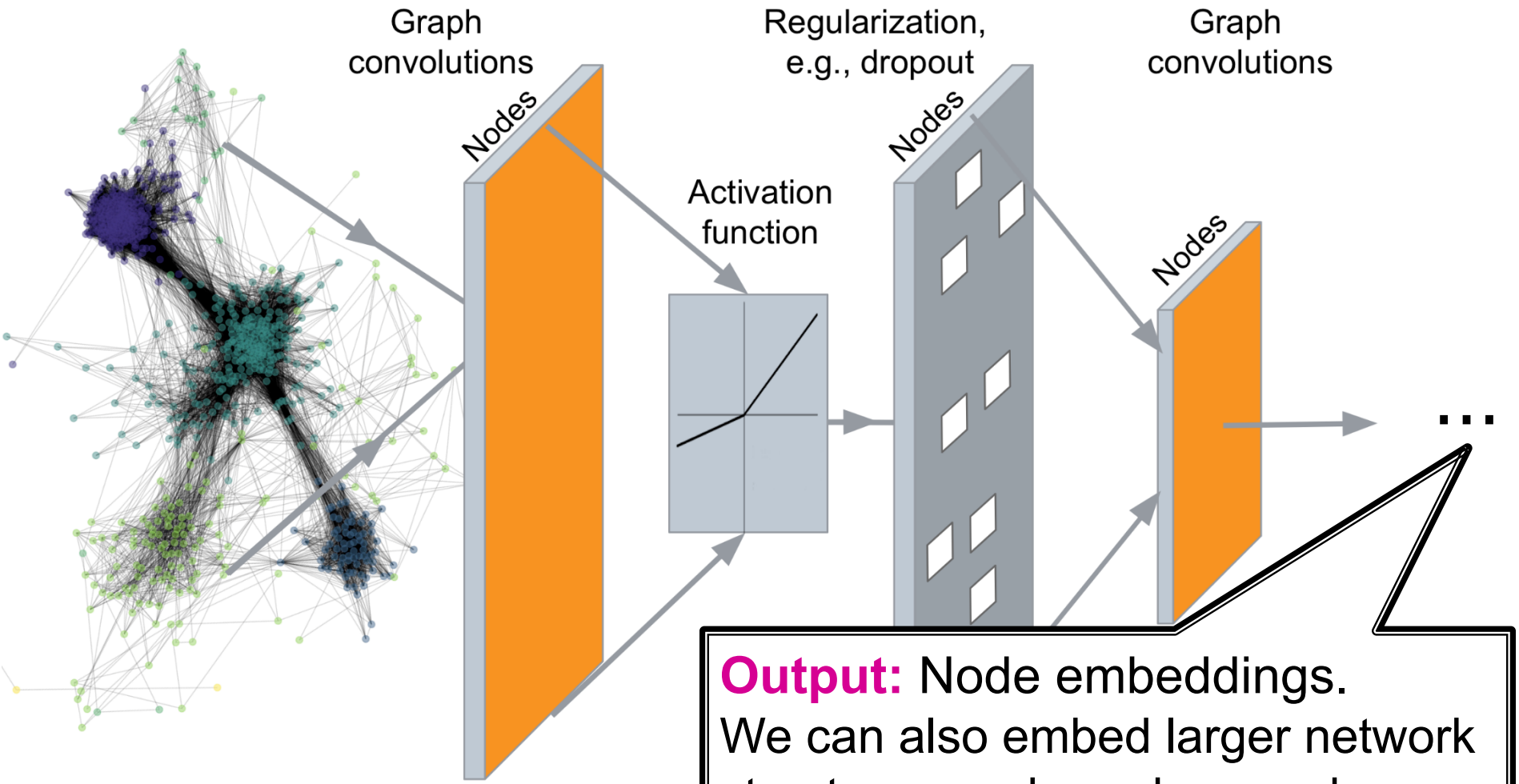
HW2 deadline postponed to next Thu, Oct 31!

We are releasing an improved version of the starter code for HW2.Q4 -- keep an eye on Piazza!

Deep Generative Models for Graphs

CS224W: Machine Learning with Graphs
Jure Leskovec, Jiaxuan You, Stanford University
http://cs224w.stanford.edu

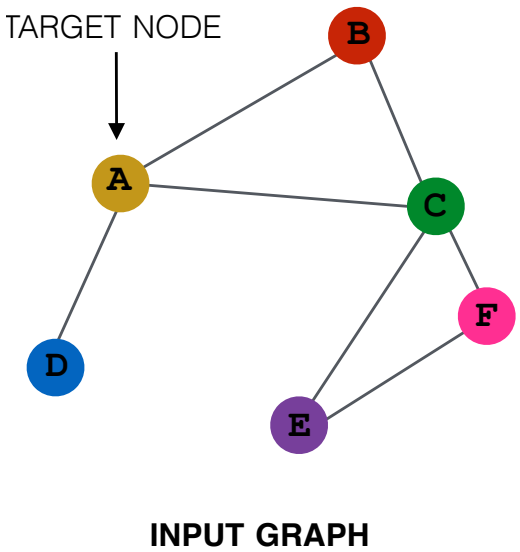
Deep Graph Encoders



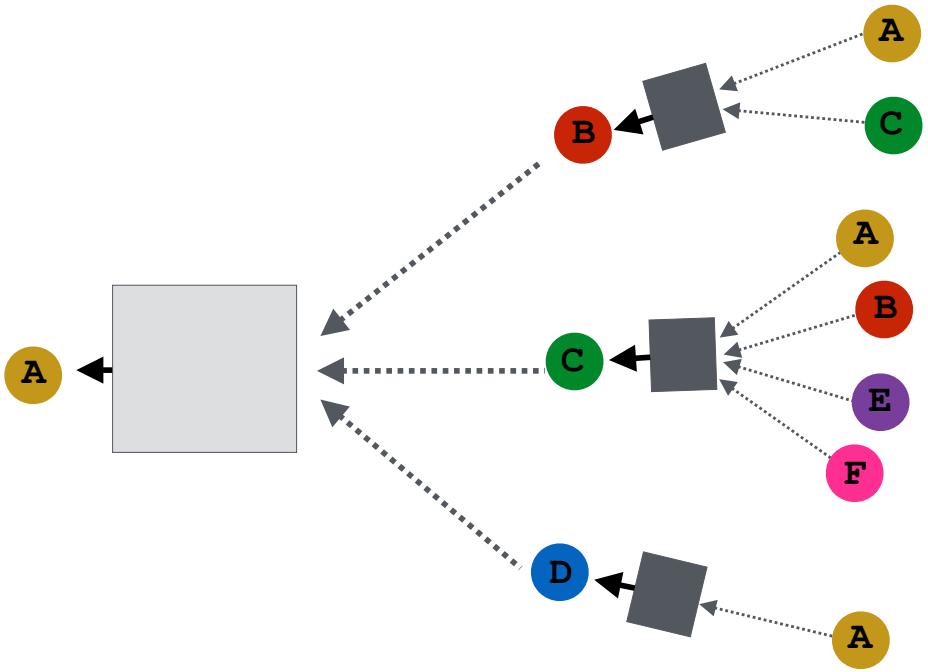
Output: Node embeddings.
We can also embed larger network structures, subgraphs, graphs

Idea: Aggregate Neighbors

- **Key idea:** Generate node embeddings based on **local network neighborhoods**

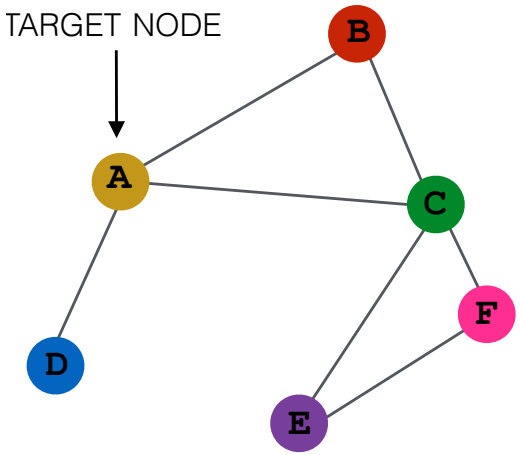


INPUT GRAPH

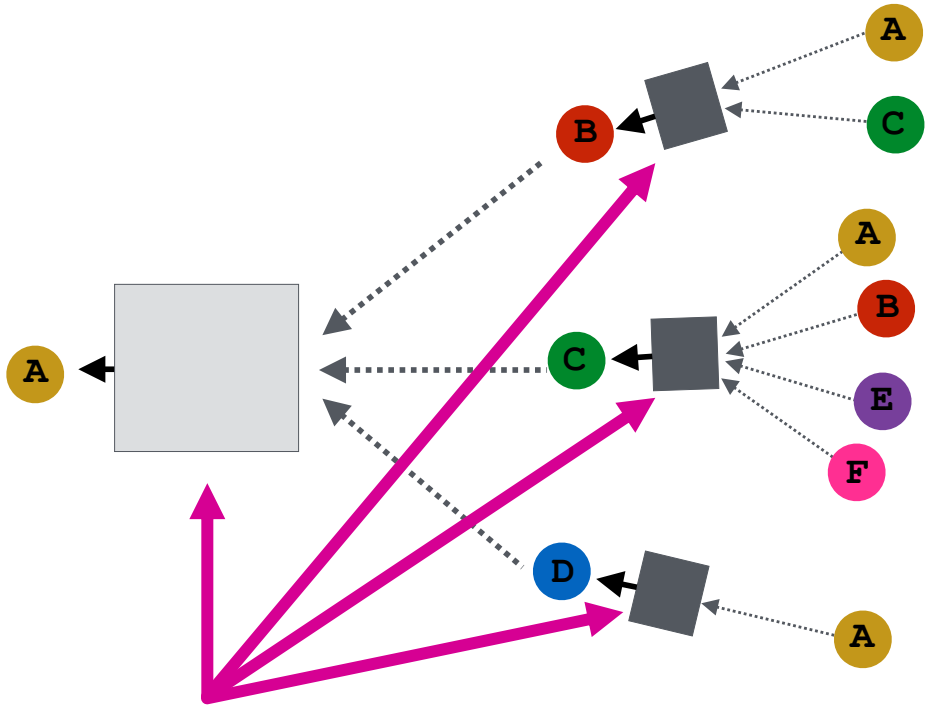


Idea: Aggregate Neighbors

- **Intuition:** Nodes aggregate information from their neighbors using neural networks



INPUT GRAPH

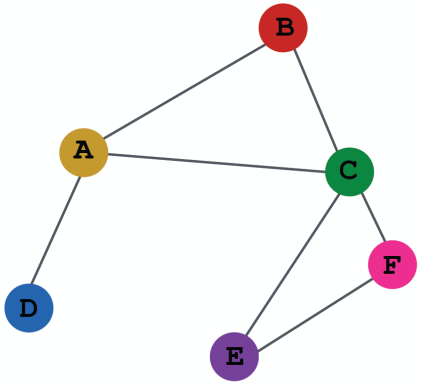


Neural networks

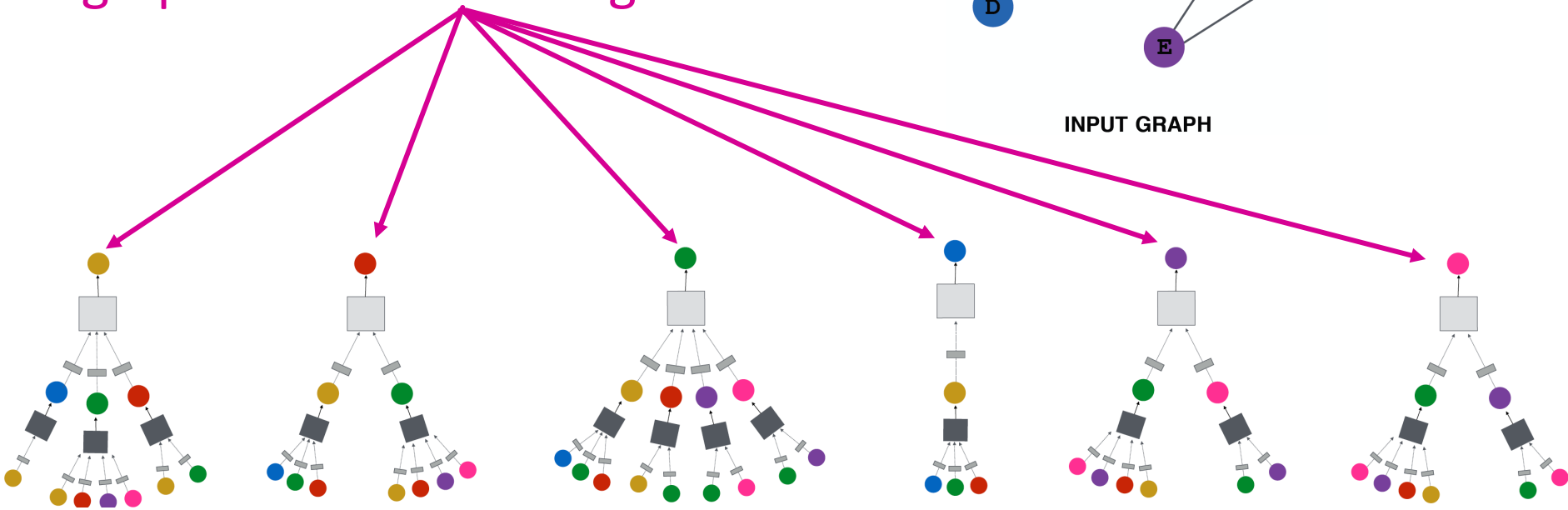
Idea: Aggregate Neighbors

- **Intuition:** Network neighborhood defines a computation graph

Every node defines a computation graph based on its neighborhood!



INPUT GRAPH

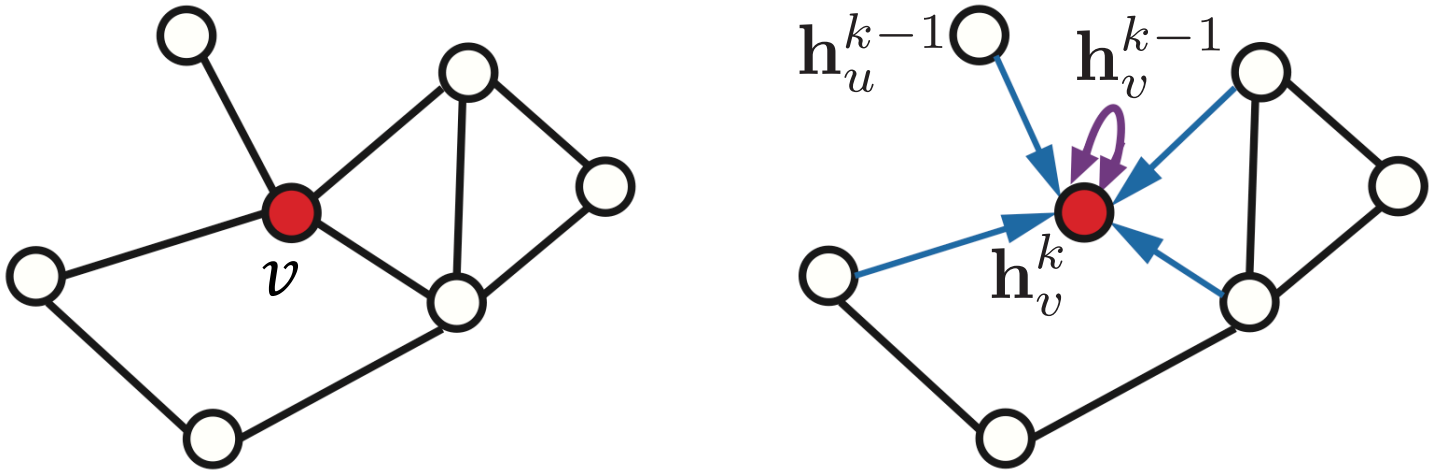


Recap: GCN, GraphSAGE

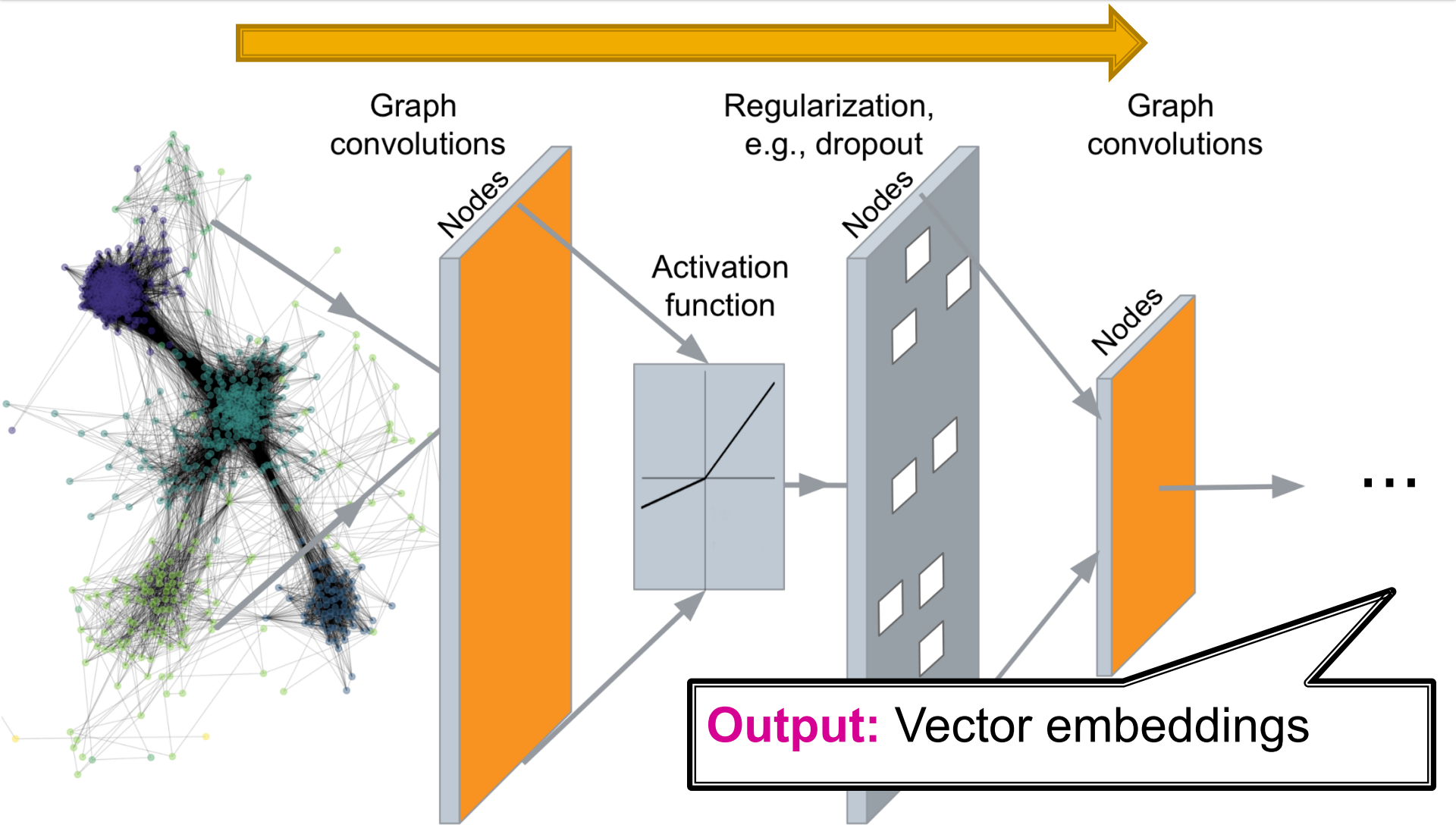
Key idea: Generate node embeddings based on **local network neighborhoods**

- Nodes aggregate "messages" from their neighbors using neural networks

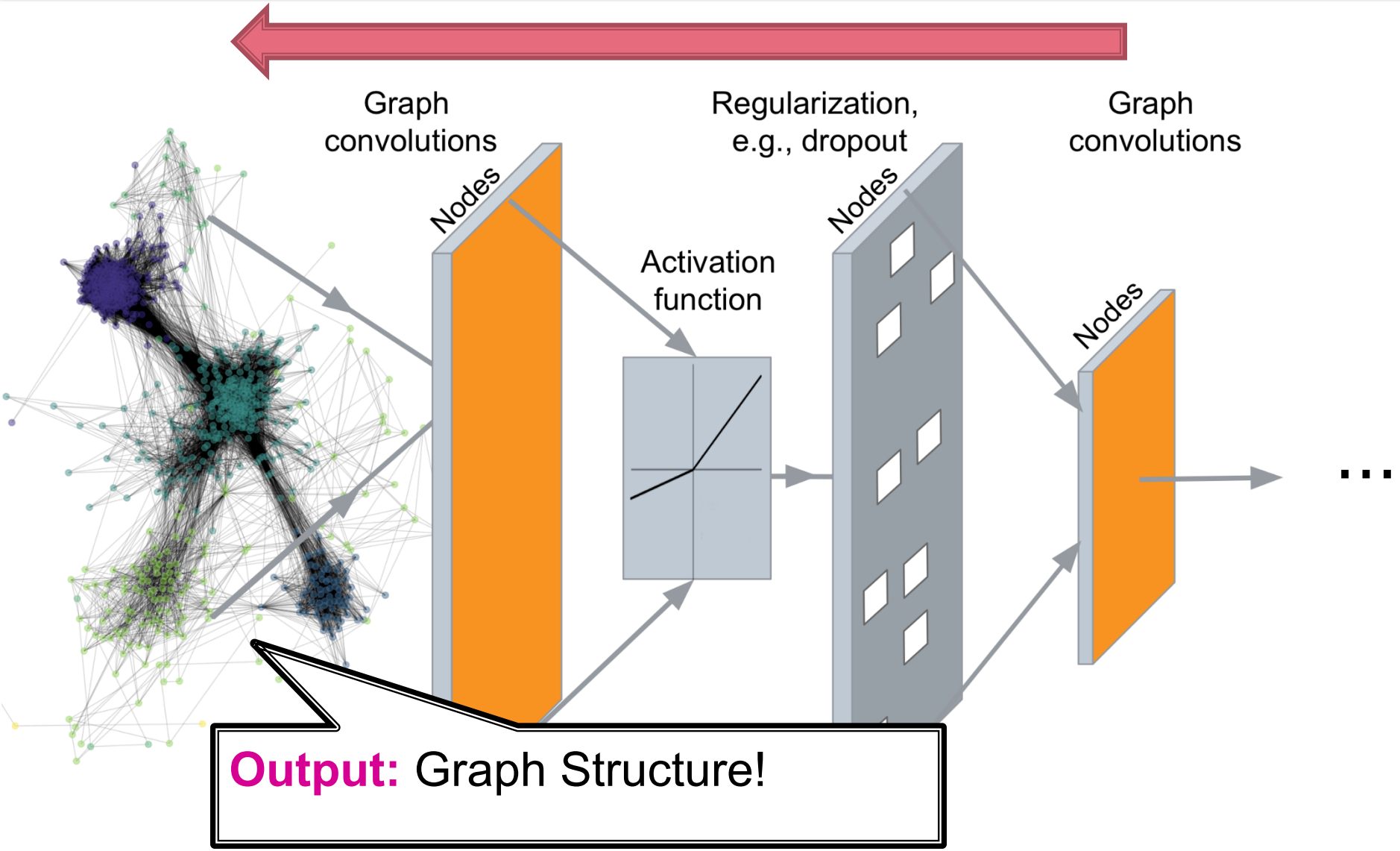
- **Graph Convolutional Neural Networks:**
  - **Basic variant:** Average neighborhood information and stack neural networks
- **GraphSAGE:**
  - Generalized neighborhood aggregation



Deep Graph Encoders



Today: Deep Graph Decoders



Outline of Today's Lecture

1. **Problem of Graph Generation**

2. **ML Basics for Graph Generation**

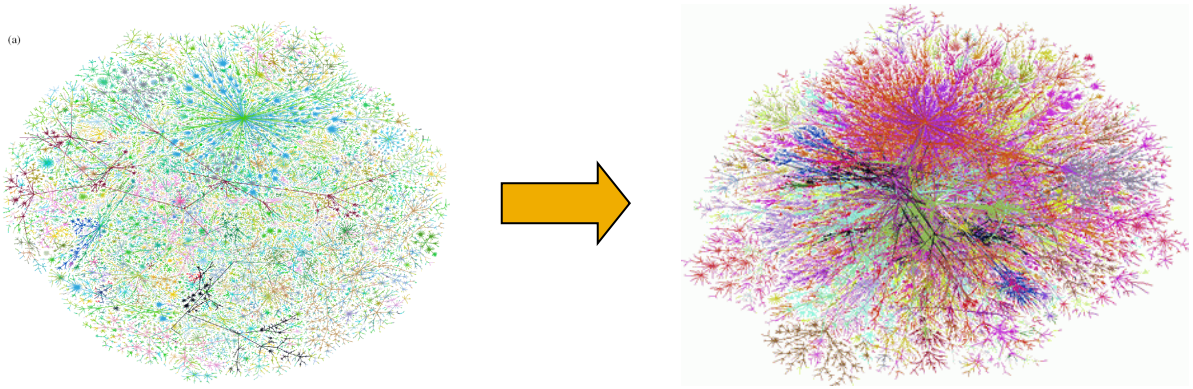
3. **GraphRNN**

4. **Applications and Open Questions**

The Problem: Graph Generation

The Problem: Graph Generation

- We want to generate realistic graphs



Given a large real graph

Generate a synthetic graph

- What is a good model?
- How can we fit the model and generate the graph using it?

Why is it Important?

- *Generation* – Gives insight into the graph formation process
- *Anomaly detection* – abnormal behavior, evolution
- *Predictions* – predicting future from the past
- *Simulations* of novel graph structures
- *Graph completion* – many graphs are partially observed
- *"What if"* scenarios

Graph Generation Tasks

Task 1: Realistic graph generation

- Generate graphs that are similar to a given set of graphs **[Focus of this lecture]**

Task 2: Goal-directed graph generation

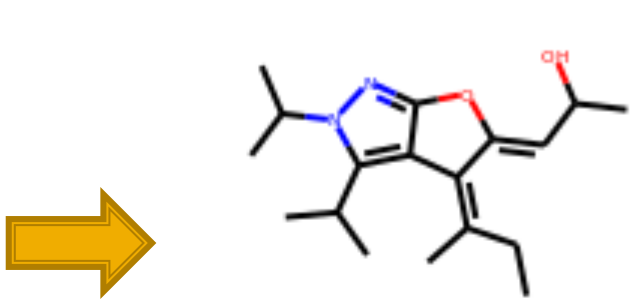
- Generate graphs that optimize given objectives/constraints
  - Drug molecule generation/optimization

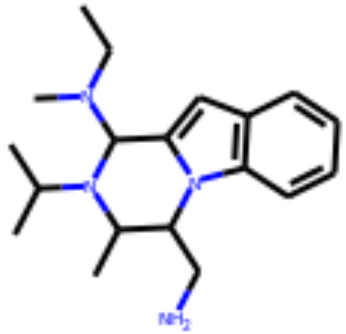
Why is it Interesting?

Drug discovery

- Discover highly drug-like molecules

Graph generative model





drug_likeness=0.94

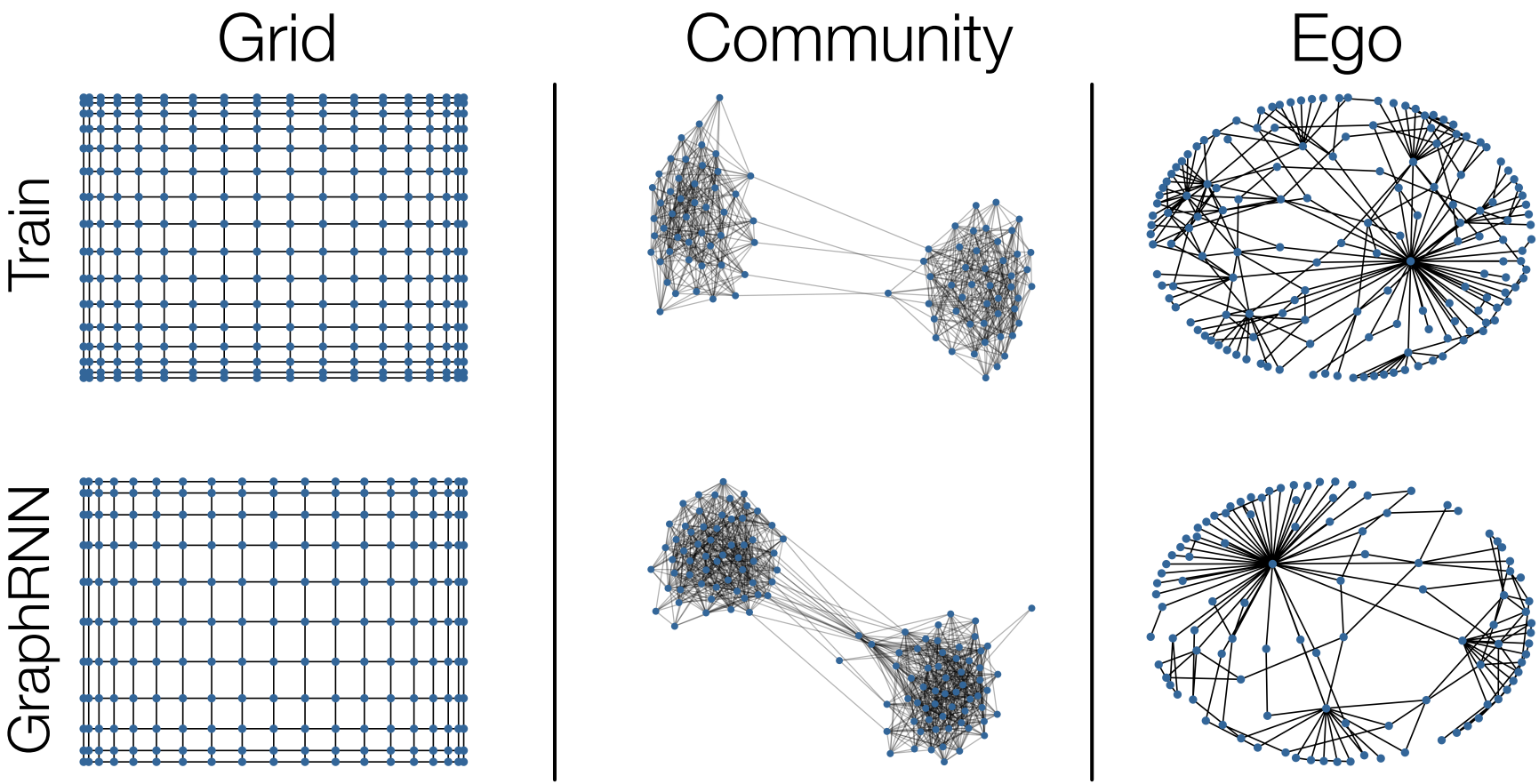
Why is it Interesting?

Drug discovery

- Complete an existing molecule to optimize a desired property

Why is it Interesting?

Discovering novel structures

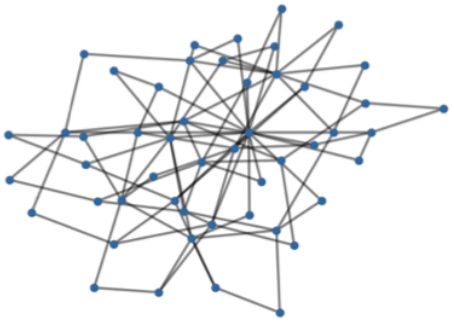


Why is it Interesting?

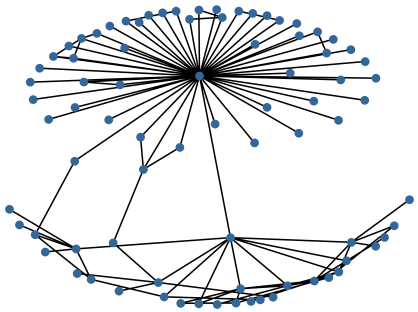
Network Science

- Null models for realistic networks

`Barabasi_Albert(n=50, m=2)` ~

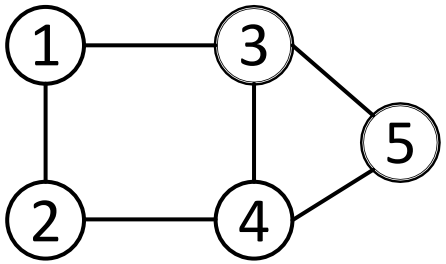


`NeuralNet_X(n=50, p=3, q=5)` ~



Why is it Hard?

- **Large and variable output space**
  - For n nodes we need to generate n^2 values
  - Graph size (nodes, edges) varies



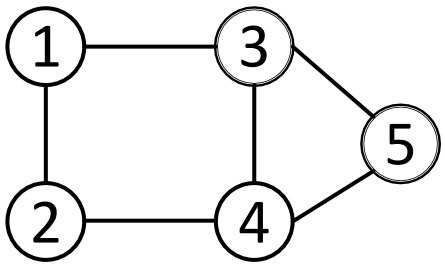
0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

5 nodes: 25 values

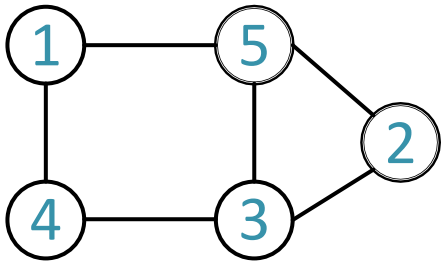
1K nodes: 1M values

Why is it Hard?

- **Non-unique representations:**
  - n-node graph can be represented in $n!$ ways
  - Hard to compute/optimize objective functions (e.g., reconstruction error)



0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0



0	0	0	1	1
0	0	1	0	1
0	1	0	1	1
1	0	1	0	0
1	1	1	0	0

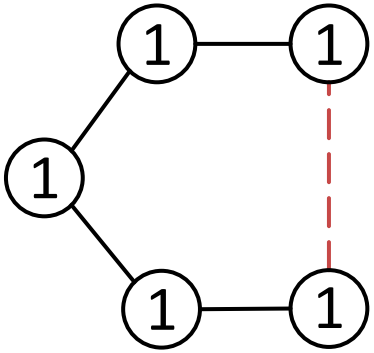
Same graph

Very different representations!

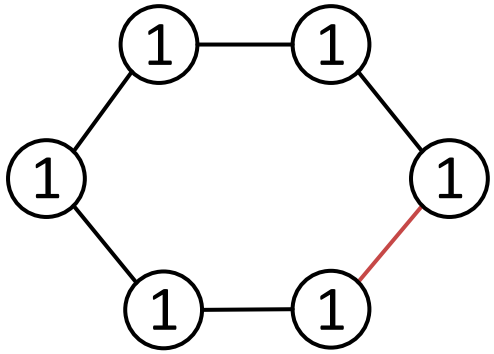
Why is it Hard?

- **Complex dependencies:**
  - Edge formation has long-range dependencies

Example: Generate a ring graph on 6 nodes:



Shouldn't have edge!



Should have edge!

Existence of an edge may depend on the entire graph!

Outline of Today's Lecture

1. **Problem of Graph Generation**
2. **ML Basics for Graph Generation**

3. **GraphRNN**
4. **Applications and Open Questions**

Machine Learning for Graph Generation

Graph Generative Models

- **Given:** Graphs sampled from $p_{data}(G)$
- **Goal:**
  - Learn the distribution $p_{model}(G)$
  - Sample from $p_{model}(G)$

Recap: Generative Models

Setup:

- Assume we want to learn a generative model from a set of data points (i.e., graphs) $\{\boldsymbol{x}_i\}$
  - $p_{data}(\boldsymbol{x})$ is the data distribution, which is never known to us, but we have sampled $\boldsymbol{x}_i \sim p_{data}(\boldsymbol{x})$
  - $p_{model}(\boldsymbol{x}; \theta)$ is the model, parametrized by θ, that we use to approximate $p_{data}(\boldsymbol{x})$
- **Goal:**
  - (1) Make $p_{model}(\boldsymbol{x}; \theta)$ close to $p_{data}(\boldsymbol{x})$
  - (2) Make sure we can sample from $p_{model}(\boldsymbol{x}; \theta)$
    - We need to generate examples (graphs) from $p_{model}(\boldsymbol{x}; \theta)$

Recap: Generative Models

(1) Make $p_{model}(x; \theta)$ close to $p_{data}(x)$

- **Key Principle**: **Maximum Likelihood**
- Fundamental approach to modeling distributions

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

  - Find parameters θ^*, such that for observed data points $\boldsymbol{x}_i \sim p_{data}$, $\sum_i \log p_{model}(\boldsymbol{x}_i; \theta^*)$ has the highest value, among all possible choices of θ
    - That is, find the model that is most likely to have generated the observed data x

Recap: Generative Models

(2) Sample from $p_{model}(x; \theta)$

- **Goal**: Sample from a complex distribution
- The most common approach:
  - **(1)** Sample from a simple noise distribution

$$z_i \sim N(0,1)$$

  - **(2)** Transform the noise z_i via $f(\cdot)$

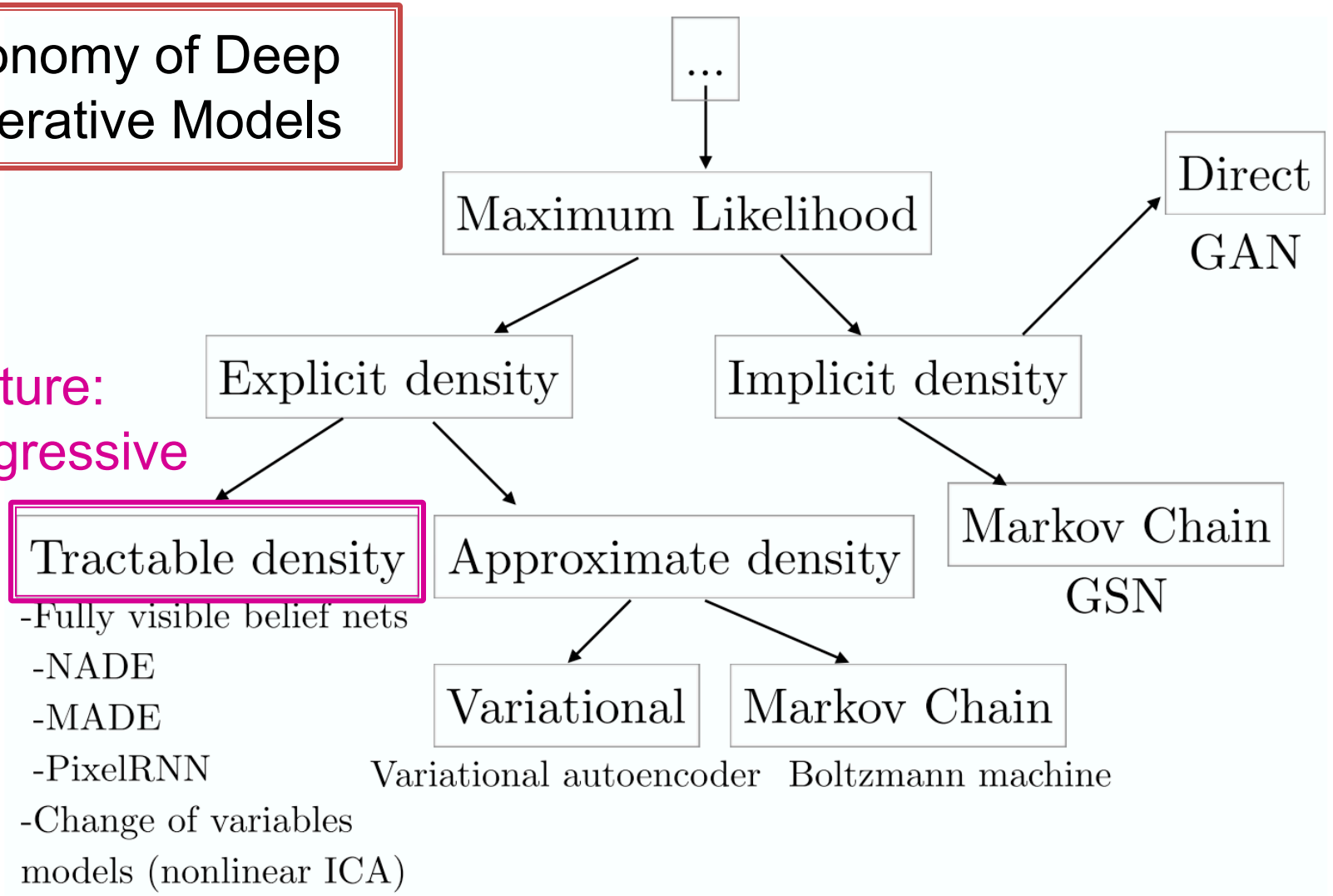
$$x_i = f(z_i; \theta)$$

  Then x_i follows a complex distribution
- **Q**: How to design $f(\cdot)$?
- **A**: Use Deep Neural Networks, and train it using the data we have!

[Goodfellow, NeurIPS 2016]

Recap: Generative Models

Taxonomy of Deep Generative Models



This lecture: Auto-regressive models:

An **autoregressive** (AR) model predicts future behavior based on past behavior.

Recap: Generative Models

Auto-regressive models

- $\boldsymbol{p_{model}(x;\theta)}$ is used for both **density estimation** and **sampling** (from the probability density)
  - (other models like Variational Auto Encoders (VAEs), Generative Adversarial Nets (GANs) have 2 or more models, each playing one of the roles)
  - **Apply chain rule**: Joint distribution is a product of conditional distributions:

$$p_{model}(\boldsymbol{x};\theta) = \prod_{t=1}^{n} p_{model}(x_t | x_1, \ldots, x_{t-1}; \theta)$$

  - E.g., $\boldsymbol{x}$ is a vector, x_t is the t-th dimension; $\boldsymbol{x}$ is a sentence, x_t is the t-th word.
  - **In our case:** x_t will be the t-th action (add node, add edge)

Outline of Today's Lecture

1. **Problem of Graph Generation**
2. **ML Basics for Graph Generation**
3. **GraphRNN** 
4. **Applications and Open Questions**

GraphRNN: Generating Realistic Graphs

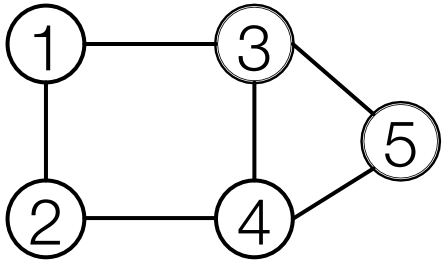
GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models. J. You, R. Ying, X. Ren, W. L. Hamilton, J. Leskovec. *International Conference on Machine Learning (ICML)*, 2018.

GraphRNN Idea

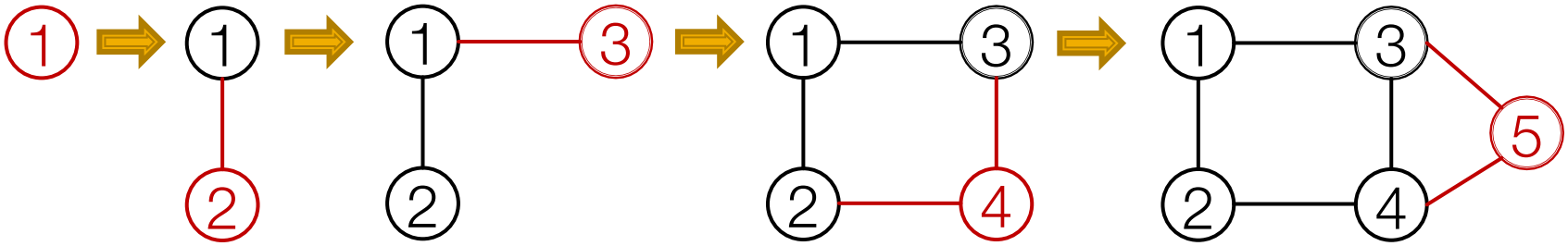
[You et al., ICML 2018]

Generating graphs via sequentially adding nodes and edges

Graph G



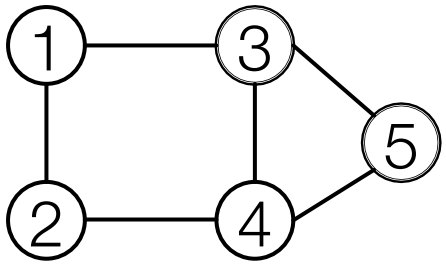
Generation process S^{π}



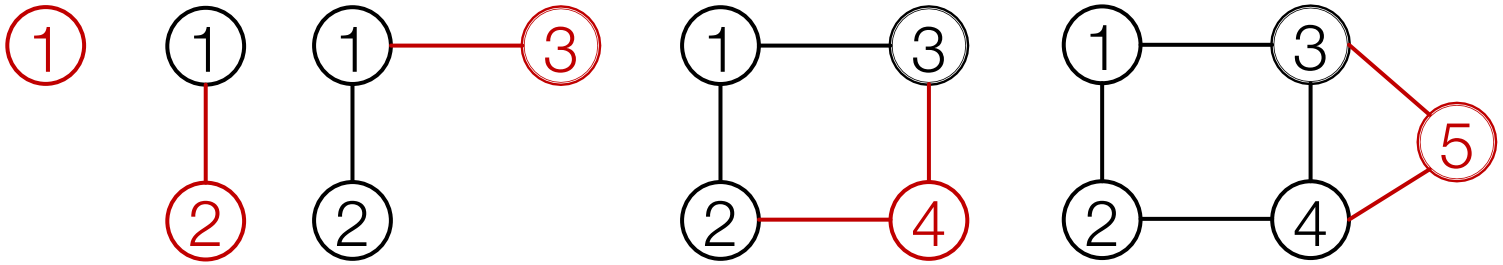
Model Graphs as Sequences

Graph $\mathbf{G}$ with node ordering $\boldsymbol{\pi}$ can be uniquely mapped into a sequence of node and edge additions $\boldsymbol{S^{\pi}}$

Graph G with node ordering π:



Sequence S^{π}:



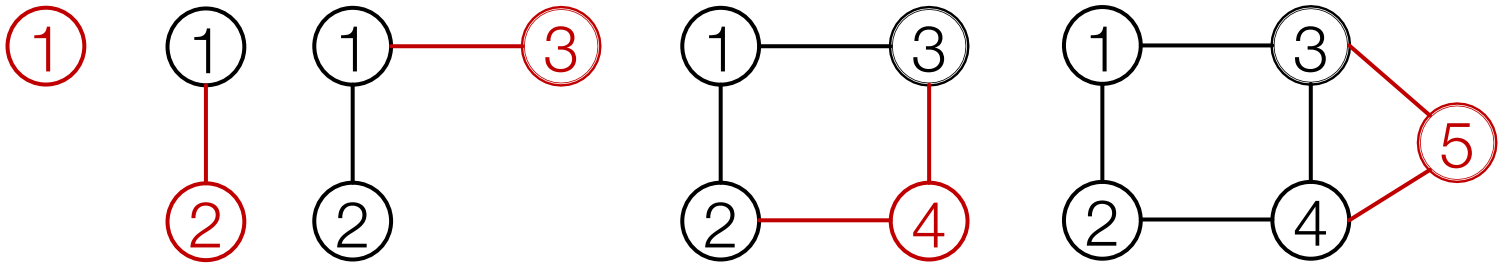
$$S^{\pi} = (S_1^{\pi} , S_2^{\pi} , S_3^{\pi} , S_4^{\pi} , S_5^{\pi})$$

Model Graphs as Sequences

The sequence S^π has **two levels** (S is a sequence of sequences):

- **Node-level:** add nodes, one at a time
- **Edge-level:** add edges between existing nodes

- **Node-level**: At each step, a new node is added



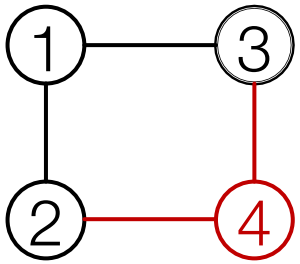
$$S^\pi = (S_1^\pi , S_2^\pi , S_3^\pi , S_4^\pi , S_5^\pi)$$

"Add node 1" . . . "Add node 5"

Model Graphs as Sequences

The sequence S^{π} has **two levels:**

- Each Node-level step is an edge-level sequence
- **Edge-level**: At each step, add a new edge



S_4^{π}

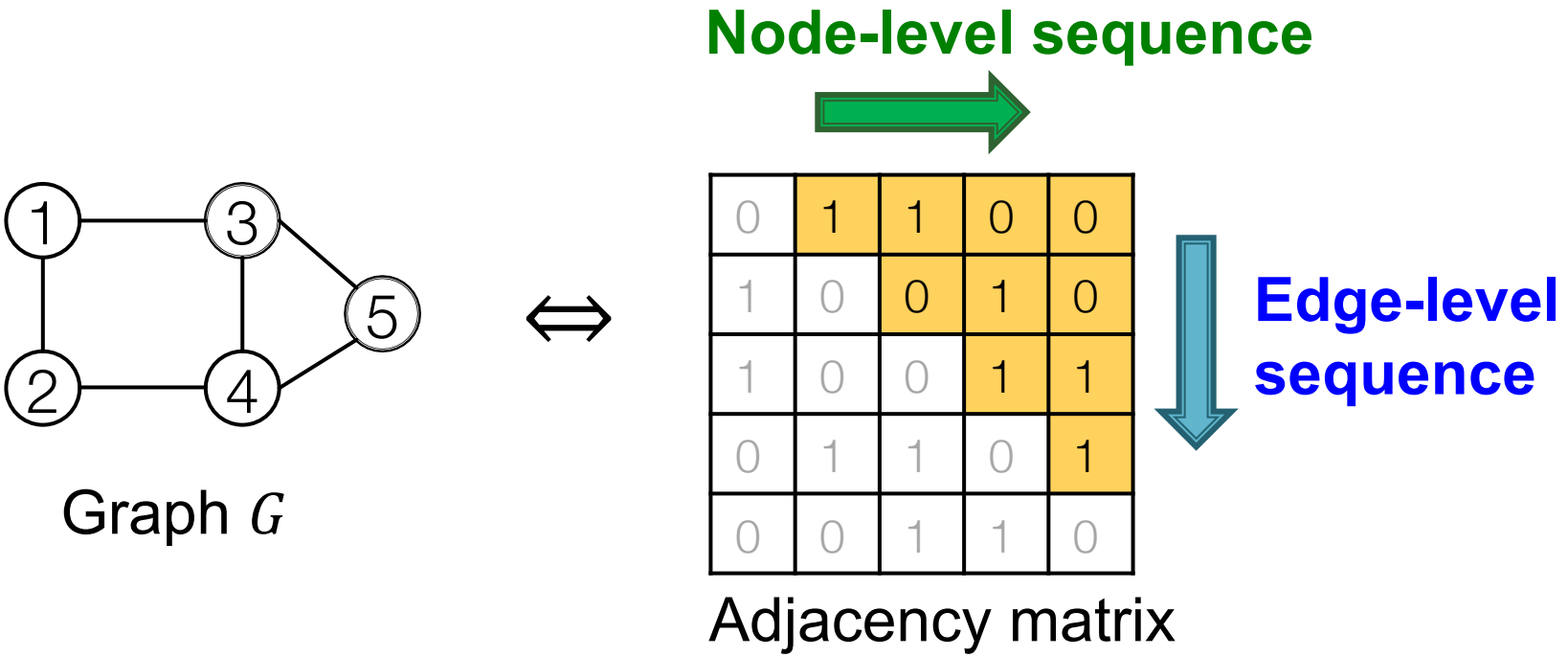
$$S_4^{\pi} = \left(S_{4,1}^{\pi}, S_{4,2}^{\pi}, S_{4,3}^{\pi} \right)$$

"Not connect 4, 1" "Connect 4, 2" "Connect 4, 3"

0 1 1

Model Graphs as Sequences

- **Summary: A graph + a node ordering = A sequence of sequences!**
- Node ordering is randomly selected (we will come back to this)



Model Graphs as Sequences

- **We have transformed graph generation problem into a sequence generation problem**
- **Need to model two processes:**
  - Generate a state for a new node (Node-level sequence)
  - Generate edges for the new node based on its state (Edge-level sequence)
- **Approach**: Use RNN to model these processes!

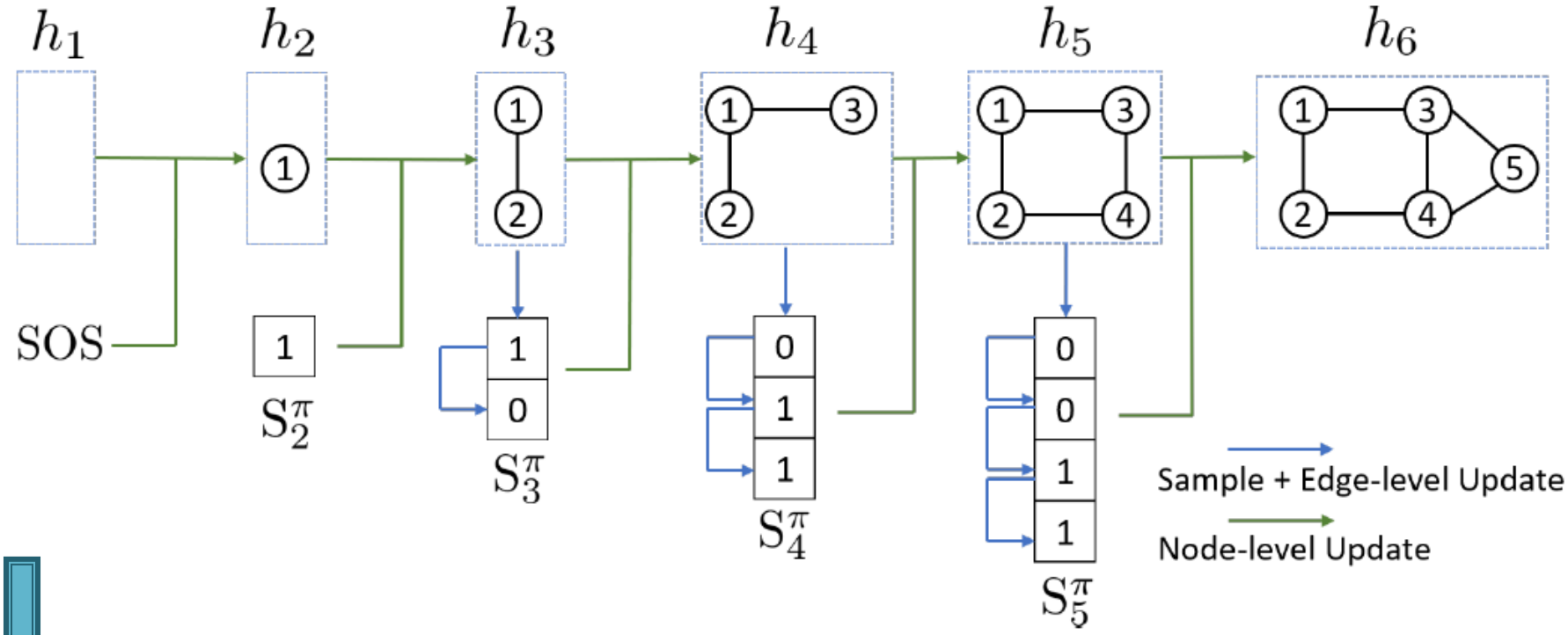
GraphRNN: Two levels of RNN

- **GraphRNN has a node-level RNN and an edge-level RNN**

- **Relationship between the two RNNs:**
  - Node-level RNN generates the initial state for edge-level RNN
  - Edge-level RNN generates edges for the new node, then updates node-level RNN state using generated results

GraphRNN: Two levels of RNN

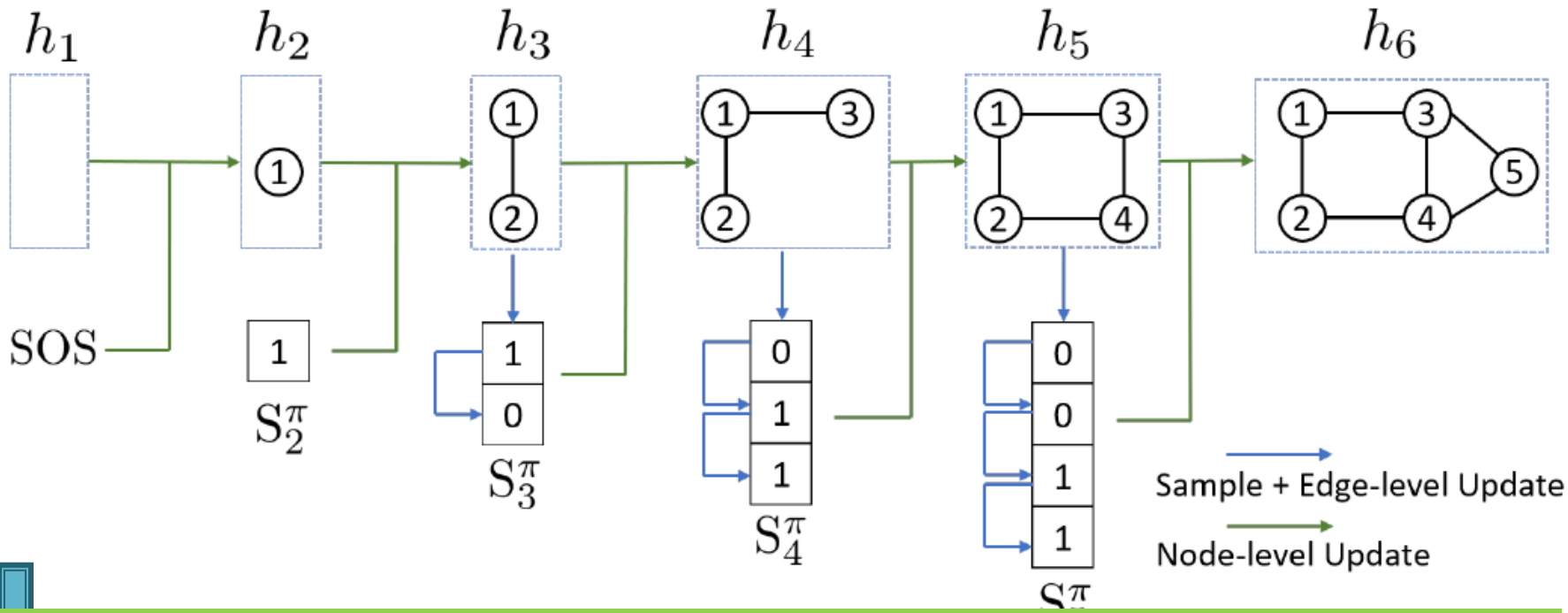
Node-level RNN generates the initial state for edge-level RNN



Edge-level RNN generates edges for the new node, then update node-level RNN state using generated results

GraphRNN: Two levels of RNN

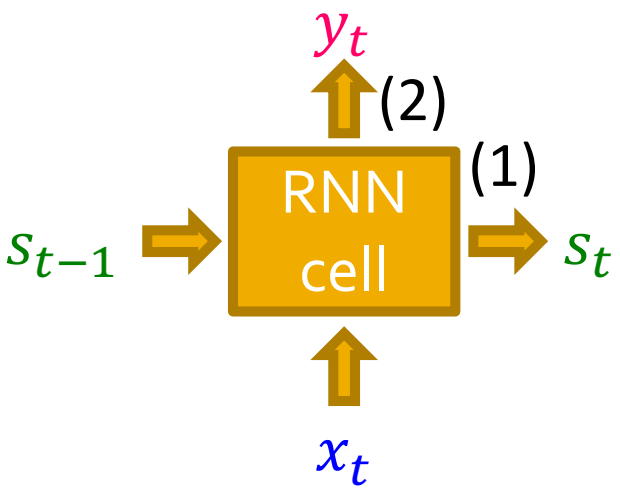
Node-level RNN generates the initial state for edge-level RNN



Next: How to generate a sequence with RNN?

Recurrent Neural Networks

- s_t: State of RNN after time t
- x_t: Input to RNN at time t
- y_t: Output of RNN at time t
- W, U, V: parameter matrices, $\sigma(\cdot)$: non-linearity



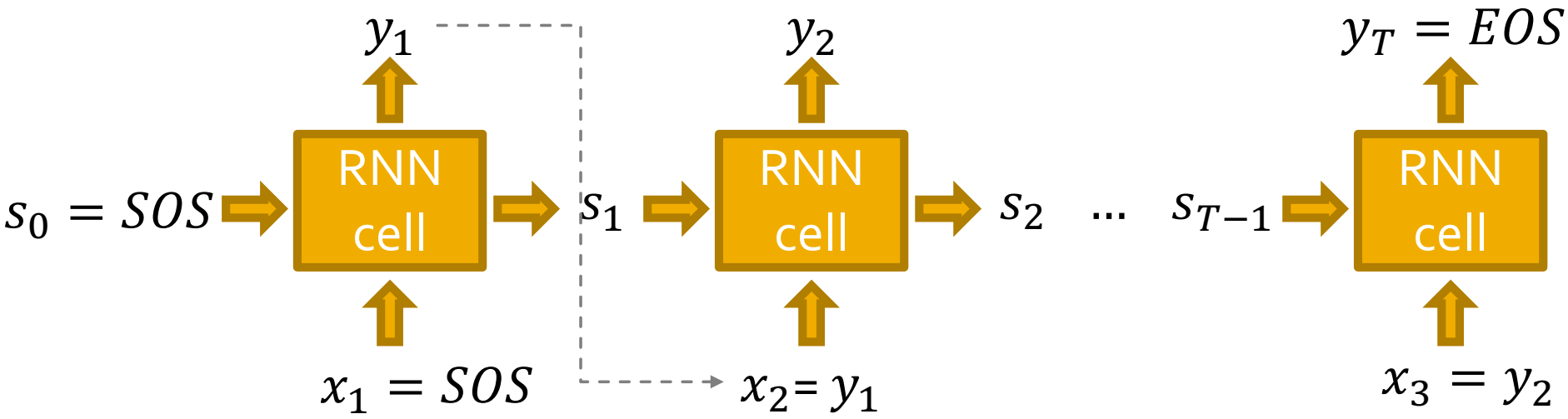
(1) $s_t = \sigma(W \cdot x_t + U \cdot s_{t-1})$

(2) $y_t = V \cdot s_t$

- **More expressive cells:** GRU, LSTM, etc.

RNN for Sequence Generation

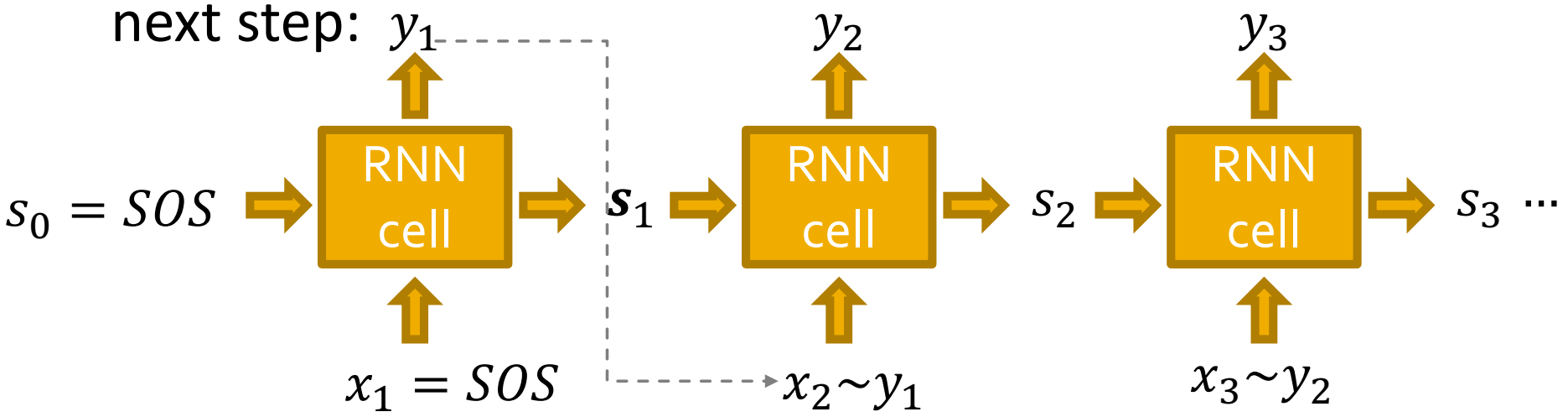
- **Q:** How to use RNN to generate sequences?
- **A:** Let $x_{t+1} = y_t$!
- **Q:** How to initialize s_0, x_1? When to stop generation?
- **A:** Use start/end of sequence token (SOS, EOS)- e.g., zero vector



- This is good, but this model is deterministic

RNN for Sequence Generation

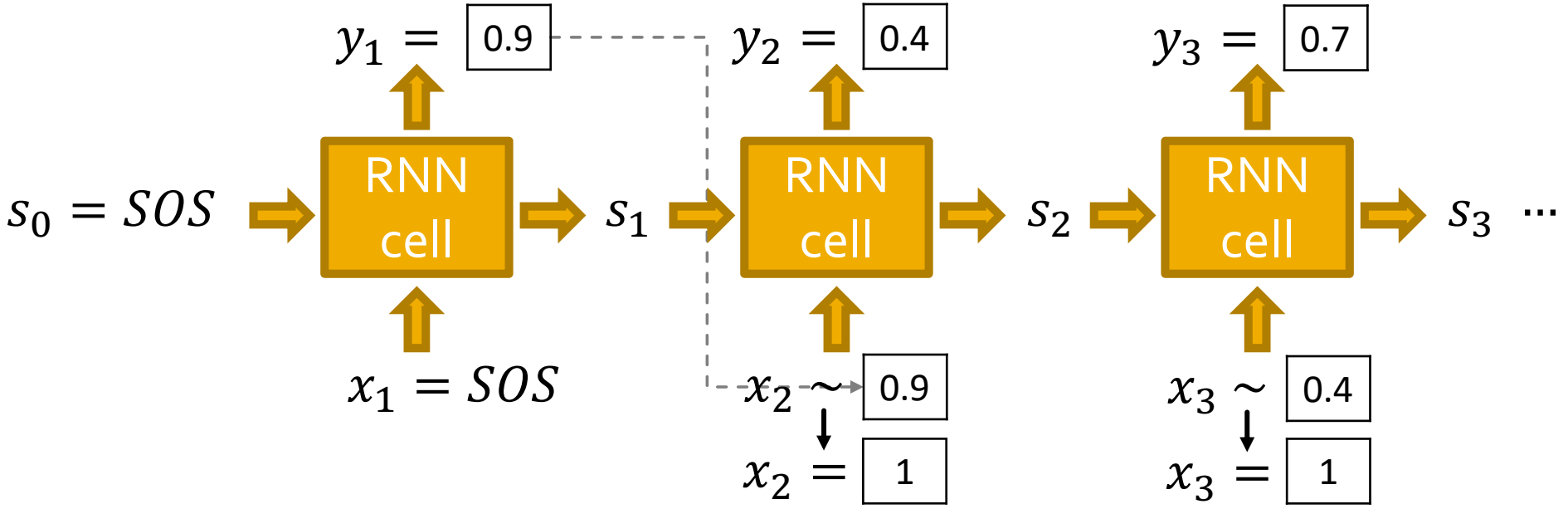
- **Remember our goal**: Use RNN to model $\prod_{k=1}^{n} p_{model}(x_t | x_1, \ldots, x_{t-1}; \theta)$
- Let $y_t = p_{model}(x_t | x_1, \ldots, x_{t-1}; \theta)$
- Then x_{t+1} is a sample from y_t: $x_{t+1} \sim y_t$
  - Each step of RNN outputs a probability vector
  - We then sample from the vector, and feed sample to next step:



RNN at Test Time

Suppose we already have trained the model

- y_t follows **Bernoulli distribution** (choice of p_{model})
- $\boxed{p}$ means value 1 has prob. p, value 0 has prob. $1-p$

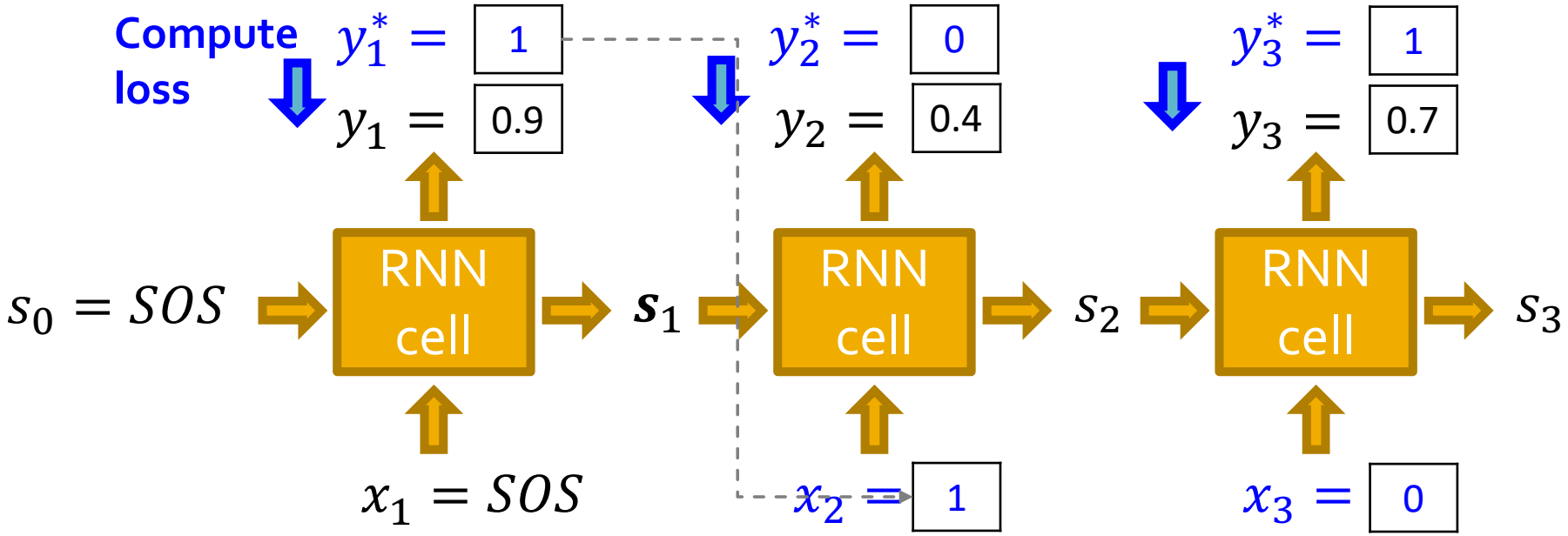


- Right now everything is generated by the model
- **How do we use training data $x_1, x_2, \ldots, x_n$?**

RNN at Training Time

Training the model

- We observe a sequence y^* of edges [1,0,…]
- **Principle**: Teacher Forcing -- Replace input and output by the real sequence



RNN at Training Time

- **Loss L : Binary cross entropy**
- **Minimize**:

$$L = -[y_1^* \log(y_1) + (1 - y_1^*)\log(1 - y_1)]$$

Compute loss

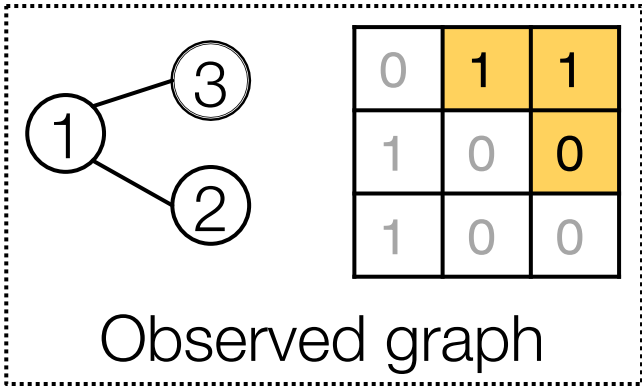
$y_1^* =$ 1

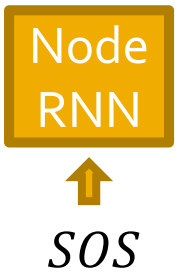
$y_1 =$ 0.9

- If $y_1^* = 1$, we minimize $-\log(y_1)$, making y_1 higher
- If $y_1^* = 0$, we minimize $-\log(1 - y_1)$, making y_1 lower
- This way, y_1 is **fitting** the data samples y_1^*
- **Reminder**: y_1 is computed by RNN, this loss will **adjust RNN parameters accordingly**, using back propagation!

Put Things Together: Training

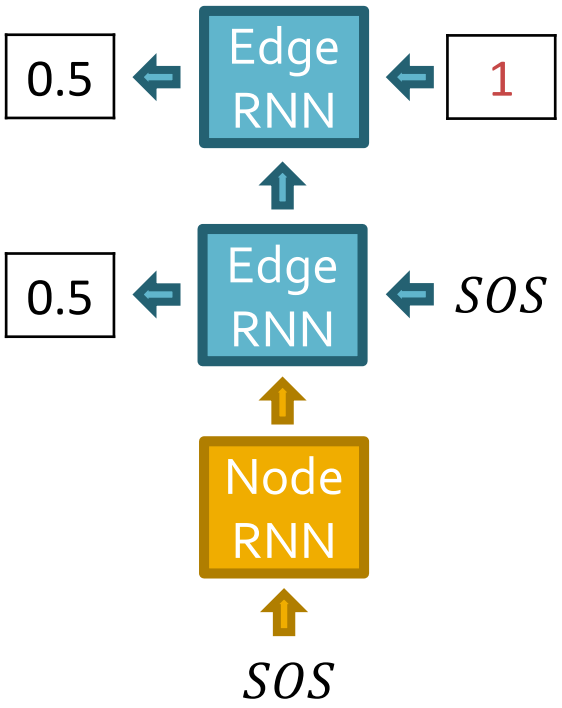
Assuming Node 1 is in the graph
Now adding Node 2

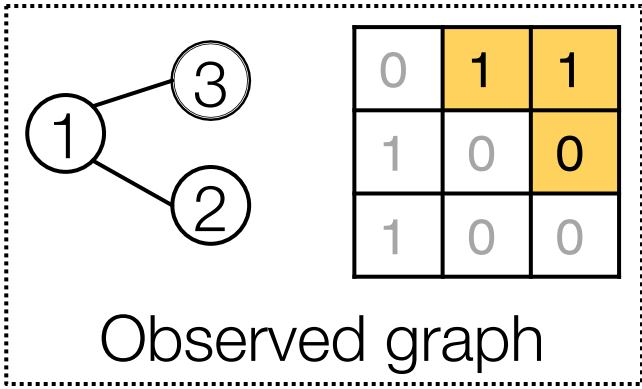




Put Things Together: Training

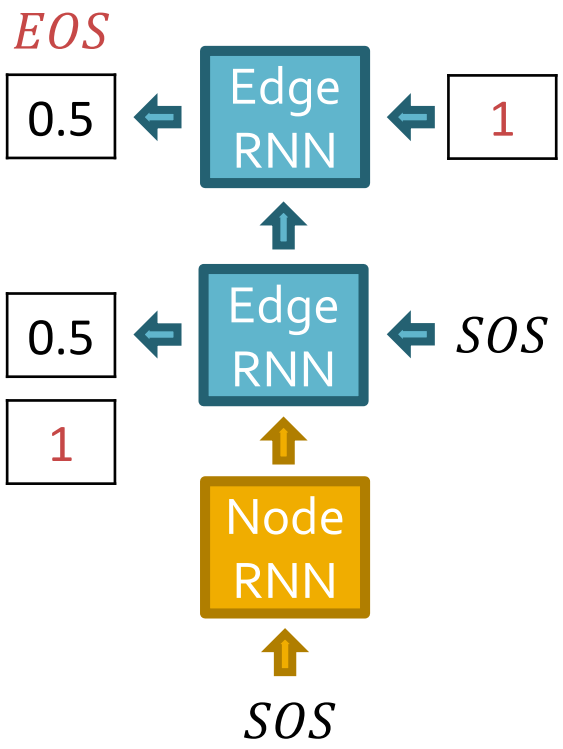
Edge RNN predicts how
Node 2 connects to Node 1

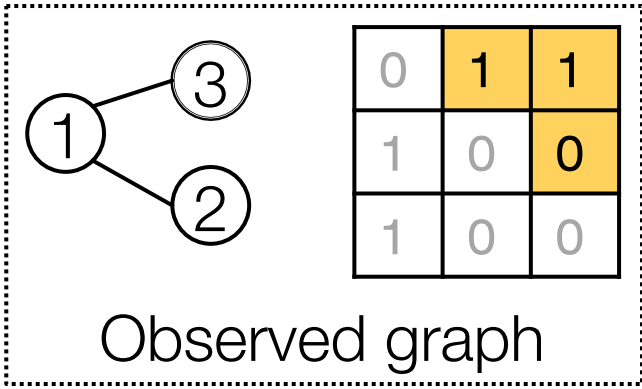




Put Things Together: Training

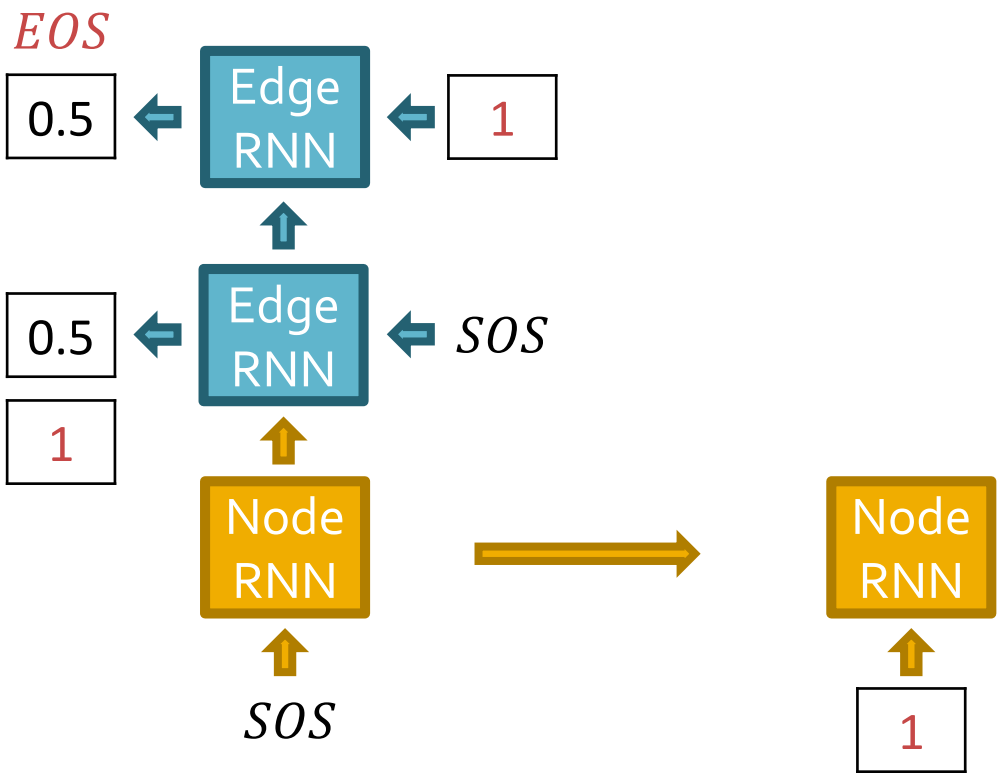
Edge RNN gets supervisions from ground truth

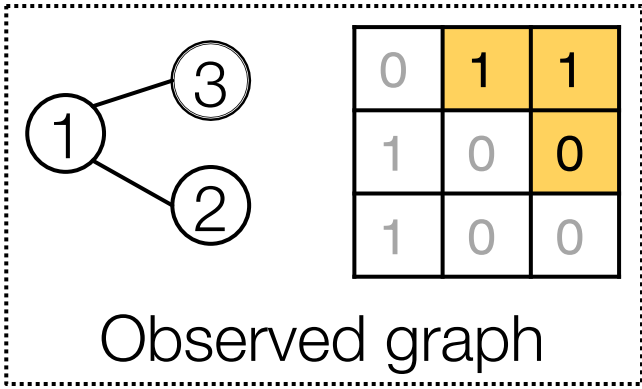




Put Things Together: Training

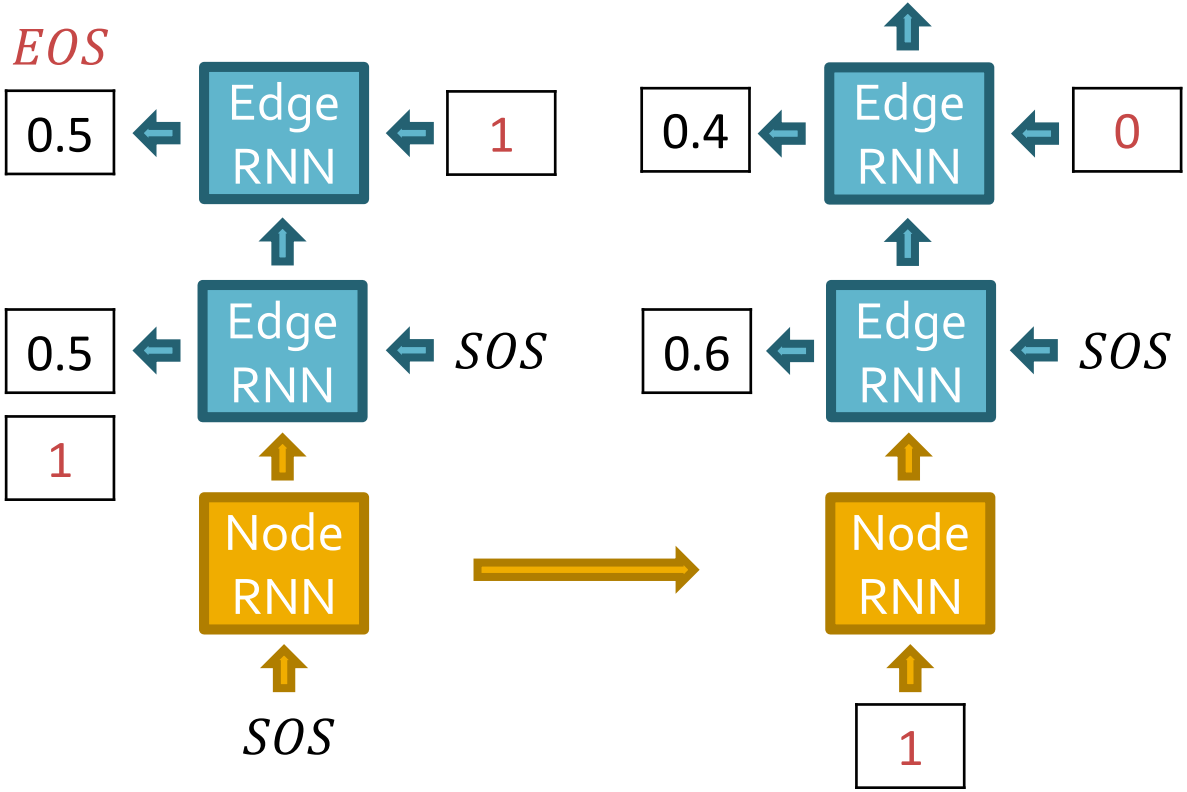
New edges are used to update Node RNN

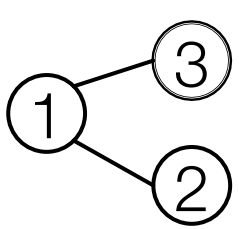




Put Things Together: Training

Edge RNN predicts how Node 3 connects to Node 2



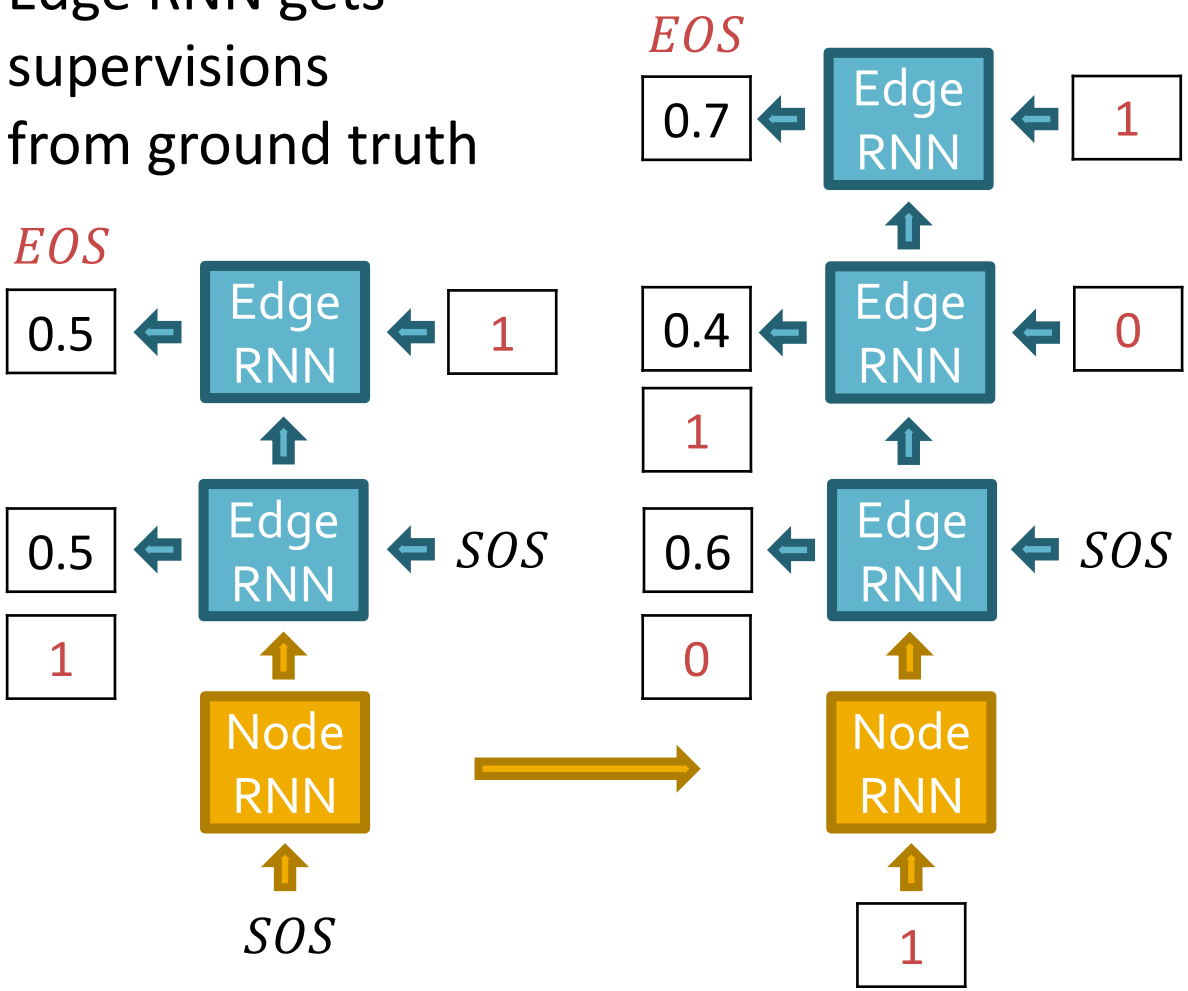


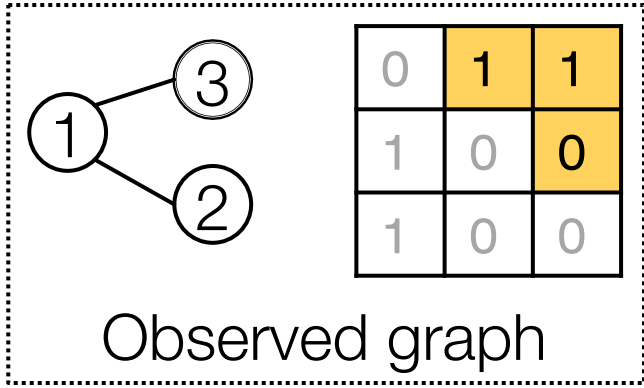
0	1	1
1	0	0
1	0	0

Observed graph

Put Things Together: Training

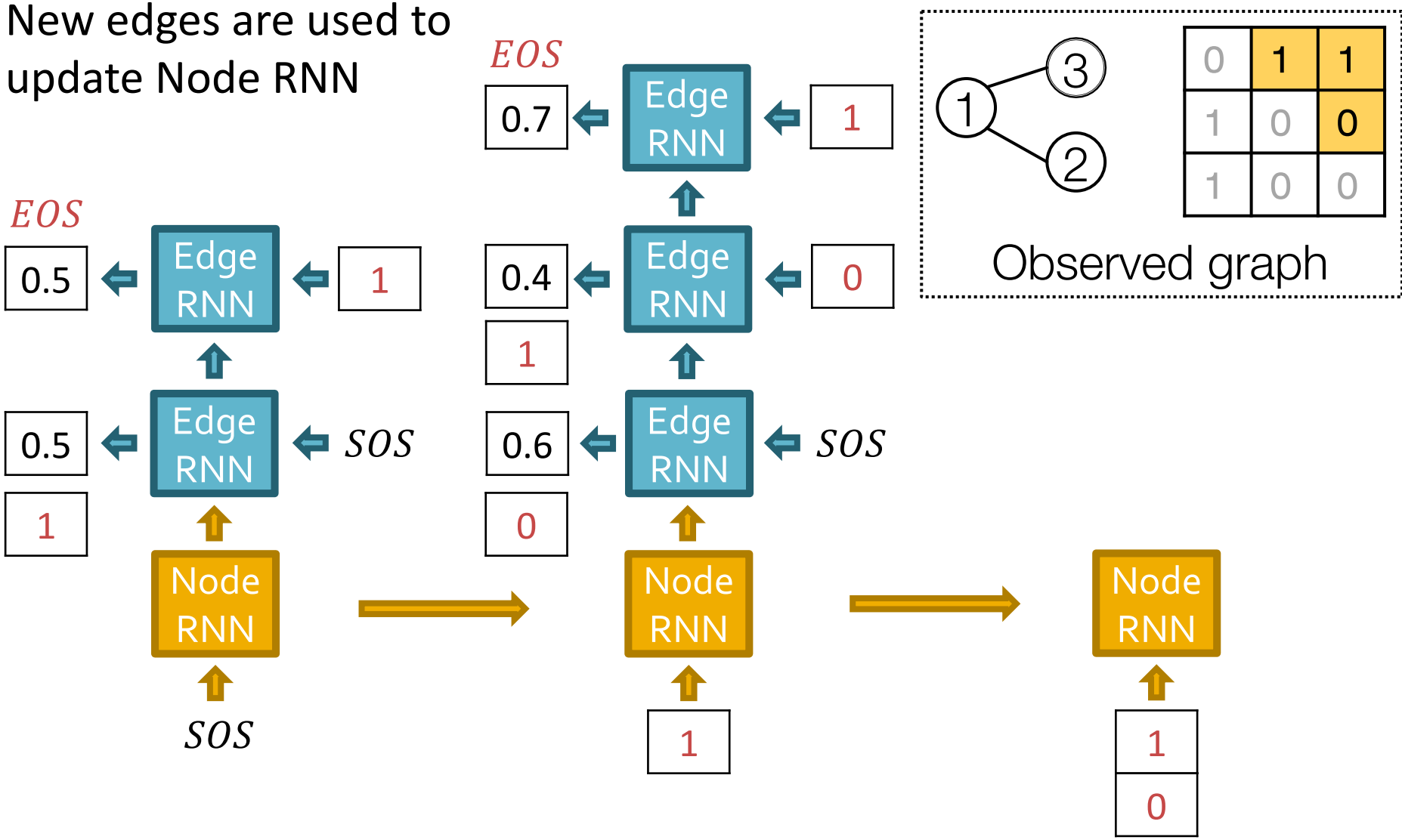
Edge RNN gets supervisions from ground truth





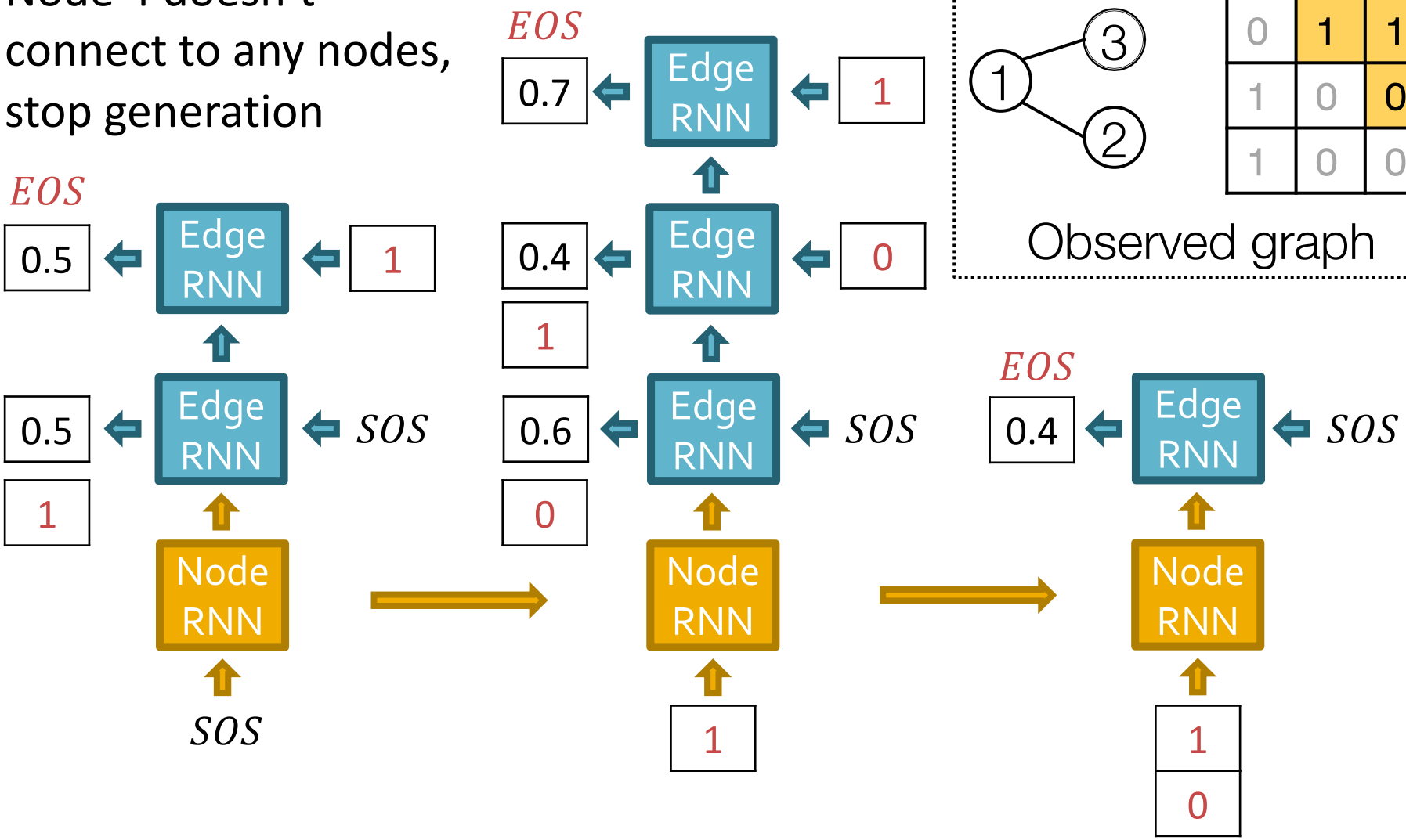
Put Things Together: Training

New edges are used to update Node RNN



Put Things Together: Training

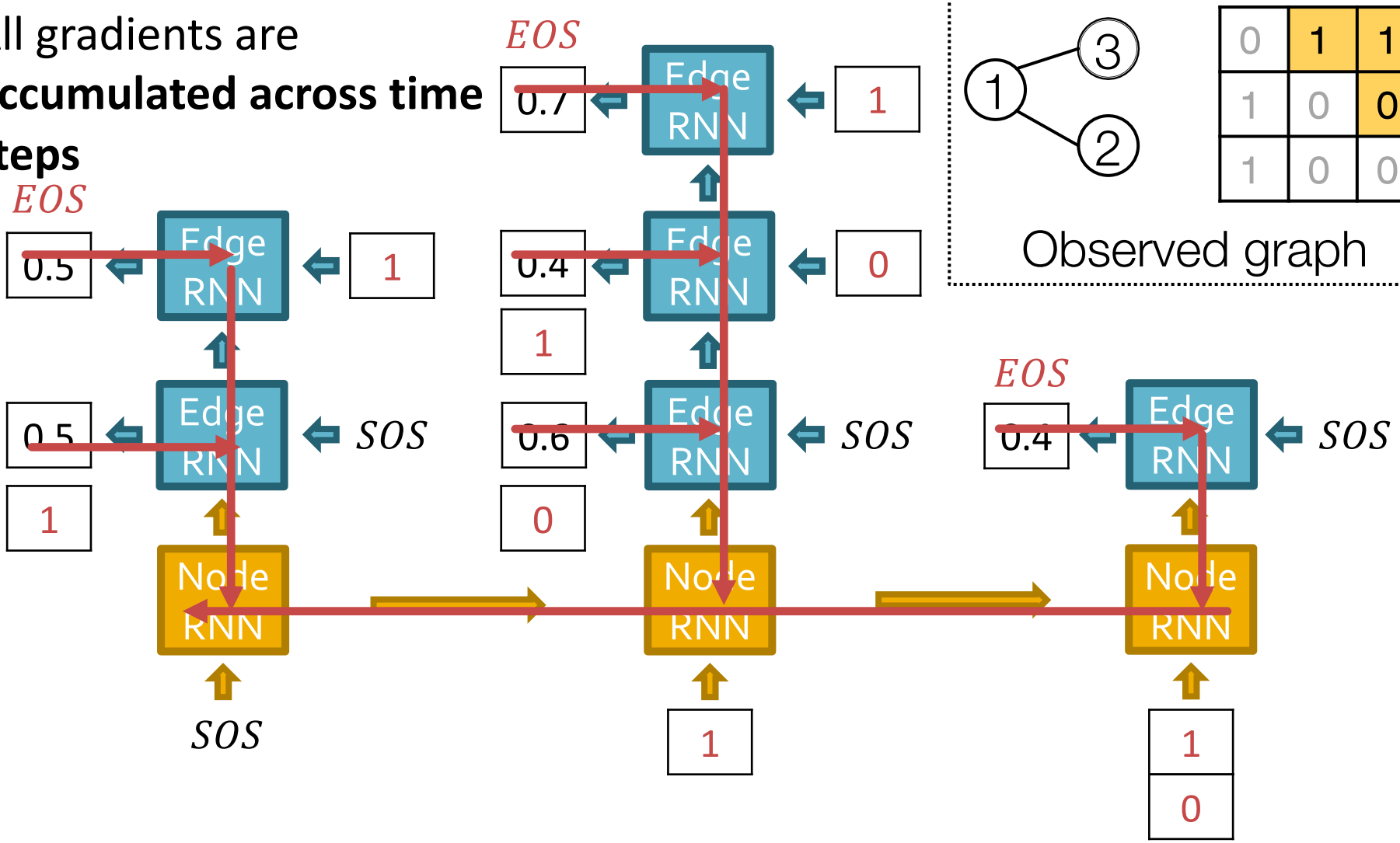
Node 4 doesn't connect to any nodes, stop generation



Put Things Together: Training

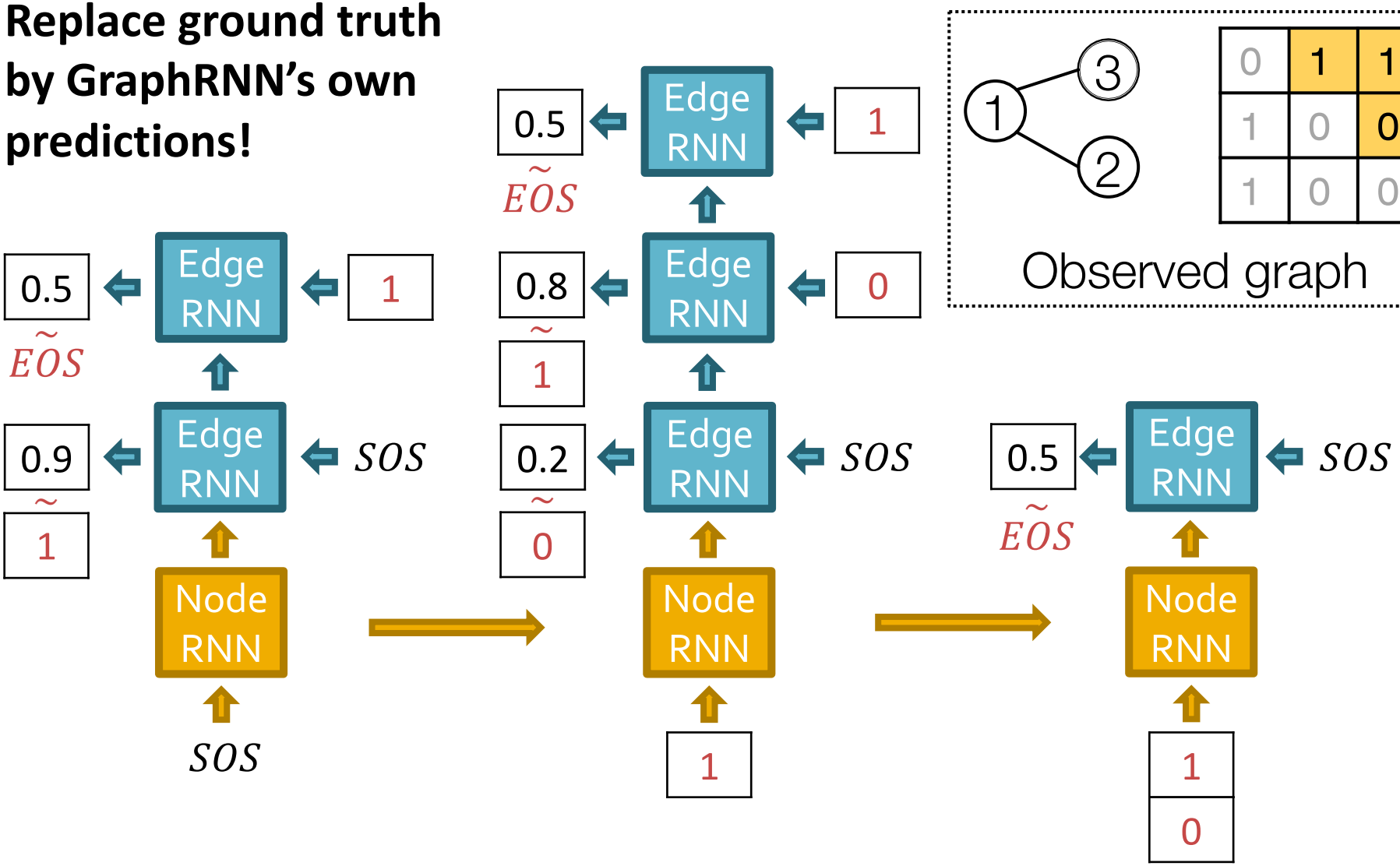
Backprop through time:

All gradients are **accumulated across time steps**



Put Things Together: Test

Replace ground truth by GraphRNN's own predictions!

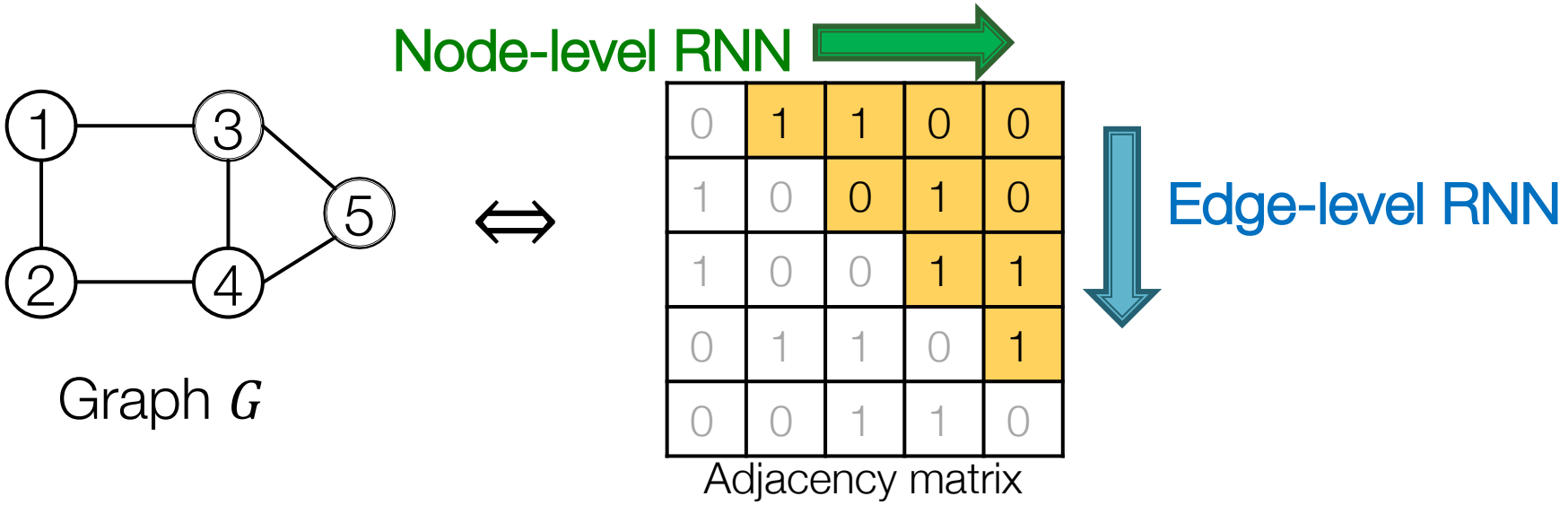


GraphRNN: Two levels of RNN

Quick Summary of GraphRNN:

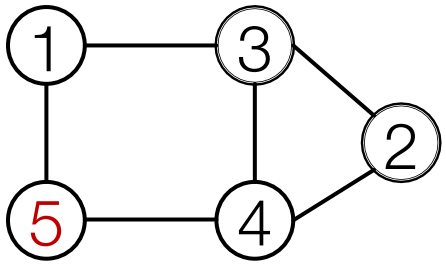
- Generate a graph by generating a two level sequence
- Use RNN to generate the sequences

- **Next**: Making GraphRNN tractable, proper evaluation



Issue: Tractability

- Any node can connect to any prior node
- Too many steps for edge generation
  - Need to generate full adjacency matrix
  - Complex too-long edge dependencies



"Recipe" to generate the left graph:

- Add node 1
- Add node 2
- Add node 3
- Connect 3 with 1 and 2
- Add node 4
- …

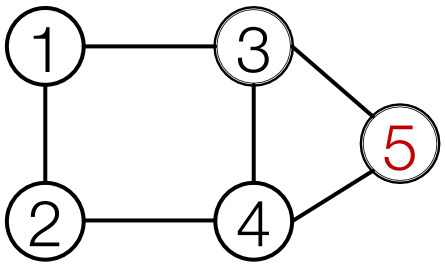
Random node ordering:

Node 5 may connect to any/all previous nodes

How do we limit this complexity?

Solution: Tractability via BFS

- **Breadth-First Search node ordering**



BFS ordering

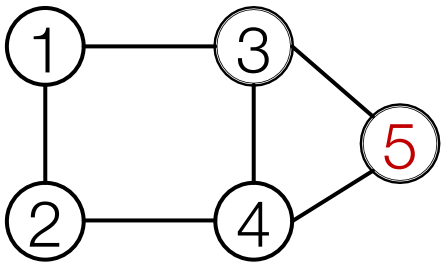
"Recipe" to generate the left graph:

- Add node 1
- Add node 2
- Connect 2 with 1
- Add node 3
- Connect 3 with 1
- Add node 4
- Connect 4 with 2 and 3

- **BFS node ordering:**
  - Since Node 4 doesn't connect to Node 1
  - We know all Node 1's neighbors have already been traversed
  - Therefore, Node 5 and the following nodes will never connect to node 1
  - We only need memory of 2 "steps" rather than $n - 1$ steps

Solution: Tractability via BFS

- **Breadth-First Search node ordering**



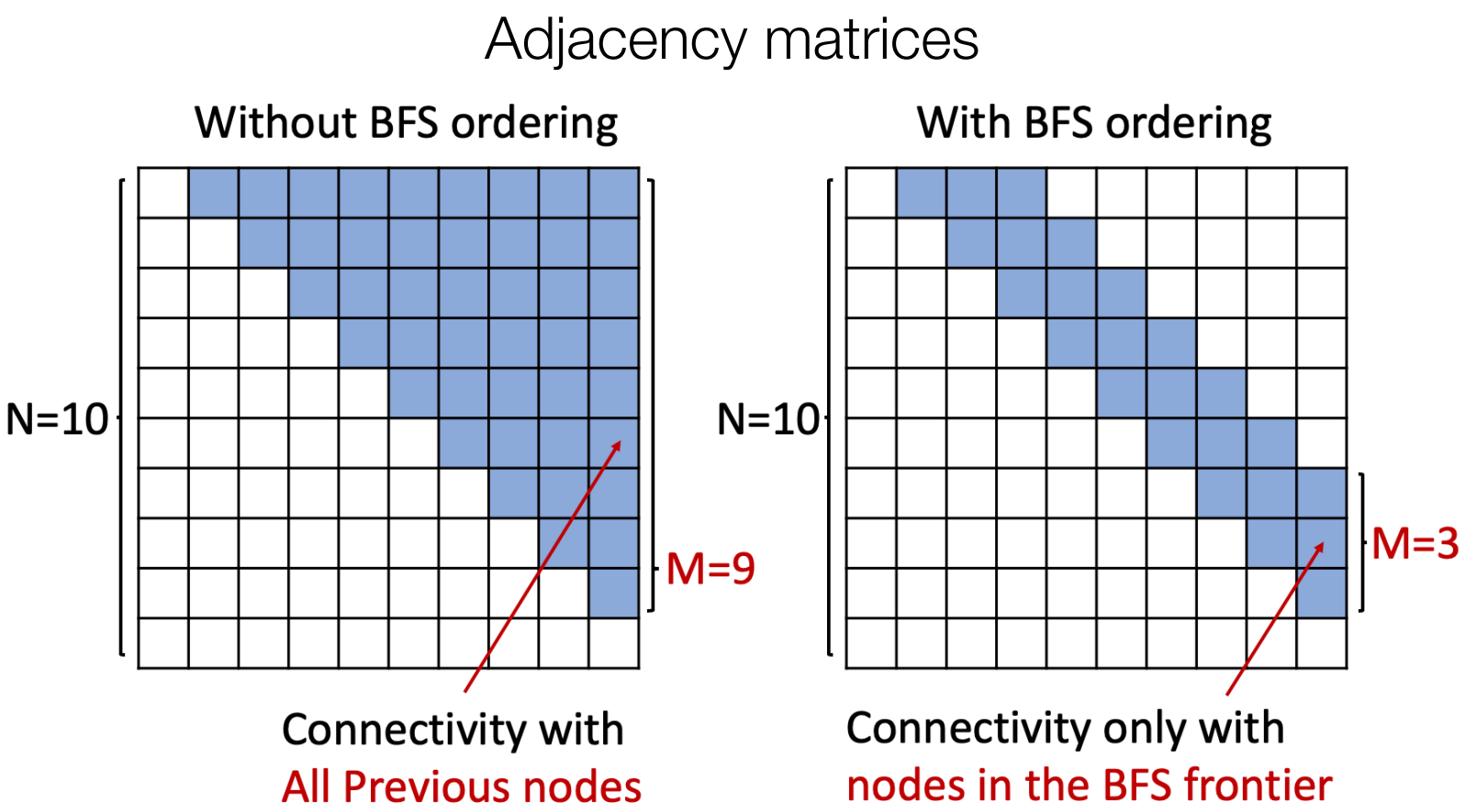
BFS ordering

BFS node ordering: Node 5 will never connect to node 1
(only need memory of 2 "steps" rather than $n-1$ steps)

- **Benefits:**
  - Reduce possible node orderings
    - From $O(n!)$ to number of distinct BFS orderings
  - Reduce steps for edge generation
    - Reducing number of previous nodes to look at

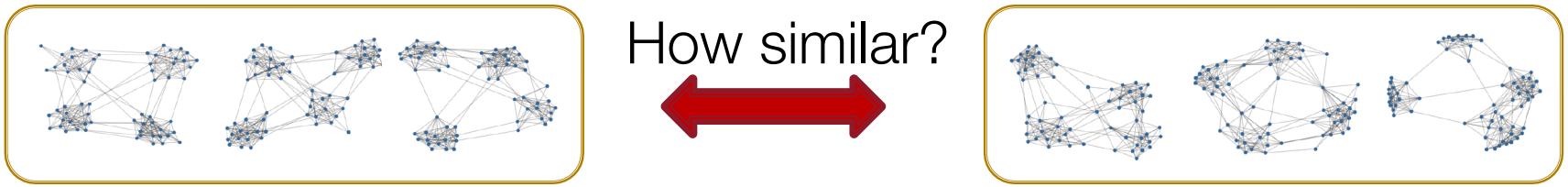
Solution: Tractability via BFS

- **BFS reduces the number of steps for edge generation**



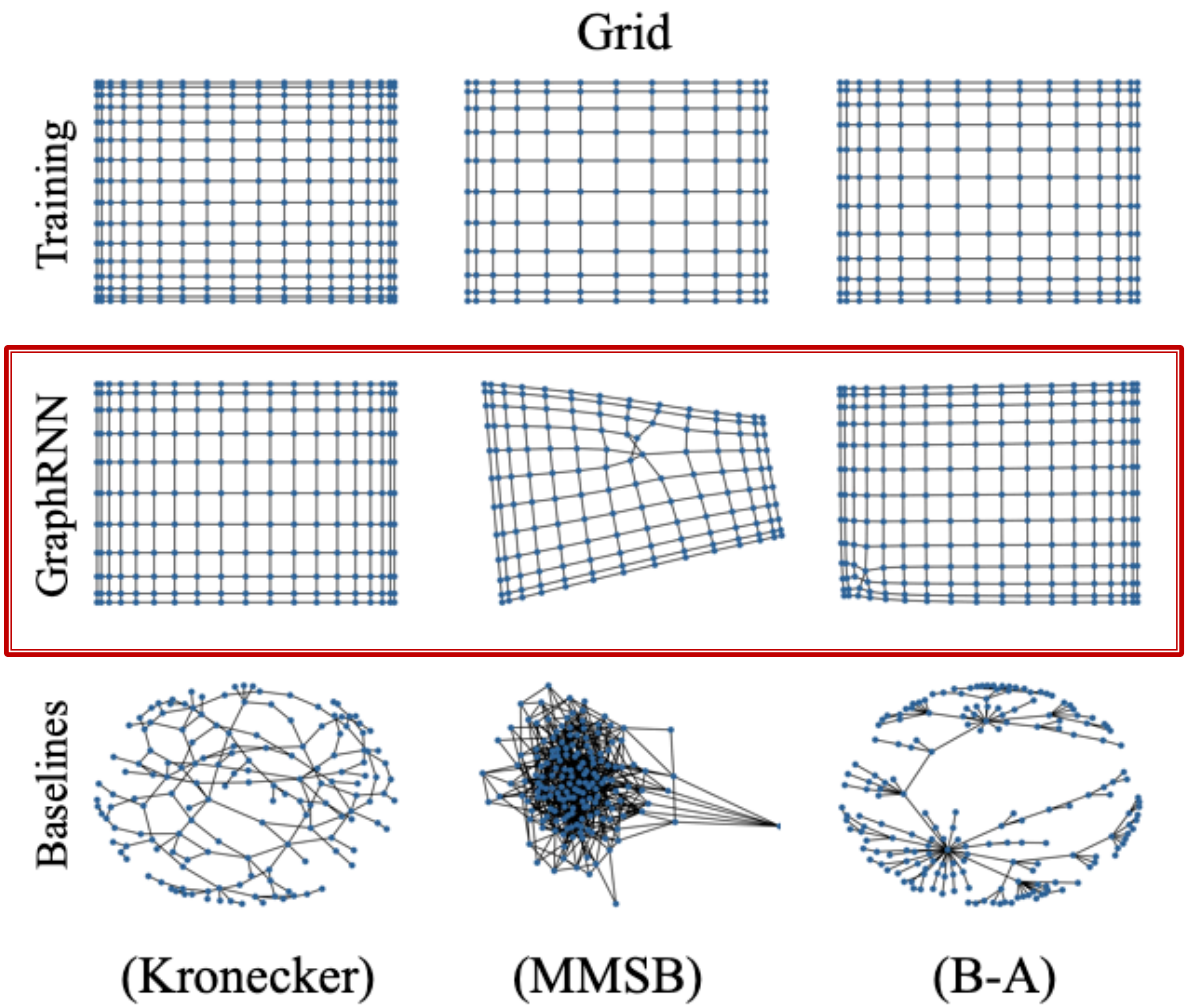
Evaluating Generated Graphs

- **Task: Compare two sets of graphs**

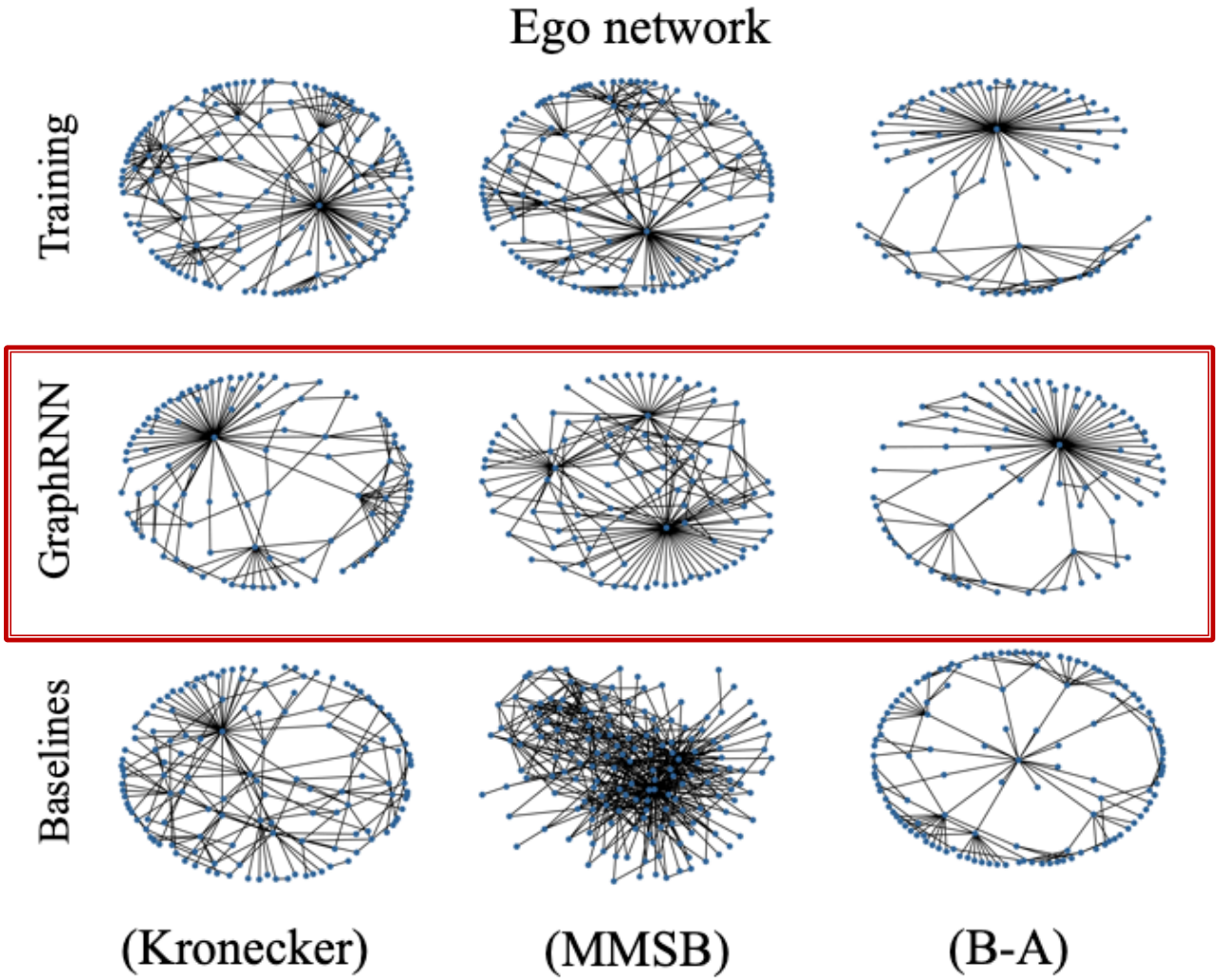


- **Goal:** Define similarity metrics for graphs
- **Challenge:** There is no efficient Graph Isomorphism test that can be applied to **any class** of graphs!
- **Solution**
  - Visual similarity
  - Graph statistics similarity

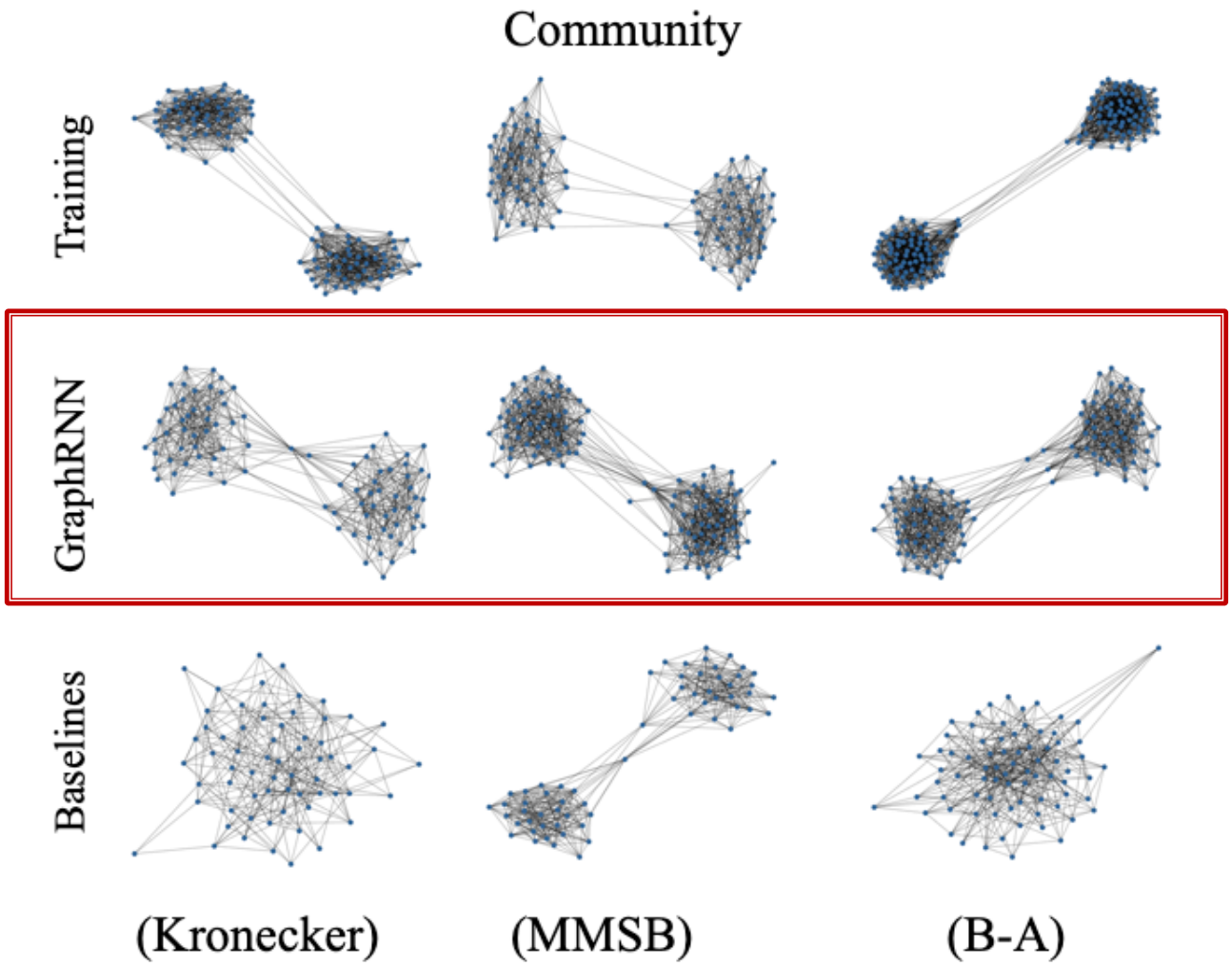
Visual Similarity



Visual Similarity



Visual Similarity



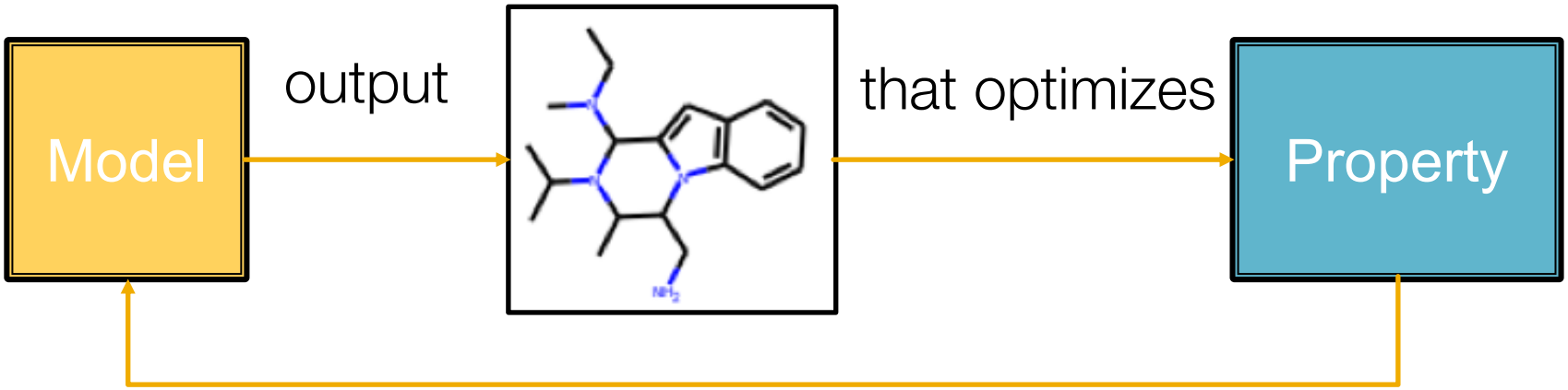
Outline of Today's Lecture

1. **Problem of Graph Generation**
2. **ML Basics for Graph Generation**
3. **GraphRNN**
4. **Applications and Open Questions**

[You et al., NeurIPS 2018]

Application: Drug Discovery

Question: Can we learn a model that can generate **valid** and **realistic** molecules with **high value of a given chemical property**?



Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. J. You, B. Liu, R. Ying, V. Pande, J. Leskovec. *Neural Information Processing Systems (NeurIPS)*, 2018.

Goal-Directed Graph Generation

Generating graphs that:

- **Optimize a given objective** (High scores)
  - e.g., drug-likeness (black box)
- **Obey underlying rules** (Valid)
  - e.g., chemical valency rules
- **Are learned from examples** (Realistic)
  - e.g., Imitating a molecule graph dataset

Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. J. You, B. Liu, R. Ying, V. Pande, J. Leskovec. *Neural Information Processing Systems (NeurIPS)*, 2018.

Model: GCPN

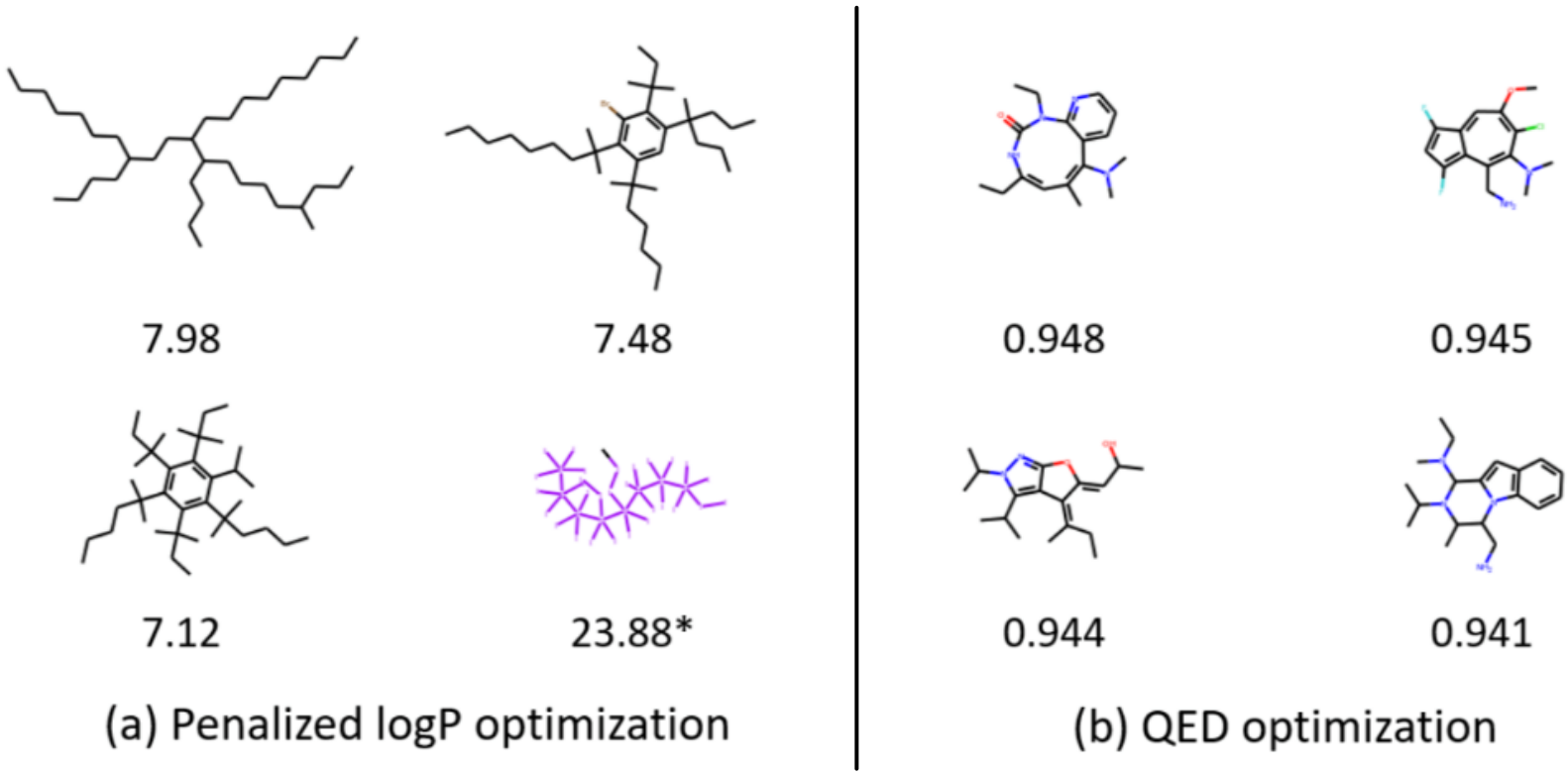
Graph Convolutional Policy Network combines graph representation + RL:

- Graph Neural Network captures complex structural information, and enables validity check in each state transition (Valid)
- Reinforcement learning optimizes intermediate/final rewards (High scores)
- Adversarial training imitates examples in given datasets (Realistic)

Qualitative Results

Visualization of GCPN graphs:

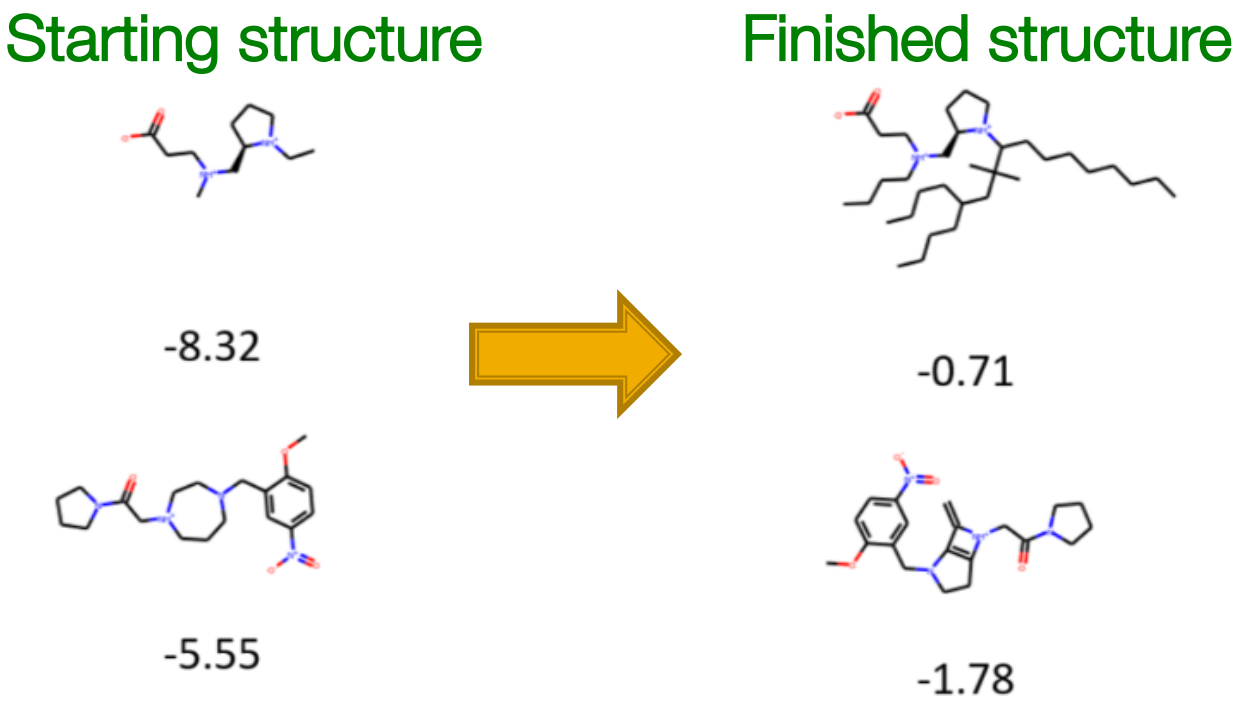
Generate graphs with high property scores



Qualitative Results

Visualization of GCPN graphs:

Edit given graph for higher property scores



(c) Constrained optimization of penalized logP

Open Problems

- Generating graphs in other domains
  - 3D shapes, point clouds, scene graphs, etc.
- Scale up to large graphs
  - Hierarchical action space, allowing high-level action like adding a structure at a time
- Other applications: Anomaly detection
  - Use generative models to estimated prob. of real graphs vs. fake graphs

Outline of Today's Lecture

1. **Problem of Graph Generation** ✓
2. **ML Basics for Graph Generation** ✓
3. **GraphRNN** ✓
4. **Applications and Open Questions** ✓