EECE 571F: Deep Learning with Structures

Lecture 7: Deep Generative Models of Graphs
Autoregressive Models

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

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)

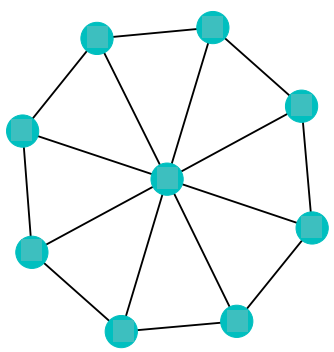
……

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

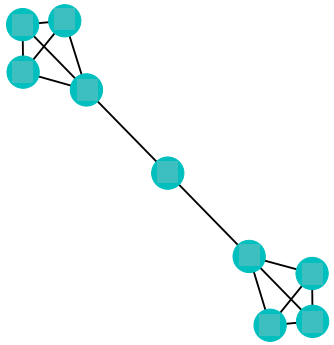
Learning could be ***probabilistic*** or *deterministic*!

Deep Generative Models of Graphs

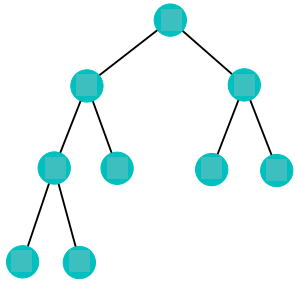
G



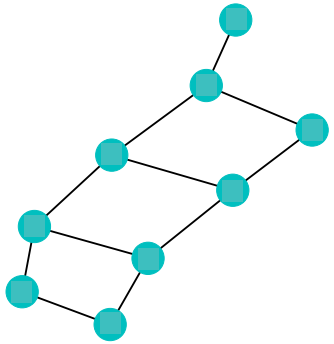
Wheel



Barbell



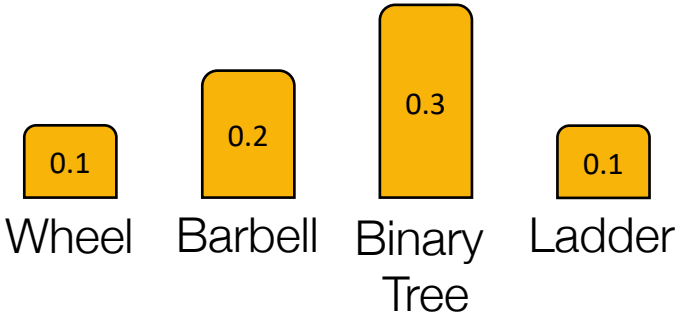
Binary Tree



Ladder

.....

$P(G)$



.....

$G \sim P(G)$

Applications

Drug discovery, e.g. molecules [1]

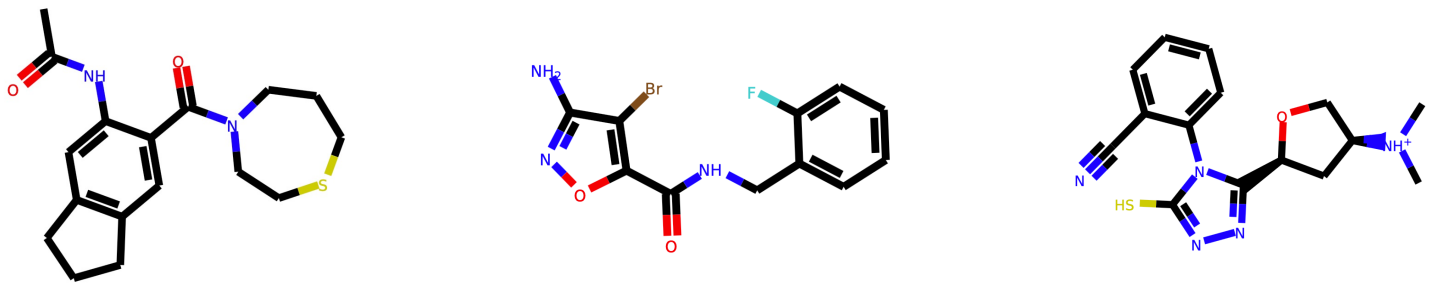


Image Credit: [1]

Applications

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

Image Credit: [2]

Applications

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

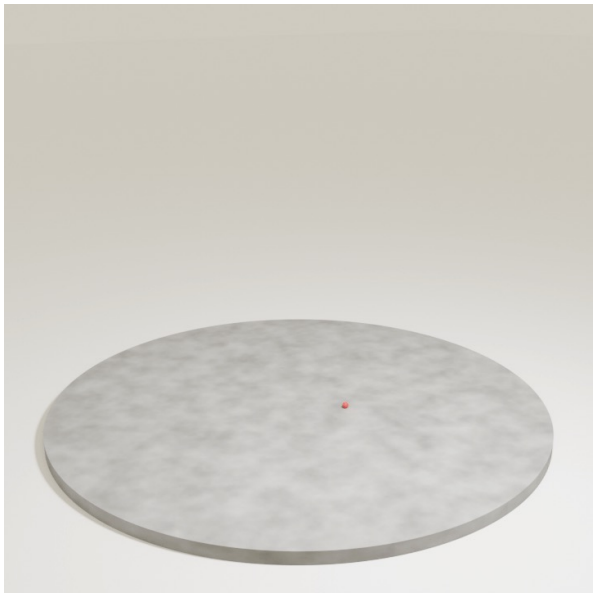


Image Credit: [3]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

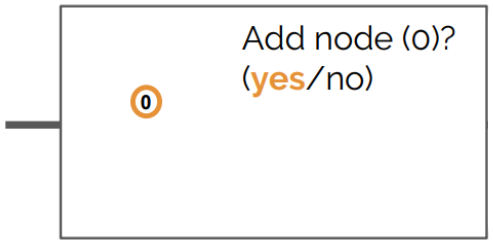


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

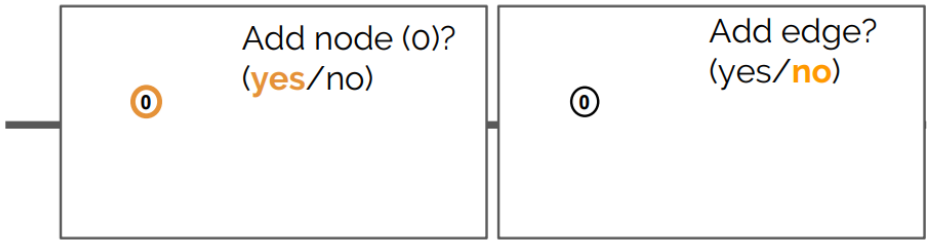


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

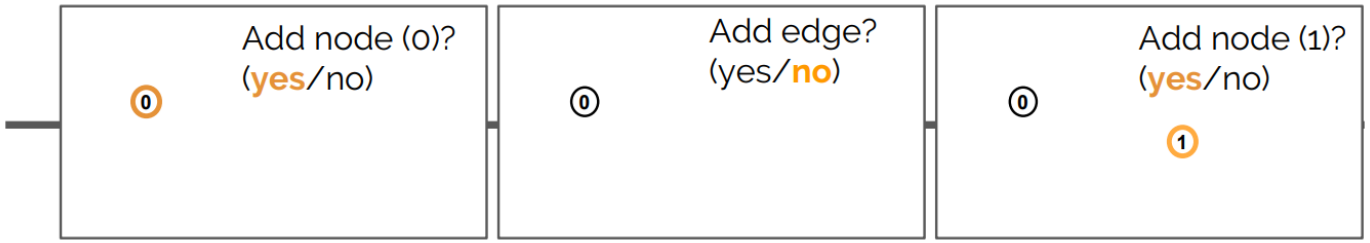


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

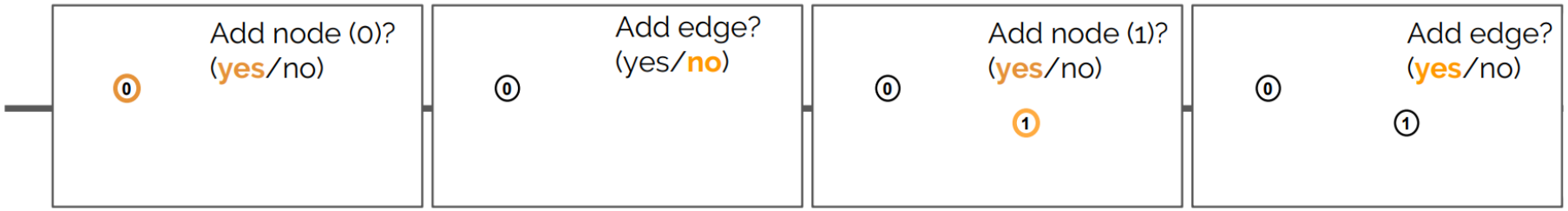


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

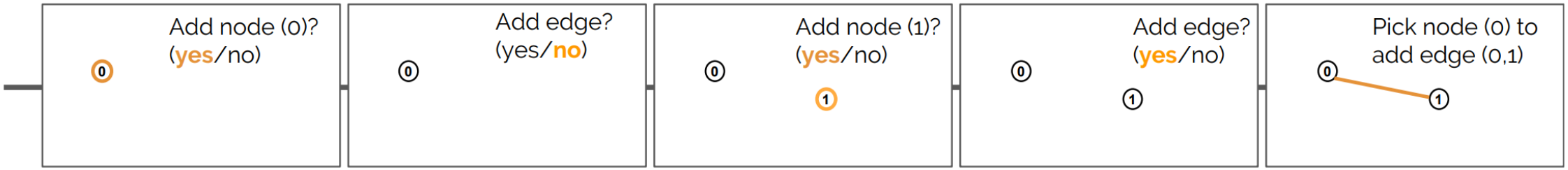


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

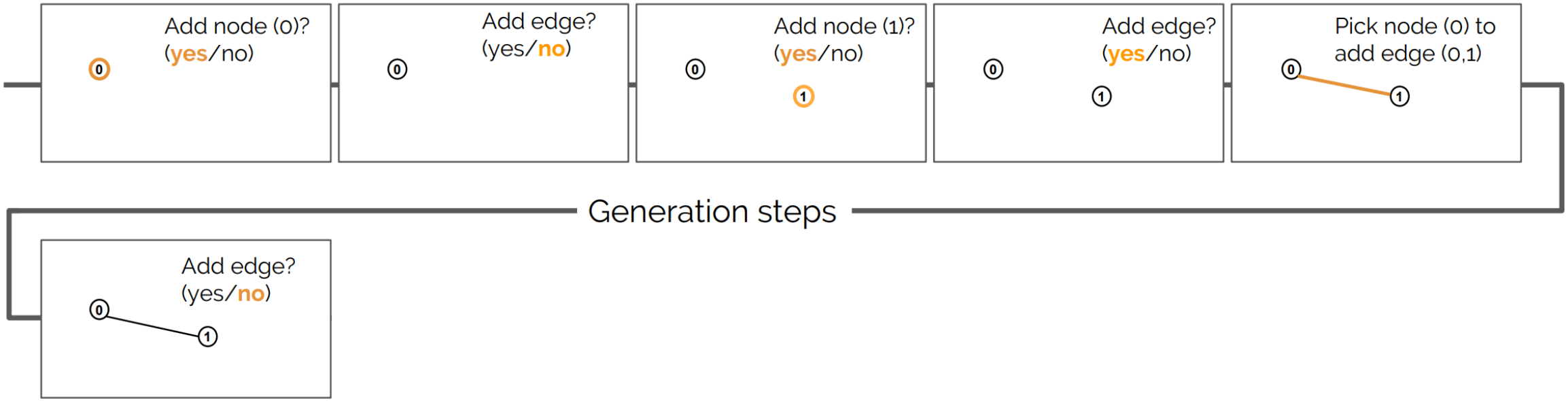


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

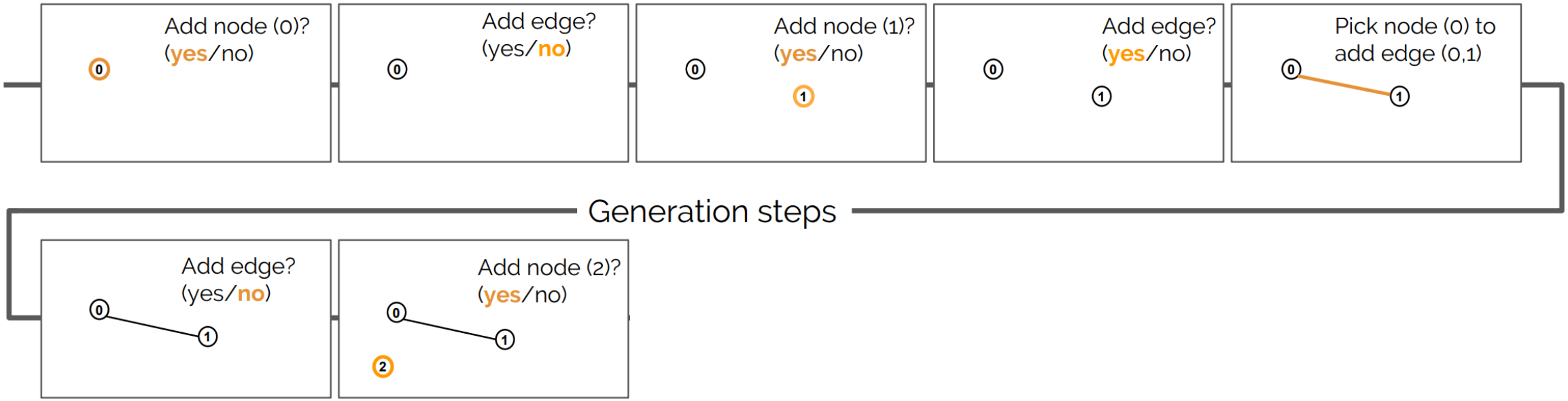


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

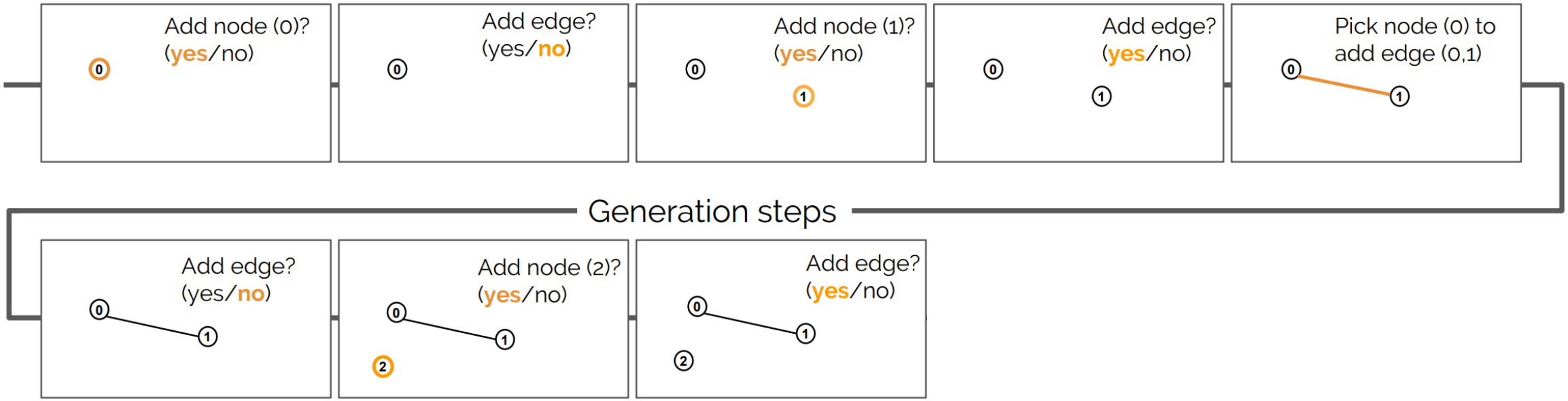


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

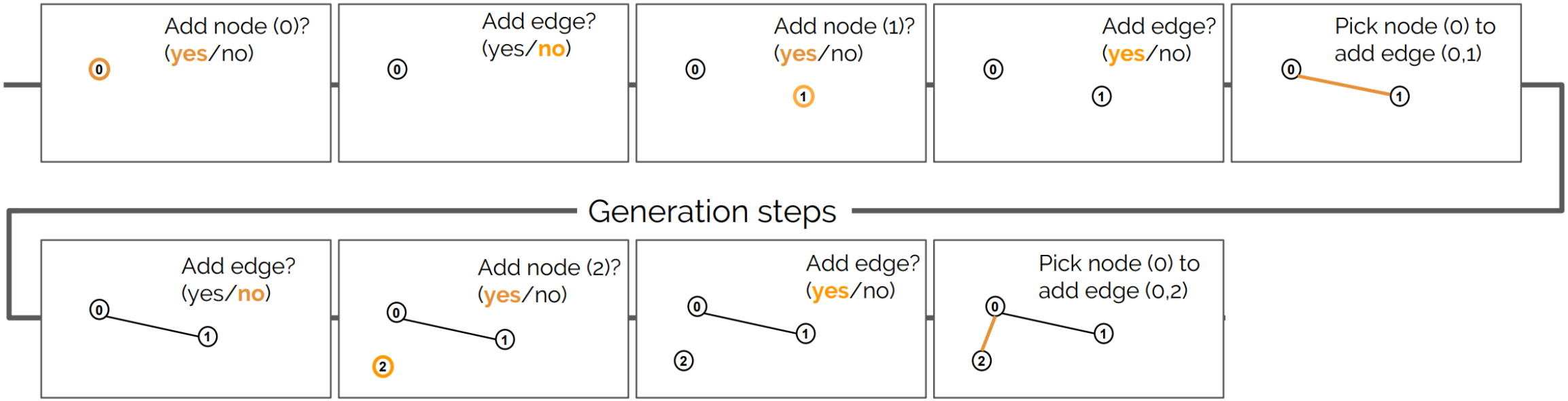


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

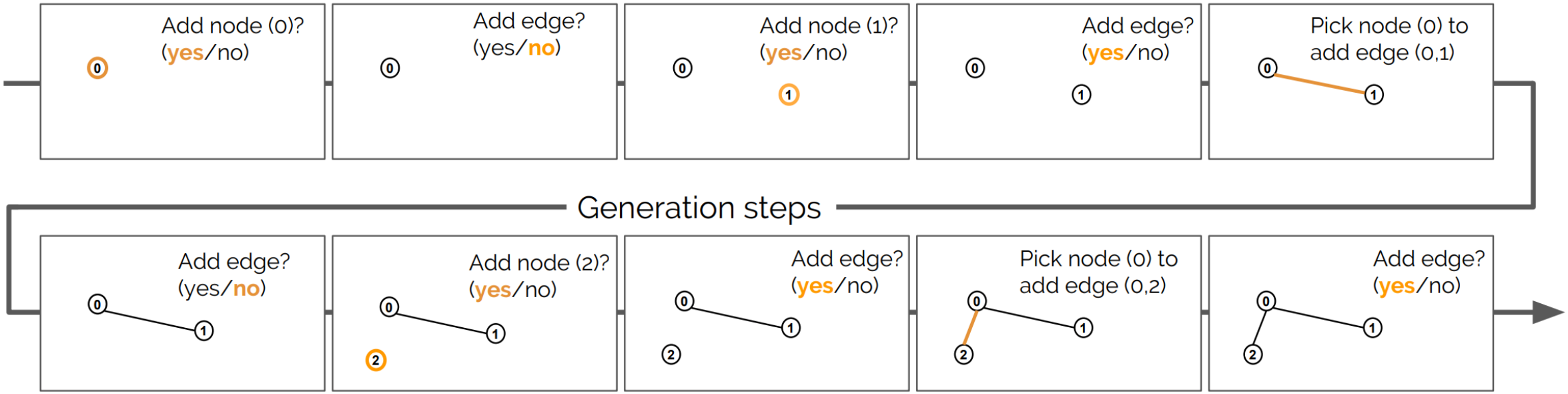
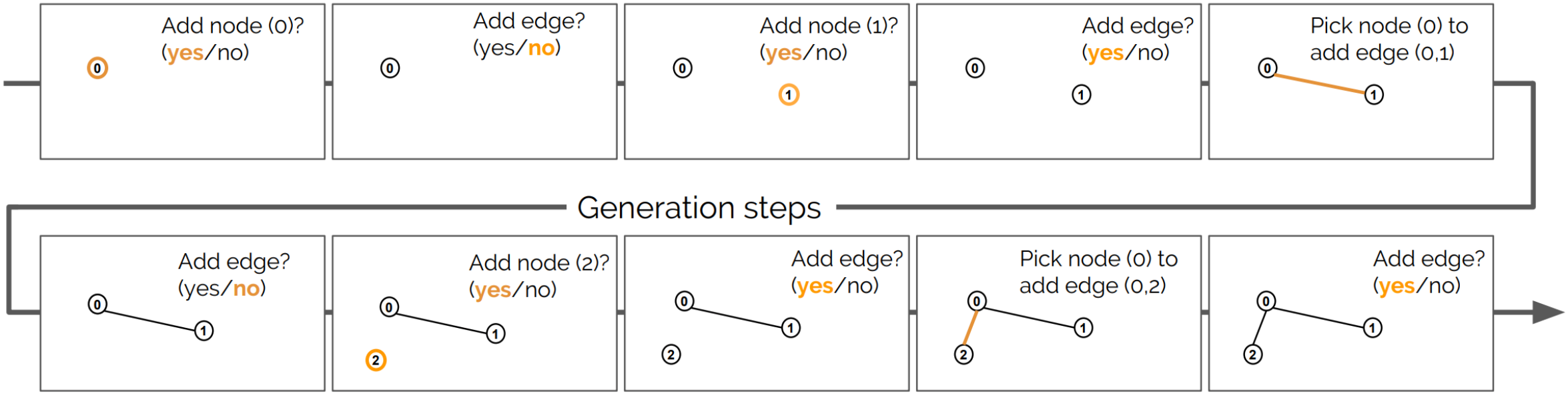


Image Credit: [4]

Autoregressive Models for Graphs

Let us start with an intuitive generation process:

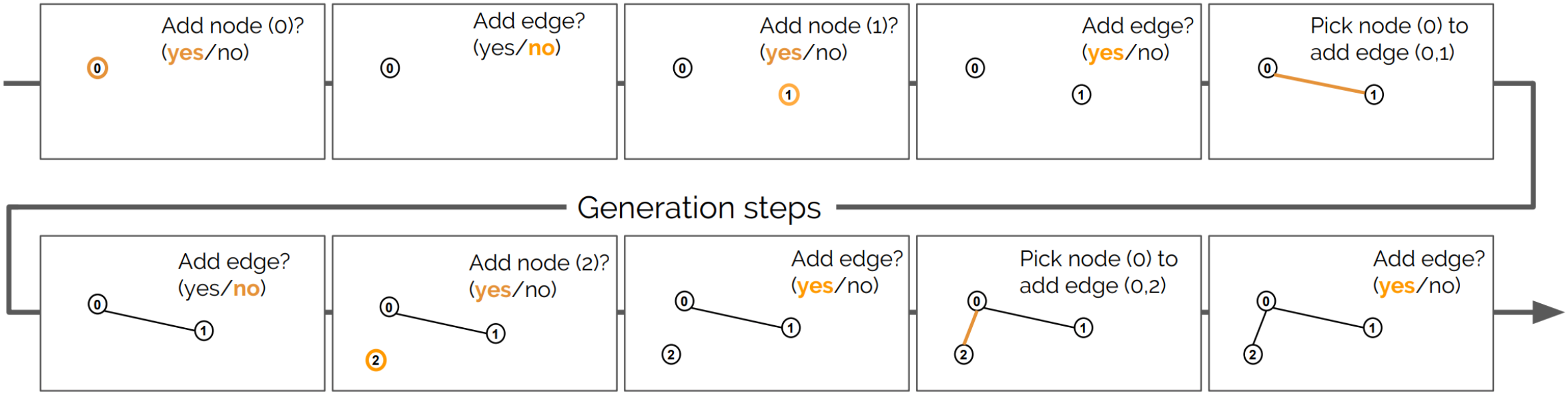


- *Graph Generation = Sequential Decision Making!*

Image Credit: [4]

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!*

Image Credit: [4]

Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

Image Credit: [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*

Image Credit: [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*

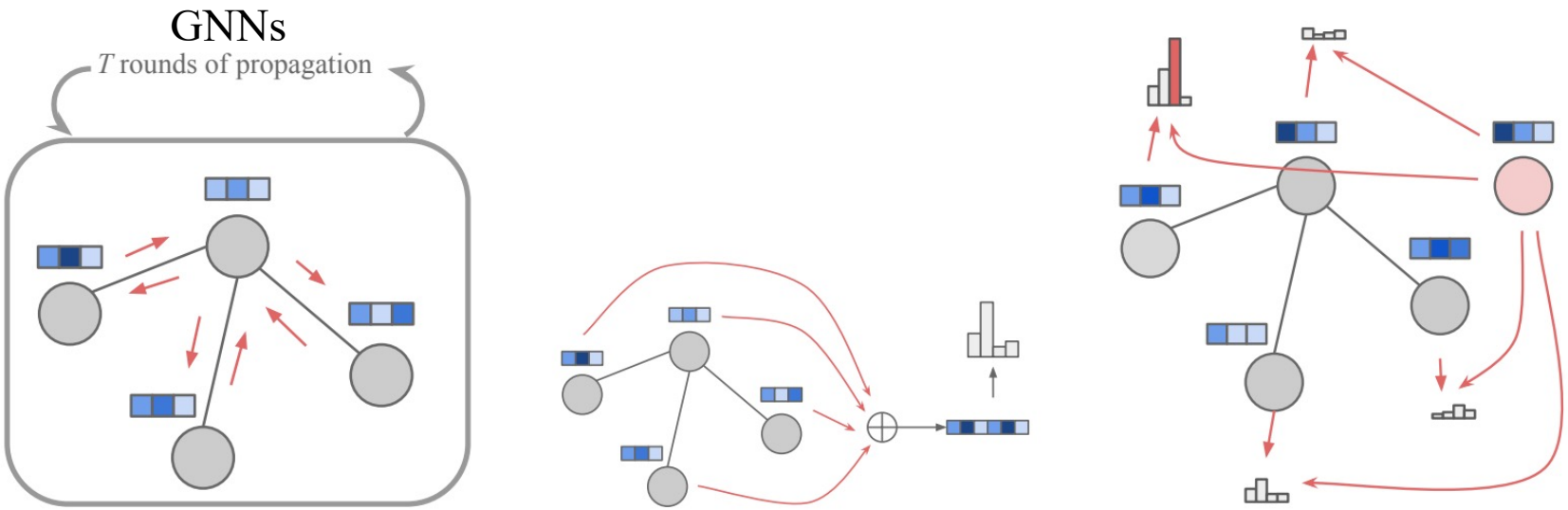


Image Credit: [4]

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*

$$\mathbf{h}_V^{(T)} = \text{prop}^{(T)}(\mathbf{h}_V, G)$$
$$\mathbf{h}_G = R(\mathbf{h}_V^{(T)}, G)$$
$$f_{addnode}(G) = \text{softmax}(f_{an}(\mathbf{h}_G))$$
$$f_{addedge}(G, v) = \sigma(f_{ae}(\mathbf{h}_G, \mathbf{h}_v^{(T)}))$$
$$s_u = f_s(\mathbf{h}_u^{(T)}, \mathbf{h}_v^{(T)}), \quad \forall u \in V$$
$$f_{nodes}(G, v) = \text{softmax}(\mathbf{s})$$

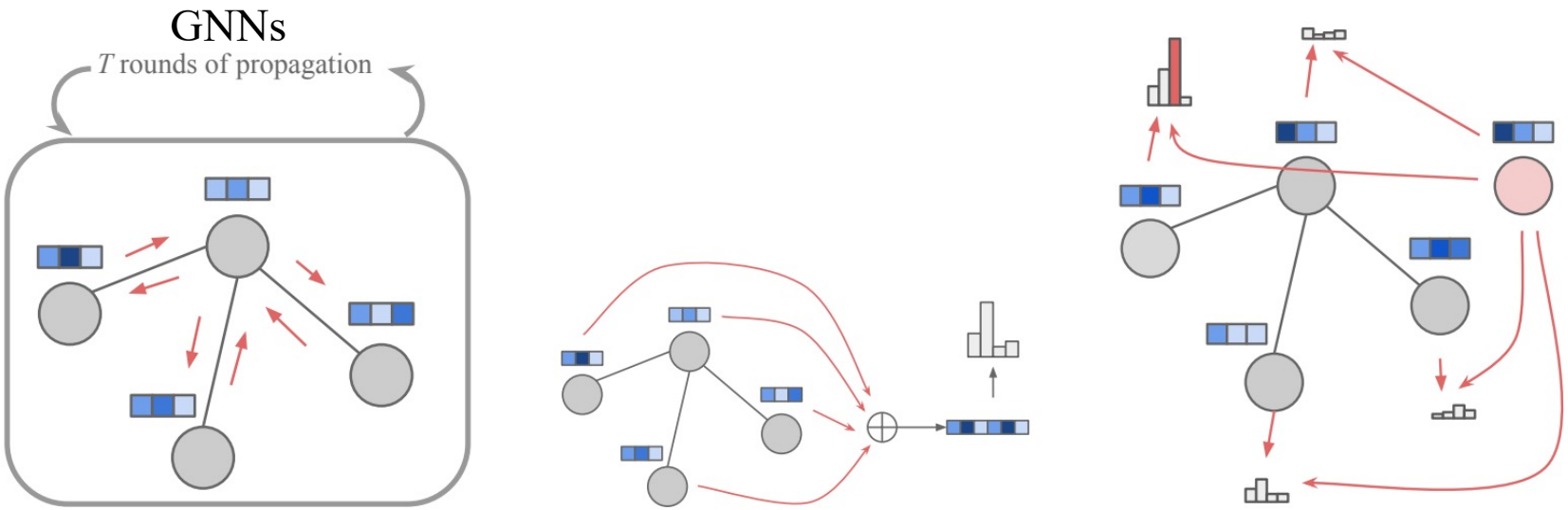


Image Credit: [4]

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*

Recurrent Neural Network

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

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

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

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

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

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

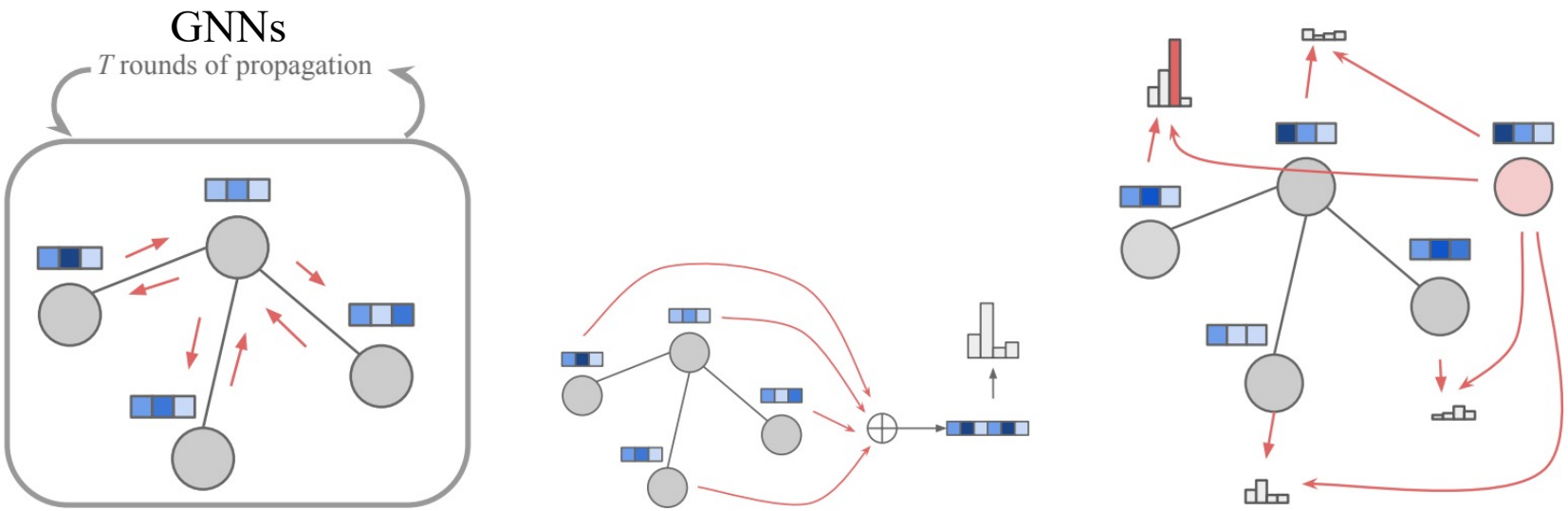


Image Credit: [4]

Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

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

Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

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

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

Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

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

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

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

Autoregressive Models for Graphs

GNN-RNN-based Autoregressive Model [4]

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

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

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

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:

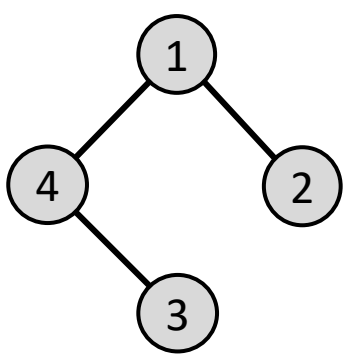
- Autoregressive models can capture the dependencies among nodes/edges
- Enable the usage of GNNs and RNNs for graph generation

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]

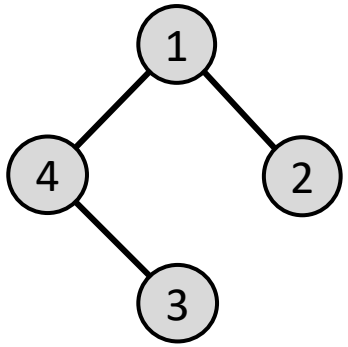


	1	2	3	4
1	0	1	0	1
2	1	0	0	0
3	0	0	0	1
4	1	0	1	0

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

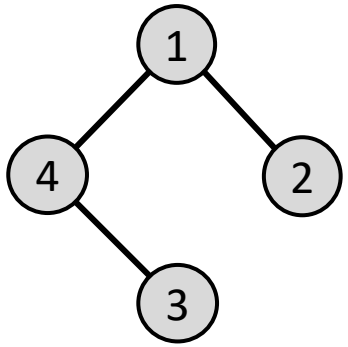
$$S_i^\pi = (A_{1,i}^\pi, ..., A_{i-1,i}^\pi)^T, \forall i \in \{2, ..., n\}$$



	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]



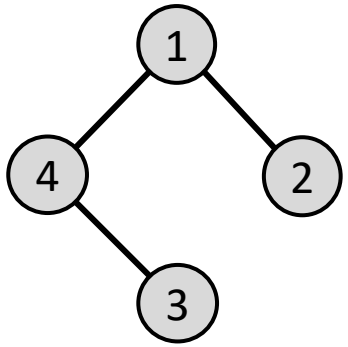
	1	2	3	4
1	0	1	0	1
2	1	0	0	0
3	0	0	0	1
4	1	0	1	0

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

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

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

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

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

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

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

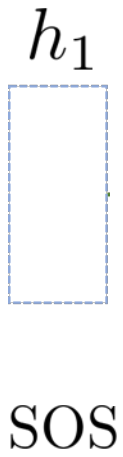
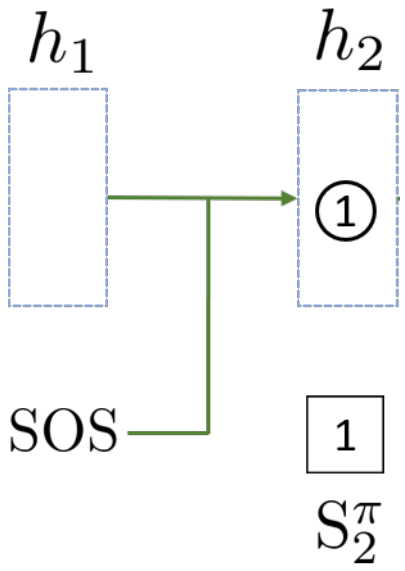


Image Credit: [5]

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

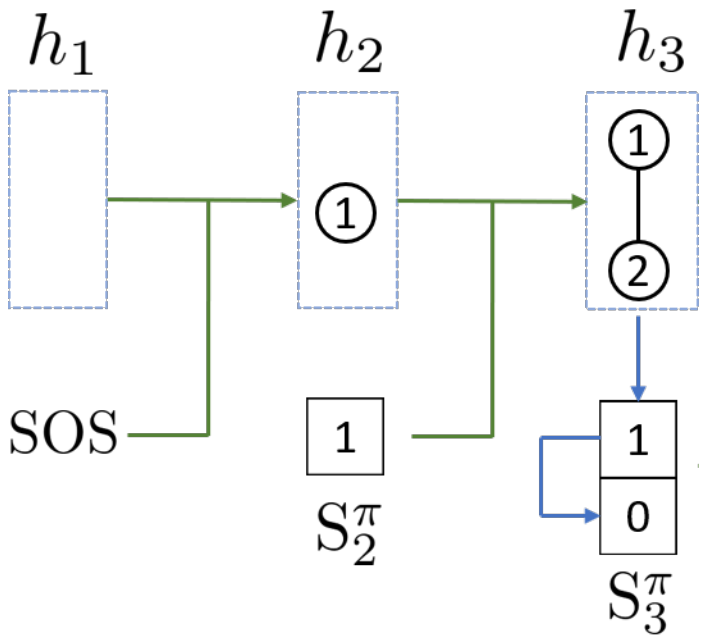


$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$
$$\theta_i = f_{\text{out}}(h_i)$$
$$S_i^\pi \sim p_{\theta_i}$$

Image Credit: [5]

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

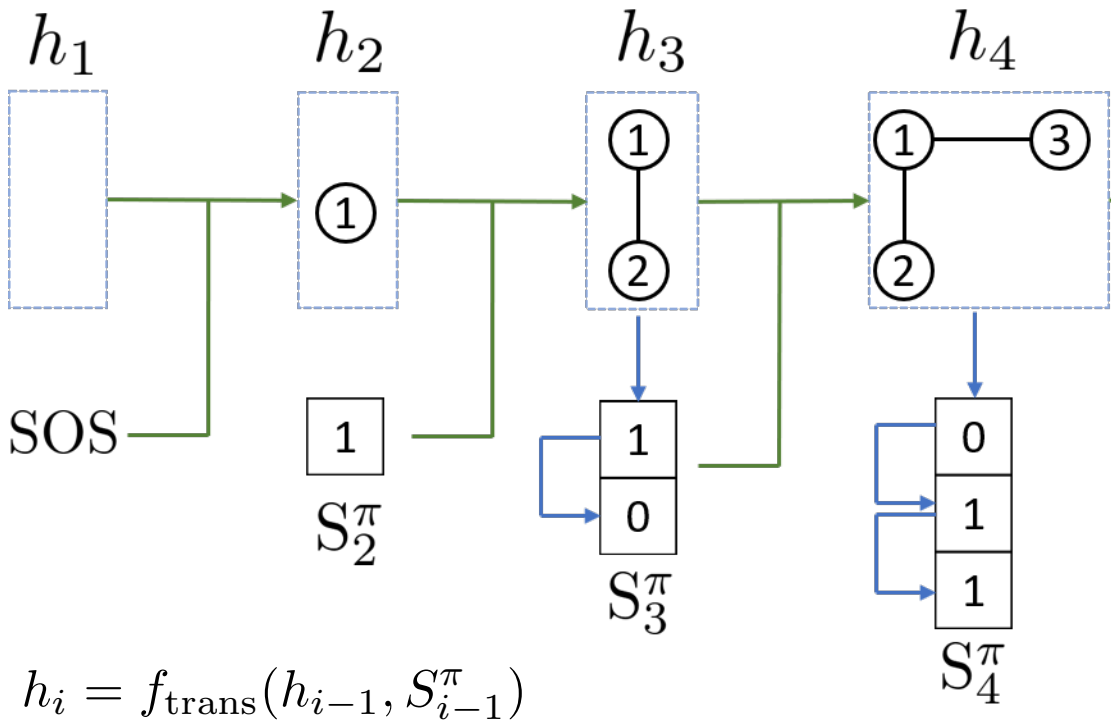


$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$
$$\theta_i = f_{\text{out}}(h_i)$$
$$S_i^\pi \sim p_{\theta_i}$$

Image Credit: [5]

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

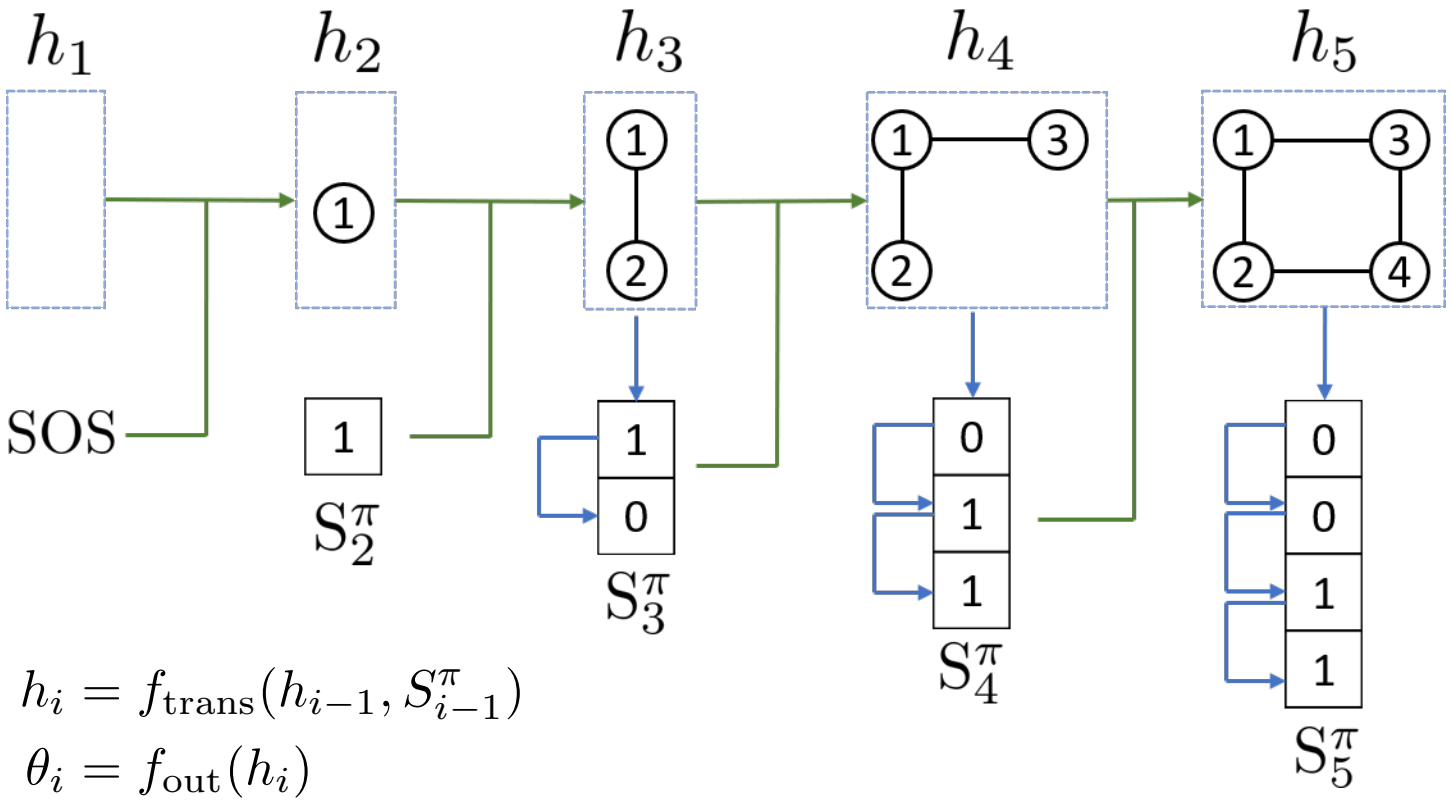


$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$
$$\theta_i = f_{\text{out}}(h_i)$$
$$S_i^\pi \sim p_{\theta_i}$$

Image Credit: [5]

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

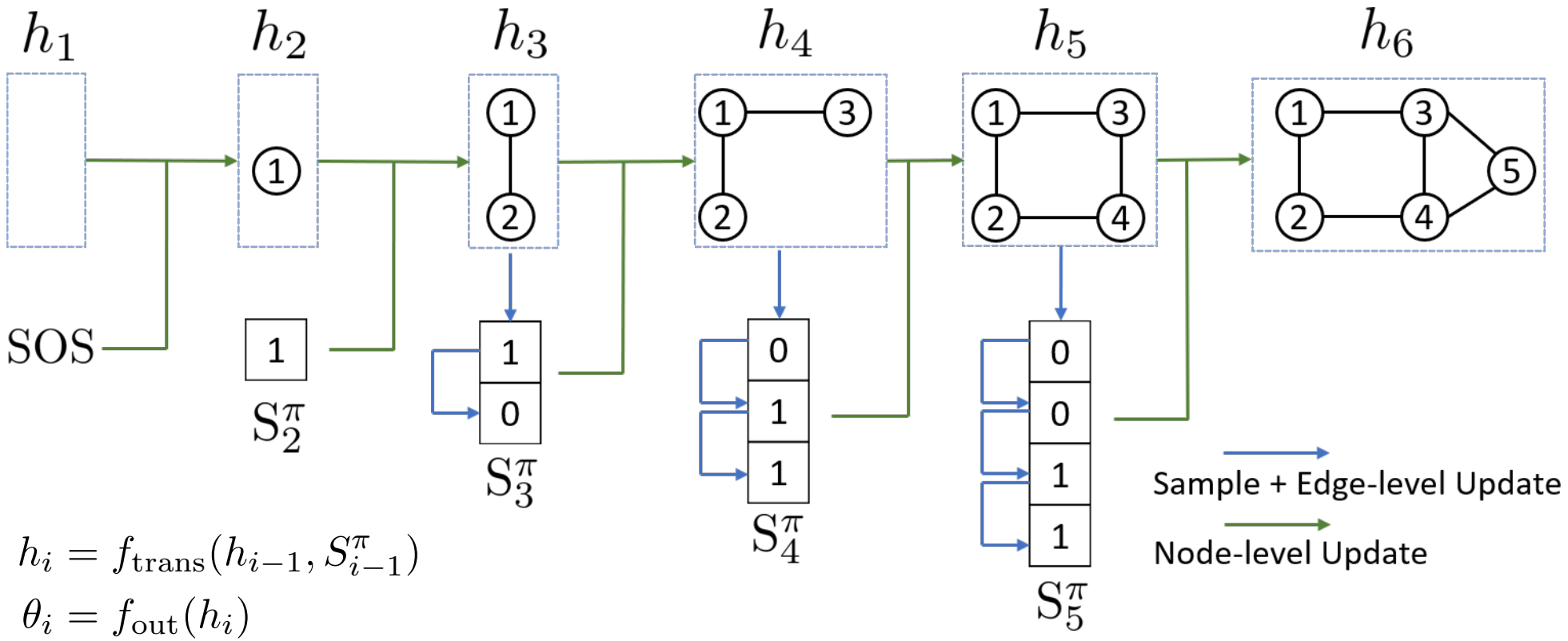


$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$
$$\theta_i = f_{\text{out}}(h_i)$$
$$S_i^\pi \sim p_{\theta_i}$$

Image Credit: [5]

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

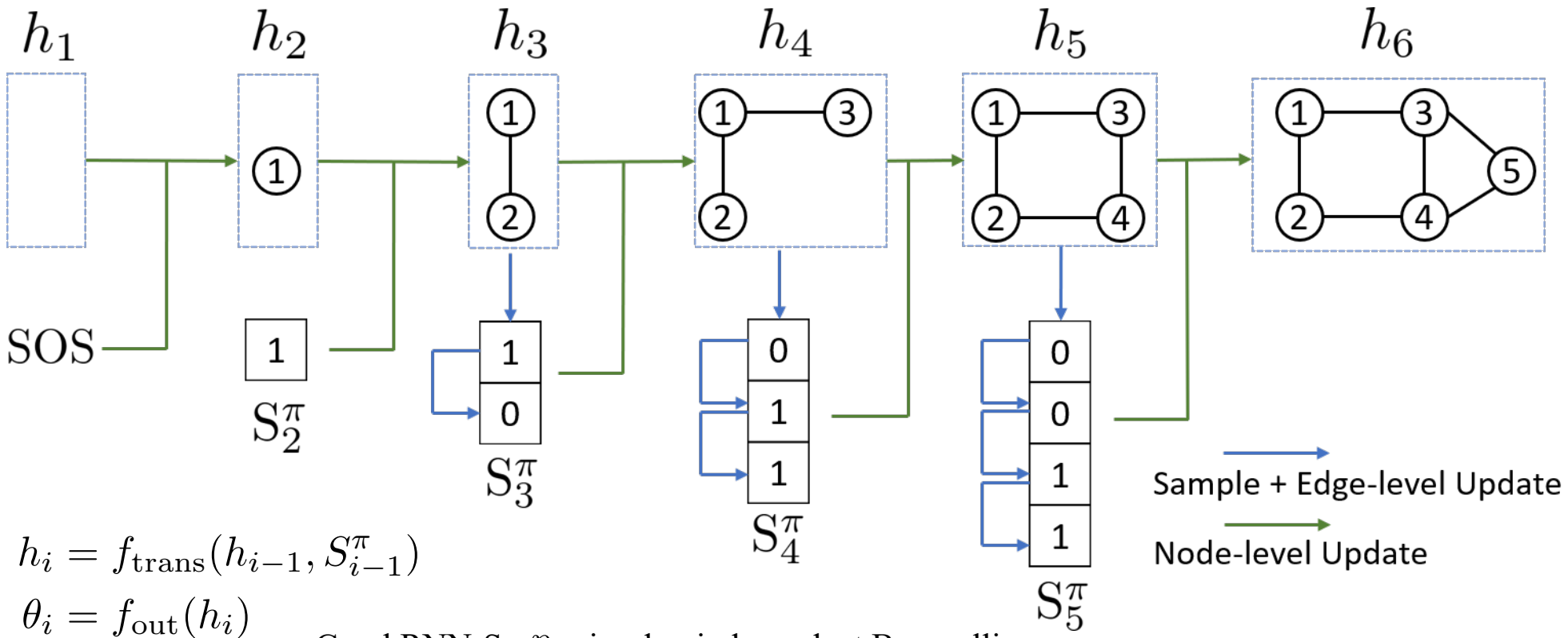


$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$
$$\theta_i = f_{\text{out}}(h_i)$$
$$S_i^\pi \sim p_{\theta_i}$$

Image Credit: [5]

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]



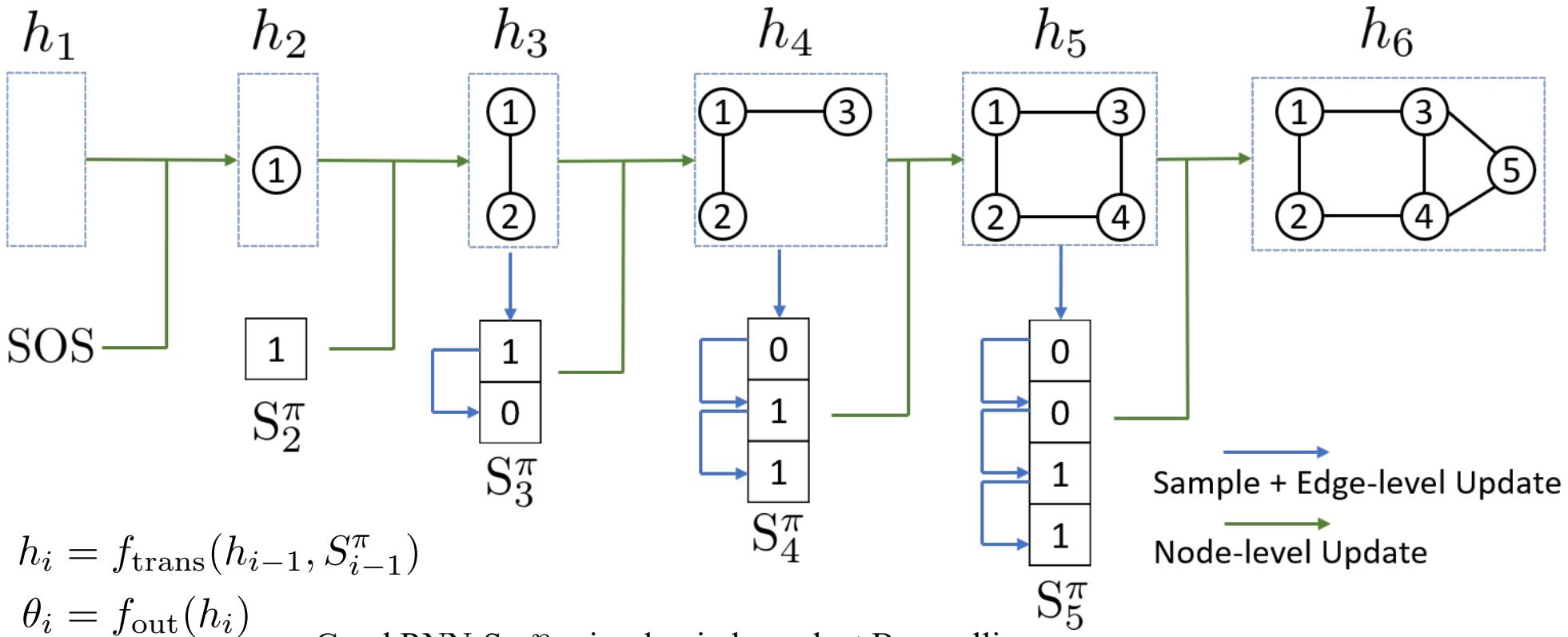
$$h_i = f_{\text{trans}}(h_{i-1}, S_{i-1}^\pi)$$
$$\theta_i = f_{\text{out}}(h_i)$$
$$S_i^\pi \sim p_{\theta_i}$$

GraphRNN-S: p_{θ_i} is edge-independent Bernoulli

Image Credit: [5]

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_{θ_i} is edge-independent Bernoulli

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

Image Credit: [5]

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

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

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

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

Learning via Back Propagation Through Time (BPTT)

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

What are their relationship?

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

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

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

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

Learning via Back Propagation Through Time (BPTT)

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

What are their relationship?

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

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

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

BFS ordering with random starting node!

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

Learning via Back Propagation Through Time (BPTT)

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GraphRNN [5]

Pros:

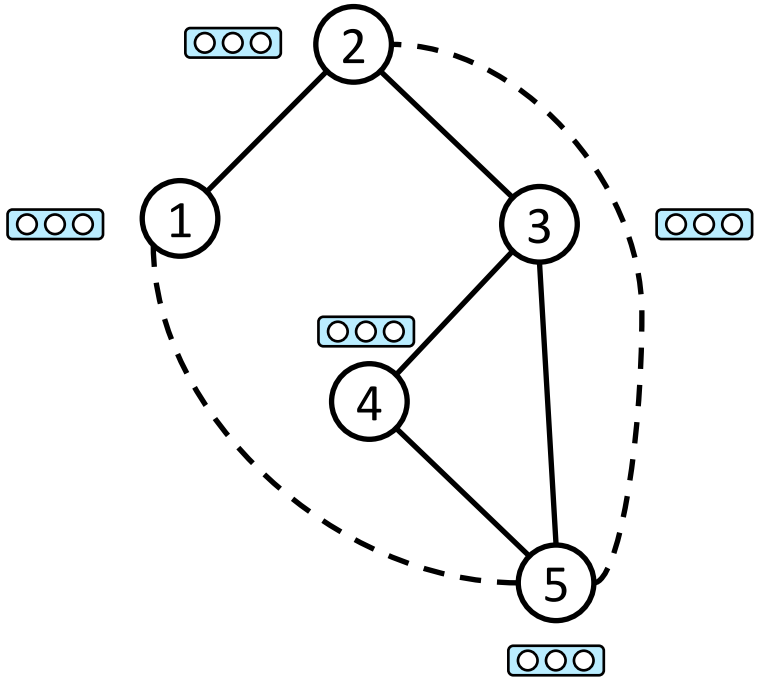
- Autoregressive models can capture the dependencies among nodes/edges
- Enable the usage of RNNs for graph generation
- It could generate one row (i.e., multiple edges) of adjacency matrix at one step

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

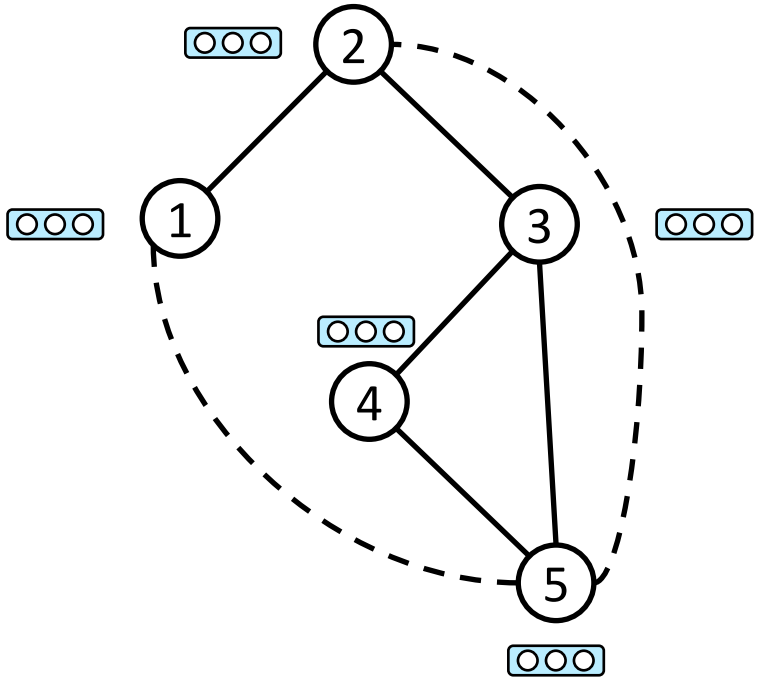
Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]



Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]

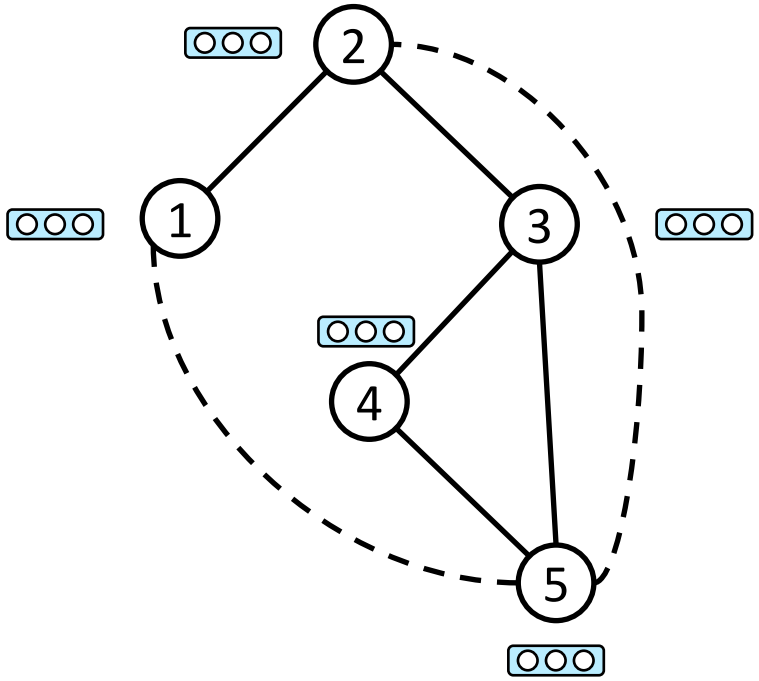


- Sequential Order Bias

  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

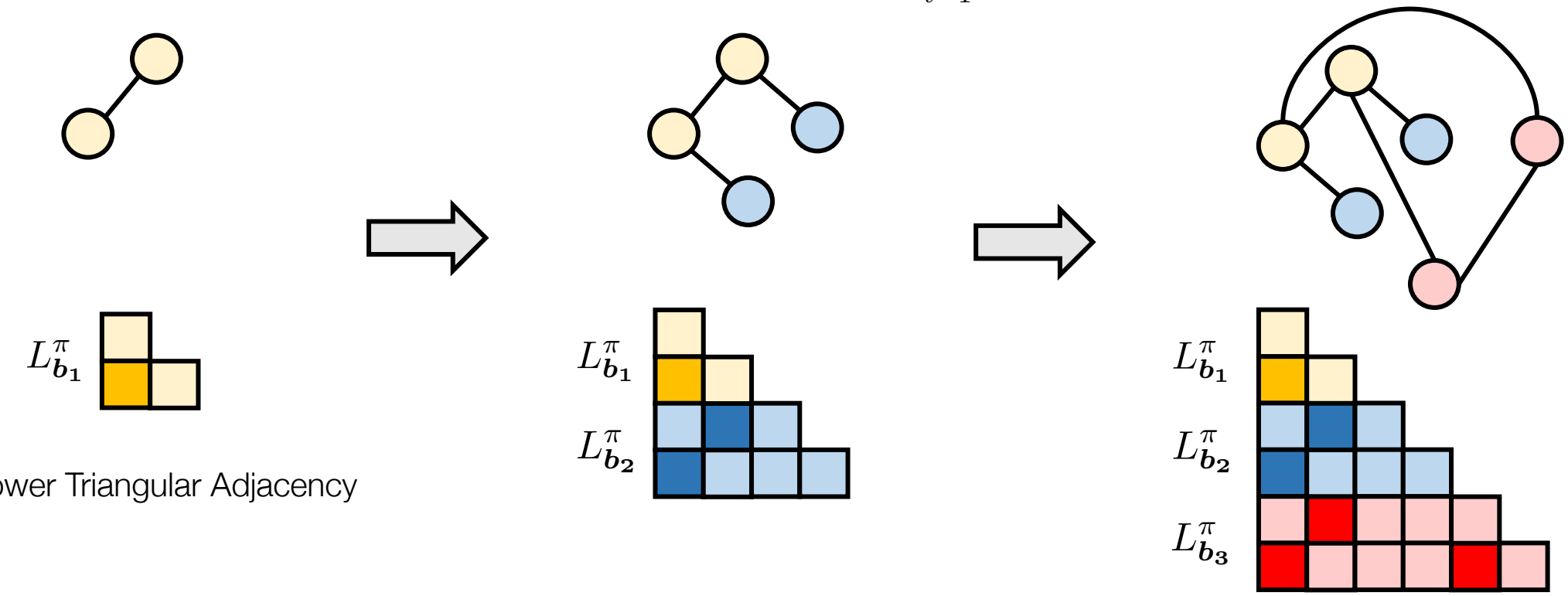
- Training Inefficiency

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

Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]

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

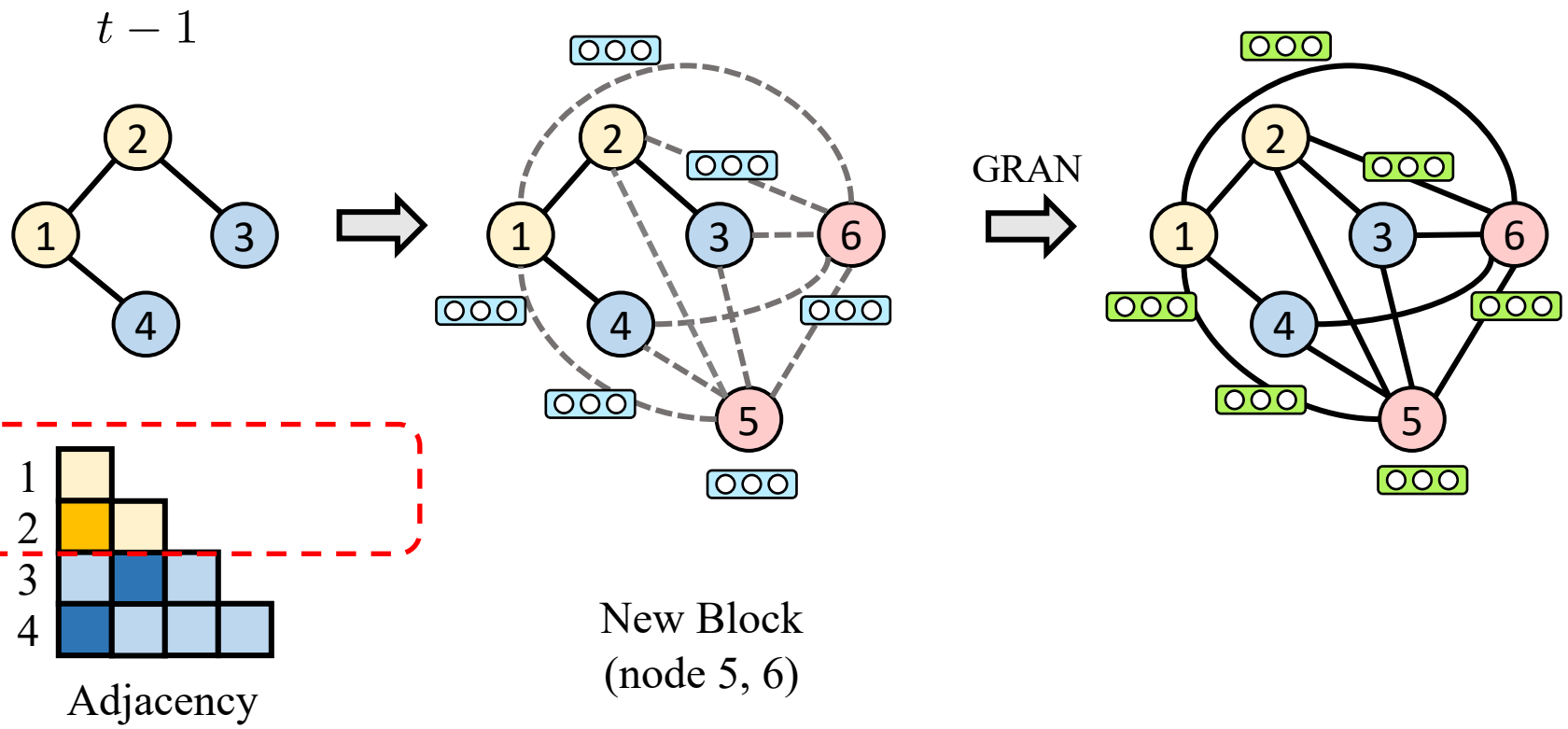


Lower Triangular Adjacency

- Sequential Order Bias — Graph Neural Network + Attention!
- Training Inefficiency — Block Generation + Parallel Training!

Autoregressive Models for Graphs

GNN-based Autoregressive Model, GRAN [6]



$$m_{ij}^t = f(h_i^t - h_j^t)$$
$$\tilde{h}_i^t = [h_i^t, x_i]$$
$$a_{ij}^t = \text{Sigmoid}\left(g(\tilde{h}_i^t - \tilde{h}_j^t)\right)$$
$$h_i^{t+1} = \text{GRU}(h_i^t, \sum_{j \in \mathcal{N}(i)} a_{ij}^t m_{ij}^t)$$

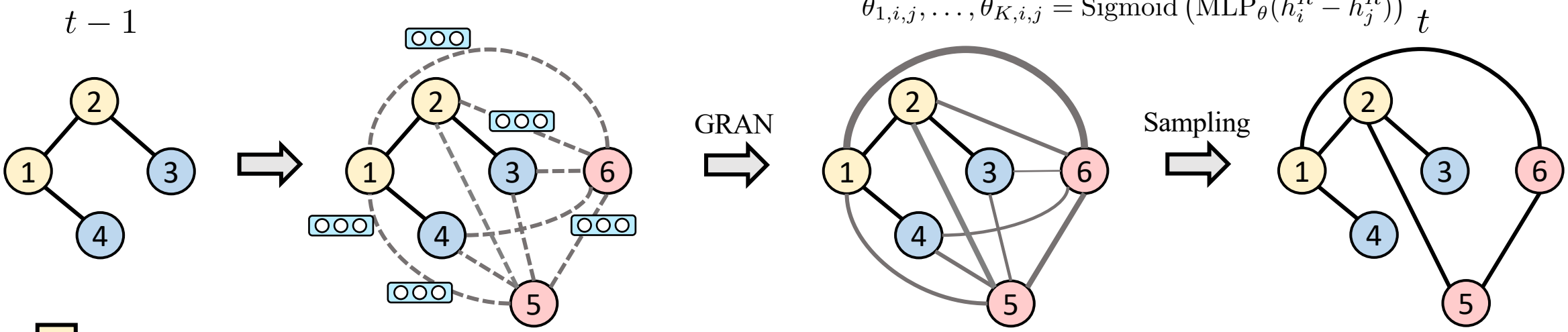
Autoregressive Models for Graphs

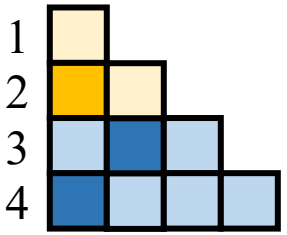
GNN-based Autoregressive Model, GRAN [6]

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

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

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

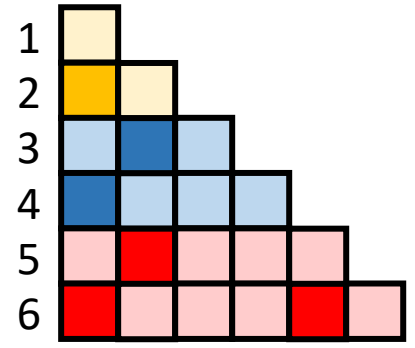




Adjacency

New Block
(node 5, 6)

Distribution over ***dashed edges***



Parallel Subgraph Training!

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GRAN [6]

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

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

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

Parallel Subgraph Learning via Back Propagation. No BPTT !

Autoregressive Models for Graphs

RNN-based Autoregressive Model, GRAN [6]

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

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

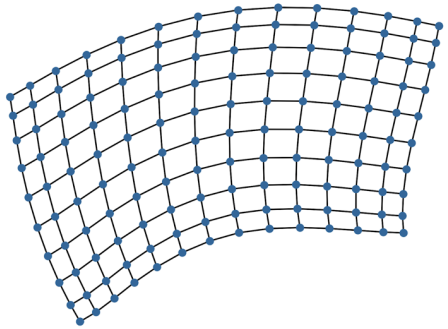
$$\max_{\theta} \quad \log \left(\sum_{\pi \sim q(\pi|G)} \left[\frac{p_\theta(G, \pi)}{q(\pi|G)} \right] \right)$$

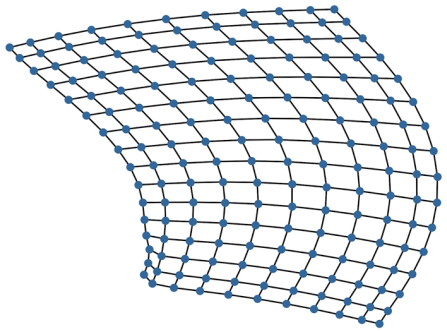
- BFS ordering starting from the largest degree node
- DFS ordering starting from the largest degree node
- K-core descending ordering
- Node degree ascending ordering
- Node degree descending ordering

Parallel Subgraph Learning via Back Propagation. No BPTT !

Random Grid Graphs

Train



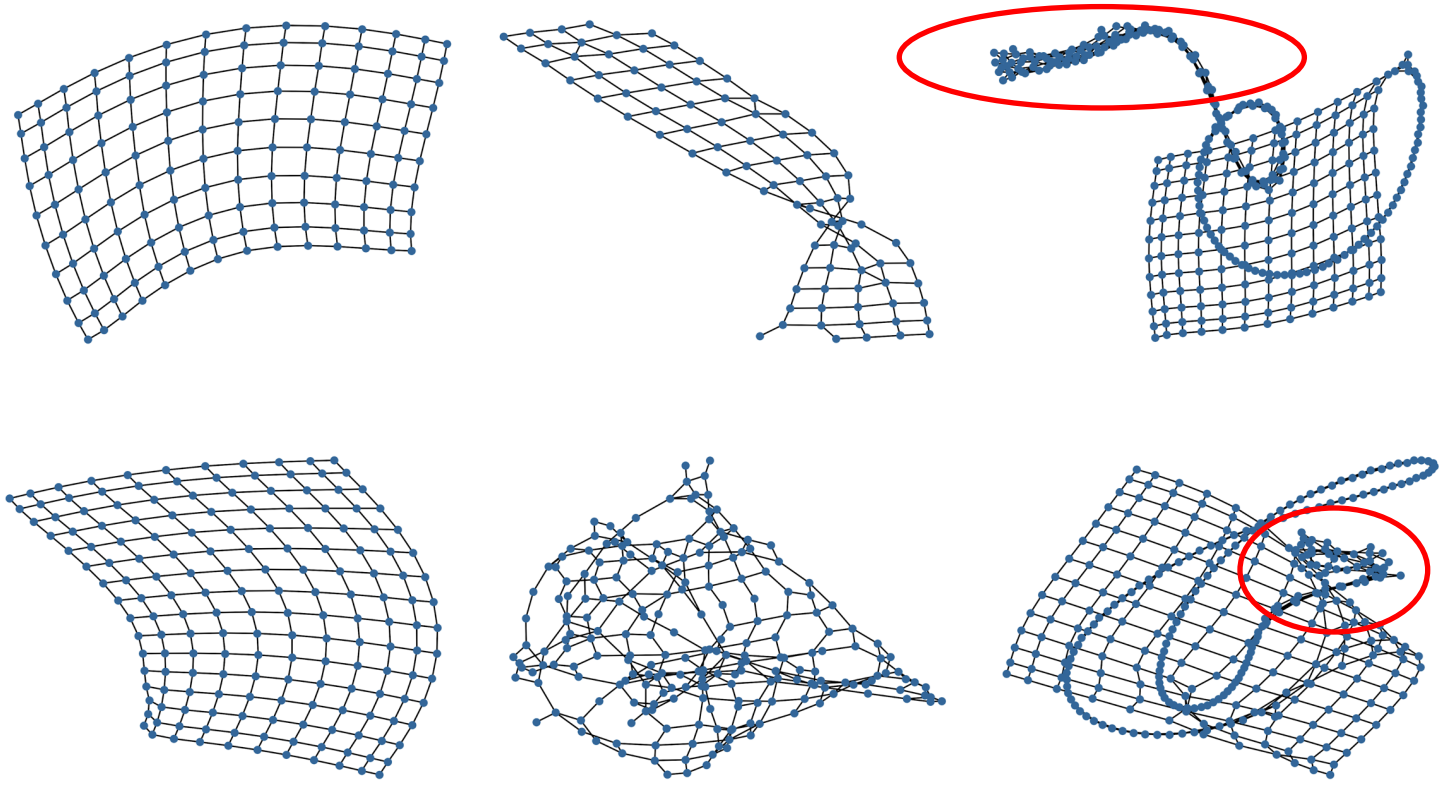


Random Grid Graphs

Train

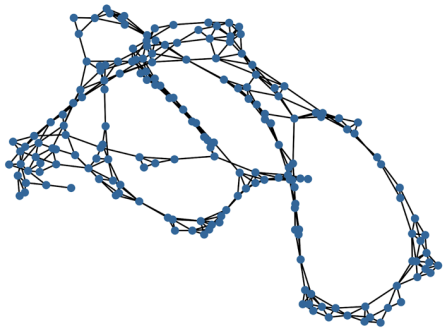
Graph VAE

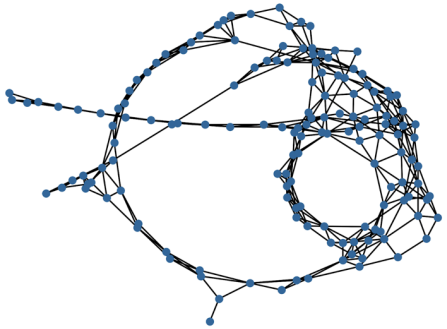
Graph RNN



Protein Graphs

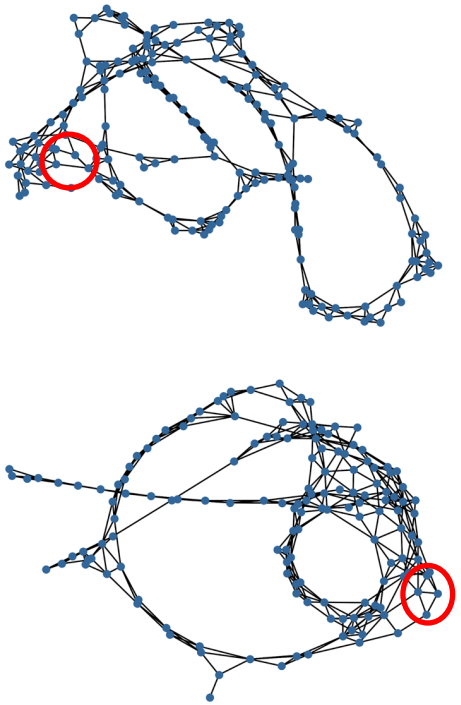
Train



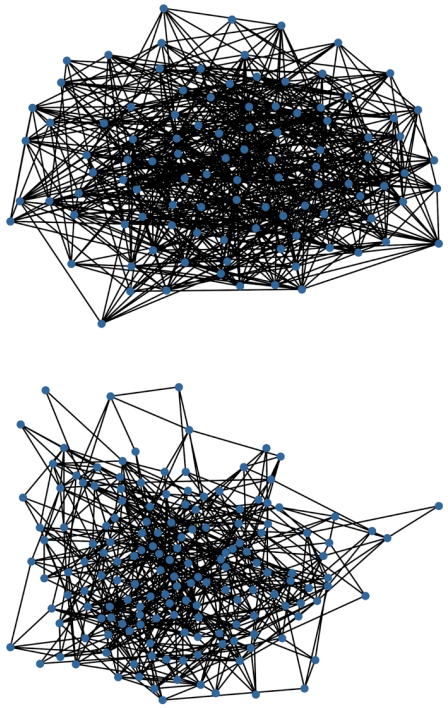


Protein Graphs

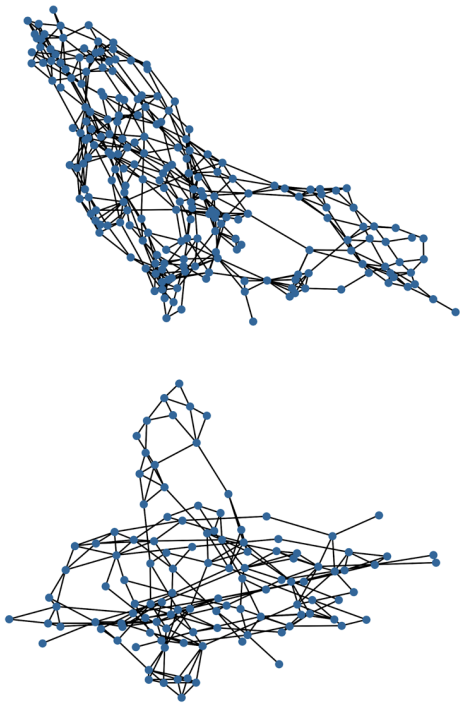
Train



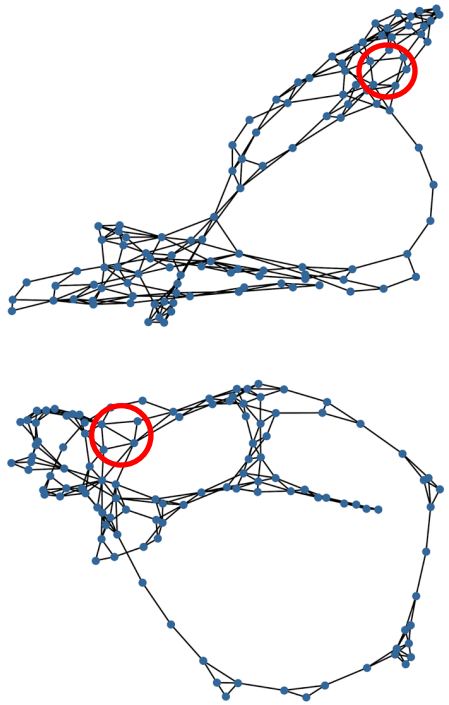
Graph VAE



Graph RNN

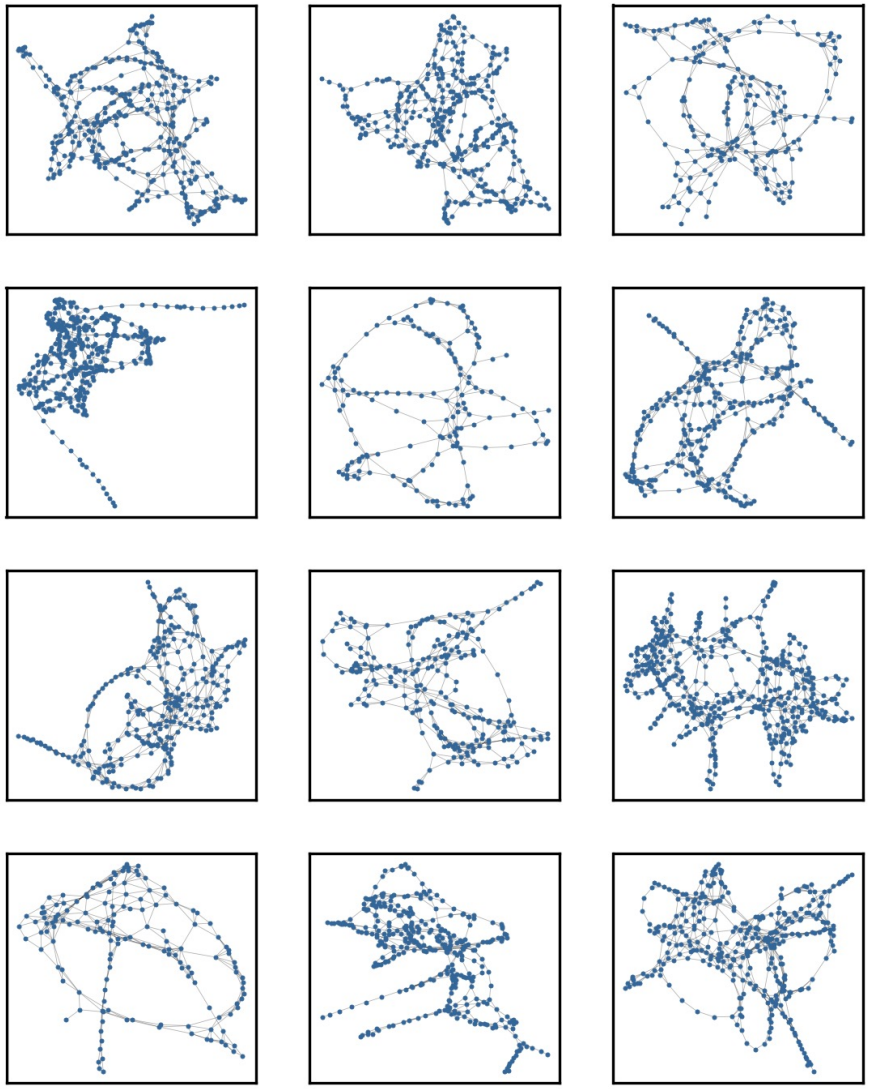


Ours

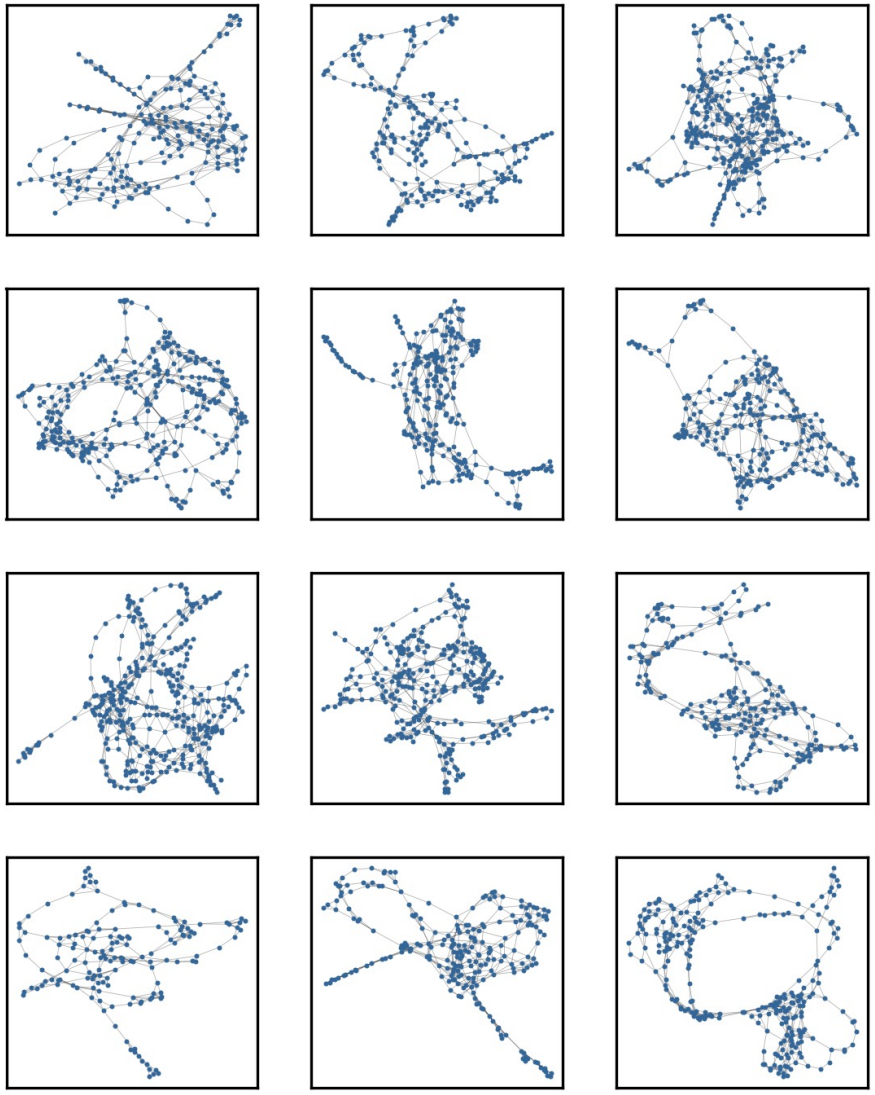


Protein Graphs

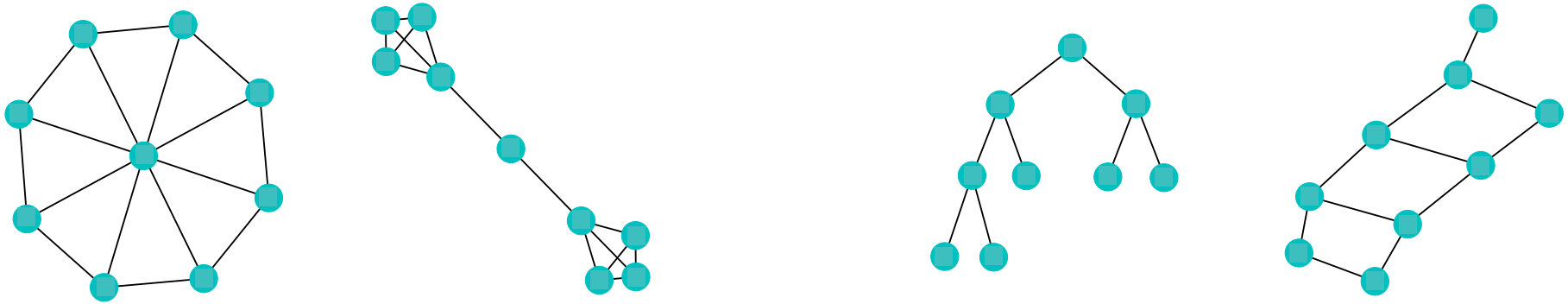
Train



Ours



Quantitative Evaluation



Observation

Generation

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	**0.0479**	0.230	0.210
GraphRNN	0.0106	0.140	0.880	0.0188
Ours	**0.00198**	0.0486	**0.130**	**0.00513**

*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 auto-regressive 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?