sum_{\pi \sim q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right] \right)$$

Learning via Back Propagation Through Time (BPTT)

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

What are their relationship?

$$G, \pi, S^{\pi}$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

BFS ordering with random starting node!

$$\max_{\theta} \log \left( \sum_{\pi \sim q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right] \right)$$

Learning via Back Propagation Through Time (BPTT)

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

Pros:

- Autoregressive models can capture the dependencies among nodes/edges
- Enable the usage of RNNs for graph generation
- It could generate one row (i.e., multiple edges) of adjacency matrix at one step

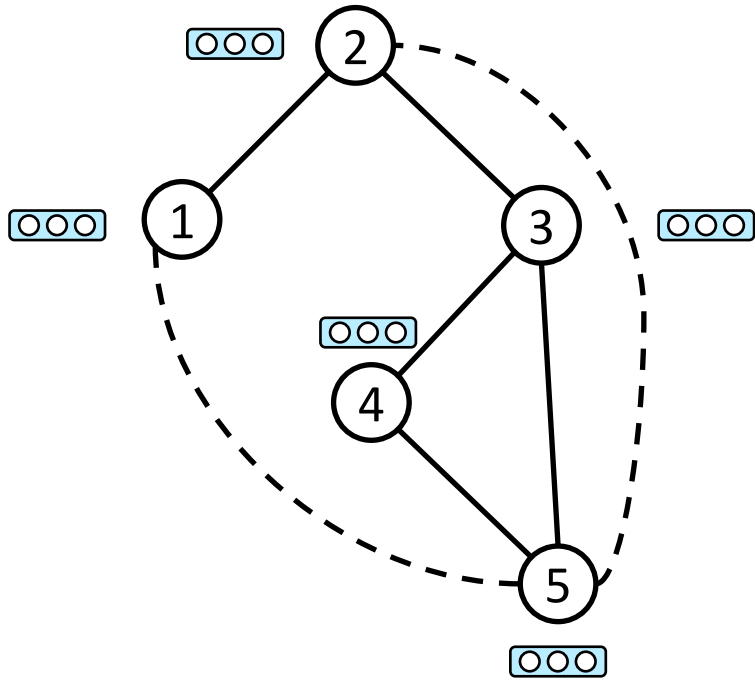
Cons:

- The likelihood is permutation-dependent
- Generating a medium-sized graph could still require many generation/unrolling steps
- Learning long-unrolled RNNs with BPTT is hard



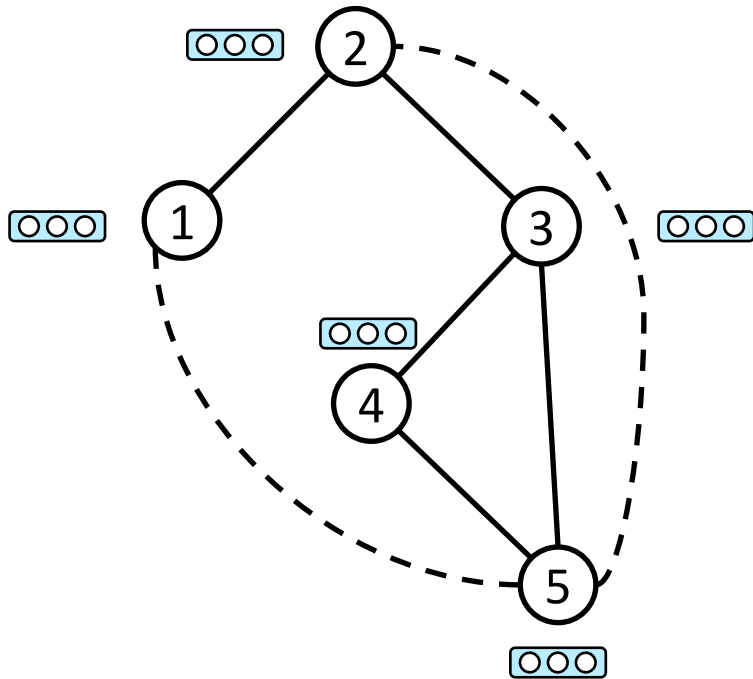
# Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]



# Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]

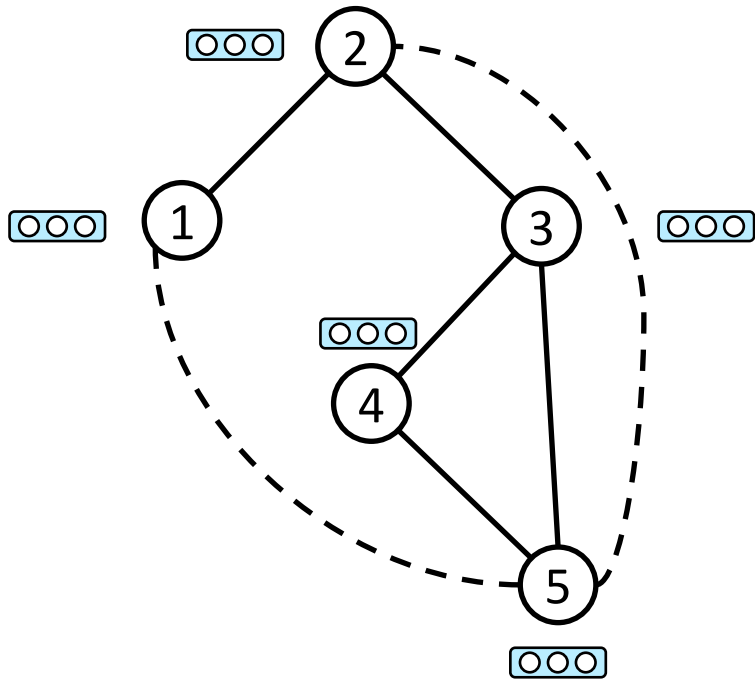


- Sequential Order Bias

More serious in RNNs [5] than GNNs [4] (permutation invariant), e.g., 5 often links to 3,4 rather than 1,2

# Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]



- Sequential Order Bias

More serious in RNNs [5] than GNNs [4] (permutation invariant), e.g., 5 often links to 3,4 rather than 1,2

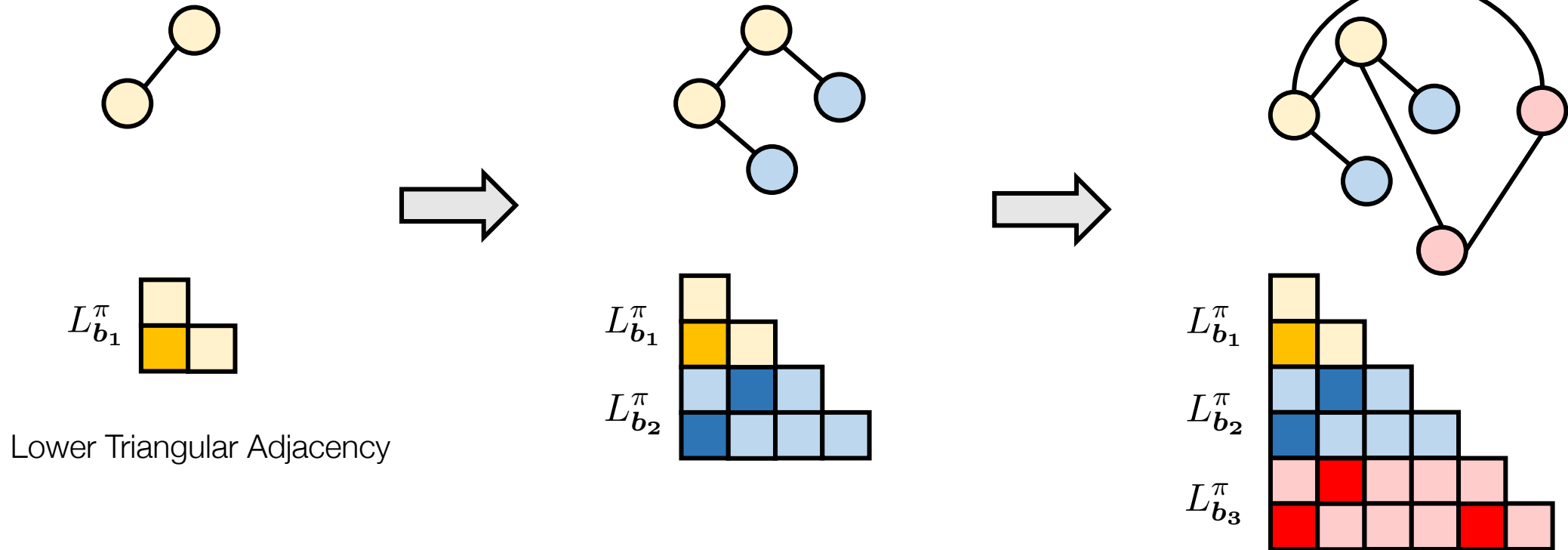
- Training Inefficiency

Training RNNs via BPTT [4,5] is impractical for moderately large graphs (~1k)

# Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]

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



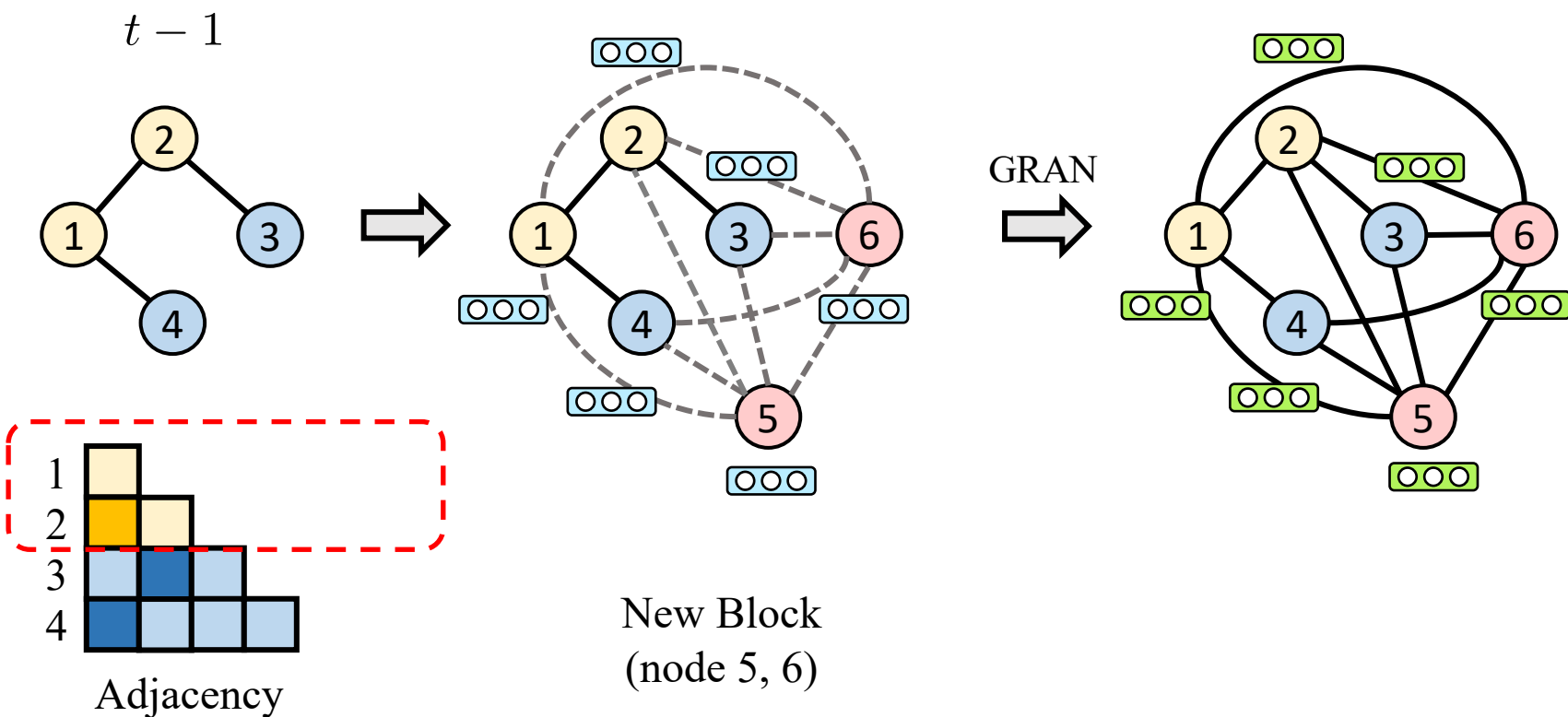
- Sequential Order Bias
- Training Inefficiency

**Graph Neural Network + Attention!**

**Block Generation + Parallel Training!**

# Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]



$$m_{ij}^t = f(h_i^t - h_j^t)$$

$$\tilde{h}_i^t = [h_i^t, x_i]$$

$$a_{ij}^t = \text{Sigmoid} \left( g(\tilde{h}_i^t - \tilde{h}_j^t) \right)$$

$$h_i^{t+1} = \text{GRU} \left( h_i^t, \sum_{j \in \mathcal{N}(i)} a_{ij}^t m_{ij}^t \right)$$

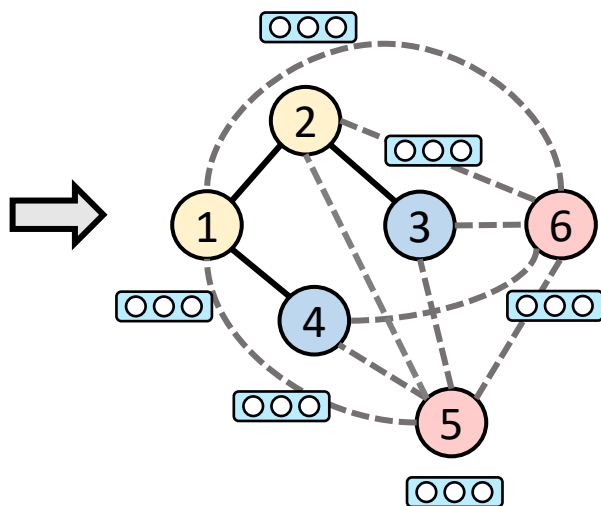
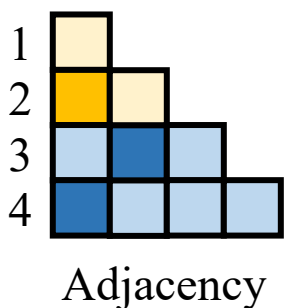
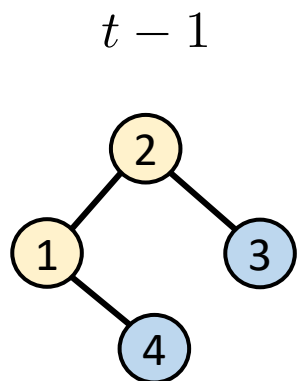
# Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]

$$p(L_{\mathbf{b}_t}^\pi | L_{\mathbf{b}_1}^\pi, \dots, L_{\mathbf{b}_{t-1}}^\pi) = \sum_{k=1}^K \alpha_k \prod_{i \in \mathbf{b}_t} \prod_{1 \leq j \leq i} \theta_{k,i,j},$$

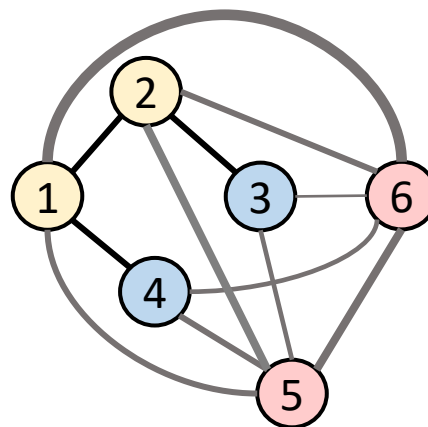
$$\alpha_1, \dots, \alpha_K = \text{Softmax} \left( \sum_{i \in \mathbf{b}_t, 1 \leq j \leq i} \text{MLP}_\alpha(h_i^R - h_j^R) \right),$$

$$\theta_{1,i,j}, \dots, \theta_{K,i,j} = \text{Sigmoid} (\text{MLP}_\theta(h_i^R - h_j^R)) \quad t$$



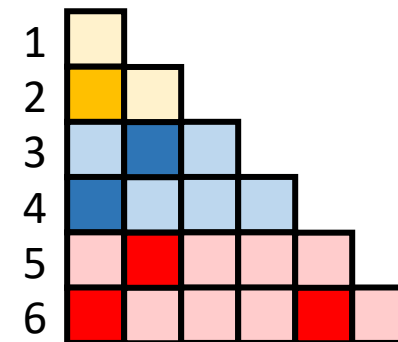
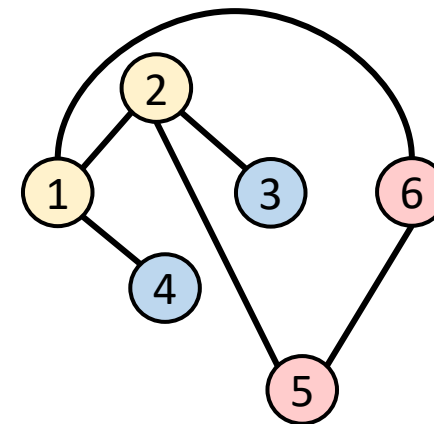
New Block  
(node 5, 6)

GRAN



Distribution over *dashed edges*

Sampling



Parallel Subgraph Training!

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GRAN [6]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

$$\max_{\theta} \log \left( \sum_{\pi \sim q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right] \right)$$

Parallel Subgraph Learning via Back Propagation. No BPTT !

# Autoregressive Models for Graphs

RNN-based Autoregressive Model, GRAN [6]

$$\max_{\theta} \log p_{\theta}(G) = \log \left( \sum_{\pi} p_{\theta}(G, \pi) \right)$$

$$p_{\theta}(G) = \sum_{\pi} p_{\theta}(G, \pi) = \mathbb{E}_{q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right]$$

- BFS ordering starting from the largest degree node
- DFS ordering starting from the largest degree node
- K-core descending ordering
- Node degree ascending ordering
- Node degree descending ordering

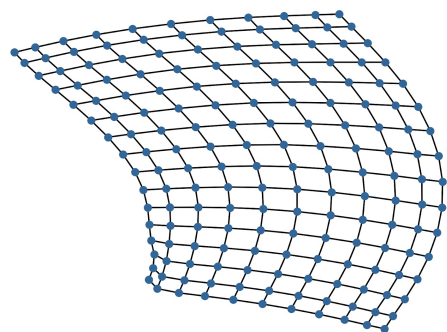
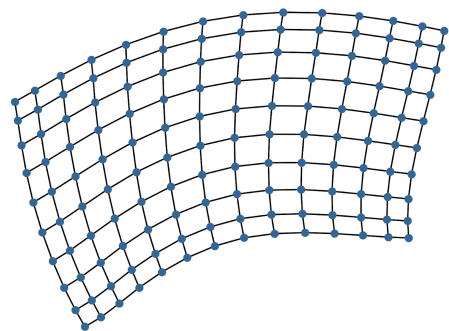
$$\max_{\theta} \log \left( \sum_{\pi \sim q(\pi|G)} \left[ \frac{p_{\theta}(G, \pi)}{q(\pi|G)} \right] \right)$$

Parallel Subgraph Learning via Back Propagation. No BPTT !



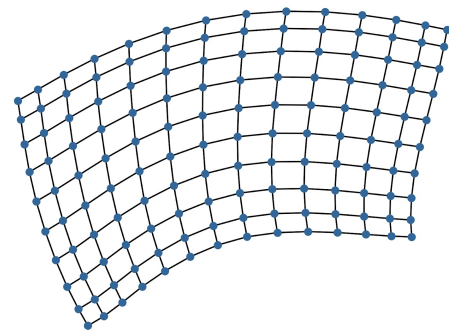
# Random Grid Graphs

Train

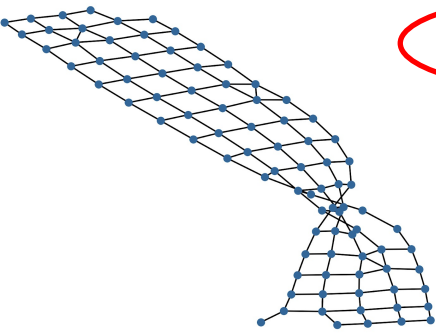


# Random Grid Graphs

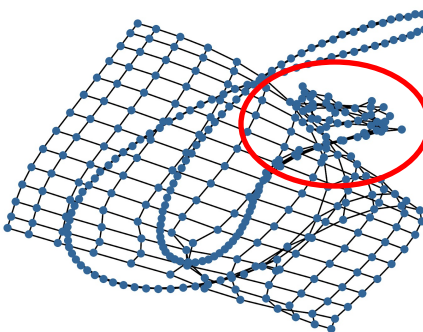
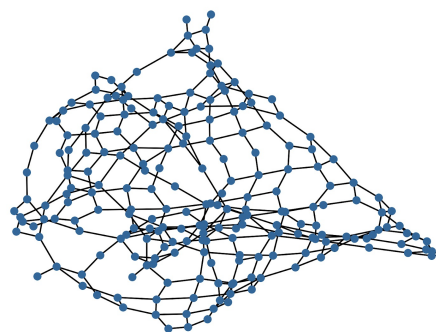
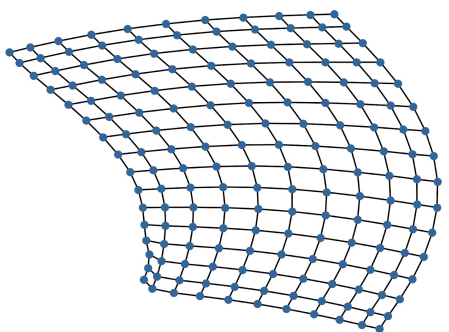
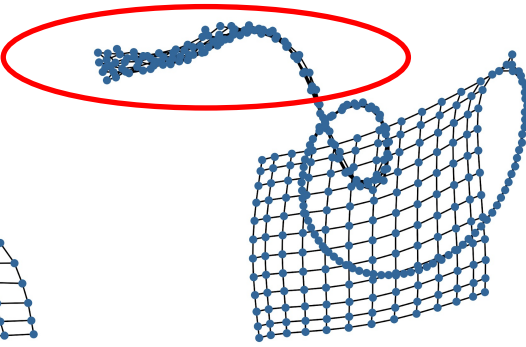
Train



Graph VAE

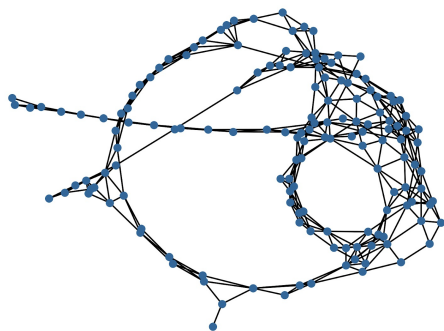
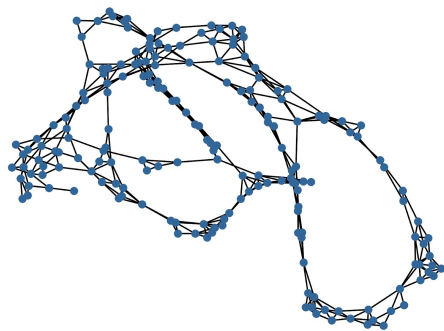


Graph RNN



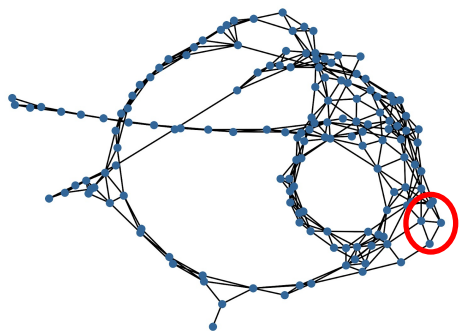
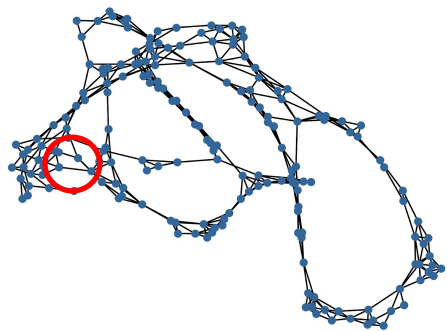
# Protein Graphs

Train

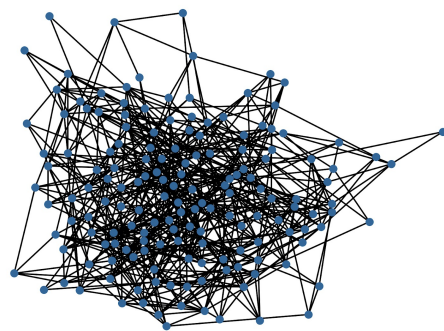
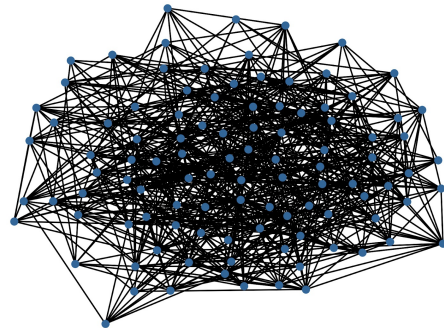


# Protein Graphs

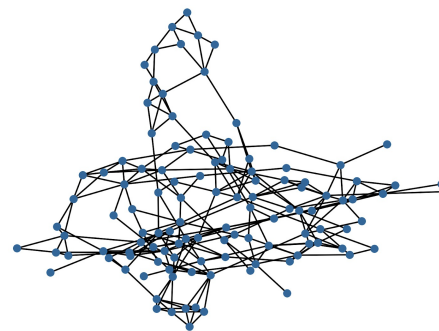
Train



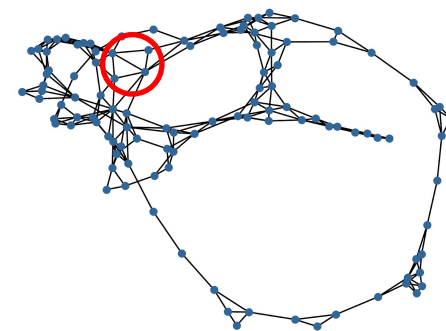
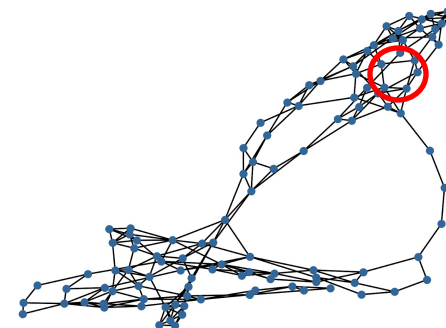
Graph VAE



Graph RNN

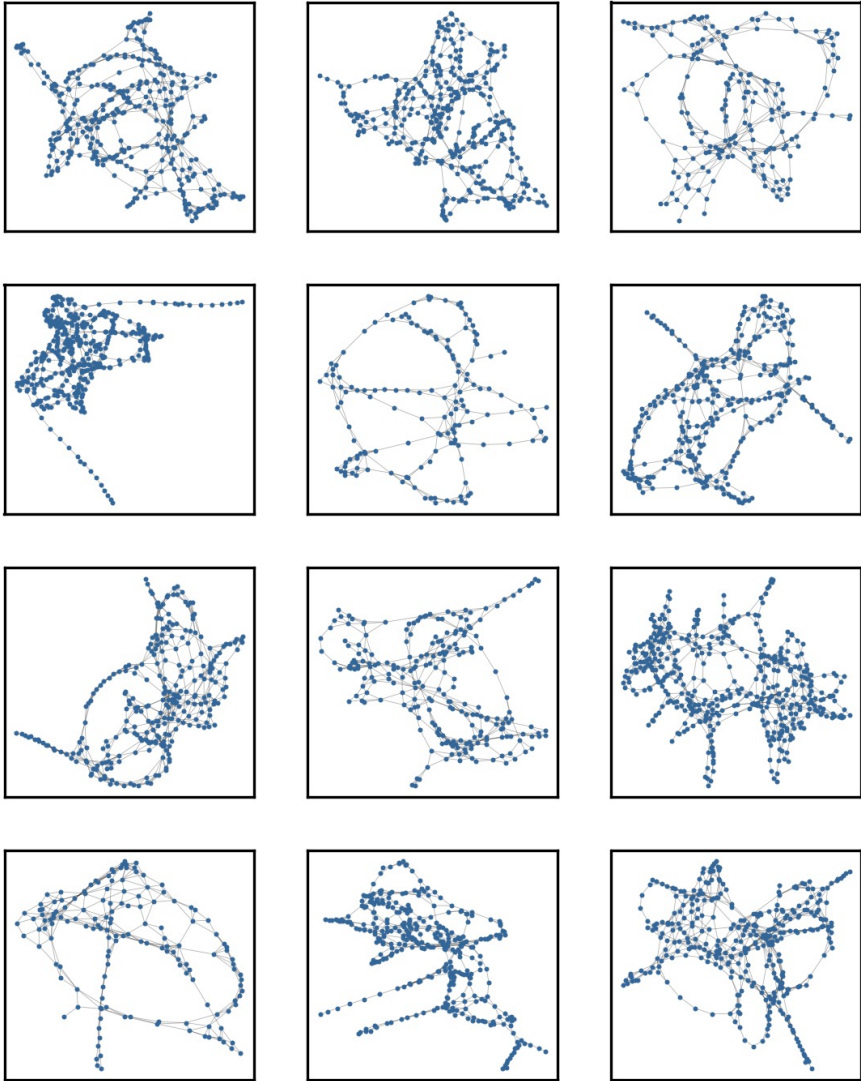


Ours

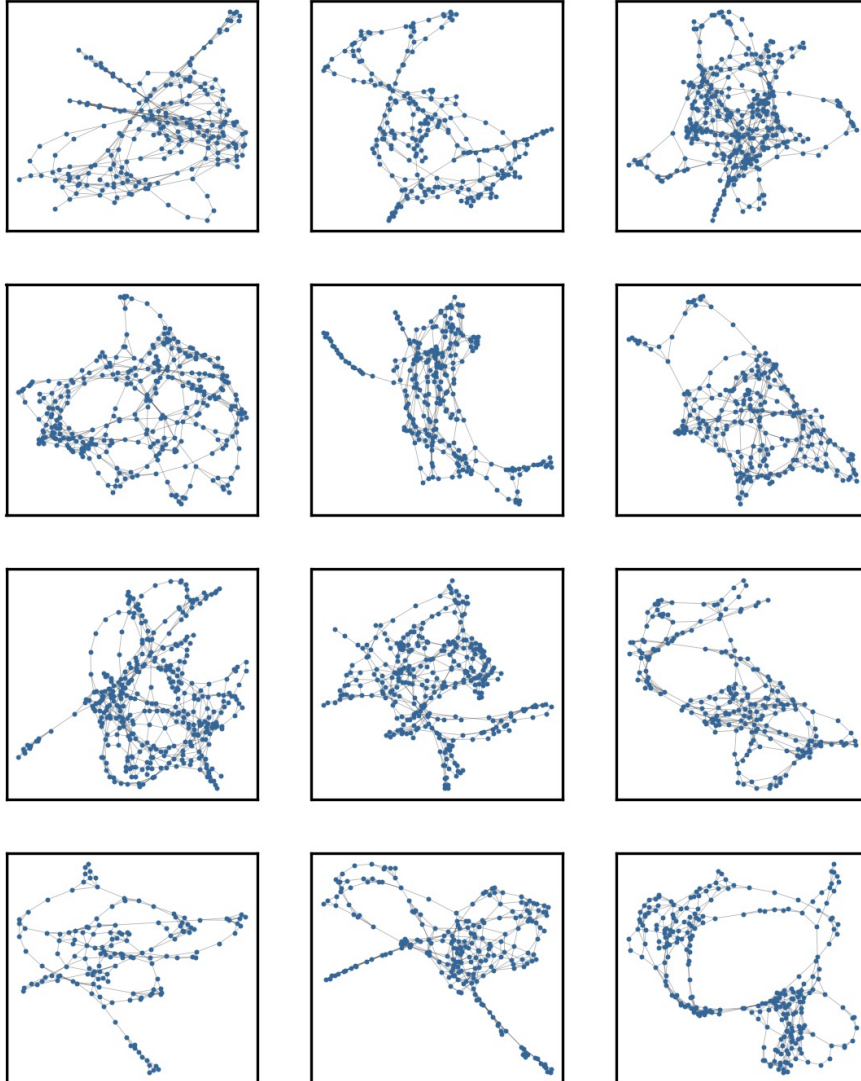


# Protein Graphs

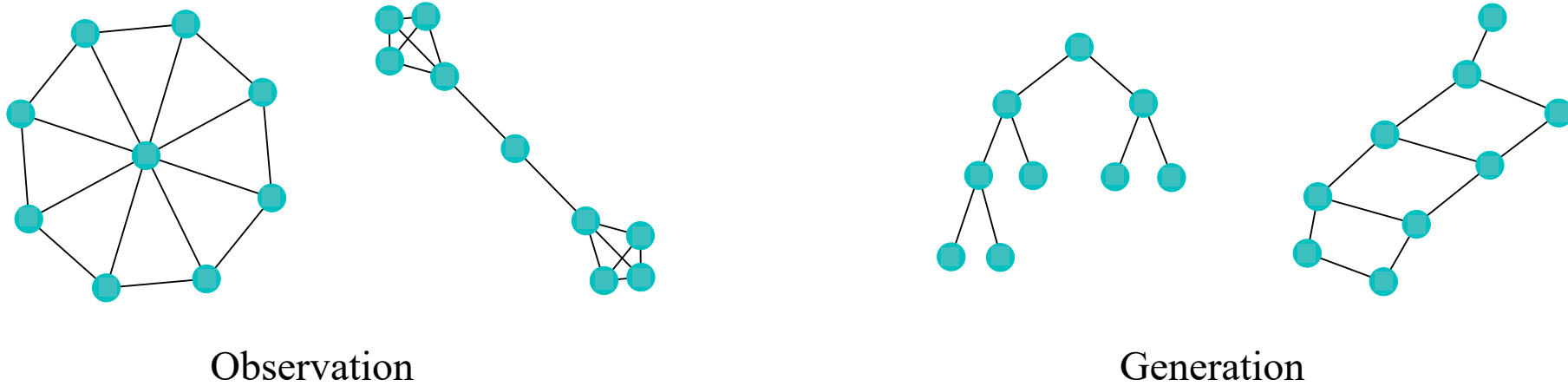
Train



Ours



# Quantitative Evaluation



Maximum Mean Discrepancy (MMD) between two distributions of graph statistics

$$\text{MMD}(P, Q) = \sup_{f \in \mathcal{F}} \mathbb{E}_{X \sim P}[f(X)] - \mathbb{E}_{Y \sim Q}[f(Y)]$$

- Degree distribution
- Clustering Coefficients
- # 4-node orbits
- Spectrum

# Protein Graphs

# train/dev/test graphs = 734/183/184

Max: # nodes = 500, # edges = 1575

Mean: # nodes = 258, # edges = 646

Models	Degree	Clustering Coeff.	Orbits	Spectrum
Erdős-Rényi	0.0564	1.00	1.54	0.0913
GraphVAE	0.480	0.0714	0.740	0.110
GraphRNN-S	0.0402	<b>0.0479</b>	0.230	0.210
GraphRNN	0.0106	0.140	0.880	0.0188
Ours	<b>0.00198</b>	0.0486	<b>0.130</b>	<b>0.00513</b>

\*For all metrics, the lower the better

# References

- [1] Jin, W., Barzilay, R. and Jaakkola, T., 2018, July. Junction tree variational autoencoder for molecular graph generation. In International conference on machine learning (pp. 2323-2332). PMLR.
- [2] Chu, H., Li, D., Acuna, D., Kar, A., Shugrina, M., Wei, X., Liu, M.Y., Torralba, A. and Fidler, S., 2019. Neural turtle graphics for modeling city road layouts. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 4522-4530).
- [3] Nash, C., Ganin, Y., Eslami, S.A. and Battaglia, P., 2020, November. Polygen: An autoregressive generative model of 3d meshes. In International Conference on Machine Learning (pp. 7220-7229). PMLR.
- [4] Li, Y., Vinyals, O., Dyer, C., Pascanu, R. and Battaglia, P., 2018. Learning deep generative models of graphs. arXiv preprint arXiv:1803.03324.
- [5] You, J., Ying, R., Ren, X., Hamilton, W. and Leskovec, J., 2018, July. Graphrnn: Generating realistic graphs with deep autoregressive models. In International conference on machine learning (pp. 5708-5717). PMLR.
- [6] Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D.K., Urtasun, R. and Zemel, R., 2019. Efficient graph generation with graph recurrent attention networks. Advances in Neural Information Processing Systems, 32.



Questions?