Summary Questions of the lecture

- Describe the key aspects of Graph Diffusion-Embedding Networks.
- GDEN formulates three difusion operators $\mathcal{F}_d(A, H)$ from RWR, LapReg, and NLapReg. In particular, LapReg, and NLapReg are formulated from the Laplacian smoothing and regularization for personalized teleport. The graph diffusion operator is used for message passing (or aggregation) in GDEN and the embedding is conducted by the normal convolution on the aggregated graph signals.

Model	Diffusion operator $\mathcal{F}_d(\mathbf{A}, \mathbf{H})$
RWR Eq.(9)	$(1-\lambda)(\mathbf{I}-\lambda\mathbf{A}\mathbf{D}^{-1})^{-1}\mathbf{H}$
LapReg Eq.(11)	$\lambda(\mathbf{D}-\mathbf{A}+\lambda\mathbf{I})^{-1}\mathbf{H}$
NLapReg Eq.(13)	$(1-\gamma)(\mathbf{I}-\gamma\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})^{-1}\mathbf{H}$

$$\mathbf{F}^{(l)} = \mathcal{F}_d(\mathbf{A}, \mathbf{H}^{(l)})$$
$$\mathbf{H}^{(l+1)} = \sigma(\mathbf{F}^{(l)}\mathbf{W}^{(l)})$$

$$\mathbf{f}_i^{(t+1)} = \lambda \sum\nolimits_{j,j\neq i} \mathbf{P}_{ij}\mathbf{f}_j^{(t)} + (1-\lambda)\mathbf{h}_i$$

$$\min_{\mathbf{F}} \ \frac{1}{2}\sum_{i,j=1}^{n} \mathbf{A}_{ij}\|\mathbf{f}_i - \mathbf{f}_j\|_2^2 + \lambda\sum_{i=1}^{n}\|\mathbf{f}_i - \mathbf{h}_i\|_2^2$$

$$\min_{\mathbf{F}} \ \frac{1}{2}\sum_{i,j=1}^{n} \mathbf{A}_{ij}\|\frac{\mathbf{f}_i}{\sqrt{\mathbf{d}_i}} - \frac{\mathbf{f}_j}{\sqrt{\mathbf{d}_j}}\|_2^2 + \lambda\sum_{i=1}^{n}\|\mathbf{f}_i - \mathbf{h}_i\|_2^2$$

Summary Questions of the lecture

- Describe the key aspects of Graph Diffusion Convolution(GDC).
- GDC genralizes graph diffusion by formulating it as an infinite weighted sum of the powers of the transition probability matrix. The output of this formulation, S, is sparsified by extracting salient connections from each column (e.g. top$-k$, thresholding) and normalizing. The resulting matrix is column stochastic and can be used with various graph convolution methods.

Generalized graph diffusion

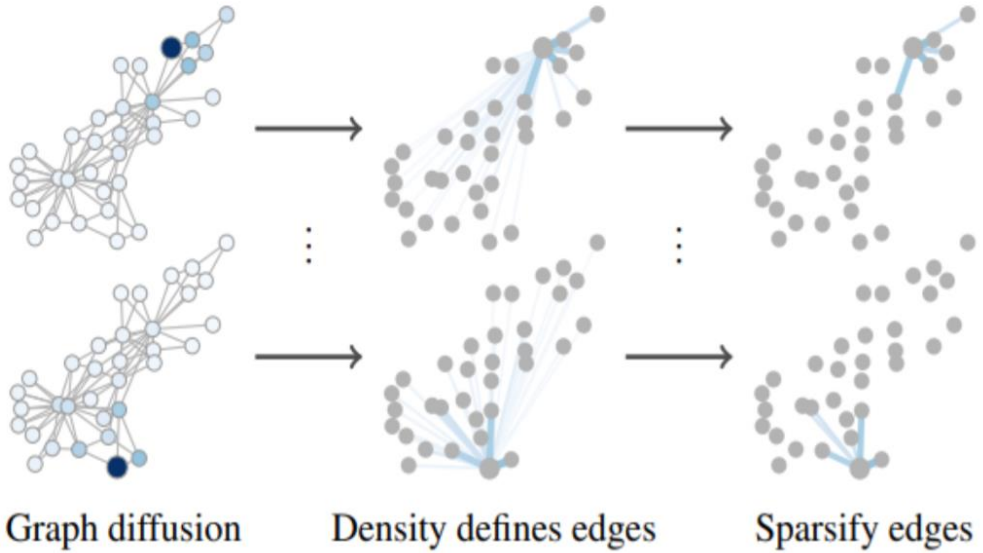
$$S = \sum_{k=0}^{\infty} \theta_k P^k ,$$

Sparsification is done additionally for $\tilde{S}$

Normalize (symmetric)

$$T_{sym}^{\tilde{S}} = D_{\tilde{S}}^{-1/2} \tilde{S} D_{\tilde{S}}^{-1/2}$$

GCN (ShevNet): $\boldsymbol{F^l} = T_{sym}^{\tilde{S}} \boldsymbol{H^l}$, $\boldsymbol{H^{l+1}} = \boldsymbol{\sigma}(\boldsymbol{F^l \Theta})$

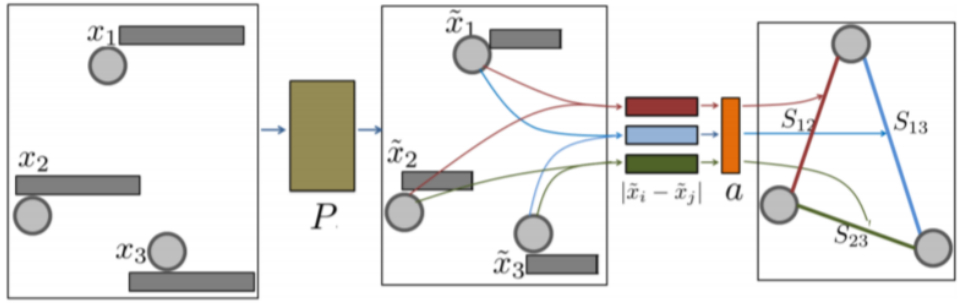


Summary Questions of the lecture

- Describe the key aspects of Graph Learning-Convolutional Networks.
- For SSL, GLCN first predicts the link between nodes and then performs graph convolution. Node connectivity values are assigned similarly as in GAT and trained such that high values are assigned between nodes that have similar features. This property is forced by the graph learning loss proposed in the paper.

$$S_{ij} = g(x_i, x_j) = \frac{A_{ij} \exp(\text{ReLU}(a^T |x_i - x_j|))}{\sum_{j=1}^{n} A_{ij} \exp(\text{ReLU}(a^T |x_i - x_j|))}$$

(when A is not available, $A_{ij} = 1$)



Graph Learning Loss:

$$\mathcal{L}_{\text{GL}} = \sum_{i,j=1}^{n} \|x_i - x_j\|_2^2 S_{ij} + \gamma \|S\|_F^2 + \beta \|S - A\|_F^2$$

a is trained

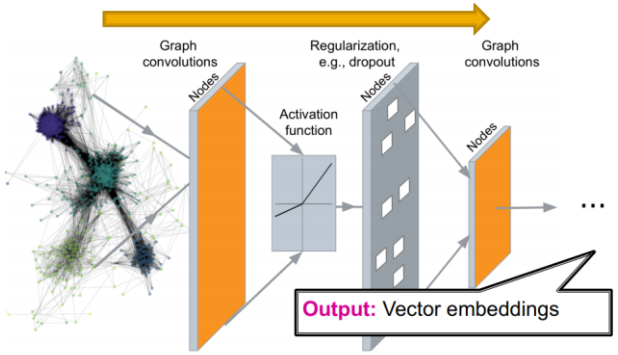
Outline of Lecture (5)

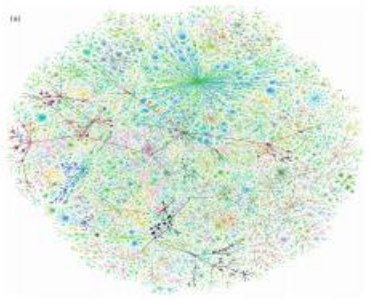
- Random Walks and Diffusion
- Diffusion in GCN
  - Propagation using graph diffusion
    - APPNP: Predict Then Propagate [ICLR'19]
    - Graph Diffusion-Embedding Networks [CVPR'19]
  - Making a new graph
    - Diffusion Improves Graph Learning [NIPS'19]
    - SSL with Graph Learning-Convolutional Networks [CVPR'19]

- Deep Generative Models For Graph
  - Problem of Graph Generation
  - ML Basics for Graph Generation
  - GraphRNN : Generating Realistic Graphs
  - Applications and Open Questions

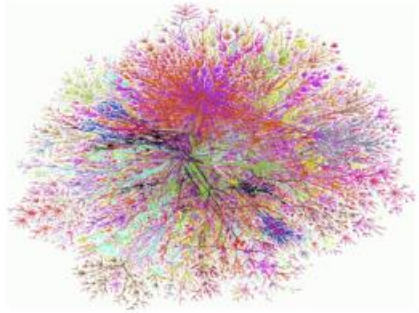
DGMG:
Deep Generative Models For Graph

Deep Graph Encoder, Graph Generation

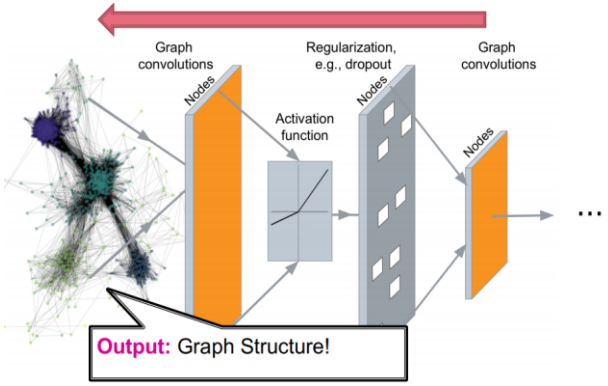




Given a large real graph

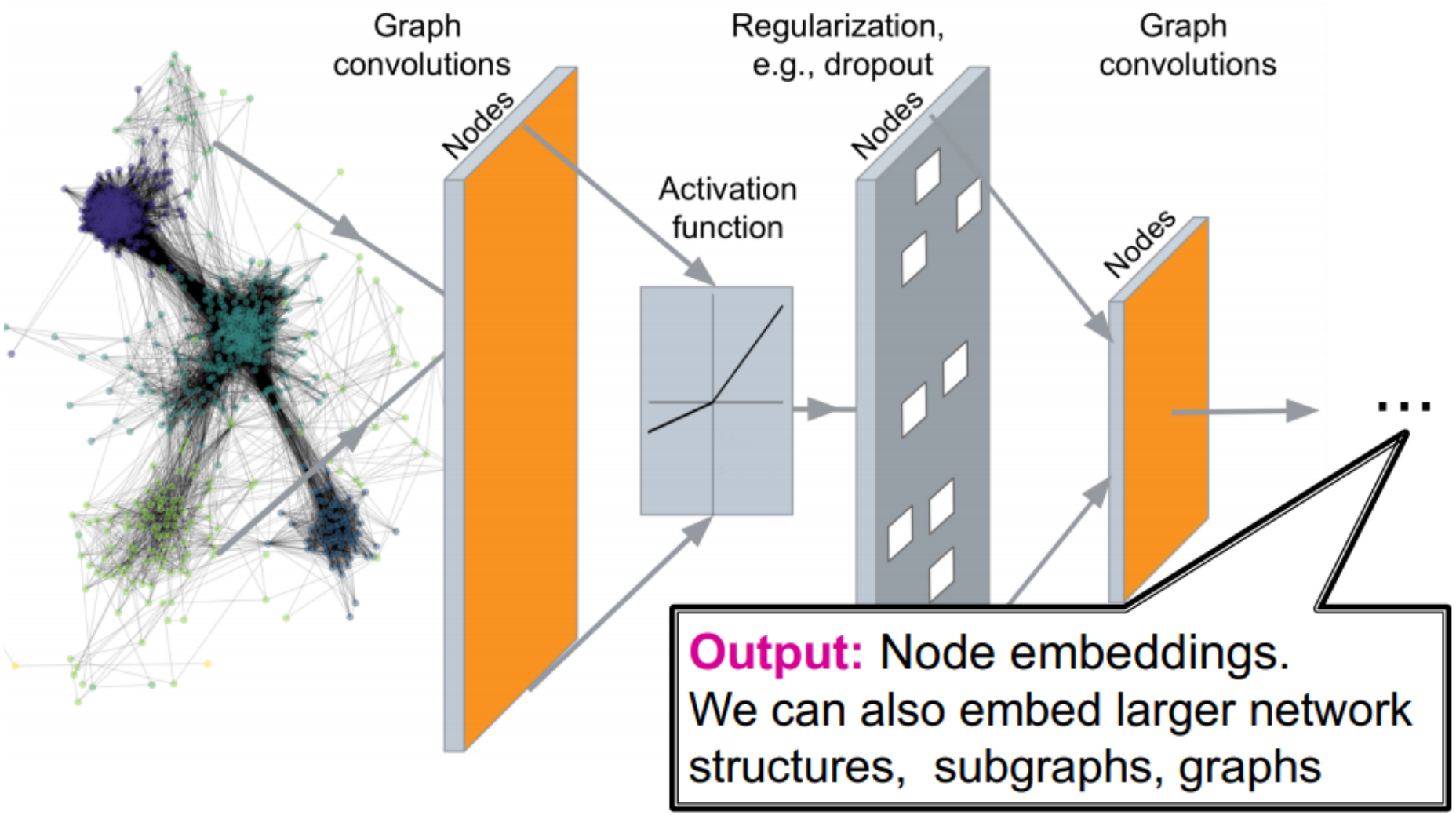


Generate a synthetic graph



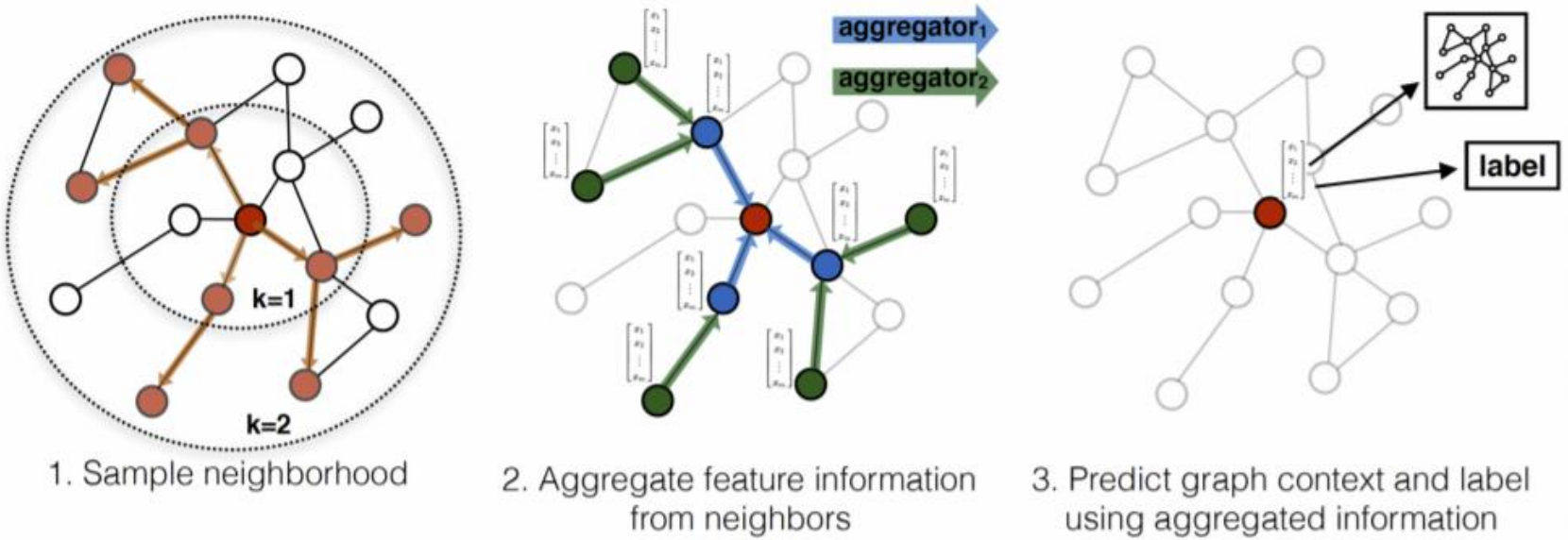
Machine Learning with Graphs, Jurij Leskovec, Stanford University
Efficient Graph Generation with Graph Recurrent Attention Networks, NeurIPS 2019 (서성욱 발표)

Deep Graph Encoders



Deep Graph Encoders

GraphSAGE



1. Sample neighborhood

2. Aggregate feature information from neighbors

3. Predict graph context and label using aggregated information

$$\boldsymbol{h}_{N_s(v_i)}^{(l+1)} = \boldsymbol{AGG}\left(\left\{\boldsymbol{h}_j^{(l)}, v_j \in N_s(v_i)\right\}\right)$$

$$\boldsymbol{h}_i^{(l+1)} = \boldsymbol{\sigma}\left(\boldsymbol{\Theta} \cdot \left[\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_{N_s(v_i)}^{(l+1)}\right]\right)$$

- Mean aggregator
- LSTM aggregator
- Pooling aggregator

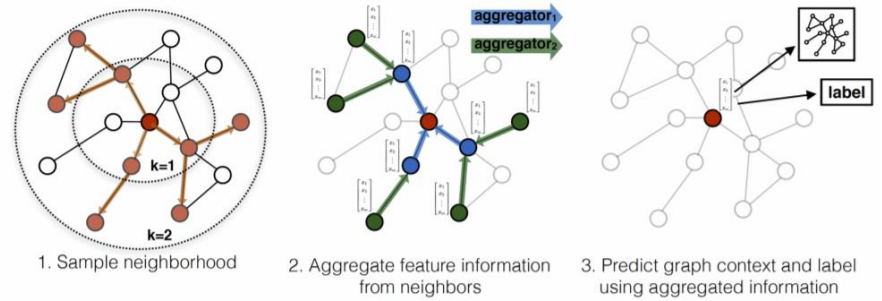
Deep Graph Encoders

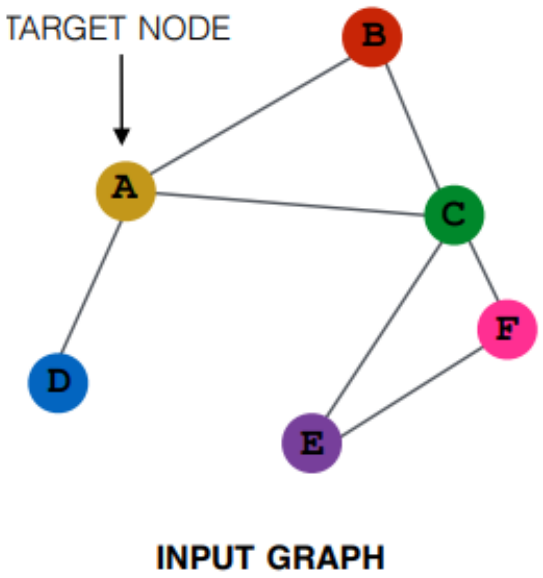
GraphSAGE

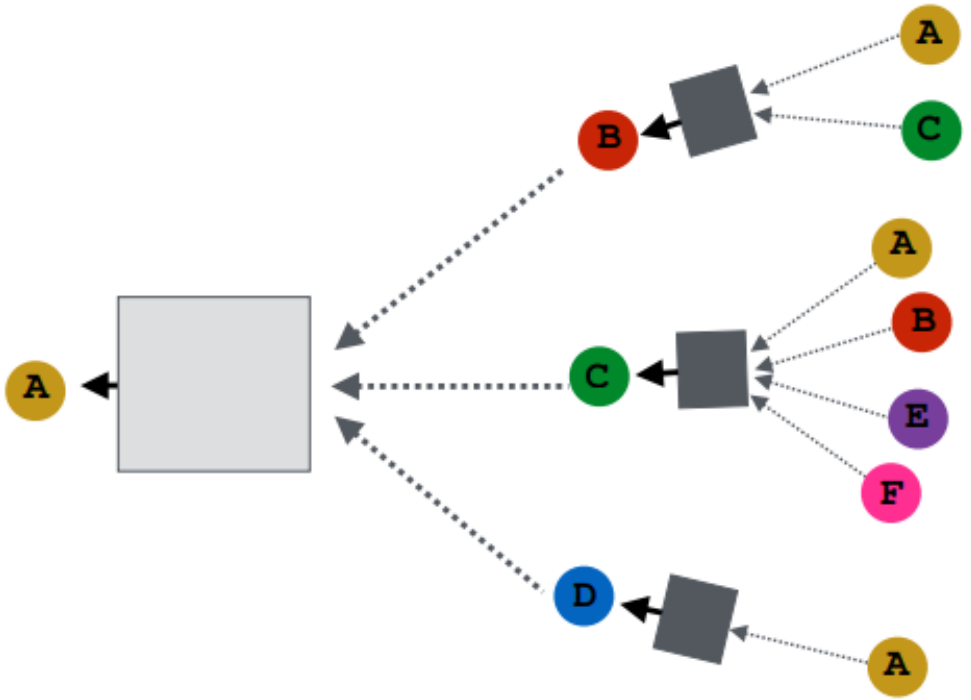
$$\boldsymbol{h}_{N_S(v_i)}^{(l+1)} = \boldsymbol{AGG}\left(\left\{\boldsymbol{h}_j^{(l)}, v_j \in N_S(v_i)\right\}\right)$$

$$\boldsymbol{h}_i^{(l+1)} = \boldsymbol{\sigma}\left(\boldsymbol{\Theta} \cdot \left[\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_{N_S(v_i)}^{(l+1)}\right]\right)$$

- Mean aggregator
- LSTM aggregator
- Pooling aggregator







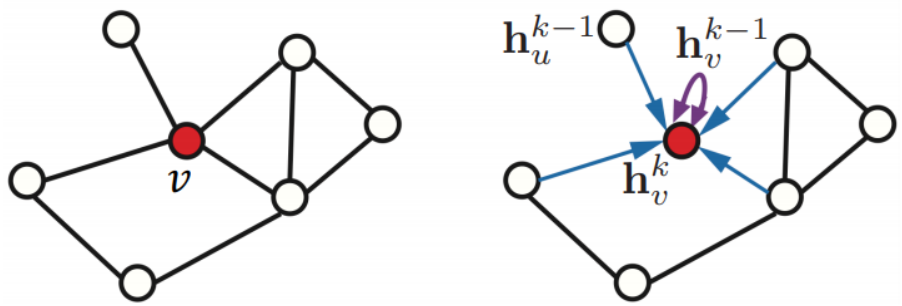
Deep Graph Encoders

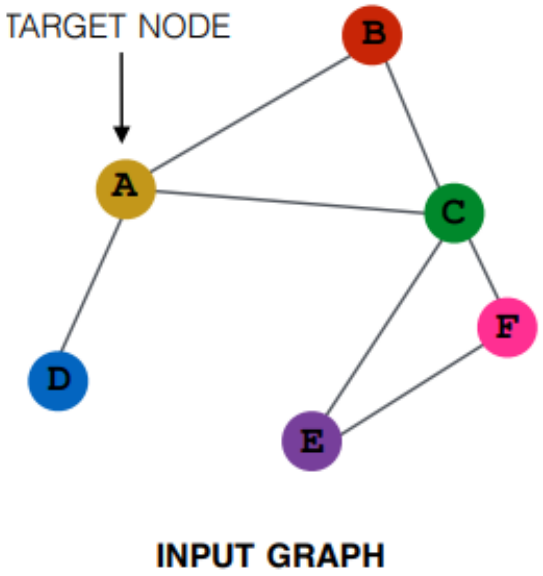
GraphSAGE

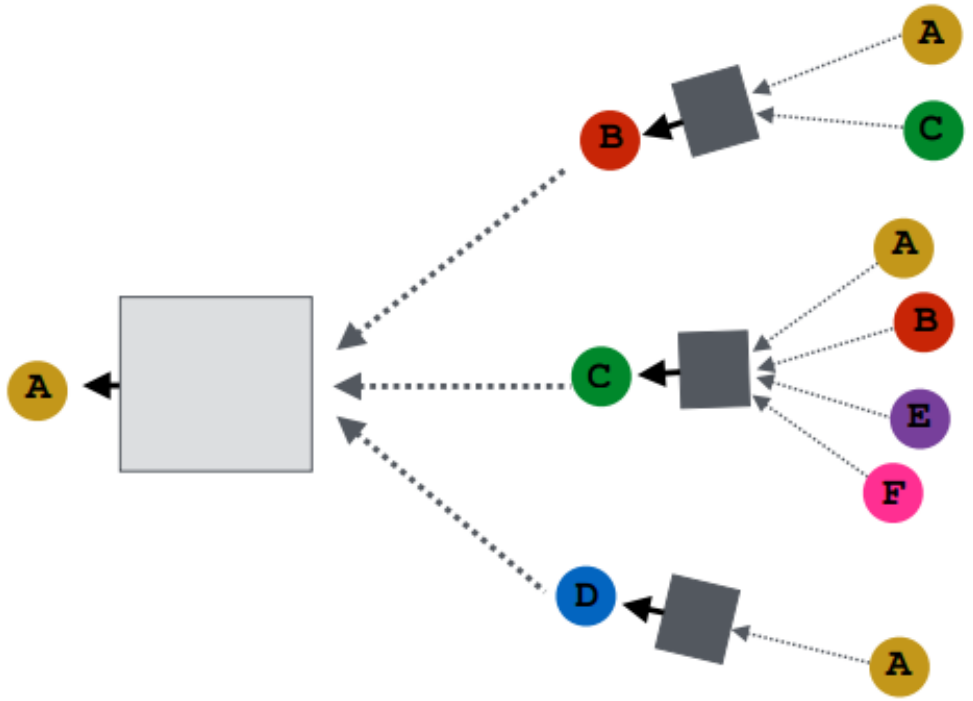
$$\boldsymbol{h}_{N_s(v_i)}^{(l+1)} = \boldsymbol{AGG}\left(\left\{\boldsymbol{h}_j^{(l)}, v_j \in N_s(v_i)\right\}\right)$$

$$\boldsymbol{h}_i^{(l+1)} = \boldsymbol{\sigma}\left(\boldsymbol{\Theta} \cdot \left[\boldsymbol{h}_i^{(l)}, \boldsymbol{h}_{N_s(v_i)}^{(l+1)}\right]\right)$$

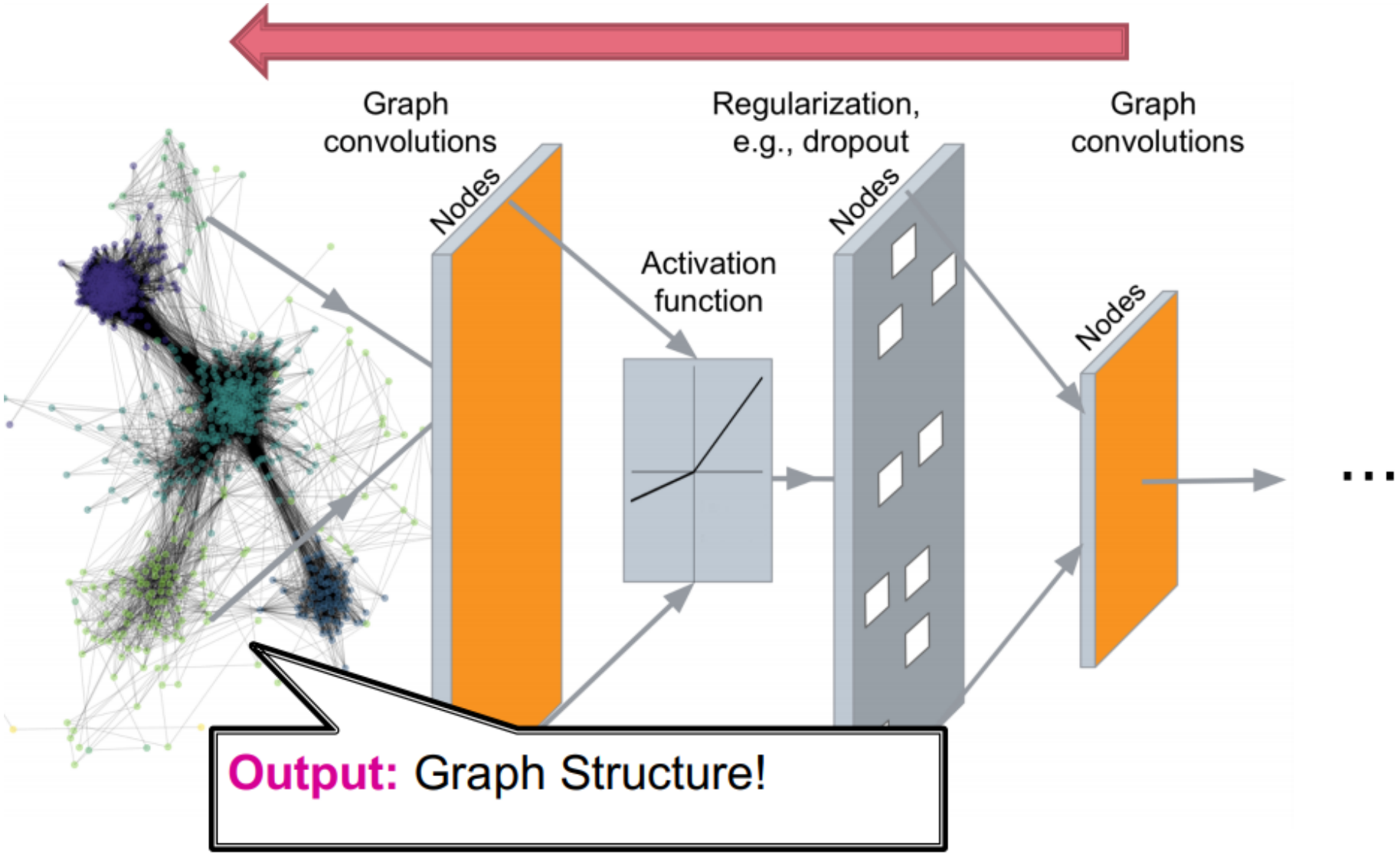
- Mean aggregator
- LSTM aggregator
- Pooling aggregator







Deep Graph Decoders



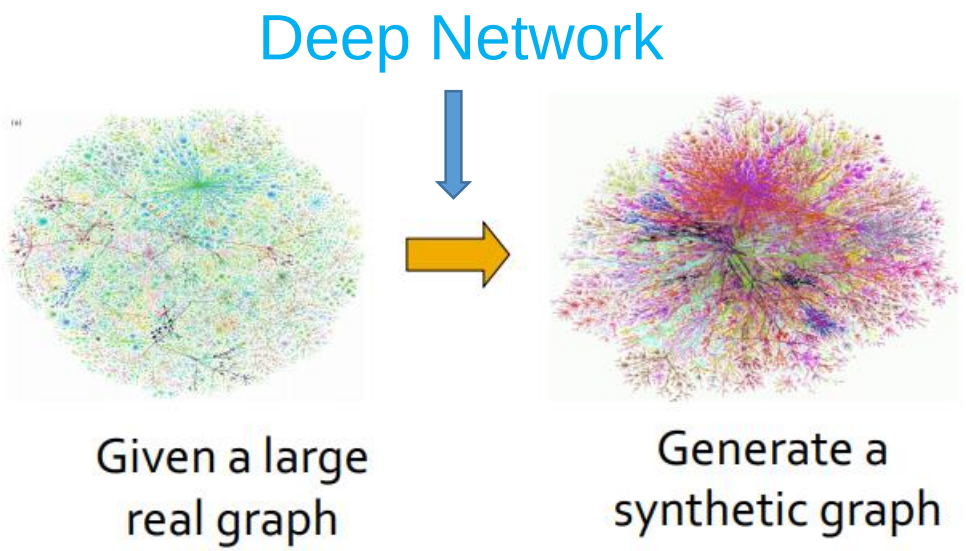
The Problem of Graph Generation

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

The Problem of Graph Generation

We want to generate realistic graphs.



What is a good model?

How can we fit the model and generate the graph using it?

Graph Generation Tasks

Task 1: Realistic graph generation

- Generate graphs that are similar to a given set of graphs
  - Focus of this lecture (Auto-Regressive Model based on RNN)

Task 2: Goal-directed graph generation

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

Deep Graph Decoders

Outline of Contents

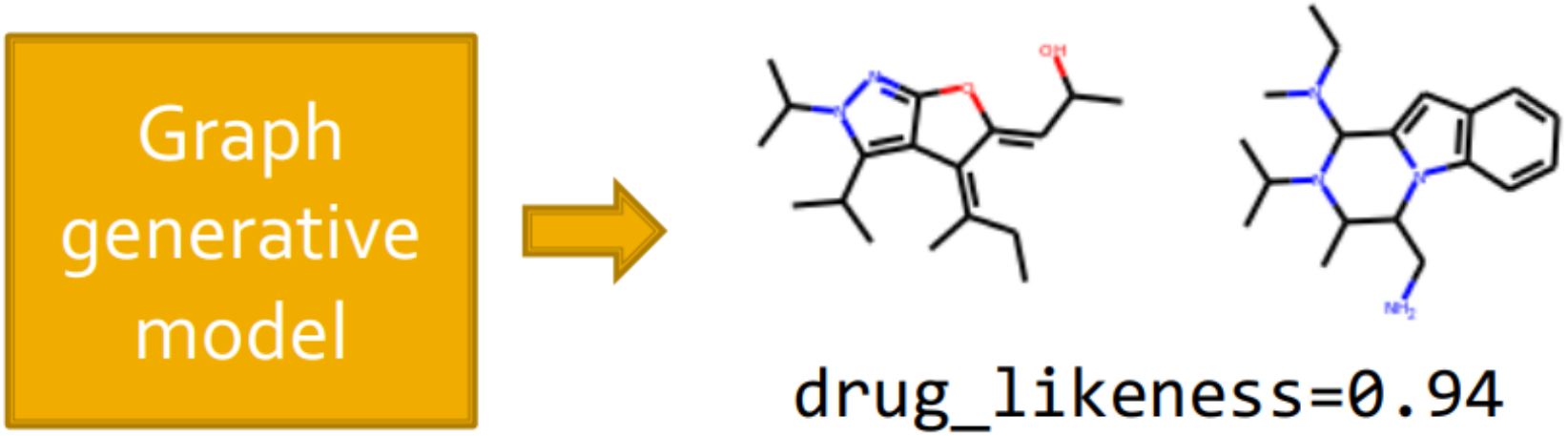
- Problem of Graph Generation
- ML Basics for Graph Generation
- GraphRNN
- Applications and Open Questions

Machine Learning with Graphs, Jurij Leskovec, Stanford University
Efficient Graph Generation with Graph Recurrent Attention Networks, NeurIPS 2019 (서성욱 발표)

Graph Generation Tasks (GGT)

Why is GGT interesting?

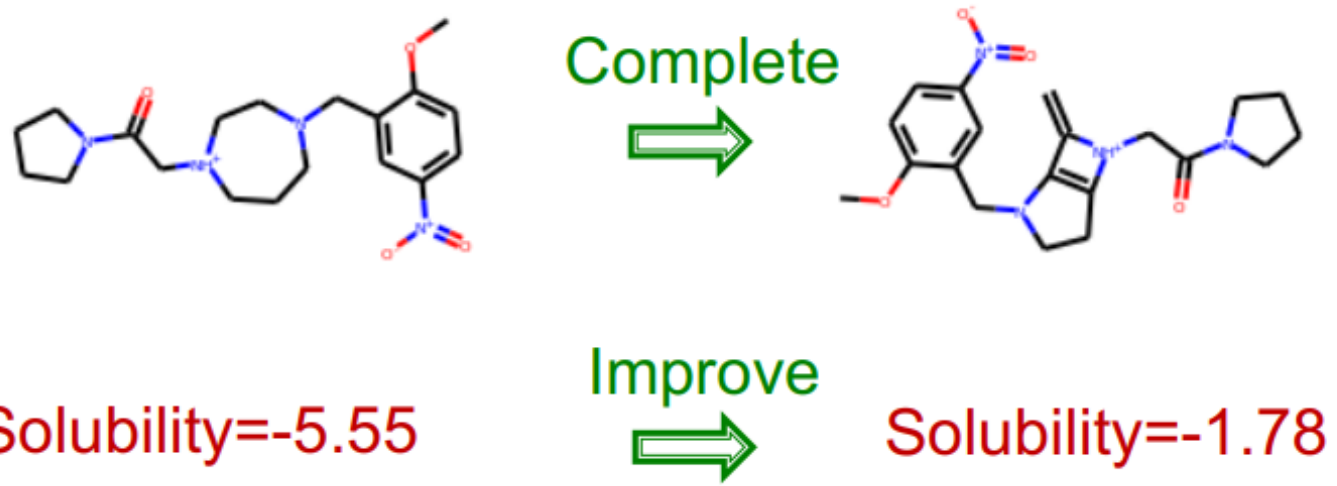
- Drug discovery
  - Discover highly drug-like molecules or novel structures



Graph Generation Tasks (GGT)

Why is GGT interesting?

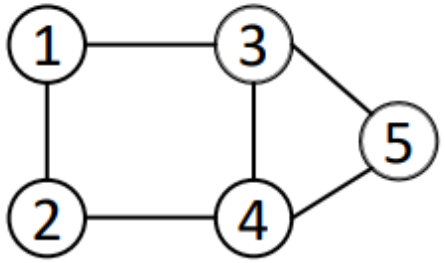
- Drug discovery
  - Complete an existing molecule to optimize a desired property



Graph Generation Tasks (GGT)

Why is GGT 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

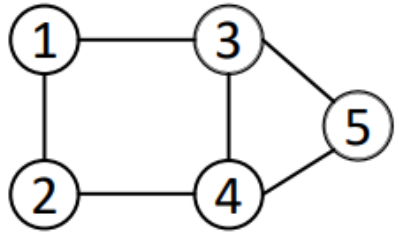
5 nodes: 25 values

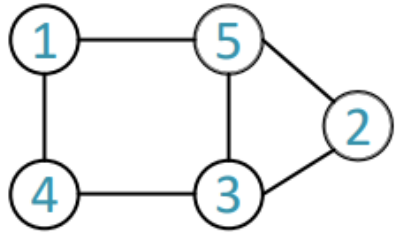
1K nodes: 1M values

Graph Generation Tasks (GGT)

Why is GGT 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

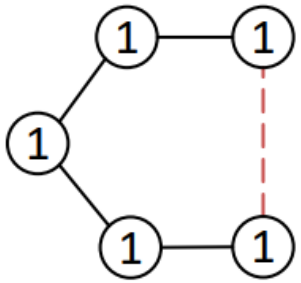
Very different representations!

Graph Generation Tasks (GGT)

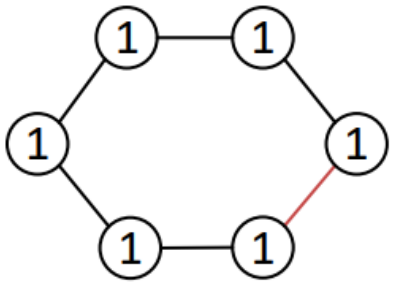
Why is GGT hard?

- 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!

Deep Graph Decoders

Outline of Contents

- Problem of Graph Generation
- ML Basics for Graph Generation
- GraphRNN
- Applications and Open Questions

Graph Generative Models

Given: Graphs sampled from $P_{data}(G)$

Goal:

- Learn the distribution $P_{model}(G|\Theta)$
- Sample from $P_{model}(G|\Theta)$

ML: 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 $\boldsymbol{x}_i \sim P_{data}(\boldsymbol{x})$
  - $P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$ is the model, parametrized by $\boldsymbol{\theta}$, that we use to approximate $P_{data}(\boldsymbol{x})$

Goal:

- Make $P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$ close to $P_{data}(\boldsymbol{x})$
- Make sure we can sample from $P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$
  - We need to generate examples (graphs) from $P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$

ML: Generative Models

Make $P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$ close to $P_{data}(\boldsymbol{x})$

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

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{\boldsymbol{x}\sim P_{data}(\boldsymbol{x})} \log\, P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$$

- Find parameters $\boldsymbol{\theta}^*$, such that for observed data points $\boldsymbol{x}_i \sim P_{data}(\boldsymbol{x})$

  $\sum_i \log\, P_{model}(\boldsymbol{x}_i|\boldsymbol{\theta})$ has the highest value, among all possible choices of $\boldsymbol{\theta}$

  - That is, find the model that is most likely to have generated the observed data $\boldsymbol{x}$

ML: Generative Models

Sample from $P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$

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

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

  - Transform the noise z_i via $f(\cdot)$ to a sample $\boldsymbol{x}_i$

$$\boldsymbol{x}_i = f(z_i; \theta)$$

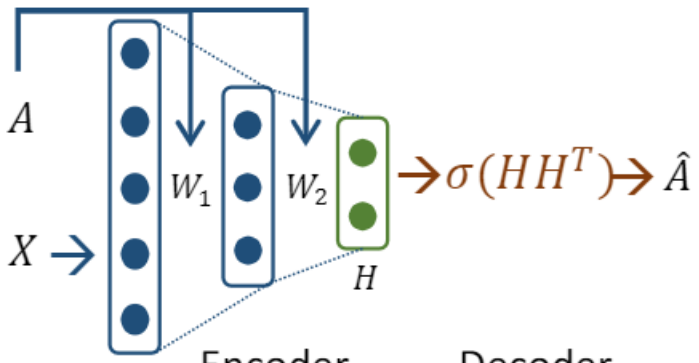
- **Q:** How to design $f(\cdot)$?
- **A:** Use Deep Neural Networks, and train it using the data we have!

ML: Generative Models

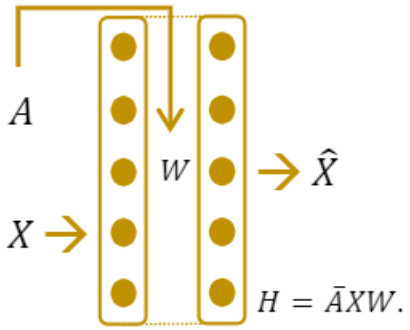
Types of Deep Generative Models

- **GAN:** Generative Adversarial Networks
- **VAE:** Variational Auto-Encoder
- **BM:** Boltzman Machine (Markov Chain)
- **AR:** Auto-Regressive Model (←This lecture)
  - AR model predicts future behavior based on past behavior
  - Node increases one by one.

ML: Generative Models(VAE)

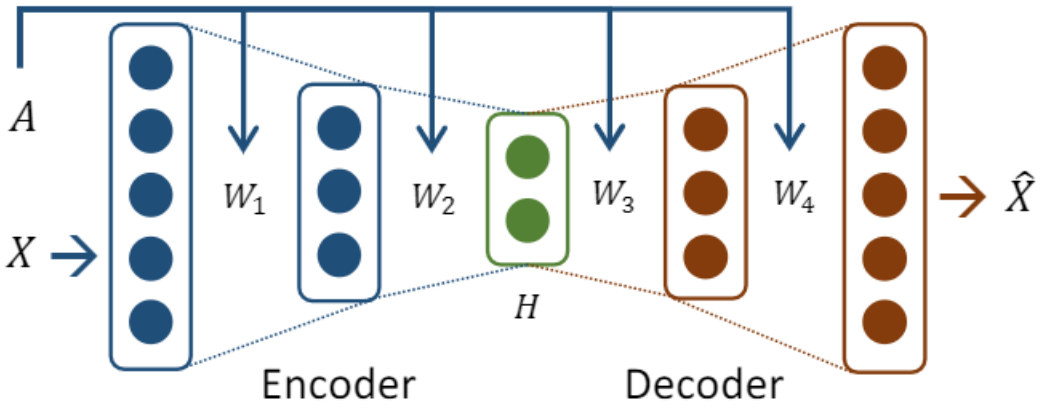


(a) VGAE [13]



(b) MGAE [35]

[Symmetric Graph Convolutional Autoencoder for Unsupervised Graph Representation Learning](), ICCV 2019

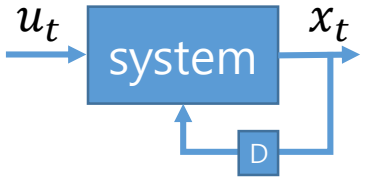


(c) Proposed autoencoder

Graph Smoothing (Encoding)
Graph Sharpening (Decoding)

ML: Generative Models

- **AR:** Auto-Regressive Model (←This lecture)
  - AR model predicts future behavior based on past behavior
  - Node increases one by one.
  - $x_t = \beta_0 u_t + \beta_1 u_{t-1} + \cdots + \beta_m u_{t-m}$ Moving-Average(AR) Model
  - $x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \cdots + \alpha_n x_{t-n}$ Auto-Regressive(MA) Model
  - $x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \cdots + \alpha_n x_{t-n} + \beta_0 u_t + \beta_1 u_{t-1} + \cdots + \beta_m u_{t-m}$ ARMA Model



ML: Generative Models

AR: Auto-Regressive Model

- $P_{model}(\boldsymbol{x}|\boldsymbol{\theta})$ is used for both density estimation and sampling (from the probability density)
- Apply chain rule: Joint distribution is a product of conditional distributions:

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

- E.g., $\boldsymbol{x}$ is a vector, x_t is the t-th element; $\boldsymbol{x}$ 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)

Deep Graph Decoders

Outline of Contents

- Problem of Graph Generation
- ML Basics for Graph Generation
- GraphRNN : Generating Realistic Graphs
- Applications and Open Questions

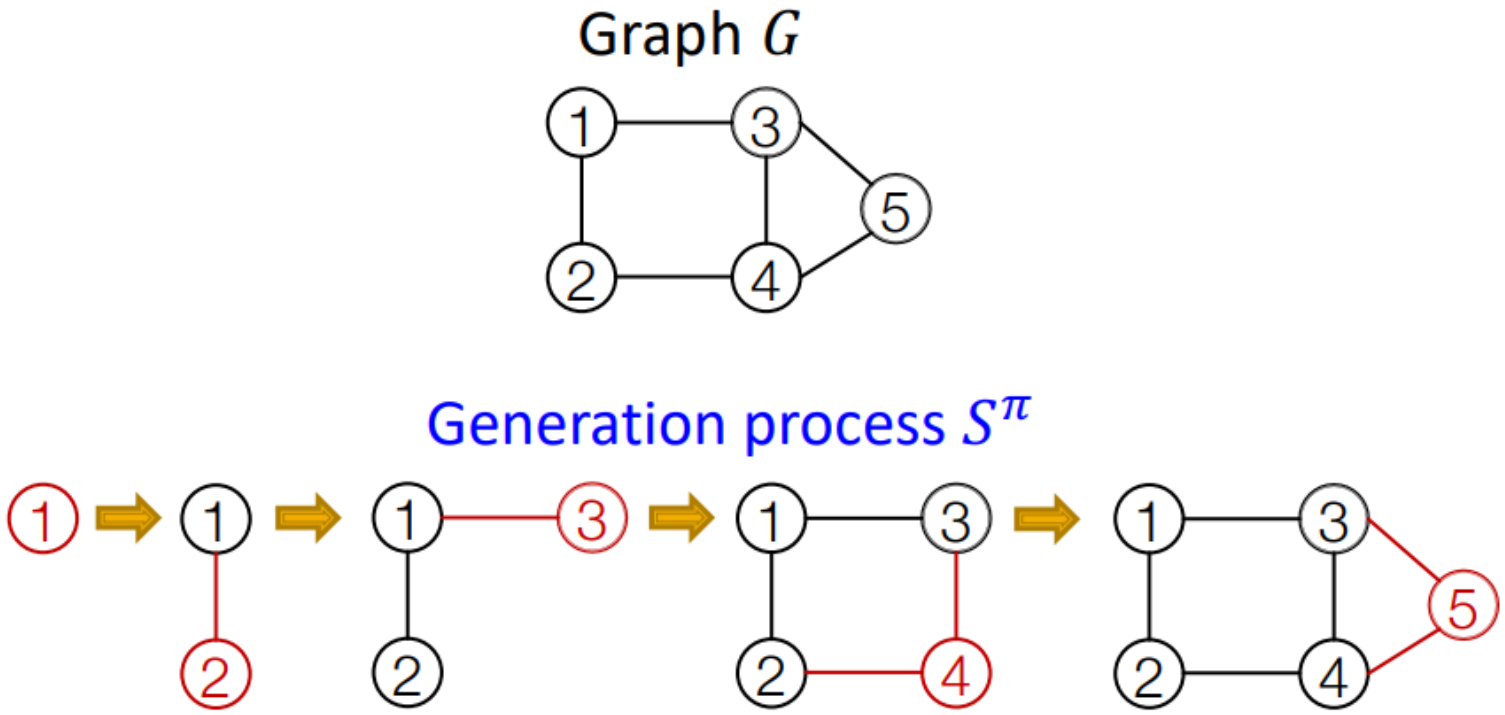
GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models

(J. You et al., ICML 2018)

GraphRNN

IDEA

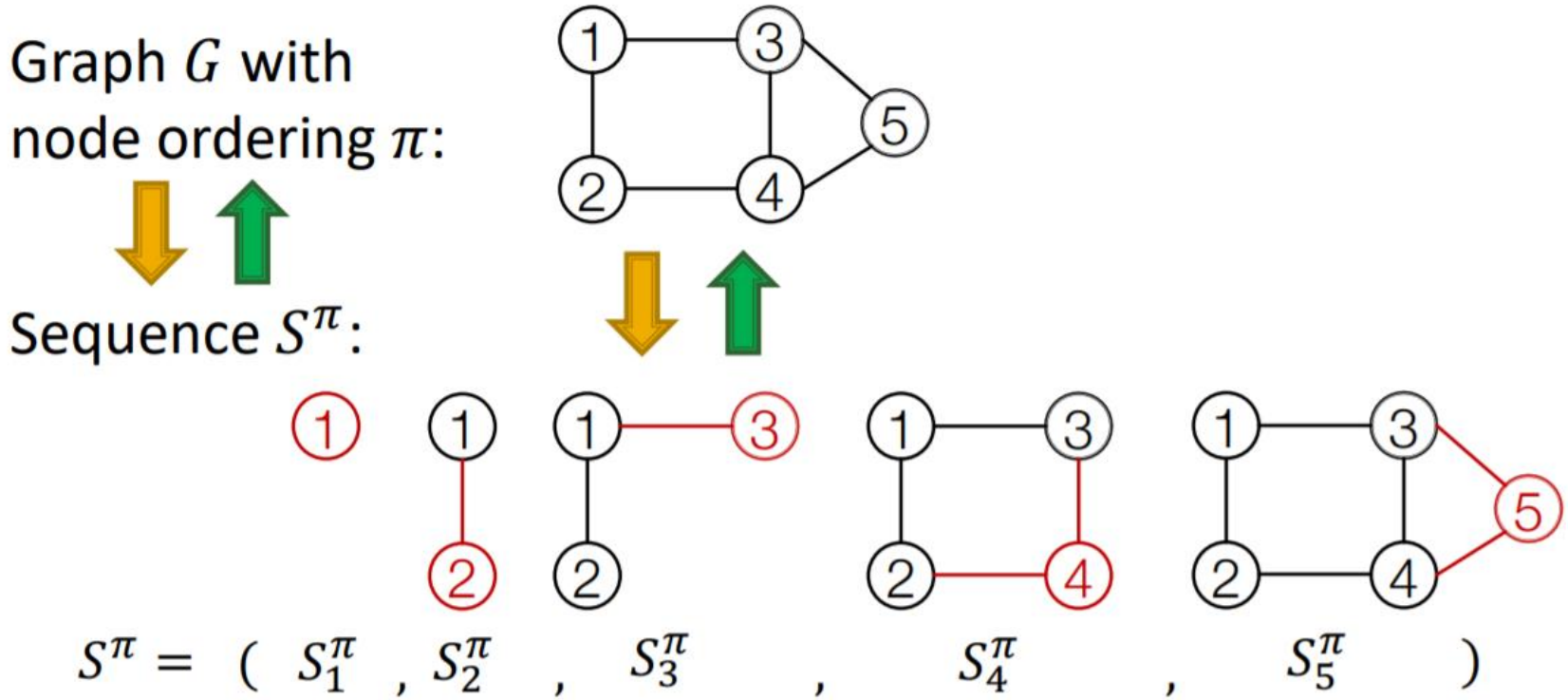
- Generating graphs via sequentially adding nodes and edges



GraphRNN

Model Graphs as Sequences

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

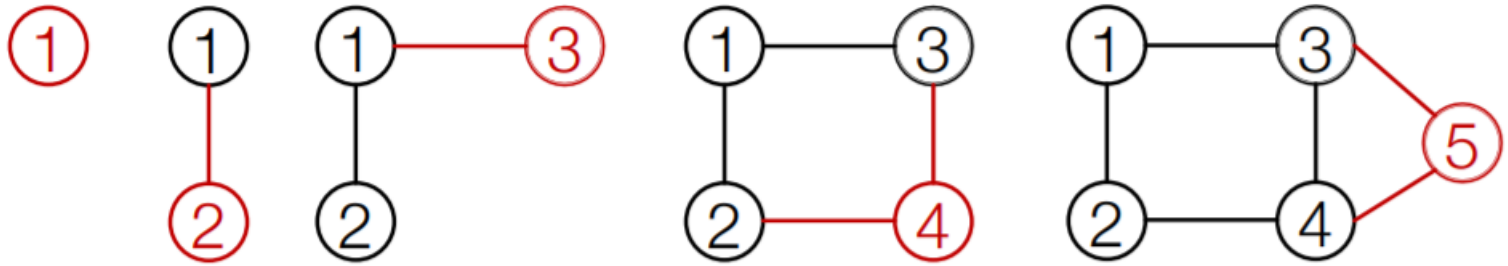


GraphRNN

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



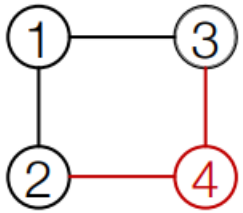
$$S^{\pi} = (S_1^{\pi}, S_2^{\pi}, S_3^{\pi}, S_4^{\pi}, S_5^{\pi})$$

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

GraphRNN

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

- Each Node-level step includes an edge-level sequence (multiple edges per each node)
- Edge-level: At each edge-level step, add a new edge



S_4^{π}

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

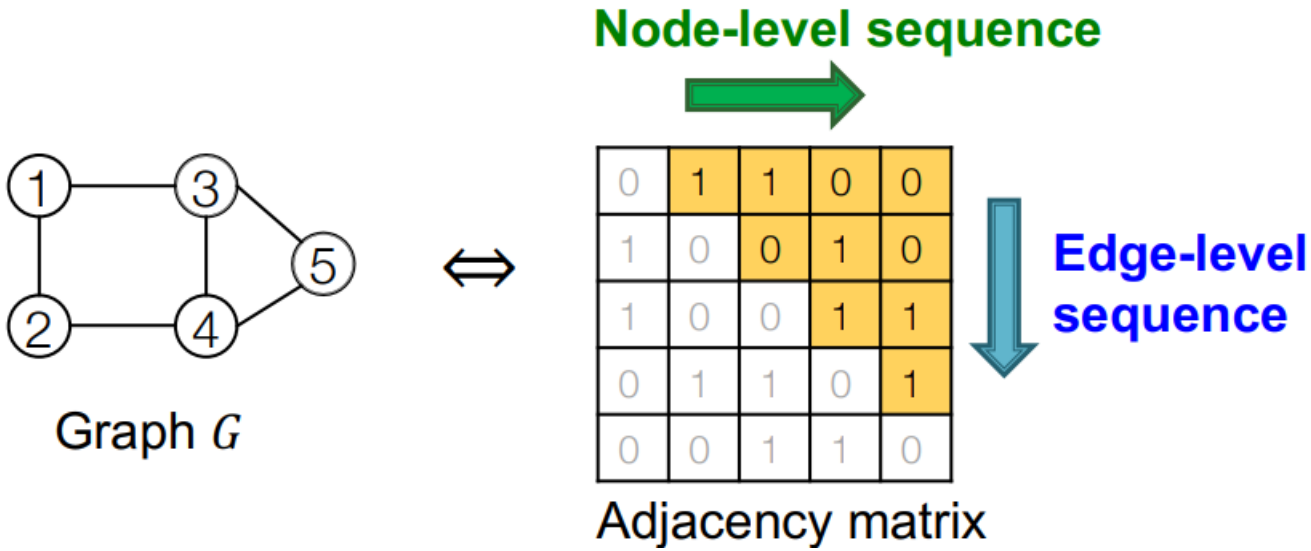
"Not connect 4, 1" "Connect 4, 2" "Connect 4, 3"

0 1 1

GraphRNN

Model Graphs as Sequences

- Summary: a graph + a node ordering → a sequence of sequences!



GraphRNN

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!

Summary Questions of the lecture

- Why is the Graph Generation Tasks hard?
- Discuss the machine learning background of auto-regressive model to obtain the graph generative models.
- Explain GraphRNN which is a sequence process to generate a graph with a node ordering.