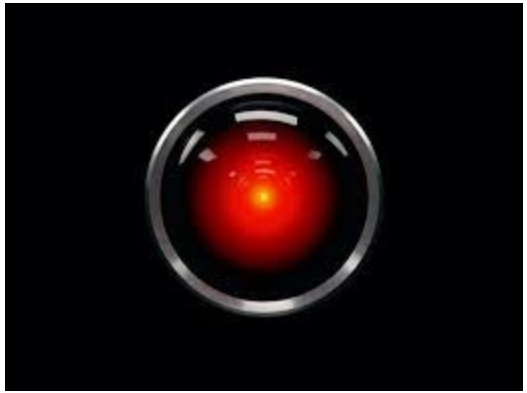


Generative Graph Learning

HAL-9000: What is going to happen?

Dave: Something wonderful.

HAL-9000: I'm afraid.

Dave: Don't be. We'll be together.

Jay Urbain, PhD - 2/5/2023

References

GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, Jure Leskovec

https://arxiv.org/abs/1802.08773

Also presentation slides.

On Discriminative vs. Generative classifiers, Andrew Ng, Michael Jordan, Stanford.

http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf

arXiv:1802.08773v3 [cs.LG] 23 Jun 2018

GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

Jiaxuan You [*1] Rex Ying [*1] Xiang Ren [2] William L. Hamilton [1] Jure Leskovec [1]

Abstract

Modeling and generating graphs is fundamental for studying networks in biology, engineering, and social sciences. However, modeling complex distributions over graphs and then efficiently sampling from these distributions is challenging due to the non-unique, high-dimensional nature of graphs and the complex, non-local dependencies that exist between edges in a given graph. Here we propose GraphRNN, a deep autoregressive model that addresses the above challenges and approximates any distribution of graphs with minimal assumptions about their structure. GraphRNN learns to generate graphs by training on a representative set of graphs and decomposes the graph generation process into a sequence of node and edge formations, conditioned on the graph structure generated so far. In order to quantitatively evaluate the performance of GraphRNN, we introduce a benchmark suite of datasets, baselines and novel evaluation metrics based on Maximum Mean Discrepancy, which measure distances between sets of graphs. Our experiments show that GraphRNN significantly outperforms all baselines, learning to generate diverse graphs that match the structural characteristics of a target set, while also scaling to graphs $50\times$ larger than previous deep models.

1. Introduction and Related Work

Generative models for real-world graphs have important applications in many domains, including modeling physical and social interactions, discovering new chemical and molecular structures, and constructing knowledge graphs. Development of generative graph models has a rich history, and many methods have been proposed that can generate graphs based on a priori structural assumptions (Newman, 2010). However, a key open challenge in this area is developing methods that can directly *learn* generative models from an observed set of graphs. Developing generative models that can learn directly from data is an important step towards improving the fidelity of generated graphs, and paves a way for new kinds of applications, such as discovering new graph structures and completing evolving graphs.

In contrast, traditional generative models for graphs (*e.g.*, Barabási-Albert model, Kronecker graphs, exponential random graphs, and stochastic block models) (Erdős & Rényi, 1959; Leskovec et al., 2010; Albert & Barabási, 2002; Airoldi et al., 2008; Leskovec et al., 2007; Robins et al., 2007) are hand-engineered to model a particular family of graphs, and thus do not have the capacity to directly learn the generative model from observed data. For example, the Barabási-Albert model is carefully designed to capture the scale-free nature of empirical degree distributions, but fails to capture many other aspects of real-world graphs, such as community structure.

Recent advances in deep generative models, such as variational autoencoders (VAE) (Kingma & Welling, 2014) and generative adversarial networks (GAN) (Goodfellow et al., 2014), have made important progress towards generative modeling for complex domains, such as image and text data. Building on these approaches a number of deep learning models for generating graphs have been proposed (Kipf & Welling, 2016; Grover et al., 2017; Simonovsky & Komodakis, 2018; Li et al., 2018). For example, Simonovsky & Komodakis 2018 propose a VAE-based approach, while Li et al. 2018 propose a framework based upon graph neural networks. However, these recently proposed deep models are either limited to learning from a single graph (Kipf & Welling, 2016; Grover et al., 2017) or generating small graphs with 40 or fewer nodes (Li et al., 2018; Simonovsky & Komodakis, 2018)—limitations that stem from three fundamental challenges in the graph generation problem:

- **Large and variable output spaces:** To generate a graph with n nodes the generative model has to output n^2 values to fully specify its structure. Also, the number of nodes n and edges m varies between different graphs and a generative model needs to accommodate such complexity and variability in the output space.

[*]Equal contribution [1]Department of Computer Science, Stanford University, Stanford, CA, 94305 [2]Department of Computer Science, University of Southern California, Los Angeles, CA, 90007. Correspondence to: Jiaxuan You <jiaxuan@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

Generative and Discriminative Machine Learning Models

- Generative models ?
- Discriminative models ?

Generative and Discriminative Machine Learning Models

- Generative models are a wide class of machine learning algorithms which make predictions by *modelling joint distribution* ***P(y, x)****.*
- Discriminative models are a class of supervised machine learning models which make predictions by *estimating conditional probability* ***P(y|x)****.*

Generative and Discriminative Models Analogy

Task: determine the language that someone is speaking

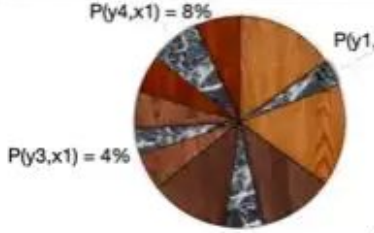
- **Generative approach**: learn each language and determine as to which language the speech belongs to.
- **Discriminative approach**: determine the linguistic differences without learning any language– a much easier task!

Conditional probability

$P(y \mid x)$

Generative model

Discriminative model



$\approx$

Joint probability

$P(y, x)$

Probability of class

$P(y)$

Probability of observation given class $P(x \mid y)$

Estimated distribution

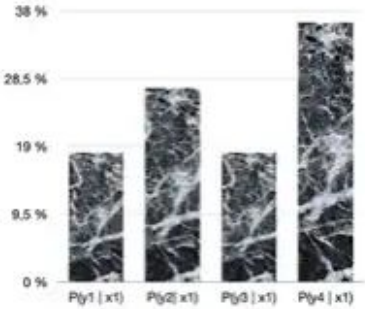
$\hat{P}(y)$

Estimated distribution

$\hat{P}(x \mid y)$

Estimated distribution

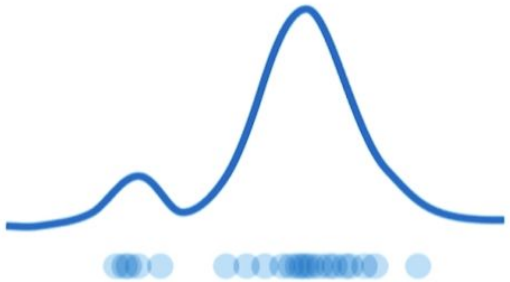
$\hat{P}(y \mid x)$



Generative Modeling

Goal: Take as input training samples from some distribution and learn a model that represents that distribution

Density Estimation



Sample Generation

Input samples

Training data ~ $P_{data}(x)$

Generated samples

Generated ~ $P_{model}(x)$

How can we learn $P_{model}(x)$ similar to $P_{data}(x)$?

Image synthesis: Which face is real?

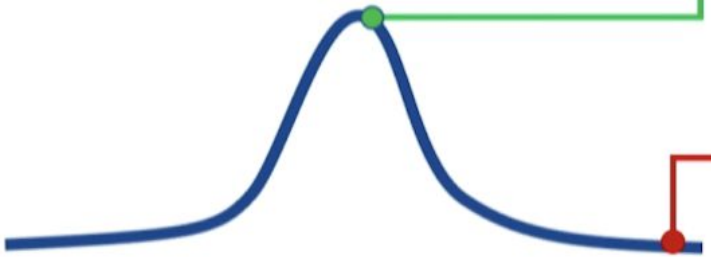
These people are not real – they were produced by our generator that allows control over different aspects of the image. Nvidia / StyleGAN

Outlier Detection

- **Problem:** How can we detect when we encounter something new or rare?
- **Strategy:** Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!

95% of Driving Data:
(1) sunny, (2) highway, (3) straight road

Detect outliers to avoid unpredictable behavior when training

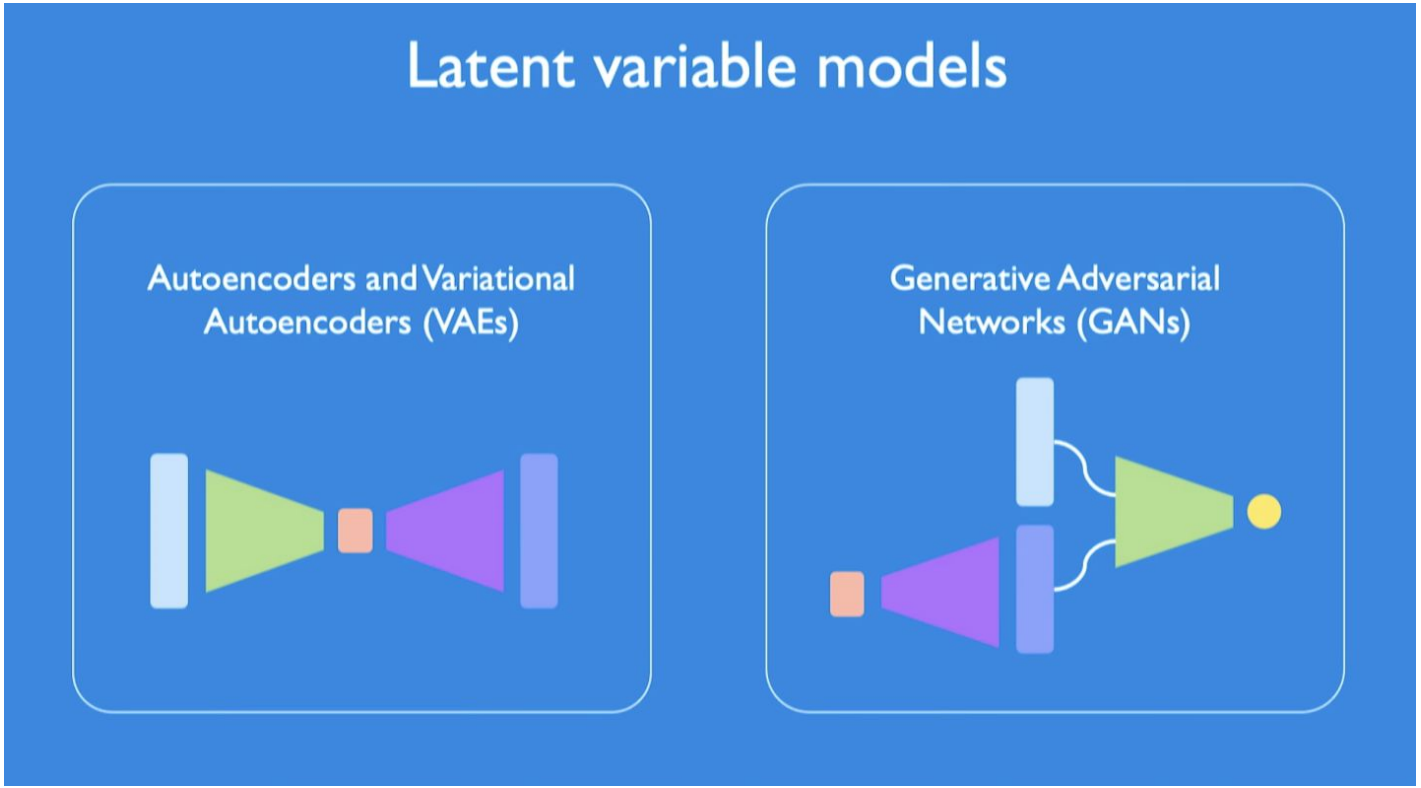


Edge Cases

Harsh Weather

Pedestrians

Latent Variable Models

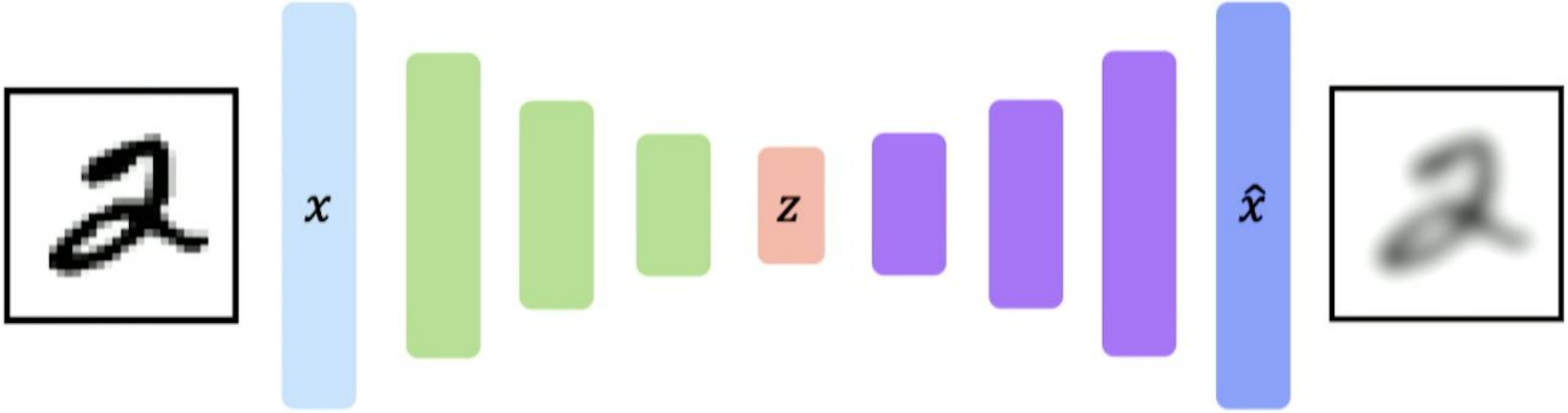


Latent variable: Plato's Myth of the Cave?

Can we learn the **true explanatory factors**, e.g. latent variables, from only observed data?

Autoencoder: Learn reconstruction of original data

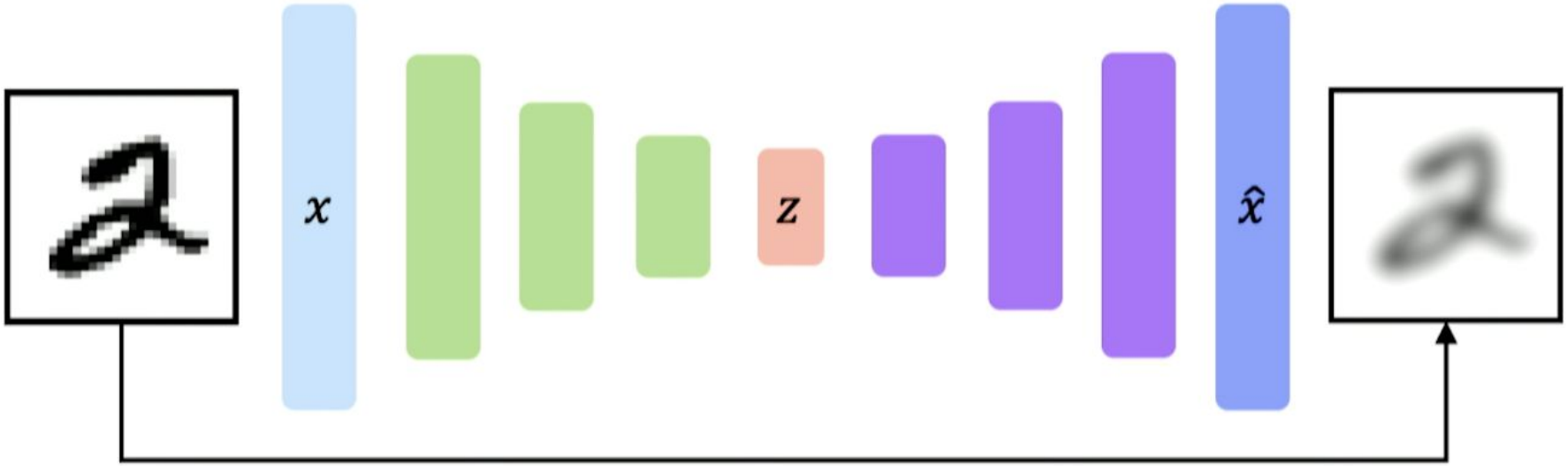
How can we learn this latent space?
Train the model to use these features to **reconstruct the original data**



"Decoder" learns mapping back from latent space, z,
to a reconstructed observation, $\hat{x}$

Autoencoder: Learn reconstruction of original data

How can we learn this latent space?
Train the model to use these features to **reconstruct the original data**



$$\mathcal{L}(x, \hat{x}) = ||x - \hat{x}||^2$$

Loss function doesn't use any labels!!

Autoencoding is a form of compression!
Smaller latent space will force a larger training bottleneck

2D latent space

5D latent space

Ground Truth

Autoencoders for representation learning

Bottleneck hidden layer forces network to learn a compressed latent representation

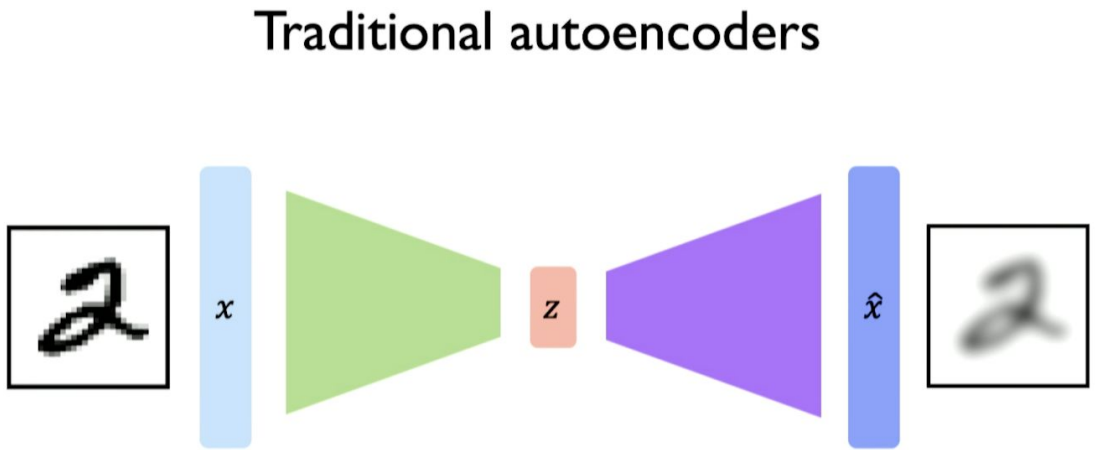
Reconstruction loss forces the latent representation to capture (or encode) as much "information" about the data as possible

Autoencoding = **Auto**matically **encoding** data; "Auto" = **self**-encoding

Traditional Autoencoders

Learns direct mapping of input to output.

Given a specific input, it always generates the same output.

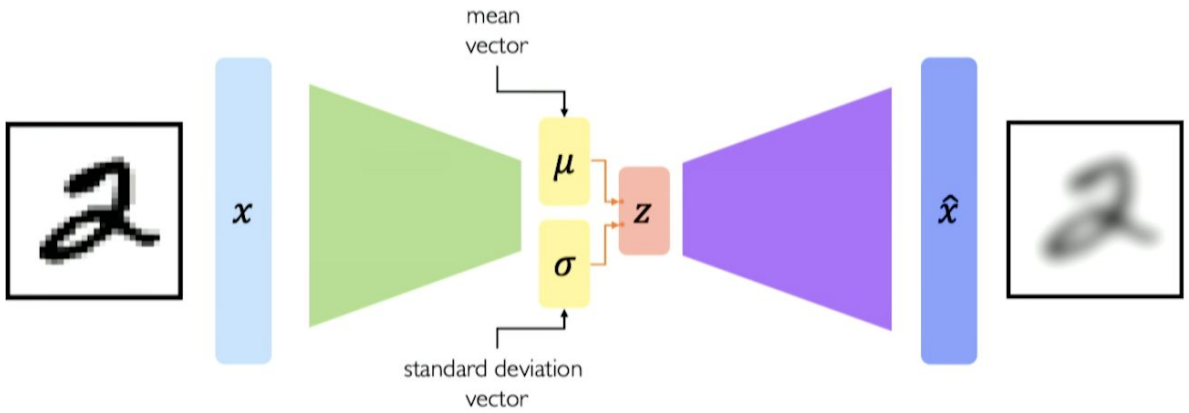


Problems with autoencoder?

VAE's: key difference with traditional Autoencoder

Introduce stochasticity.

Learn distribution, sample from distribution to generate new data.



Variational autoencoders are a probabilistic twist on autoencoders!
Sample from the mean and standard deviation to compute latent sample

On to graphs: Motivation for Graph Generation

- So far, we have been **learning from graphs**
  - We assume the graphs are given

Image credit: Medium

Social Networks

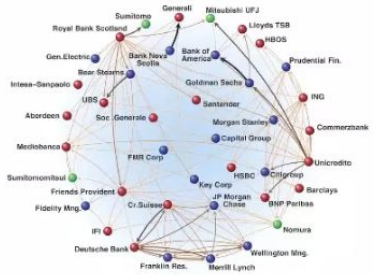


Image credit: Science

Economic Networks

Image credit: Kumen Learning

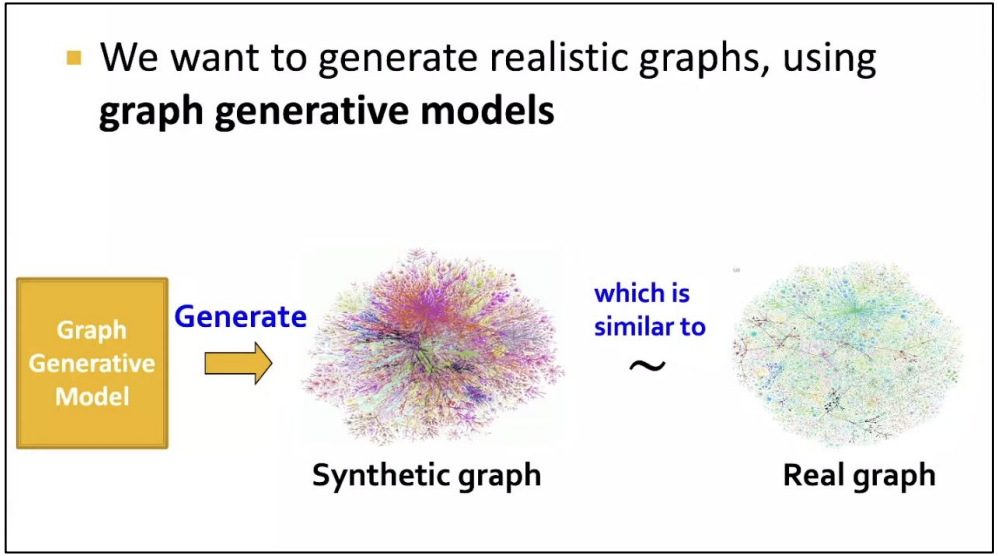
Communication Networks

- **But how are these graphs generated?**

Problem: Graph Generation

Learn a model to generate a graph that is somehow similar to underlying real-world graph:

- Generate graph of molecule or material design with specific properties
- Predict social network evolution
- Viral spread
- Malignant cell growth



Why do we study graph generation?

- Insights - We can understand the formulation of graphs
- Predictions - We can predict how will the graph further evolve
- Simulations - We can use the same process to generate novel graph instances
- Anomaly detection - We can decide if a graph is normal / abnormal
- Synthesis - Conditionally generate graph with certain properties

Steps for Graph Generation

Step 1: Properties of real world graphs

- A successful graph generative model should capture these properties.

Step 2: Traditional graph generative models

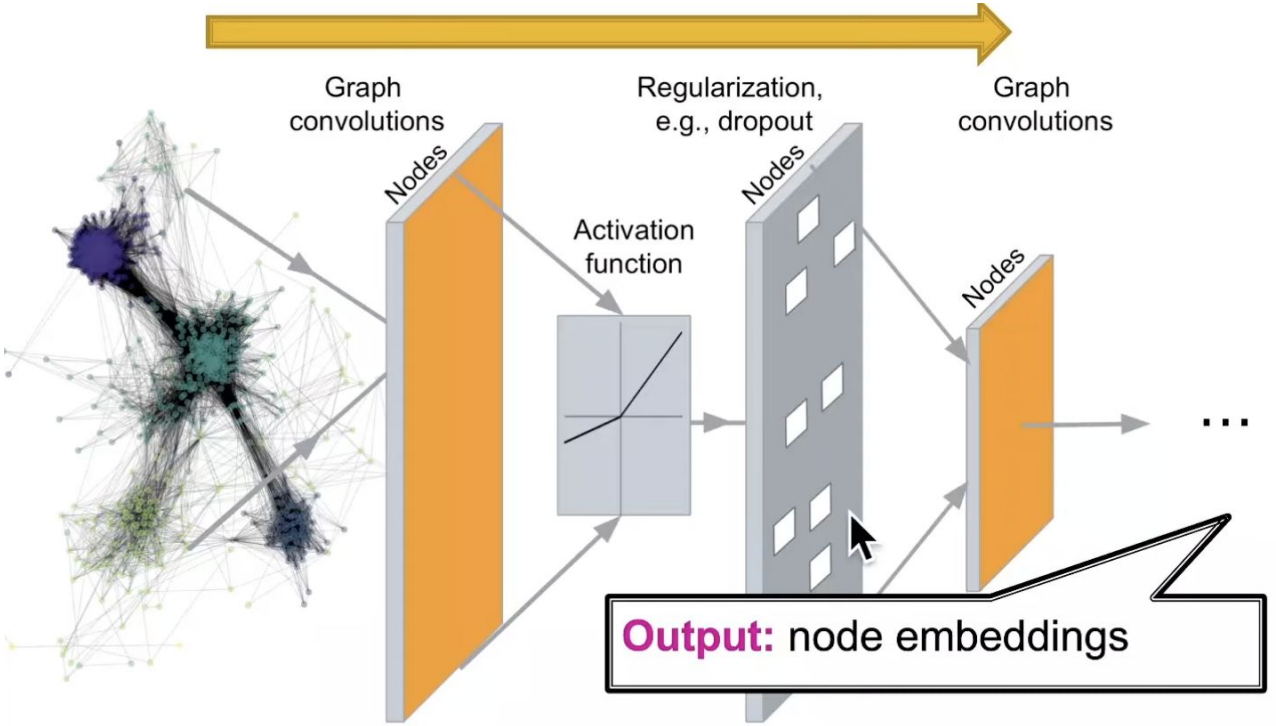
- Each come with different assumptions on the graph formulation process

Step 3: Deep graph generative models

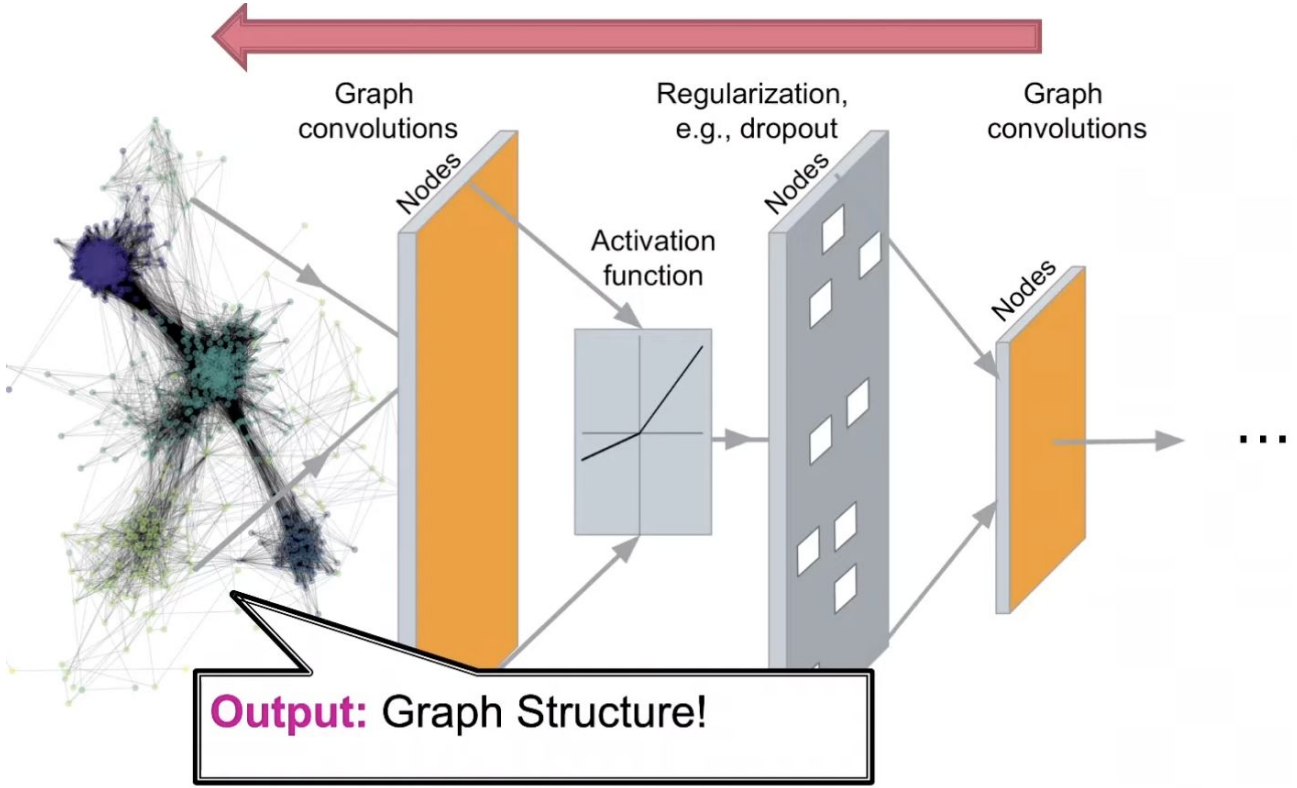
- Learn the graph formulation process from the data

Deep Graph Encoders

So far we've talked about deep graph encoders!



Deep Graph Decoding!



Graph Generation Tasks

Task 1: Realistic graph generation

- Generate graphs that are similar to a given set of graphs (focus)

Task 2: Goal-directed graph generation

- Generate graphs that optimize given objectives/constraints
- E.g., drug molecule generation.

Graph Generative Models

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

Generative Model Basics

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 $p_{model}(x; \theta)$ close to $p_{data}(x)$ **(Density estimation)**
  - **(2)** Make sure we can sample from $p_{model}(x; \theta)$ **(Sampling)**
    - We need to generate examples (graphs) from $p_{model}(\boldsymbol{x}; \theta)$

Generative Model Basics - Training

(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 $\boldsymbol{x}_i \sim p_{data}$ the $\sum_i \log p_{model}(\boldsymbol{x}_i;\theta^*)$ 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

Generative Model Basics - Inference

(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!

Generative Model Basics

Auto-regressive models:

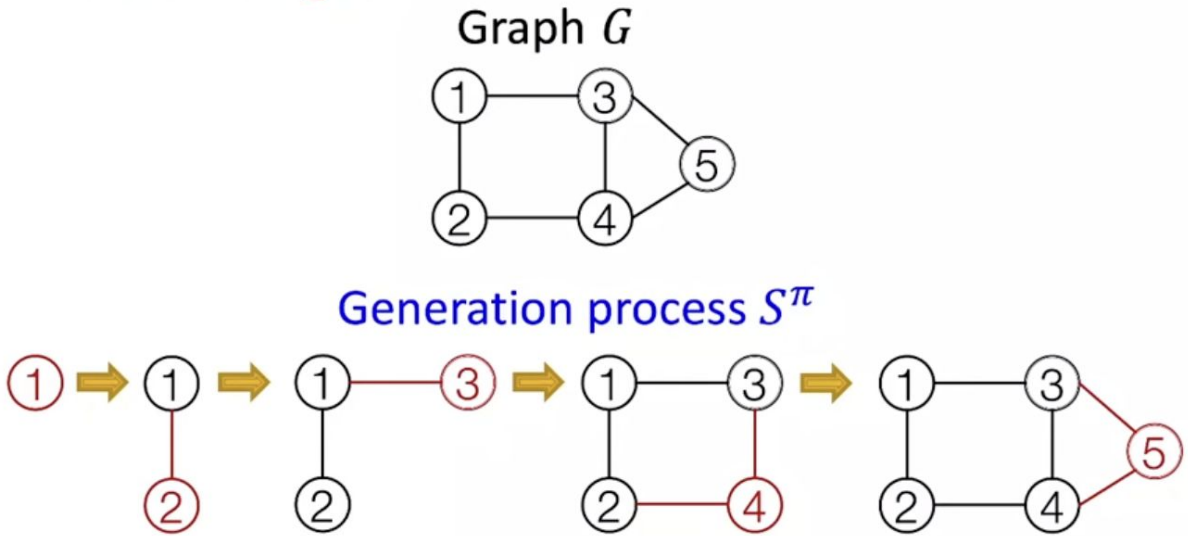
- $p_{model}(x; \theta)$ is used for **both density estimation and sampling**
  - Other models like Variational Auto Encoders (VAEs), Generative Adversarial Nets (GANs) have 2 or more models, each playing one of the roles
  - **Idea: Chain rule.** Joint distribution is a product of conditional distributions:

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

  - 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)

Generating realistic graphs: GraphRNN

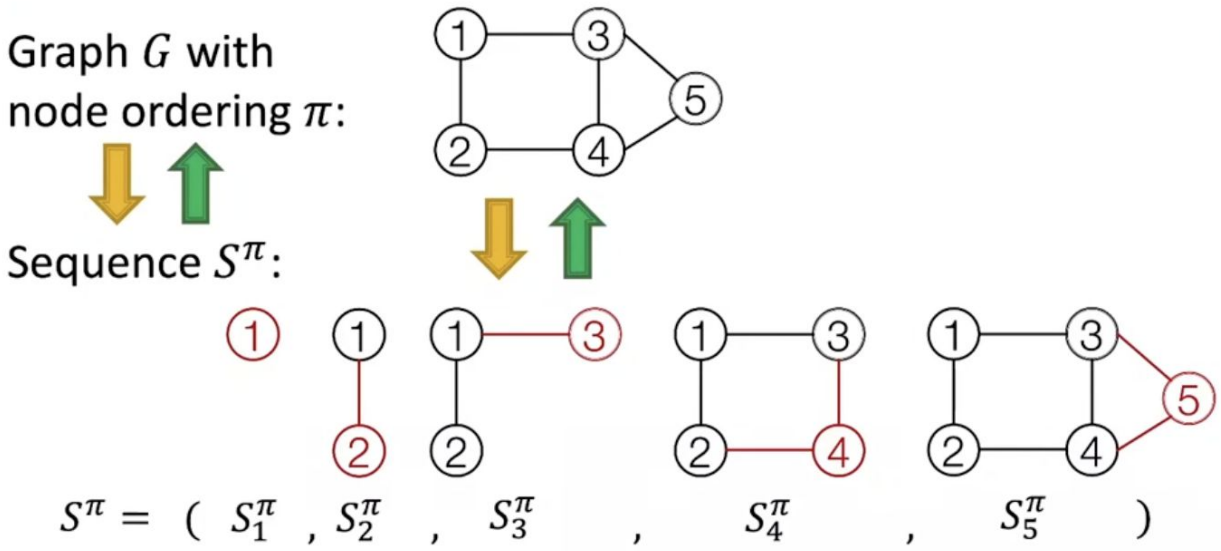
Generating graphs via sequentially adding nodes and edges



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.

Model Graphs as Sequences

Graph ***G*** with node ordering ***π*** can be uniquely mapped into a sequence of node and edge additions S^{π}.

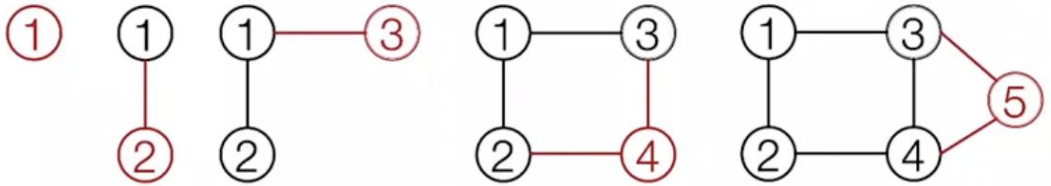


Model Graphs as Sequences

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



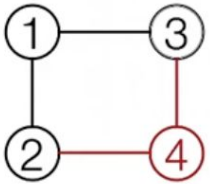
$S^{\pi} = (S_1^{\pi}, S_2^{\pi}, S_3^{\pi}, S_4^{\pi}, S_5^{\pi})$

"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}

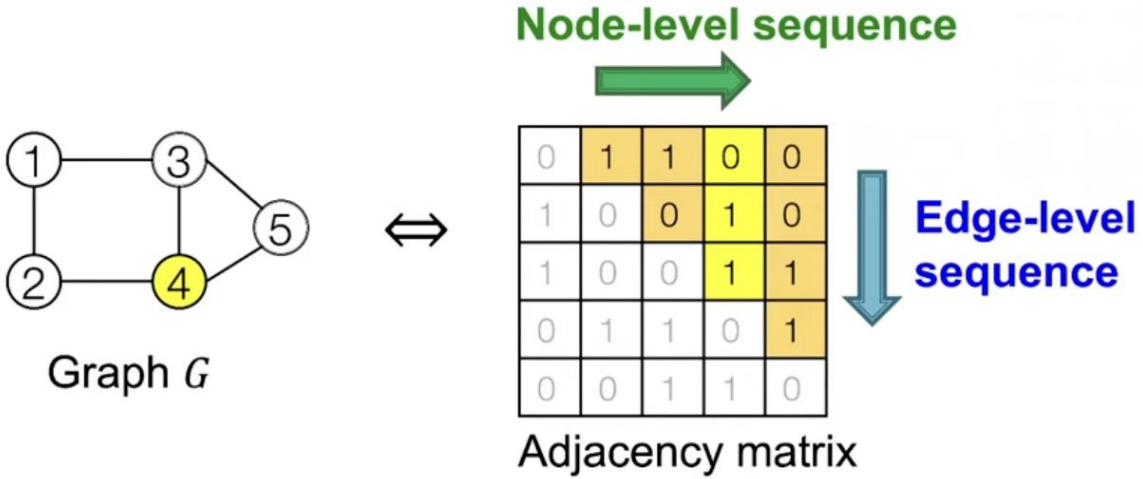
$S_4^{\pi} = (\; S_{4,1}^{\pi} \; , \; S_{4,2}^{\pi} \; , \; S_{4,3}^{\pi} \;)$

"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



Model Graphs as Sequences

Transformed graph generation problem into a sequence generation problem.

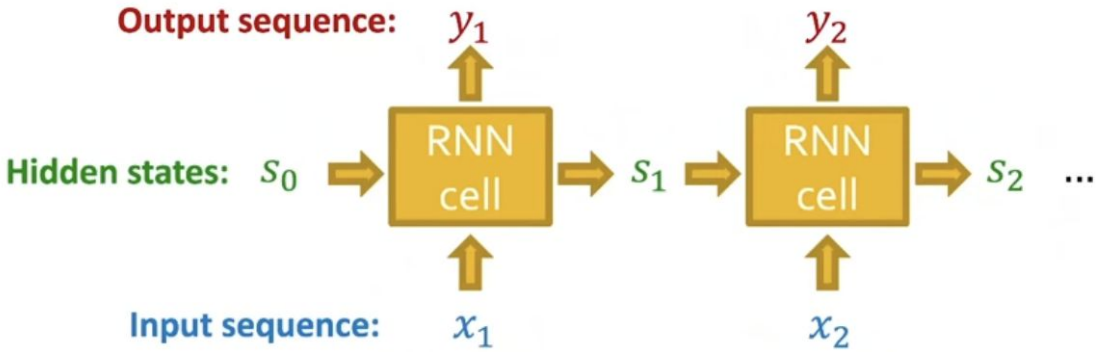
Need to model two processes:

- Generate a state for a new node (node-level sequence)
- Generate edges for the new node based on its state (edge-level sequence

Approach: **Recurrent Neural Network**

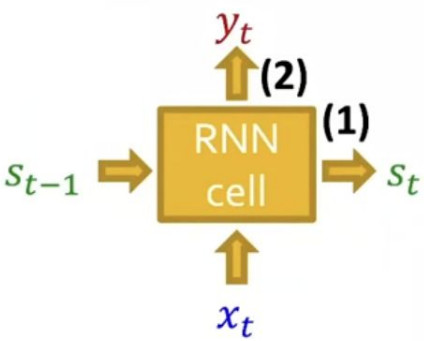
RNNs

- RNNs are designed for **sequential data**
  - RNN sequentially takes **input sequence** to **update its hidden states**
  - The **hidden states** summarize all the information input to RNN
  - The update is conducted via **RNN cells**



RNNs

- s_t: **State** of RNN after step t
- x_t: **Input** to RNN at step t
- y_t: **Output** of RNN at step t
- **RNN cell:** W, U, V: Trainable parameters



The RNN cell:

(1) Update hidden state:

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

(2) Output prediction:

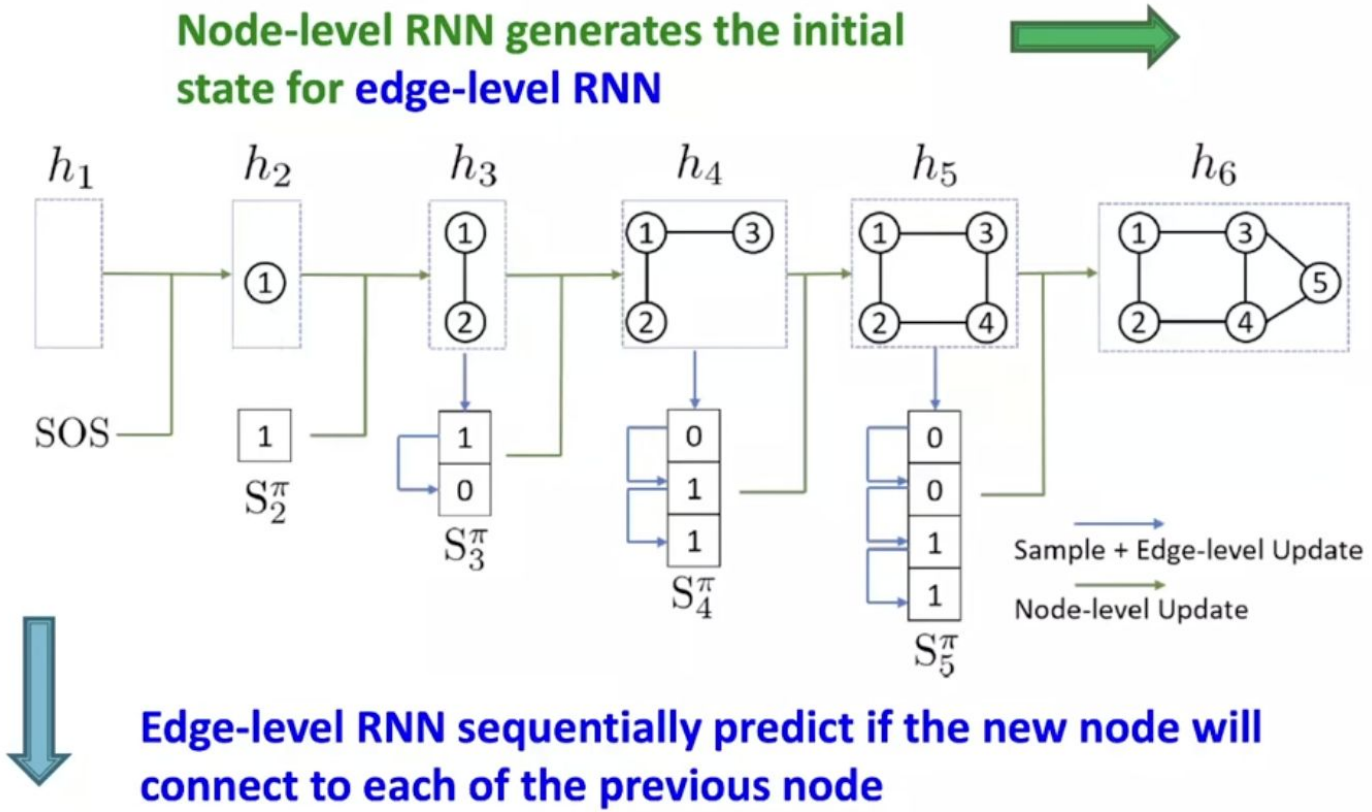
$$y_t = V \cdot s_t$$

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

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 sequentially predict if the new node will connect to each of the previous node

GraphRNN: Two levels of RNN



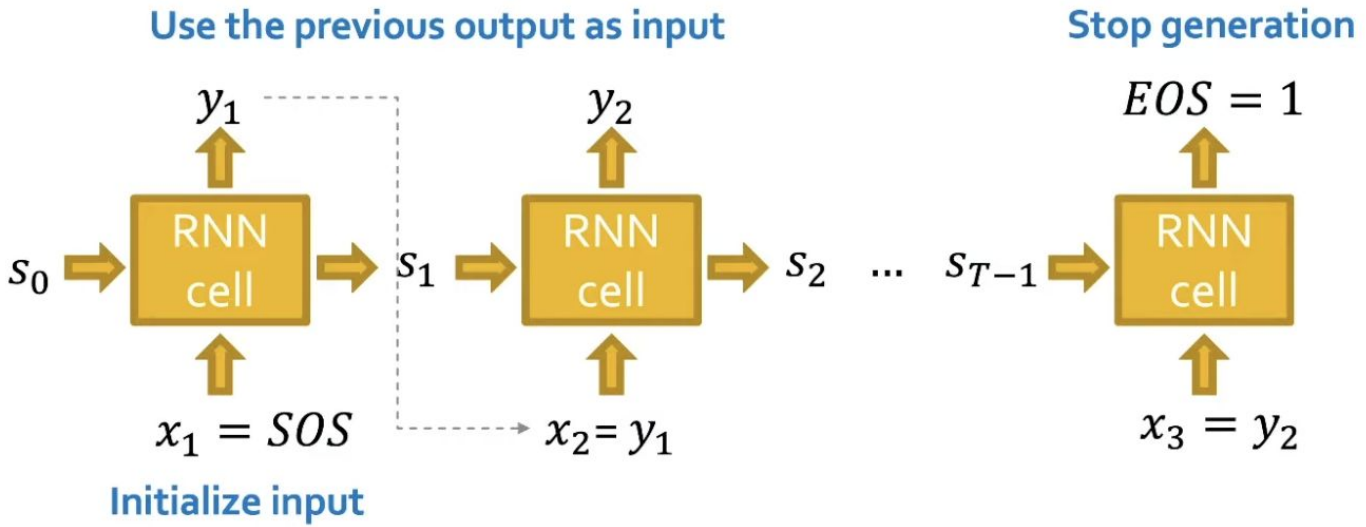
RNN for Sequence Generation - Inference (we have a trained model)

- **Q:** How to use RNN to generate sequences?
- **A:** Let $x_{t+1} = y_t$ (Use the previous output as input)

- **Q:** How to initialize the input sequence?
- **A:** Use **start of sequence token (SOS)** as the initial input
  - SOS is usually a vector with all zero/ones

- **Q:** When to stop generation?
- **A:** Use **end of sequence token (EOS)** as an **extra** RNN output
  - If output EOS=0, RNN will continue generation
  - If output EOS=1, RNN will stop generation

RNN for Sequence Generation

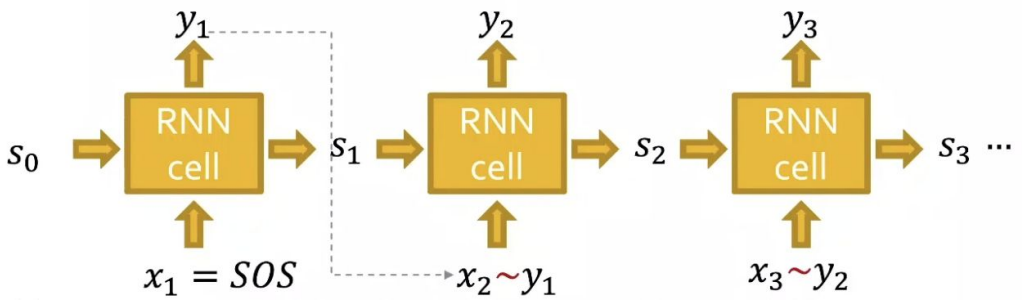


- This is good, but this model is **deterministic**

RNN for Sequence Generation

Need to introduce stochasticity in the model.

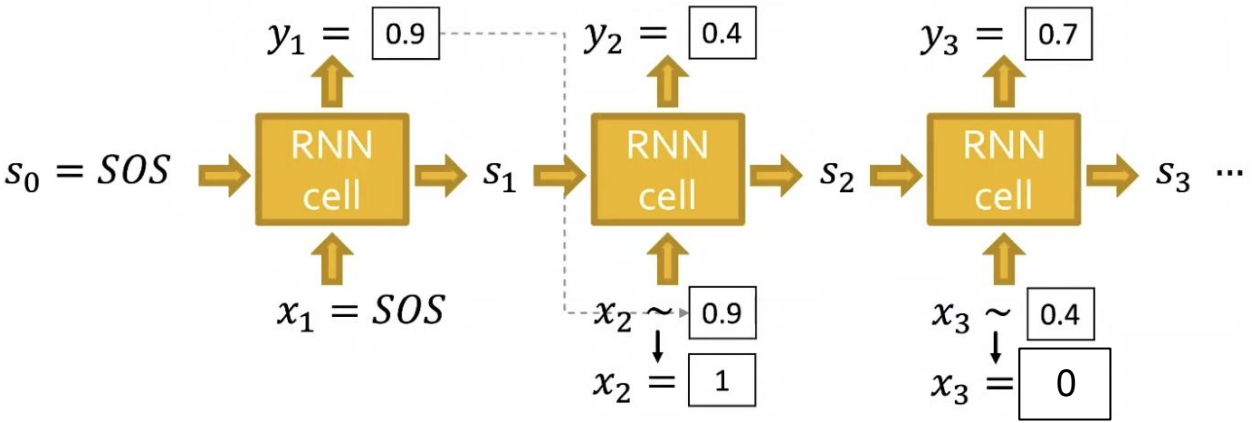
- **Remember our goal**: Use RNN to model $\prod_{k=1}^{n} p_{model}(x_t|x_1, \dots, x_{t-1}; \theta)$
- Let $y_t = p_{model}(x_t|x_1, \dots, x_{t-1}; \theta)$
- Then we need to sample x_{t+1} from y_t: $x_{t+1} \sim y_t$
  - Each step of RNN outputs a **probability of a single edge**
  - We then sample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



RNN at Test Time

Suppose we already have trained the model

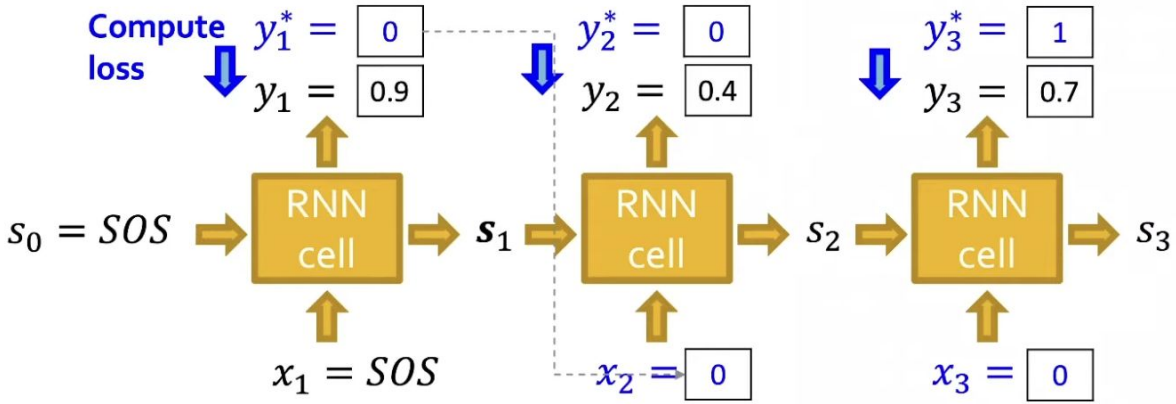
- y_t is a **scalar**, following a **Bernoulli distribution**
- $\boxed{p}$ means value 1 has prob. p, value 0 has prob. $1-p$



RNN at Training Time

Training the model:

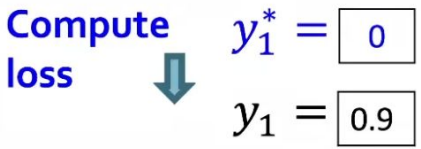
- We observe a sequence y^* of edges [1,0,…]
- **Principle**: **Teacher Forcing** -- Replace input and output by the real sequence



RNN at Training Time

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

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



- 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!

GraphRNN Approach

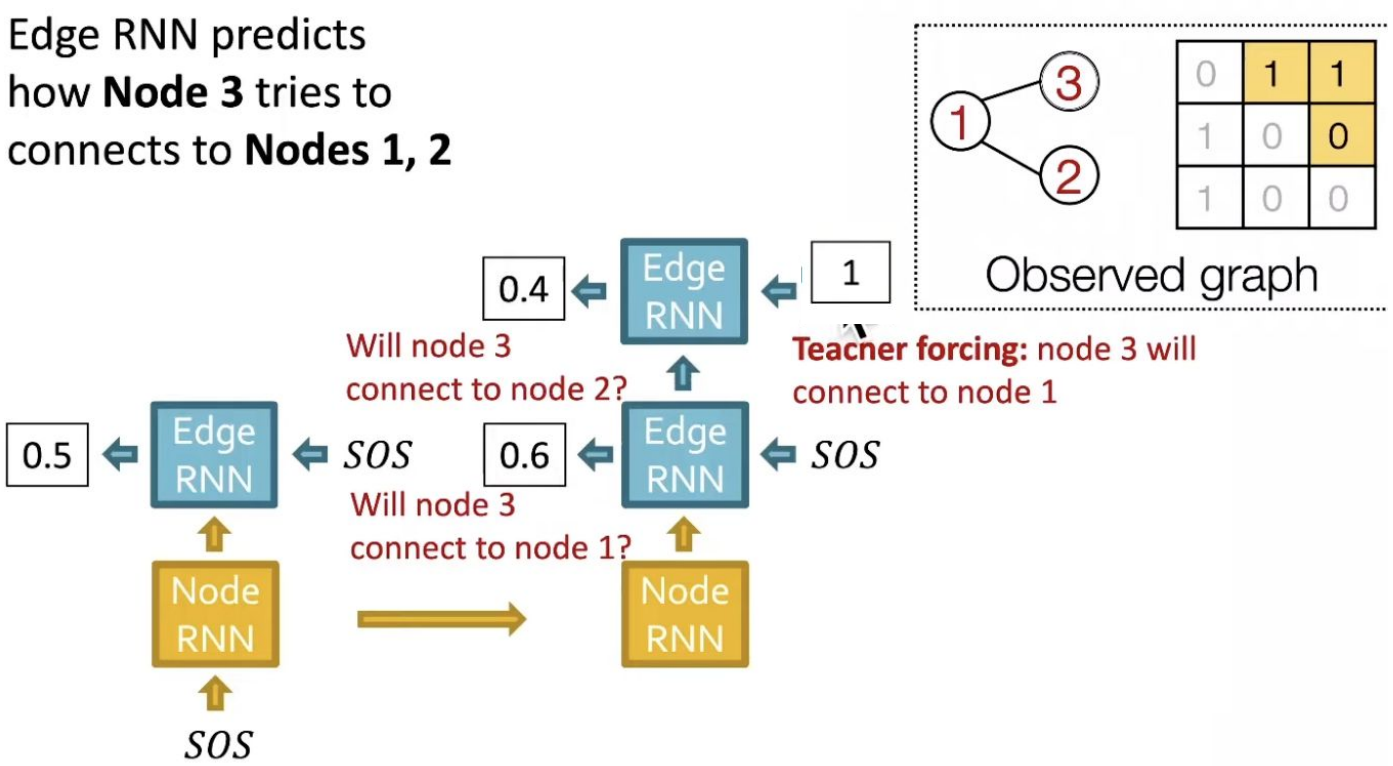
(1) **Add a new node:** We run Node RNN for a step, and use it output to initialize Edge RNN

(2) **Add new edges for the new node:** We run Edge RNN to predict if the new node will connect to each of the previous nodes

(3) **Add another new node:** We use the last hidden state of Edge RNN to run Node RNN for another step

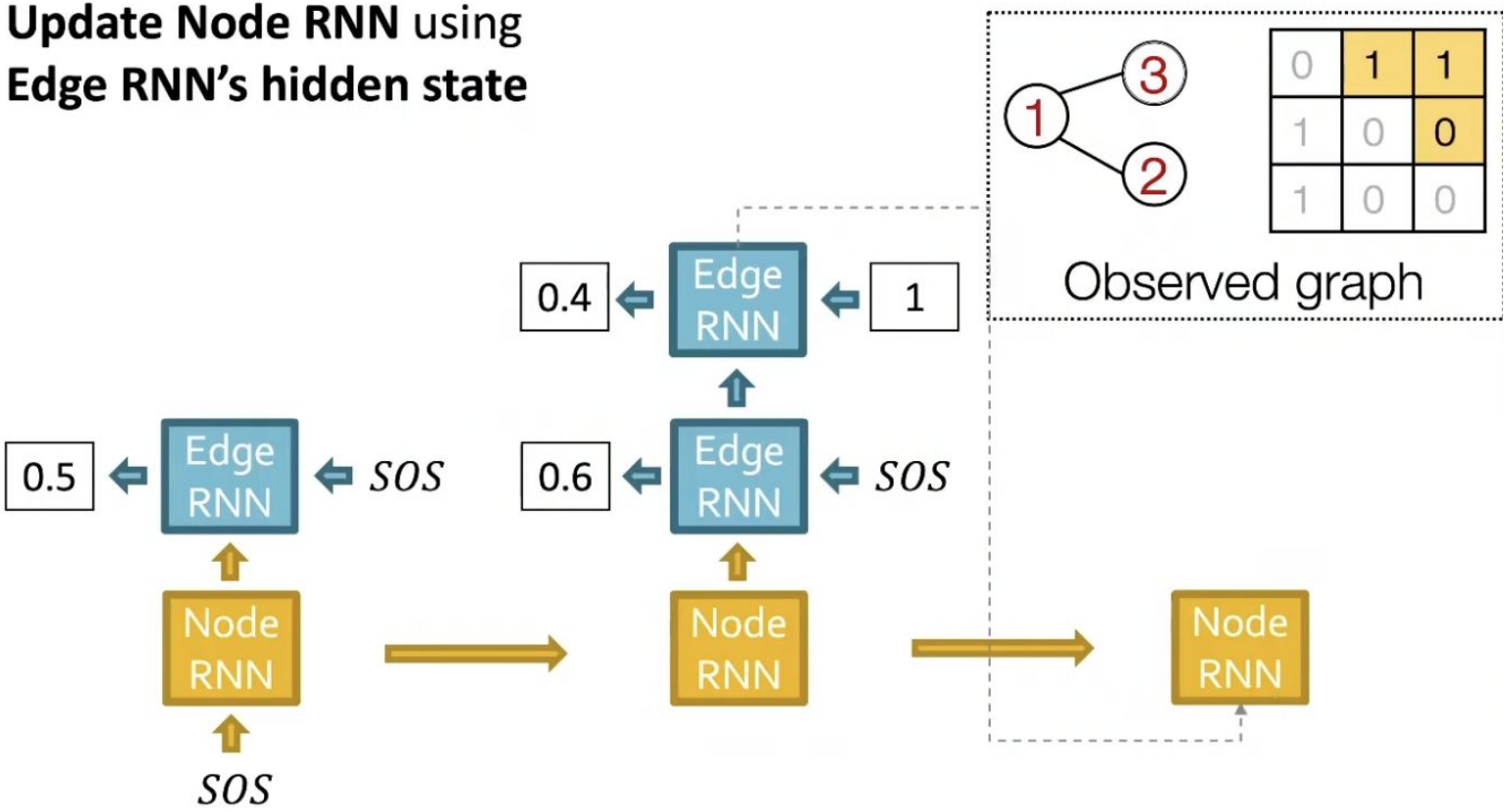
(4) **Stop graph generation:** If Edge RNN outputs EOS at step 1, we know no edges are connected to the new node. We stop the graph generation.

Training


Edge RNN predicts how **Node 3** tries to connects to **Nodes 1, 2**
3
1
2
0 1 1
1 0 0
1 0 0
Observed graph
0.4
Edge RNN
1
Will node 3 connect to node 2?
Teacher forcing: node 3 will connect to node 1
0.5
Edge RNN
SOS
0.6
Edge RNN
SOS
Will node 3 connect to node 1?
Node RNN
Node RNN
SOS

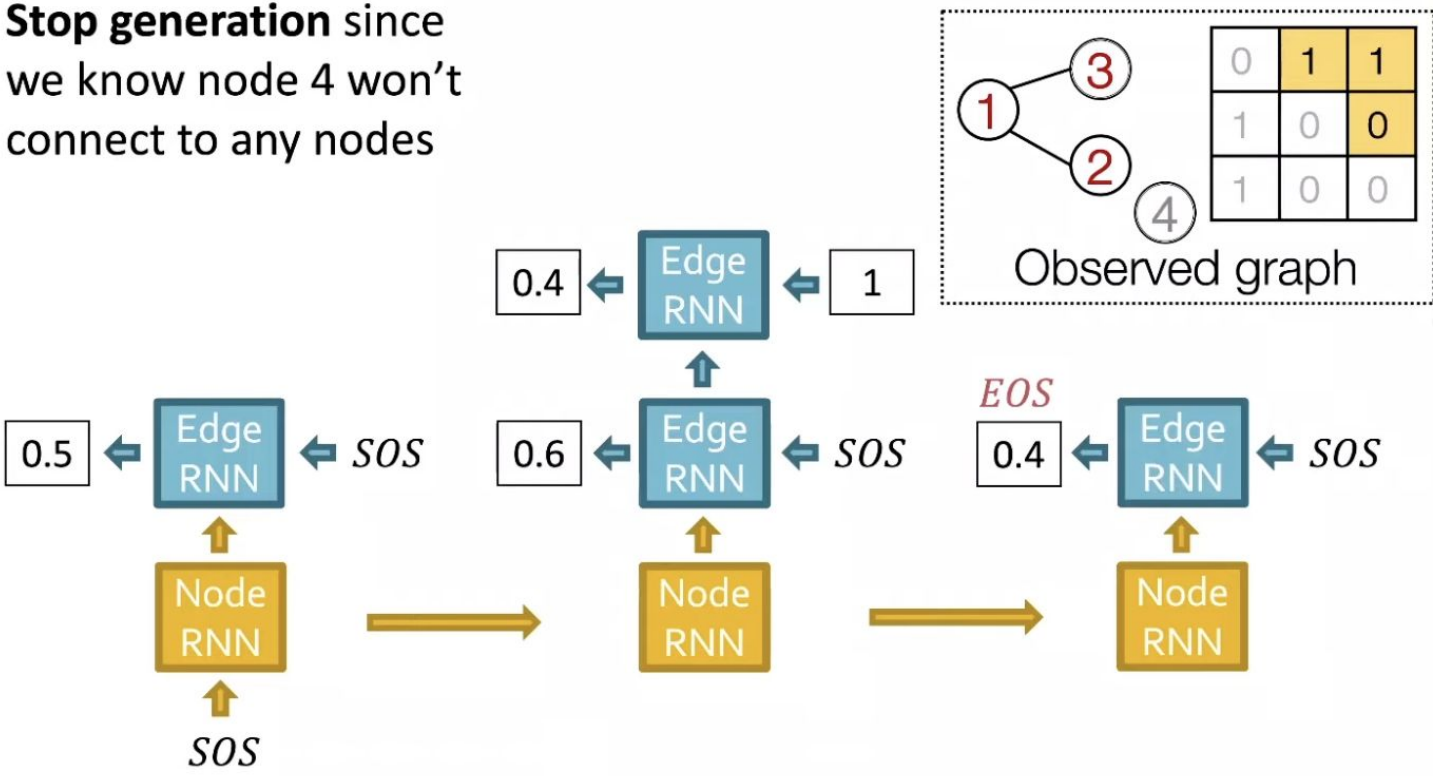
Training

Update Node RNN using
Edge RNN's hidden state

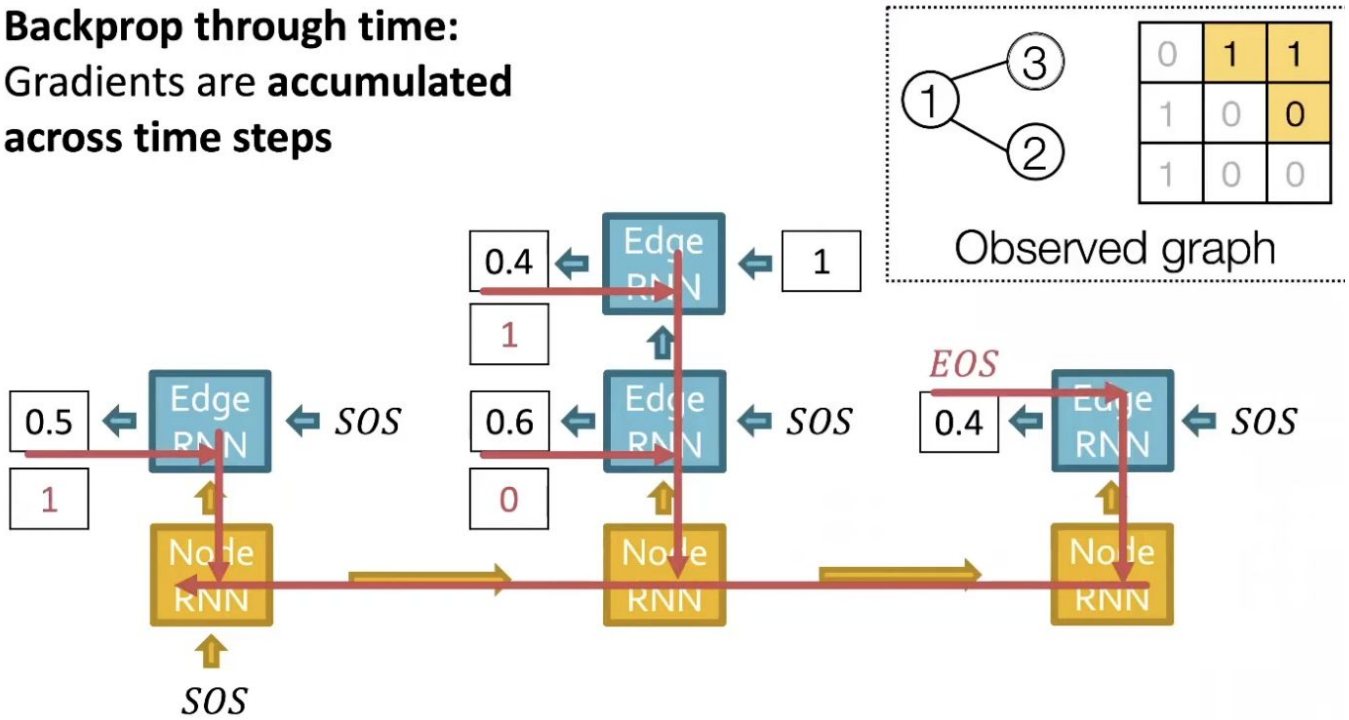


Training

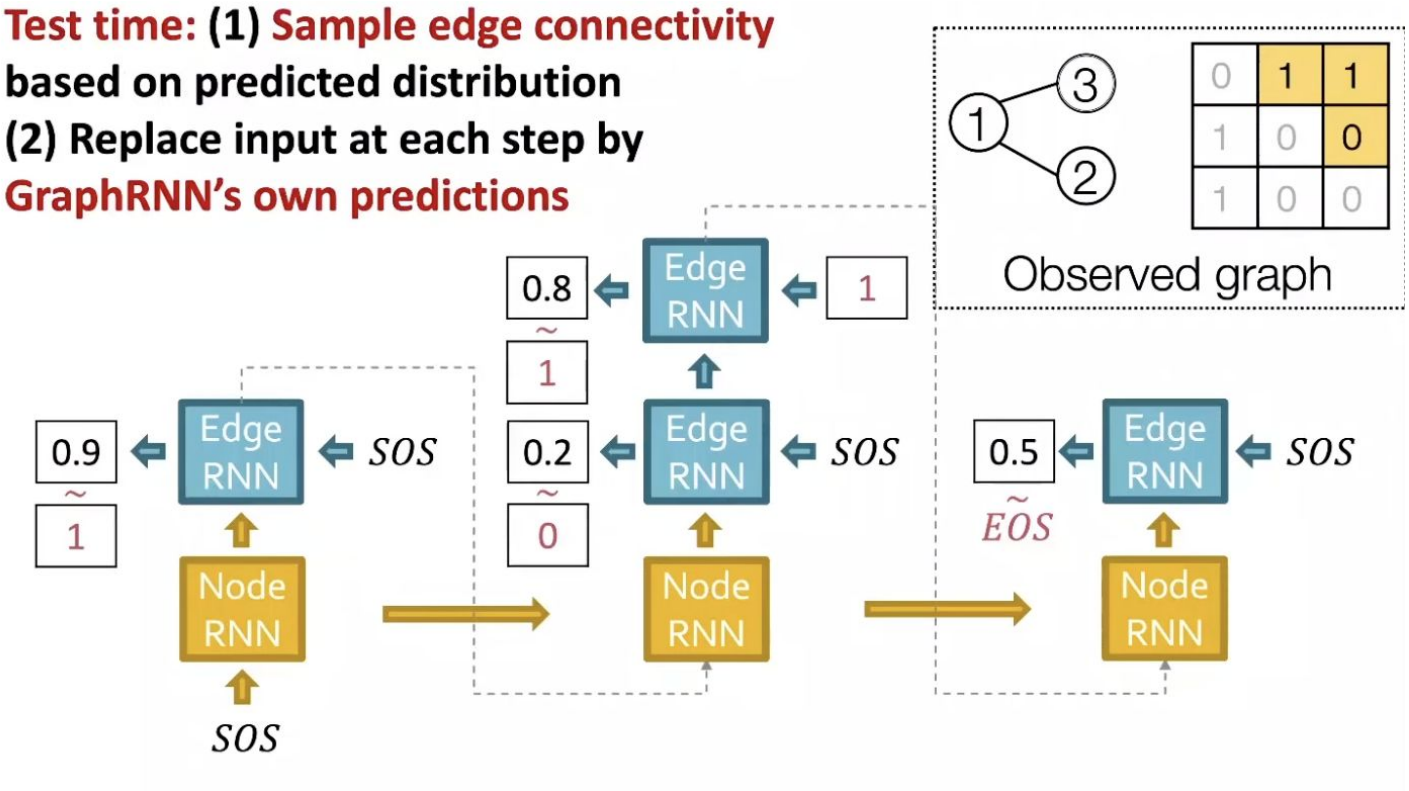
Stop generation since we know node 4 won't connect to any nodes



Backprop


Backprop through time:
Gradients are accumulated across time steps
1
2
3
0 1 1
1 0 0
1 0 0
Observed graph
0.4
1
1
Edge RNN
0.5
1
Edge RNN
SOS
0.6
0
Edge RNN
SOS
EOS
0.4
Edge RNN
SOS
Node RNN
Node RNN
Node RNN
SOS

Test


Test time: (1) Sample edge connectivity
based on predicted distribution
(2) Replace input at each step by
GraphRNN's own predictions
1
2
3
0 1 1
1 0 0
1 0 0
Observed graph
Edge RNN
Node RNN
SOS
0.9
1
0.8
1
0.2
0
0.5
EOS

Major Challenge

Now we can generate graphs by sampling from a distribution learned by our model.

Since any node can connect to any prior node, we need to generate half of the adjacency matrix which can be extremely inefficient due to the problem of quadratic explosion.

To tackle this, generate the node sequence in a BFS manner. This reduces the possible node orderings from ***O(n!)*** to a comparatively small distinct BFS orderings