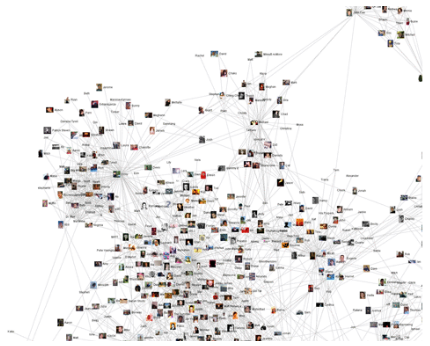
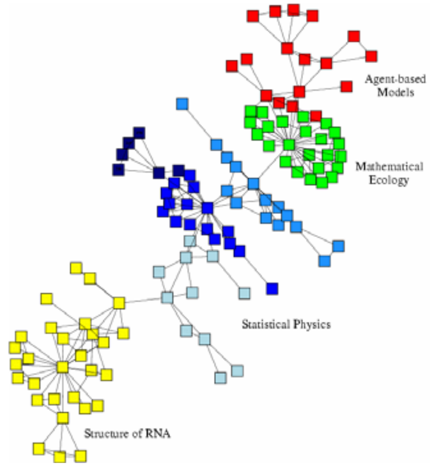
Generative Models for Graphs

Hongwei Jin

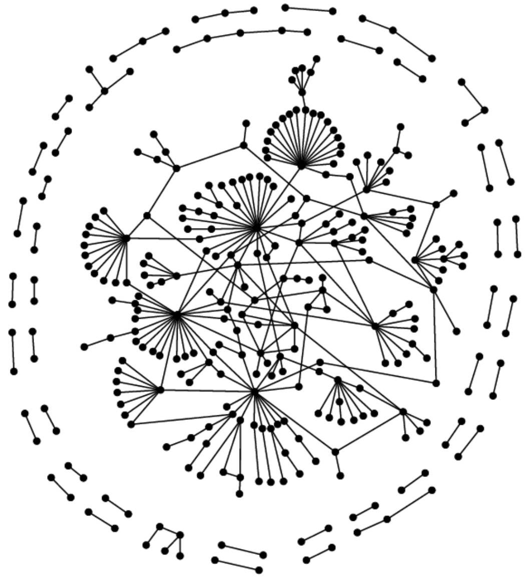
2020-10-28

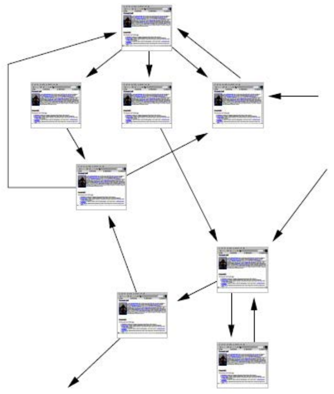
Many Data are Graphs

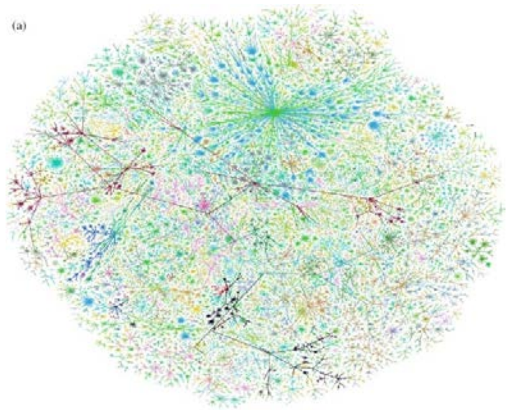

Social networks



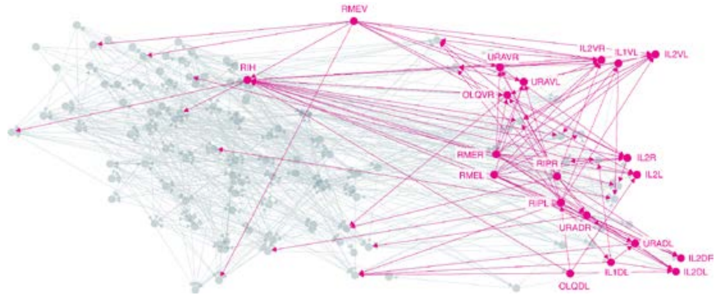
Economic networks


Biomedical networks


Information networks



Internet



Network of neurons

Why Graphs?

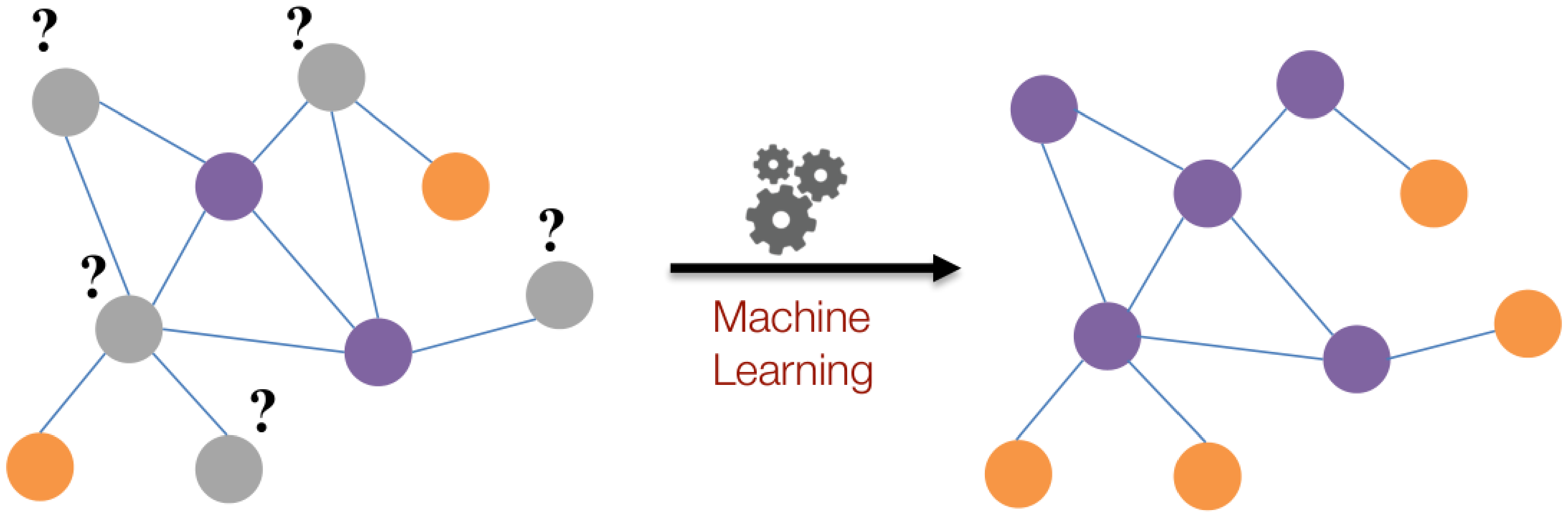
- Universal language for describing complex data
  - Networks/graphs from science, nature, and technology are more similar than one would expect
- Shared vocabulary between fields
  - Computer Science, Social science, Physics, Economics, Statistics, Biology
- Data availability (+computational challenges)
  - Web/mobile, bio, health, and medical
- Impact!
  - Social networking, Social media, Drug design

Machine Learning with Graphs

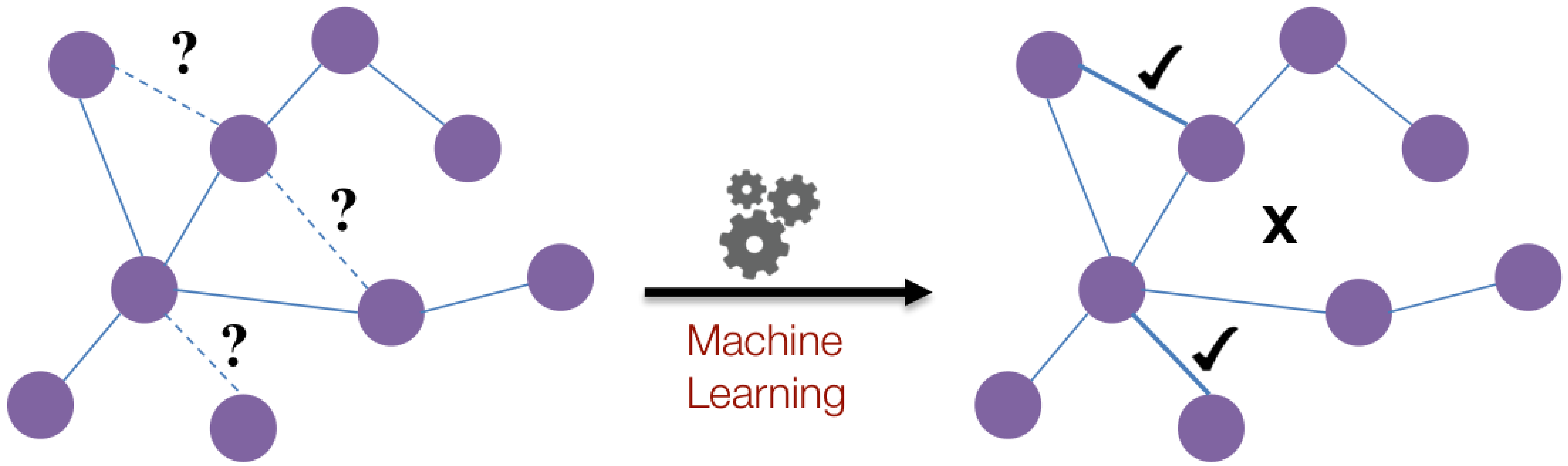
Classical ML tasks in graphs:

- Node classification
  - Predict a type of a given node
- Link prediction
  - Predict whether two nodes are linked
- Graph classification
  - Predict the label of a single graph
- Community detection
  - Identify densely linked clusters of nodes
- Network similarity
  - How similar are two (sub)networks

Node classification

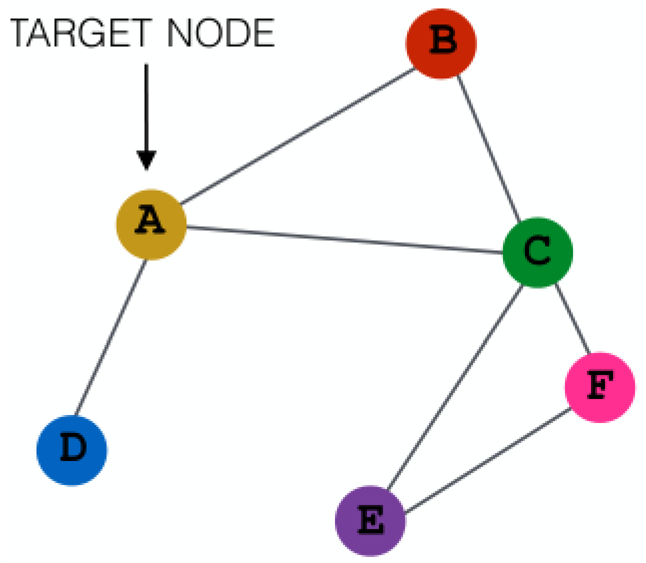


Link Prediction

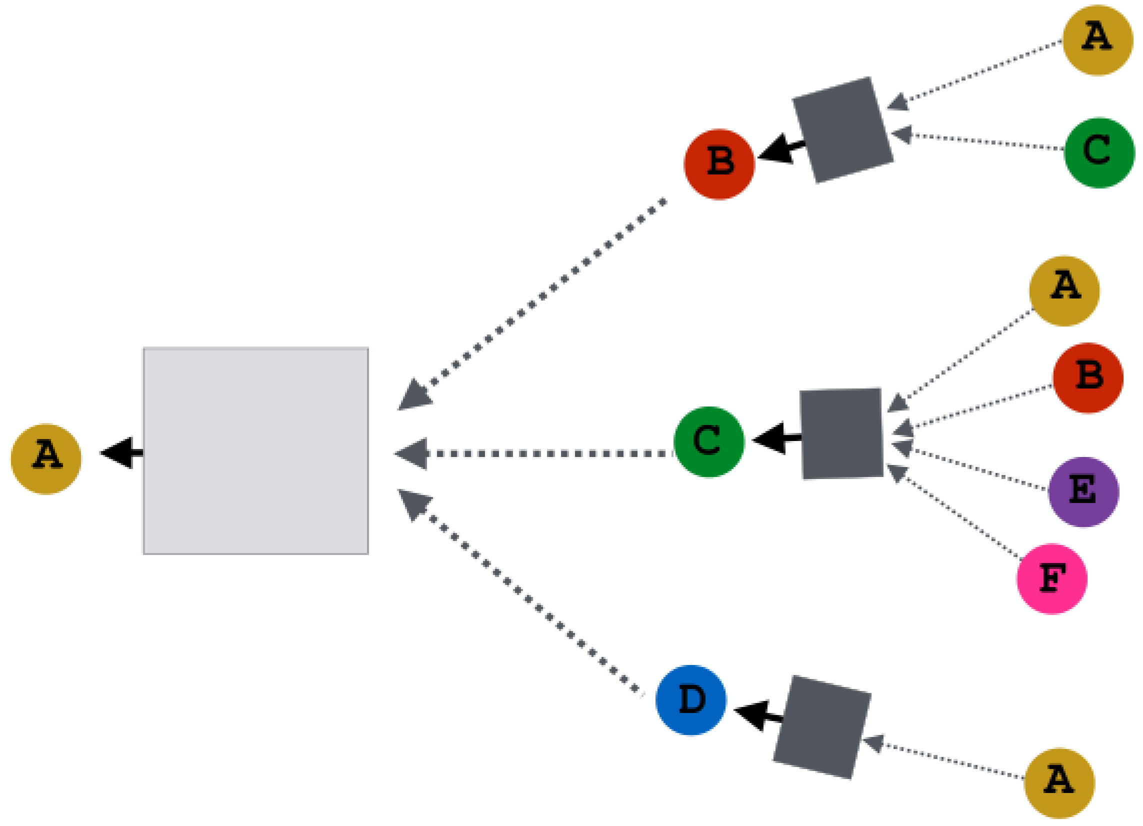


Idea – Aggregate Neighbors

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

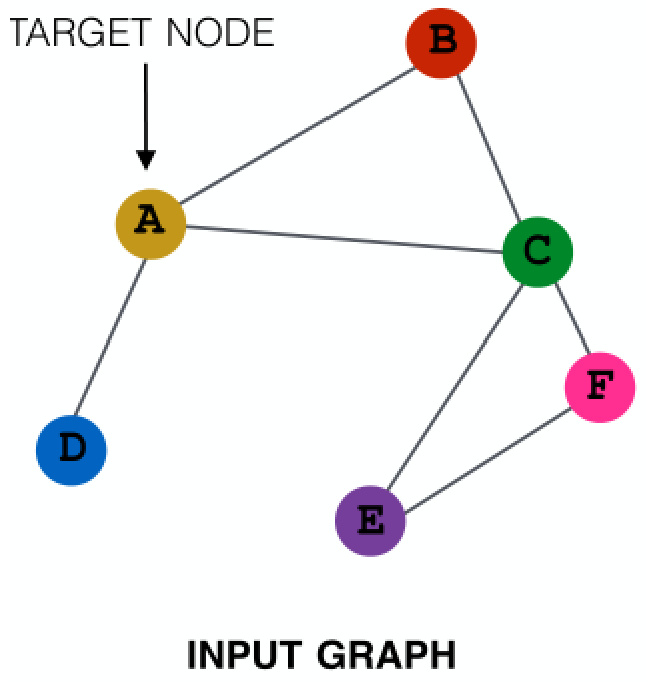


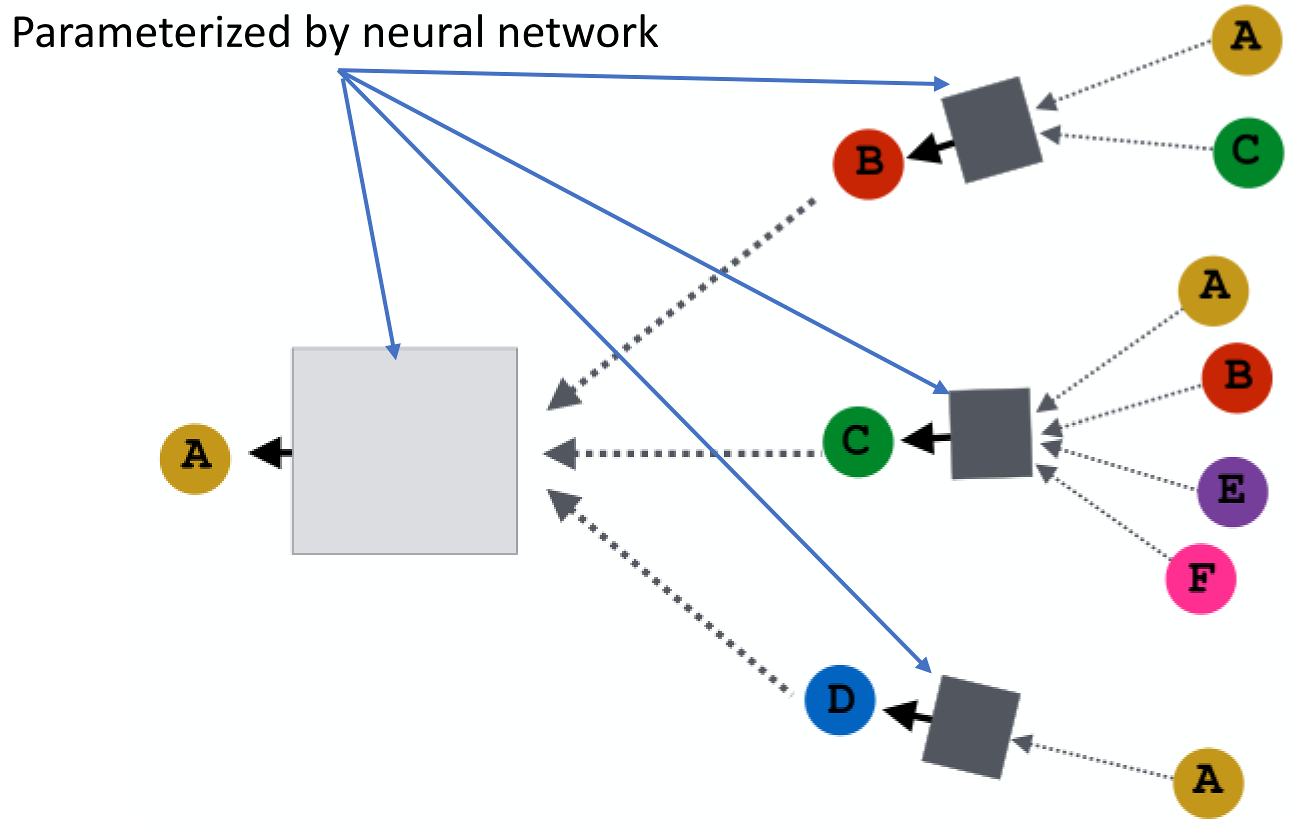
INPUT GRAPH



Idea – Aggregate Neighbors

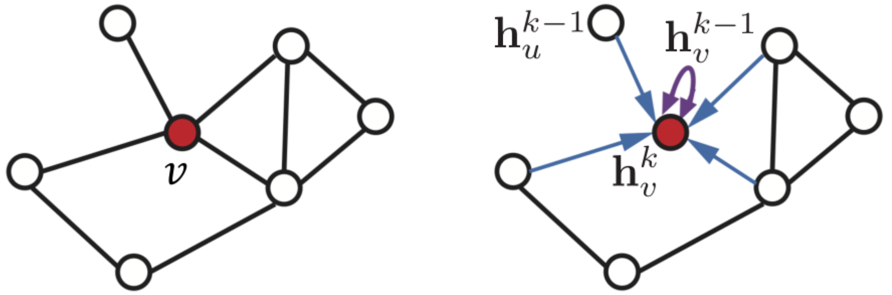
- Intuition: Nodes aggregate information from their neighbors using neural networks



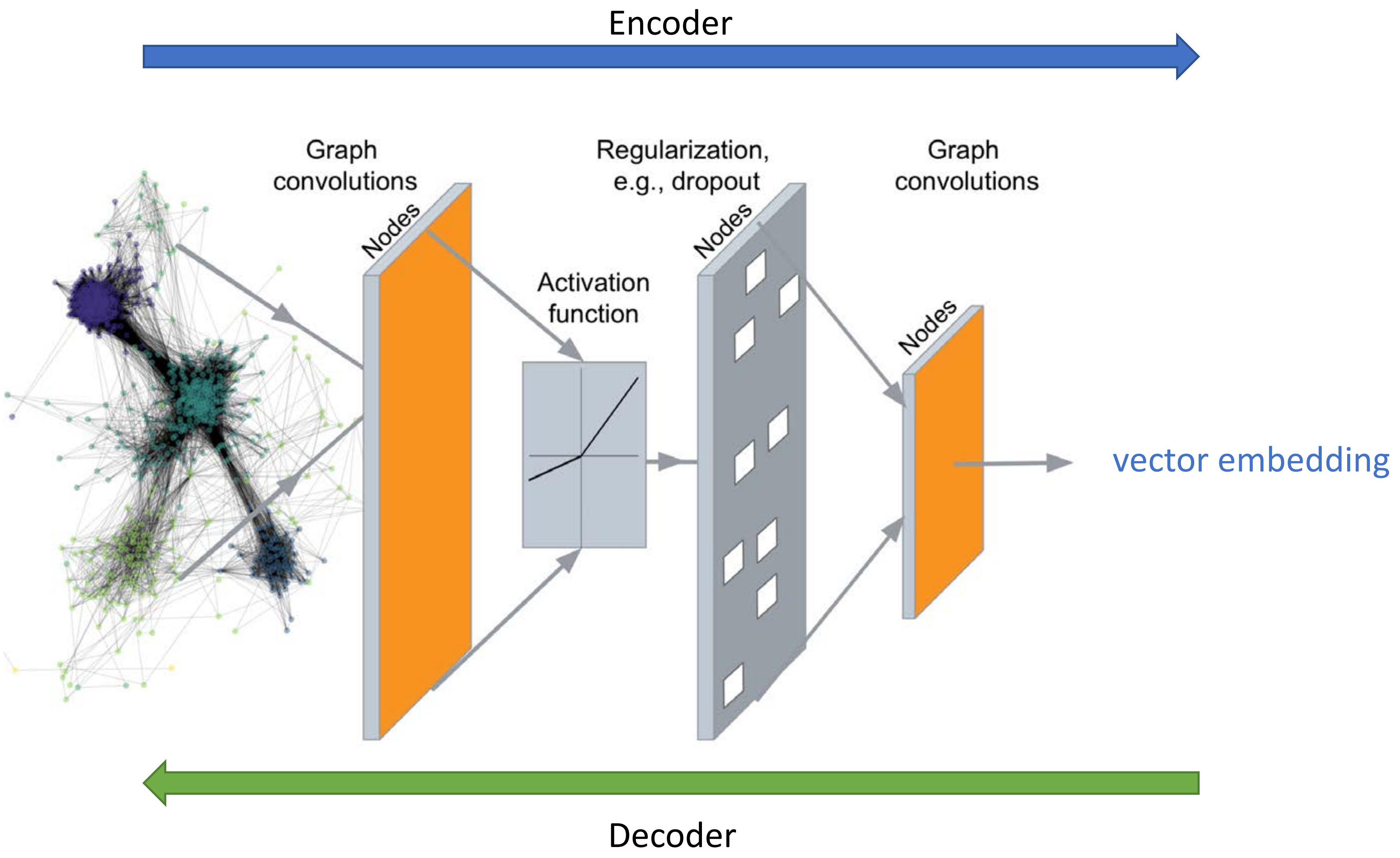


Example – GCN, GraphSAGE

- Key idea: Generate node embeddings based on local network neighborhoods
  - Nodes aggregate "messages" from their neighbors using neural networks
- Graph convolutional network
  - Basic variant: average neighborhood information and stack neural networks
- GraphSAGE
  - Generalized neighborhood aggregation

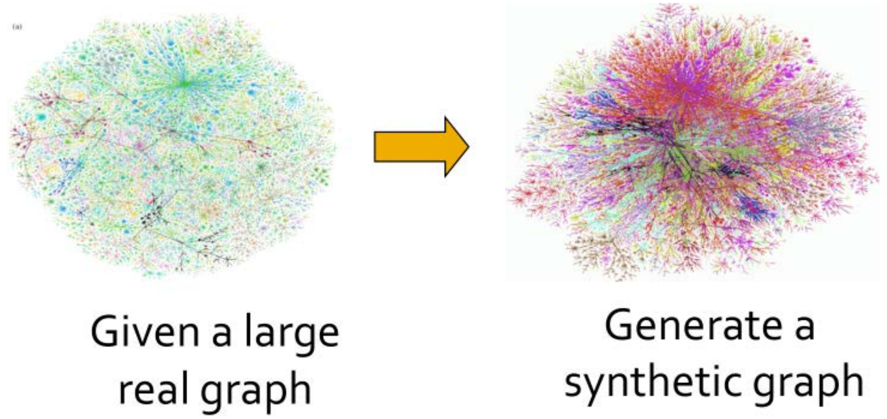


Graph encoder/decoder



Graph Generation Problem

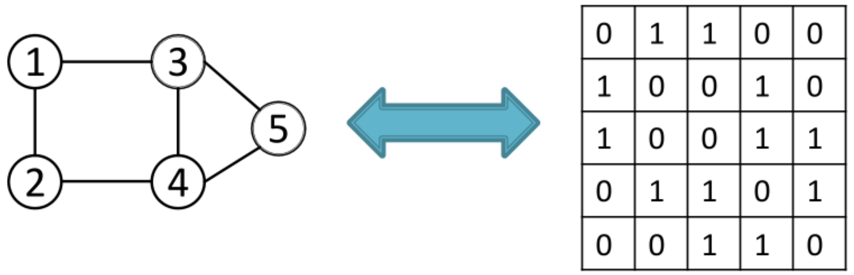
- We want to generate realistic graphs



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

Challenges for Graph Generation

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

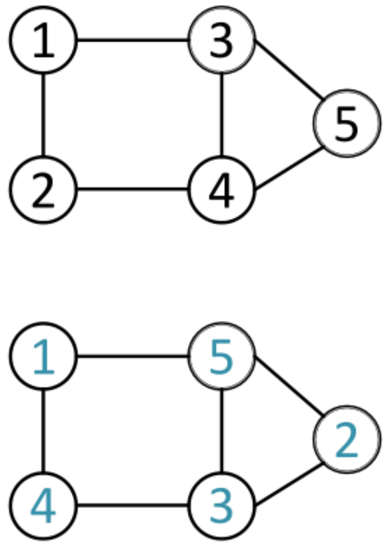


0	1	1	0	0
1	0	0	1	0
1	0	0	1	1
0	1	1	0	1
0	0	1	1	0

5 nodes: 25 values

Challenges for Graph Generation

- Isomorphic graphs
  - 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

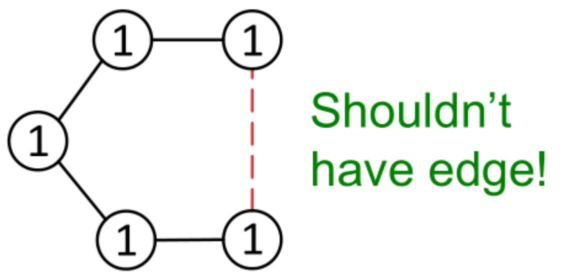
Same graph

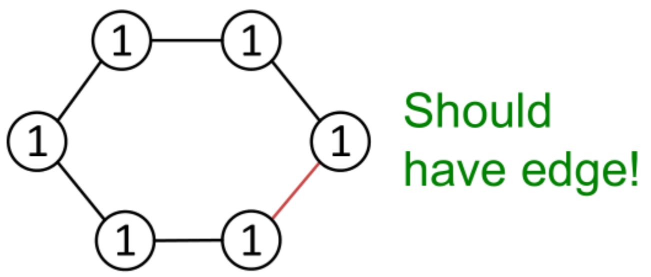
Very different representations!

Challenges for Graph Generation

- Complex dependencies
  - Edge formation has long-range dependencies

Example: Generate a ring graph on 6 nodes:

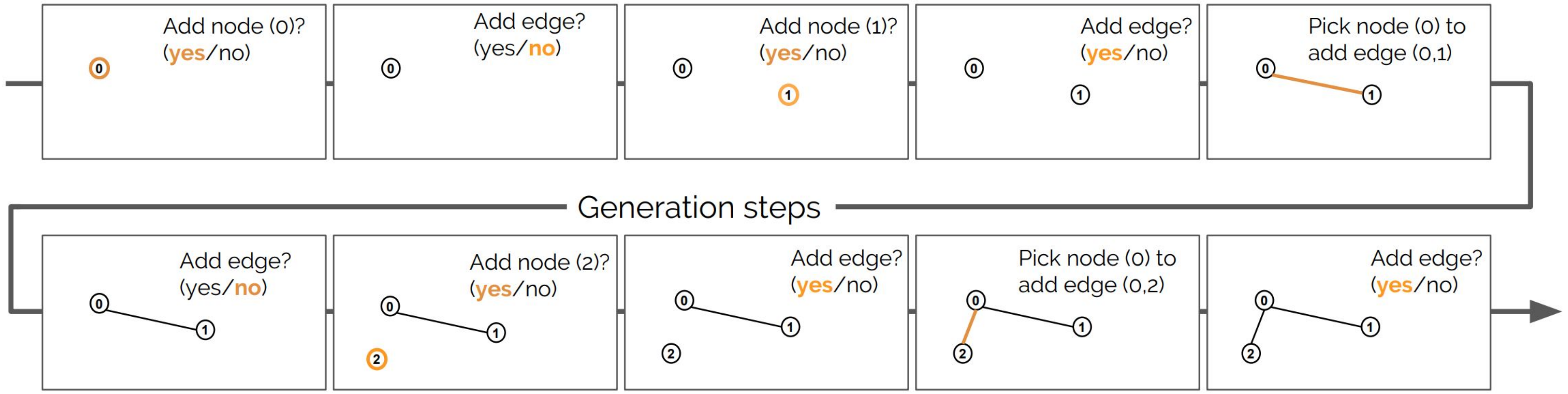




Existence of an edge may depend on the entire graph!

A very General Graph Generation Process

- Loop until not adding new nodes:
  - Add node?
  - Create node
  - Loop until not adding new edges:
    - Add edge?
    - Choose an existing node to create edge



Generative Models of Graphs

- Stochastic graph models
  - Erdos-Renyi model, Barabasi-Albert model, stochastic block model, small-world model
  - Nice theory, but limited capacity
- Tree-based models
  - Tons of tree generation models
  - Only works on trees
- Graph grammars
  - Makes hard distinction between what is in the language vs not, hard to use

Deep 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 $p_{model}(\boldsymbol{x};\theta)$, i.e., generate examples from $p_{model}(\boldsymbol{x};\theta)$

Deep Generative Models

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

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

Deep Generative Models

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

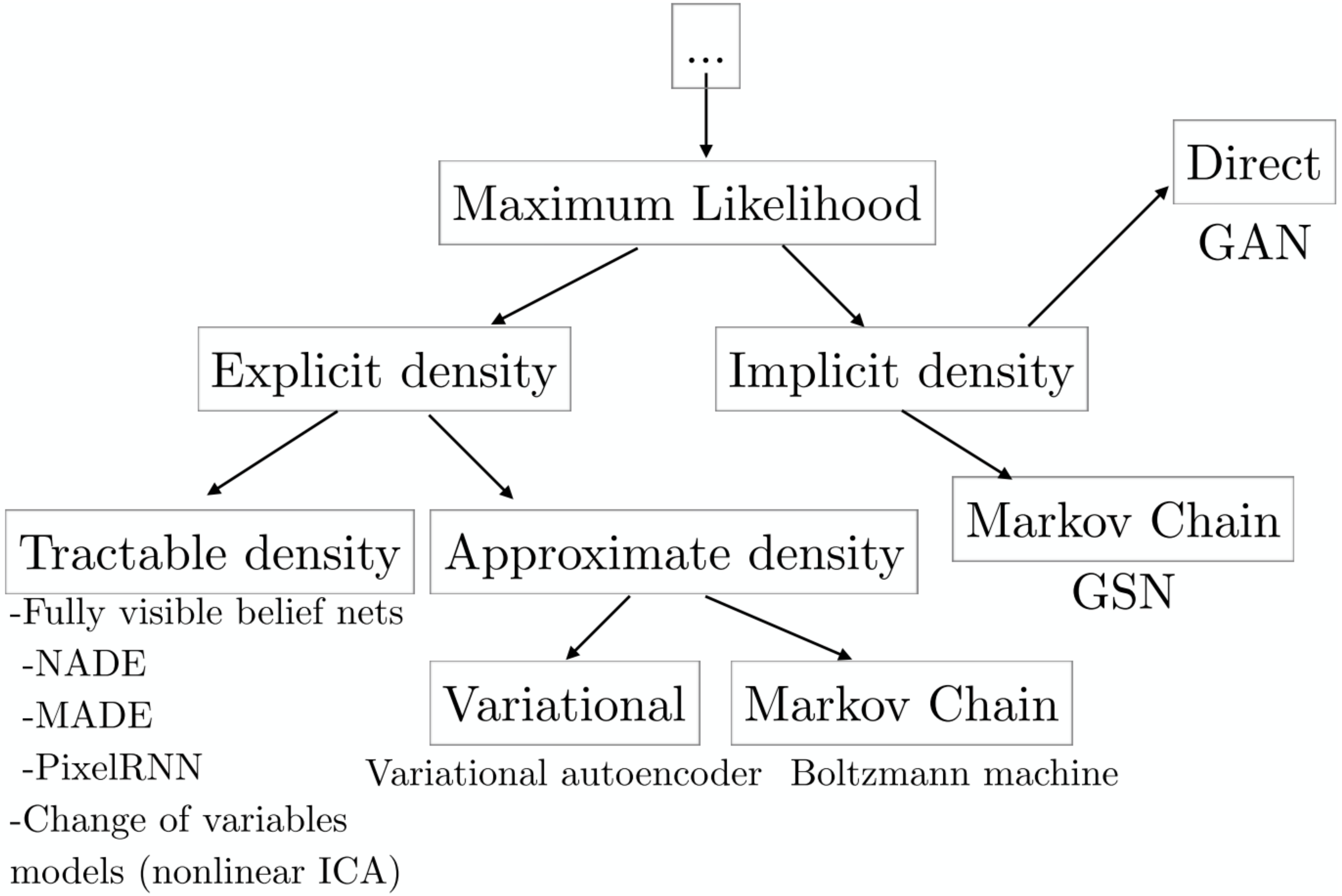
- **Goal:** sample from a complex distribution
- The most common approaches:
    - 1) Sample from a simple noise distribution

$$\boldsymbol{z}_i \sim N(0,1)$$

    - 2) Transform the noise $\boldsymbol{z}_i$ via a function $f(\cdot)$ ← Use deep neural networks to design f

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

        - $\boldsymbol{x}_i$ follows a complex distribution


...
Maximum Likelihood
Explicit density
Implicit density
Direct
GAN
Markov Chain
GSN
Tractable density
-Fully visible belief nets
-NADE
-MADE
-PixelRNN
-Change of variables models (nonlinear ICA)
Approximate density
Variational
Variational autoencoder
Markov Chain
Boltzmann machine

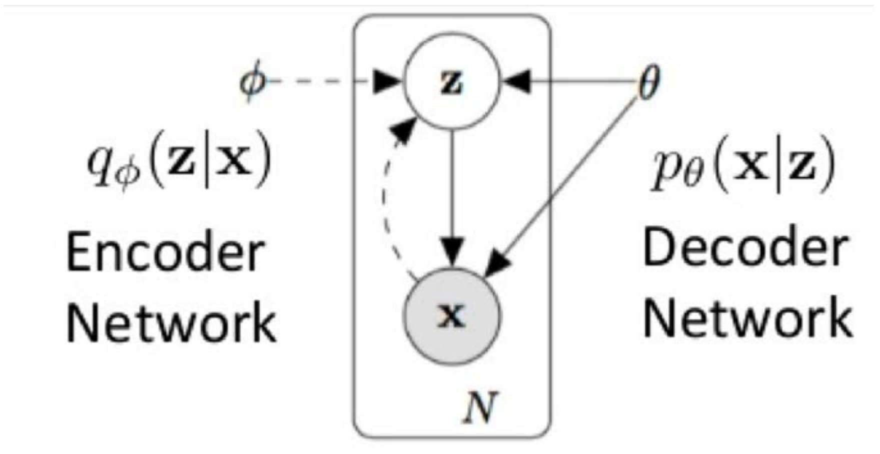
Types of Deep Generative Models

- Variational Autoencoders (VAEs)
  - VAEs, Kingma et al. 2014
- Generative Adversarial Networks (GANs)
  - GANs, Goodfellow et al. 2014
- Deep Auto-regressive Models (ARs)
  - ARs, Oord et al. 2016
- …

$\Bigg\}$ Extend to deep graph generative models

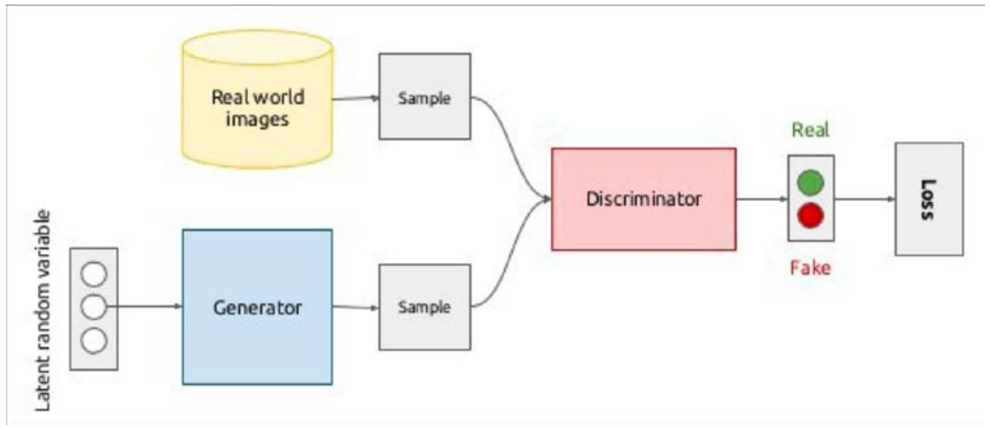
VAEs, Kingma et al. 2014

- Latent variable model
  - An encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$
  - A decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$
- Maximizing the likelihood $\log p(\boldsymbol{x})$
  - Inference intractable since $\boldsymbol{z}$ is continuous.
- Maximizing the variational lower-bound $\mathcal{L}(\phi, \theta; \boldsymbol{x})$
  - Reparametrization trick for jointly optimizing encoder and decoder



$$\mathcal{L}(\phi, \theta; x) = \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log p_\theta(\boldsymbol{x}|\boldsymbol{z})}_{\text{Reconstruction}} - \underbrace{KL[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})]}_{\text{Regularization}}$$

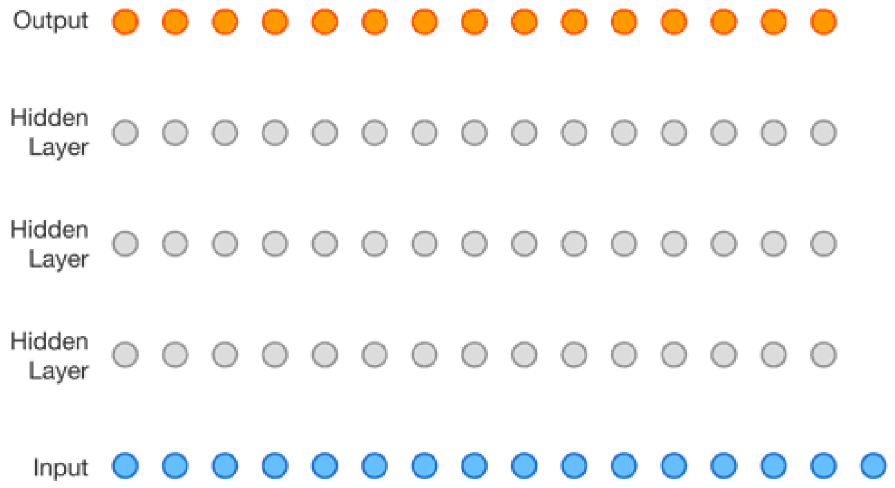
GANs, Goodfellow et al. 2014



- A two-player minimax game
  - Generator G: $\boldsymbol{z} \rightarrow \boldsymbol{x}$
  - Discriminator D: $\boldsymbol{x} \rightarrow \{0, 1\}$
- Discriminator aims to distinguish between real data and generated data
- Generator aims to fool the discriminator

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

ARs, Oord et al. 2016



- Example of deep auto-regressive model
  - Recurrent Neural Networks
- PixelRNN, Pixel CNN (Oord et al. 2016)
  - Generate an image pixel by pixel
  - A neural network is used to model the conditional distribution
- WaveNet (Oord et al. 2016)

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$

VAE based Graph Generative Model

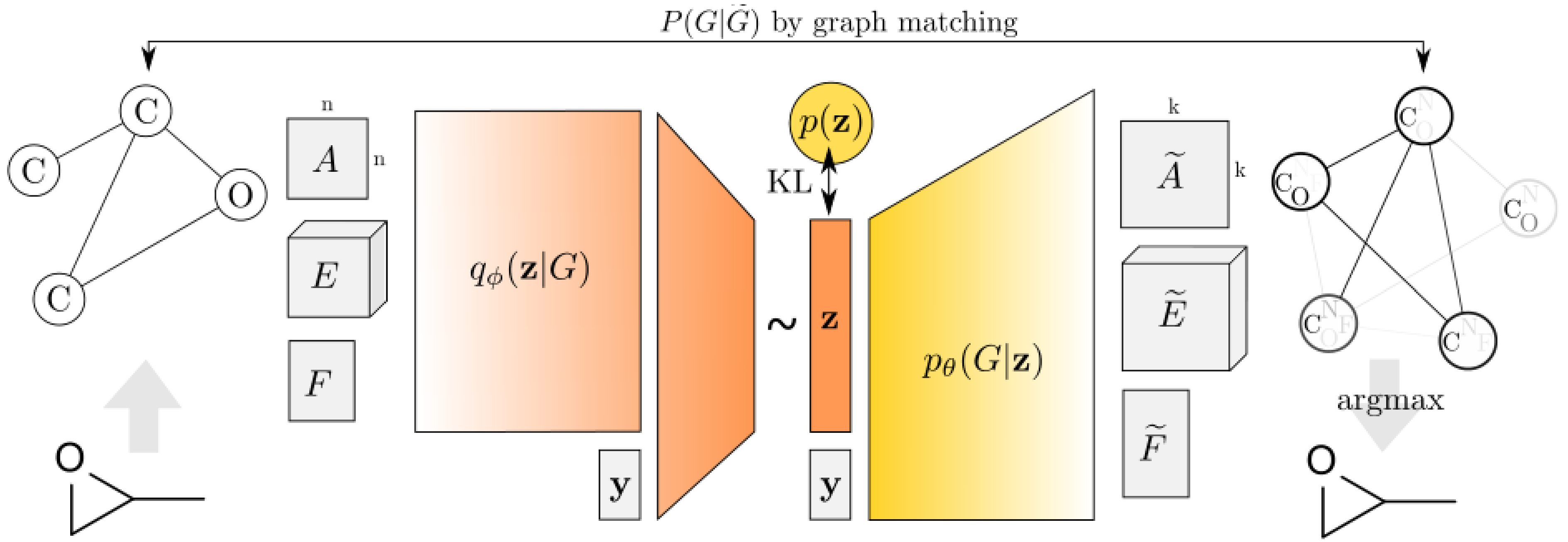
GraphVAE (Simonovsky and Komodakis, 2018)

- VAE framework for graph generation
  - Graph as input data
  - Encoder: graph neural networks + gated pooling ➔ graph representation
  - Decoder: output a probabilistic fully-connected graph of predefined maximum size
    - Model the existence of nodes, edges and their attributes independently
    - Graph matching is required

VAE based Graph Generative Model

GraphVAE (Simonovsky and Komodakis, 2018)

- Input: graph $G = (A, E, F)$
  - A: adjacency matrix, E: edge attribute tensor, F: node attribute matrix



VAE based Graph Generative Model

GraphVAE (Simonovsky and Komodakis, 2018)

- Input: graph $G = (A, E, F)$
  - A: adjacency matrix, E: edge attribute tensor, F: node attribute matrix

$$
\begin{aligned}
&\mathcal{L}(\phi, \theta; \cancel{x})^{\text{G}} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\cancel{\mathbf{x}})} \log p_\theta(\cancel{\mathbf{x}}|\mathbf{z}) - KL[q_\phi(\mathbf{z}|\cancel{\mathbf{x}})||p(\mathbf{z})]
\end{aligned}
$$

Reconstruction Regularization

- New reconstruction loss:

$$
\log p(G|z) = \lambda_A \log p(A'|z) + \lambda_F \log p(F|z) + \lambda_E \log p(E|z)
$$

VAE based Graph Generative Model

GraphVAE (Simonovsky and Komodakis, 2018), graph decoder

- Restrict the domain to the set of all graphs on maximum k nodes (k is around tens)
- Output a probabilistic fully-connected graph $\tilde{G} = (\tilde{A}, \tilde{E}, \tilde{F})$ on k nodes at once
  - Model the existence of nodes and edges as Bernoulli variables
  - Model the node and edge attributes as Multinomial variables
  - $\tilde{A} \in [0,1]^{k\times k}$ contains both node probabilities $\tilde{A}_{aa}$ and edge probabilities $\tilde{A}_{ab}$ for nodes $a \neq$b
  - $\tilde{E} \in [0,1]^{k\times k\times d_e}$ indicates the probabilities for edge attributes
  - $\tilde{F} \in [0,1]^{k\times d_n}$ indicates the probabilities for node attributes
- Inference: taking edge- and node-wise argmax in $\tilde{A}$, $\tilde{E}$, and $\tilde{F}$.
- Graph Matching must be used for calculating the reconstruction loss

$\longrightarrow$ Find corresponding $X \in \{0,1\}^{k\times n}$, mapping between G and $\tilde{G}$, COST

GAN based Graph Generative Model

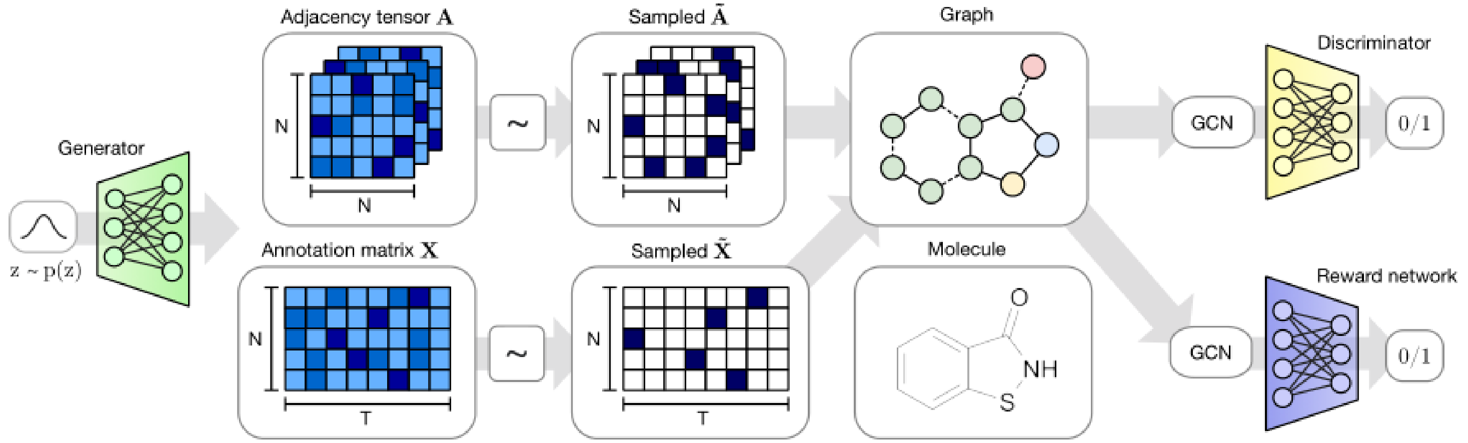
MolGAN (Cao and Kipf 2018)

- An implicit, likelihood-free generative model for molecule generation
- Combined with reinforcement learning to encourage the generated molecules with desired chemical properties
- **Generator**: generating molecules from a prior distribution
- **Discriminator**: distinguishing the generated samples and real samples
- **Reward network**:
  - Learns to assign a reward to each molecule to match a score provided by an external software
  - Invalid molecules always receive zero rewards.

GAN based Graph Generative Model

MolGAN (Cao and Kipf 2018), Generator

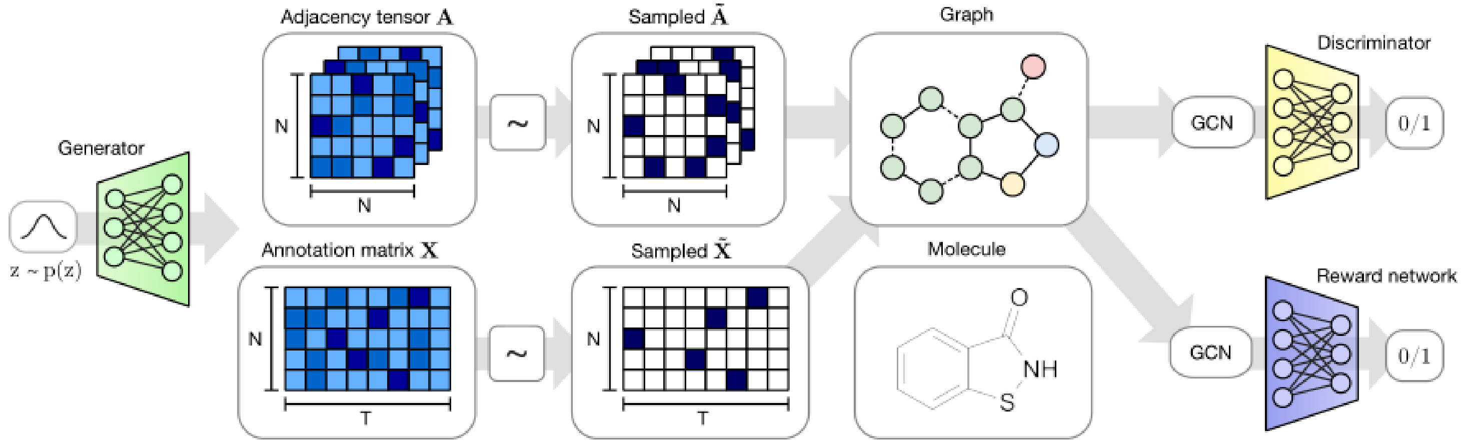
- A probabilistic fully-connected graph
  - $X \in R^{N \times T}$: atom types
  - $A \in R^{N \times N \times T}$: bond types
- Objective function: $L(\theta) = \lambda L_{WGAN} + (1-\lambda) L_{RL}$


Generator
$z \sim p(z)$
Adjacency tensor $\mathbf{A}$
N
N
Annotation matrix $\mathbf{X}$
N
T
~
Sampled $\tilde{\mathbf{A}}$
Sampled $\tilde{\mathbf{X}}$
Graph
Molecule
O
NH
S
GCN
Discriminator
0/1
GCN
Reward network
0/1

GAN based Graph Generative Model

MolGAN (Cao and Kipf 2018), Discriminator and Reward Network

- Learning molecule/graph representations with a variant of neural message passing algorithms
- Same architectures for discriminator and reward network
- Reward network for approximating the score by an external software
  - Trained with real samples and generated samples



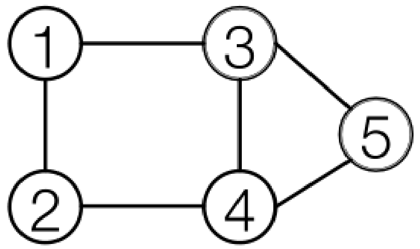
Generator
$z \sim p(z)$
Adjacency tensor $\mathbf{A}$
N
N
Annotation matrix $\mathbf{X}$
N
T
~
~
Sampled $\tilde{\mathbf{A}}$
N
N
Sampled $\tilde{\mathbf{X}}$
N
T
Graph
Molecule
O
NH
S
GCN
Discriminator
0/1
GCN
Reward network
0/1

AR based Graph Generative Model

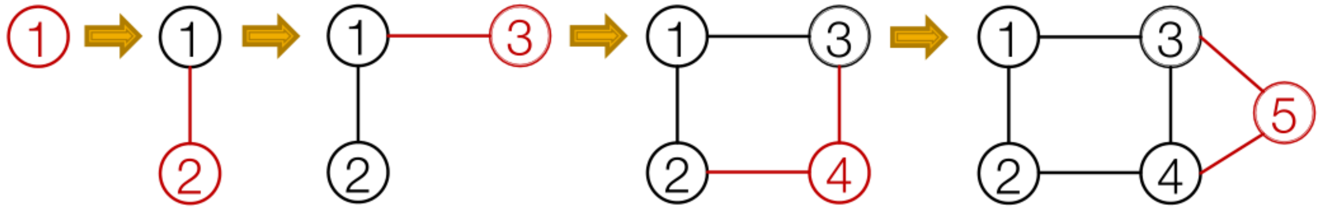
GraphRNN, You et al. 2018

- **Idea:** Generating graphs via **sequentially** adding nodes and edges

Graph G



Generation process S^{π}



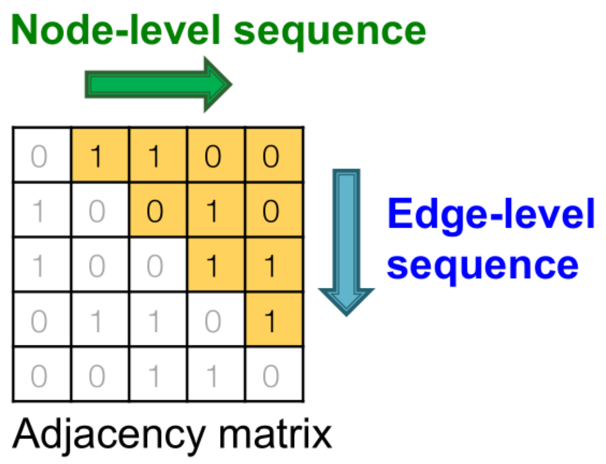
$S^{\pi} = (S_1^{\pi}, \quad S_2^{\pi}, \quad S_3^{\pi}, \quad S_4^{\pi}, \quad S_5^{\pi})$

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

AR based Graph Generative Model

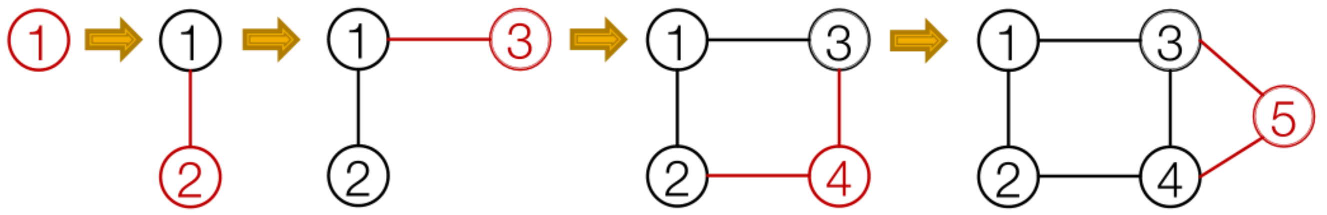
GraphRNN, You et al. 2018

- The sequence S^{π} has two levels: node and edge
- Node-level: at each step, a new node is added
- Edge-level: at each step add a new edge



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

Generation process S^{π}



AR based Graph Generative Model

GraphRNN, You et al. 2018

- Transform graph generation problem into a sequence generation problem

- Two processed required:
  - Generate a state for new node (node-level)
  - Generate edges for the new node based on its state (edge-level)

- Approach: RNN

AR based Graph Generative Model

GraphRNN, You et al. 2018

- GraphRNN has two RNNs: node-level RNN and edge-level RNN

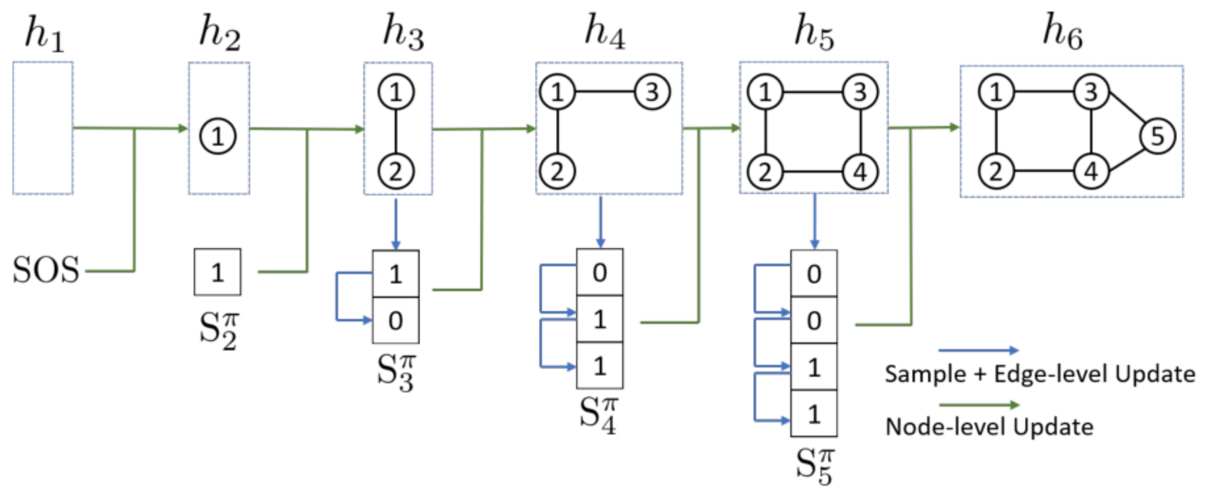
- Relationship between two RNNs:
  - Node-level RNN generates the initial state for edge-level RNN
  - Edge-level RNN generates edges for the new node, then update node-level RNN state using generated results

AR based Graph Generative Model

GraphRNN, You et al. 2018

Green arrows denote the node-level RNN that encodes the “graph state” vector h_i in its hidden state, updated by the predicted adjacency vector S_i^π for node $\pi(v_i)$

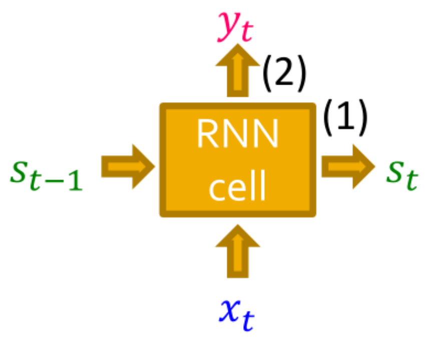
Blue arrows represent the edge-level RNN, whose hidden state is initialized by the graph-level RNN, that is used to predict the adjacency vector S_i^π for node $\pi(v_i)$



AR based Graph Generative Model

GraphRNN, You et al. 2018

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

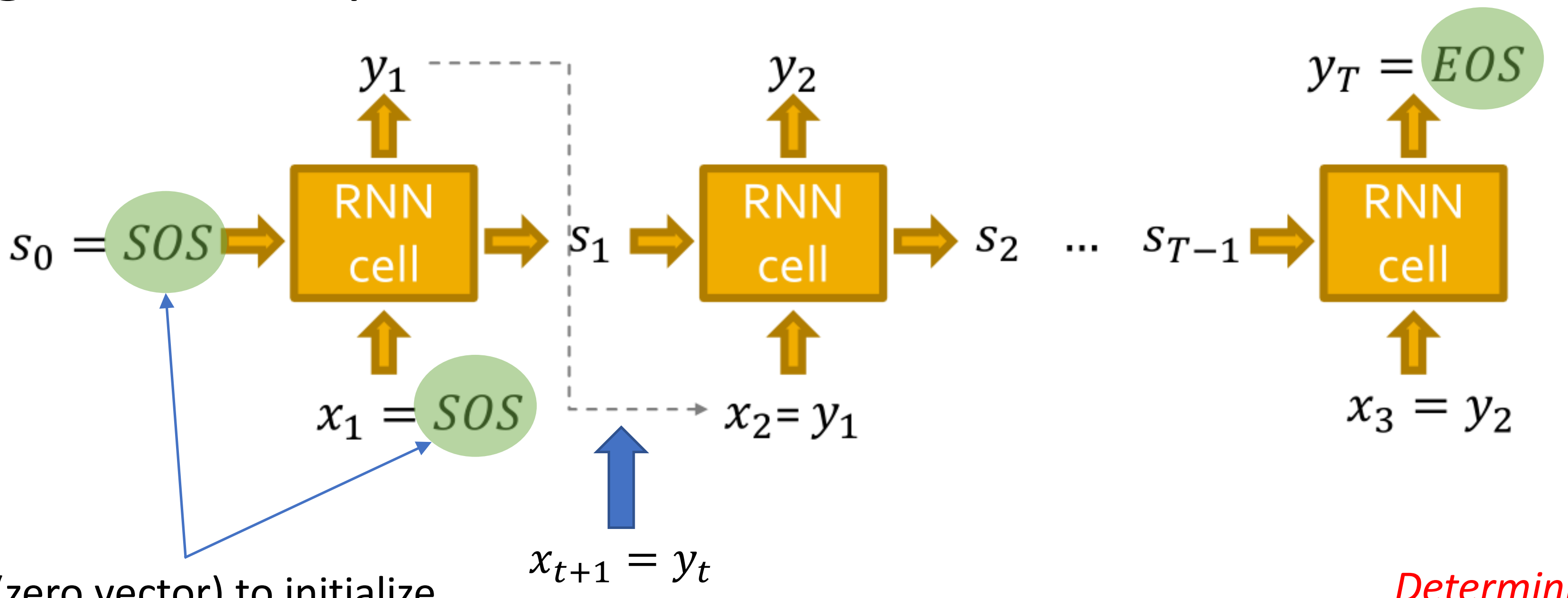
(2) $y_t = V \cdot s_t$

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

AR based Graph Generative Model

GraphRNN, You et al. 2018

- Goal: generate sequences



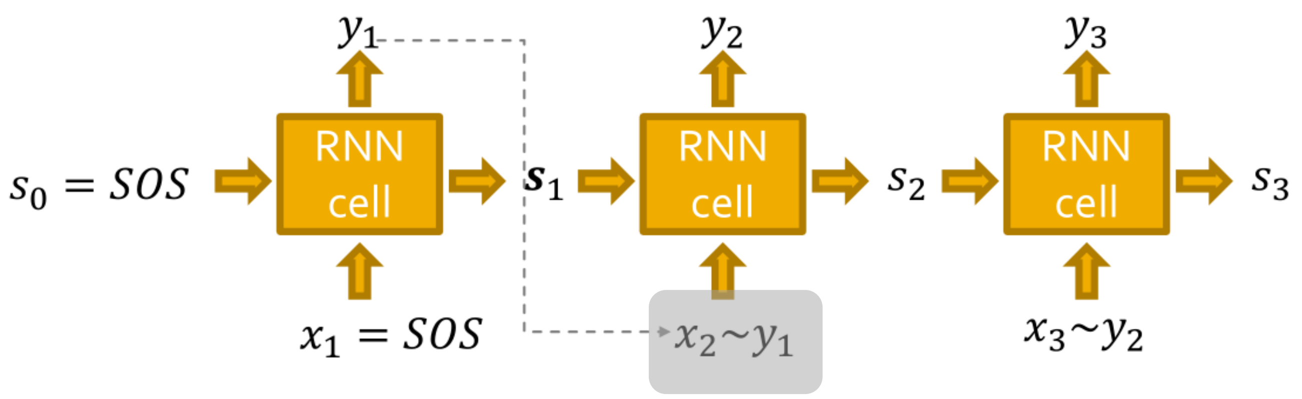
SOS (zero vector) to initialize

Deterministic, not so good

AR based Graph Generative Model

GraphRNN, You et al. 2018

- Probabilistic: $y_t = p_{model}(x_t | x_1, \cdots, x_{t-1}; \theta)$

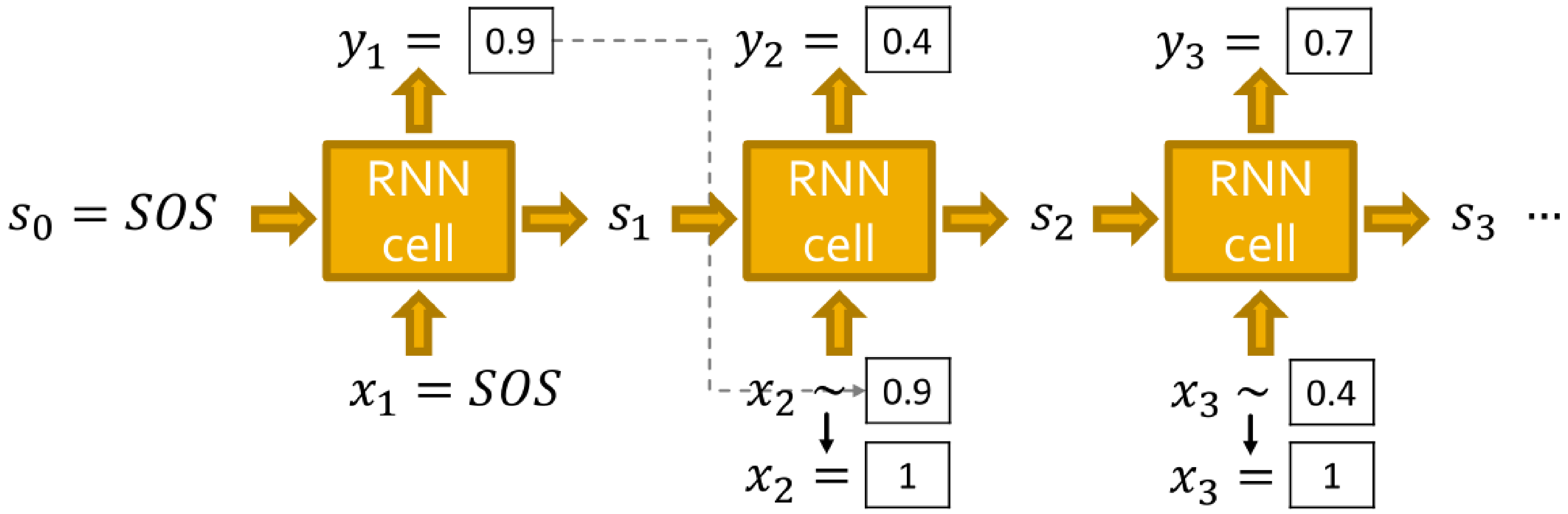


x_{t+1} is sampled from y_t: $x_{t+1} \sim y_t$

AR based Graph Generative Model

GraphRNN, You et al. 2018

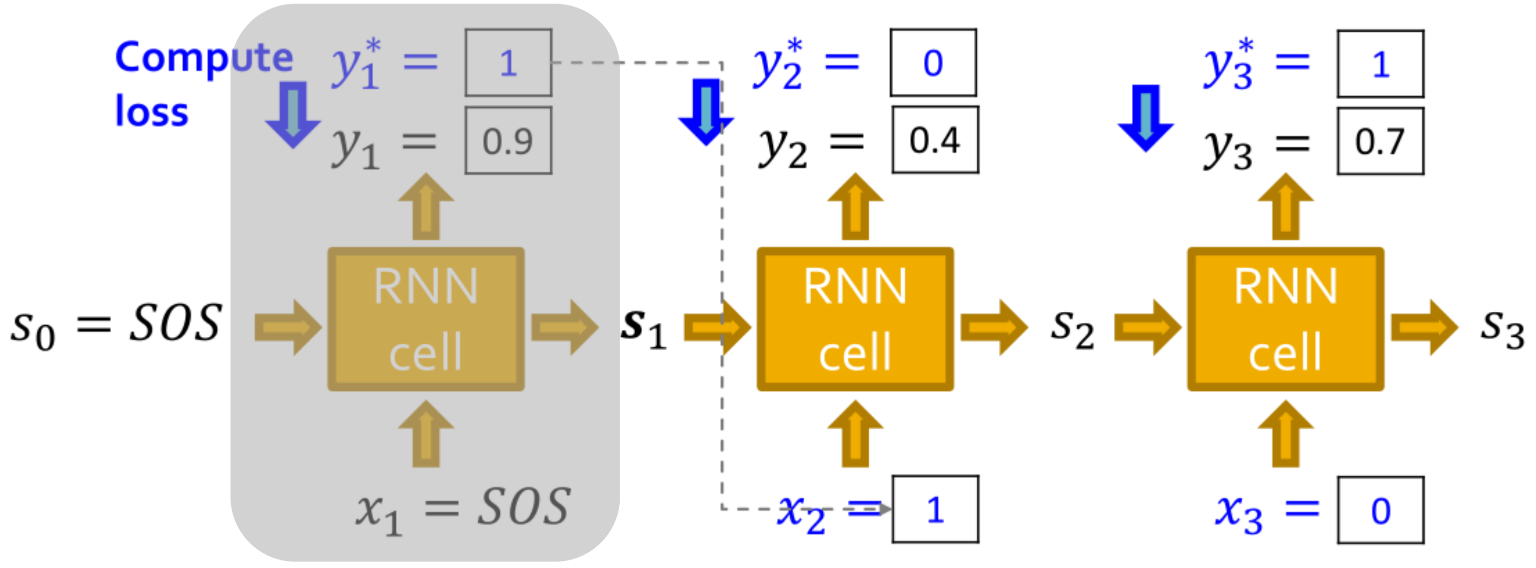
- Testing:
  - y_t follows **Bernoulli distribution** (choice of p_{model})
  - $\boxed{p}$ means value 1 has prob. p, value 0 has prob. $1-p$



AR based Graph Generative Model

GraphRNN, You et al. 2018

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



AR based Graph Generative Model

GraphRNN, You et al. 2018

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

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

Compute loss $\Downarrow$

$$y_1^* = \boxed{1}$$

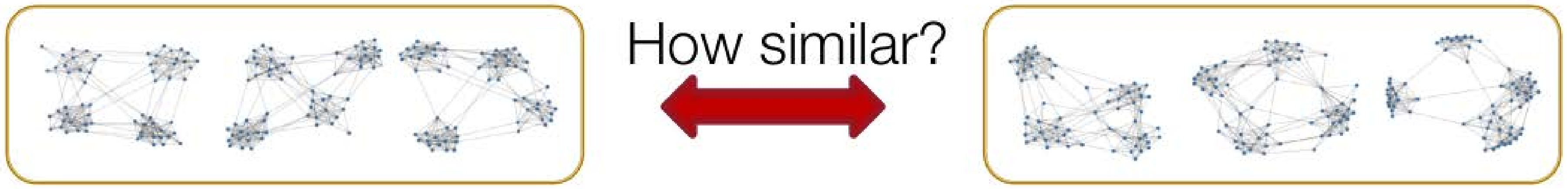
$$y_1 = \boxed{0.9}$$

  - 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^*

AR based Graph Generative Model

GraphRNN, You et al. 2018

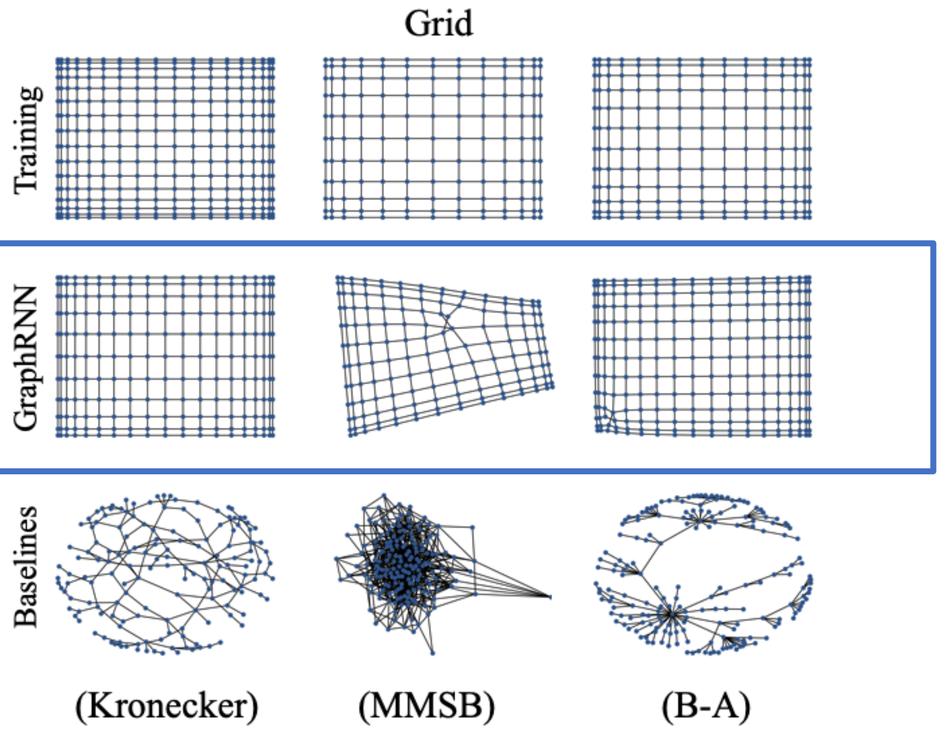
- Evaluation:
  - Define similarity metrics for graphs
  - No efficient graph isomorphism test that can be applied to any class of graphs
  - Solution:
    - Visual similarity
    - Graph statistics similarity: degree distribution, cluster coefficient, diameter of graphs, etc.



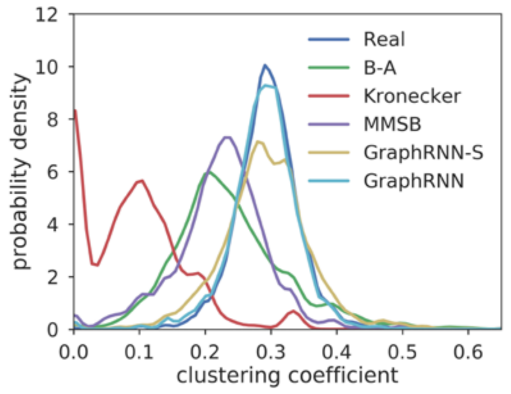
AR based Graph Generative Model

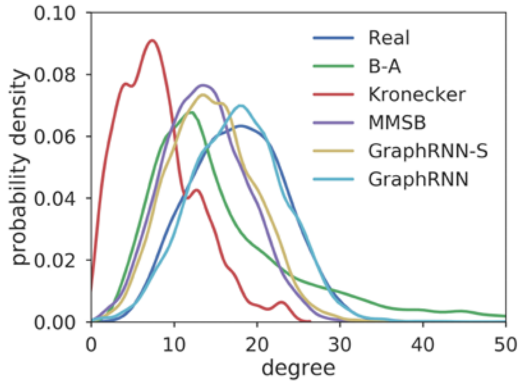
GraphRNN, You et al. 2018

- Visual Similarity:



- Graph statistics Similarity:





MDP based Graph Generative Model

GCPN: Graph Convolutional Policy Network, You et al. 2018

- Molecule generation as sequential decisions
  - Add nodes and edges
  - A Markov decision process
- Goal: discover molecules that optimize desired properties while incorporating chemical rules.
- GCPN: A general model for **goal-directed graph generation** with RL
  - Optimize adversarial loss and domain-specific rewards with policy gradients
  - Acts in an environment that incorporates domain-specific rules.

MDP based Graph Generative Model

GCPN: Graph Convolutional Policy Network, You et al. 2018

- Goal-Directed Graph Generation
  - 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

MDP based Graph Generative Model

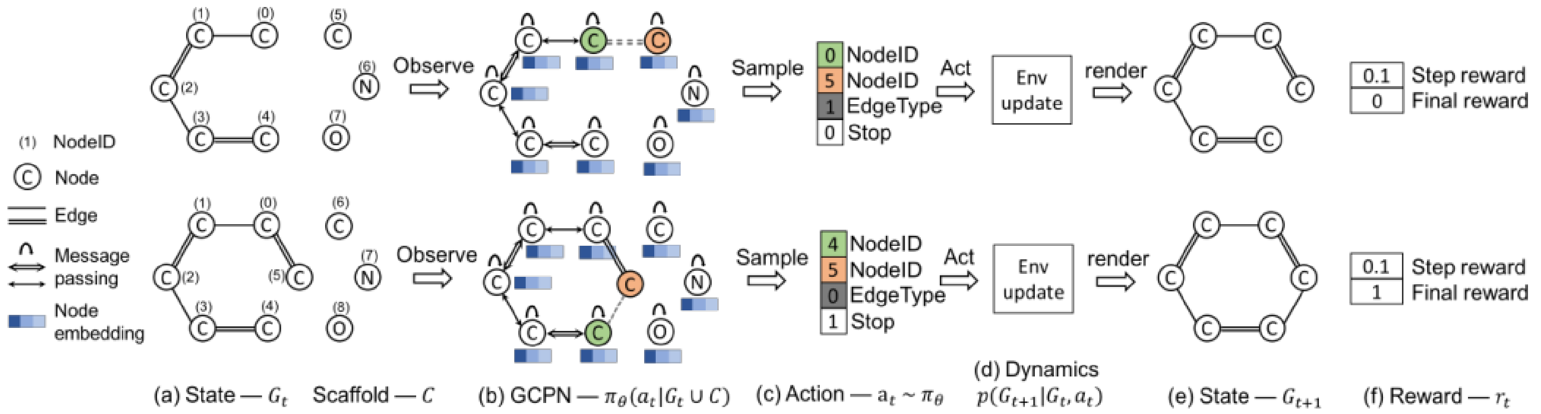
GCPN: Graph Convolutional Policy Network, You et al. 2018

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

MDP based Graph Generative Model

GCPN, You et al. 2018, MDP

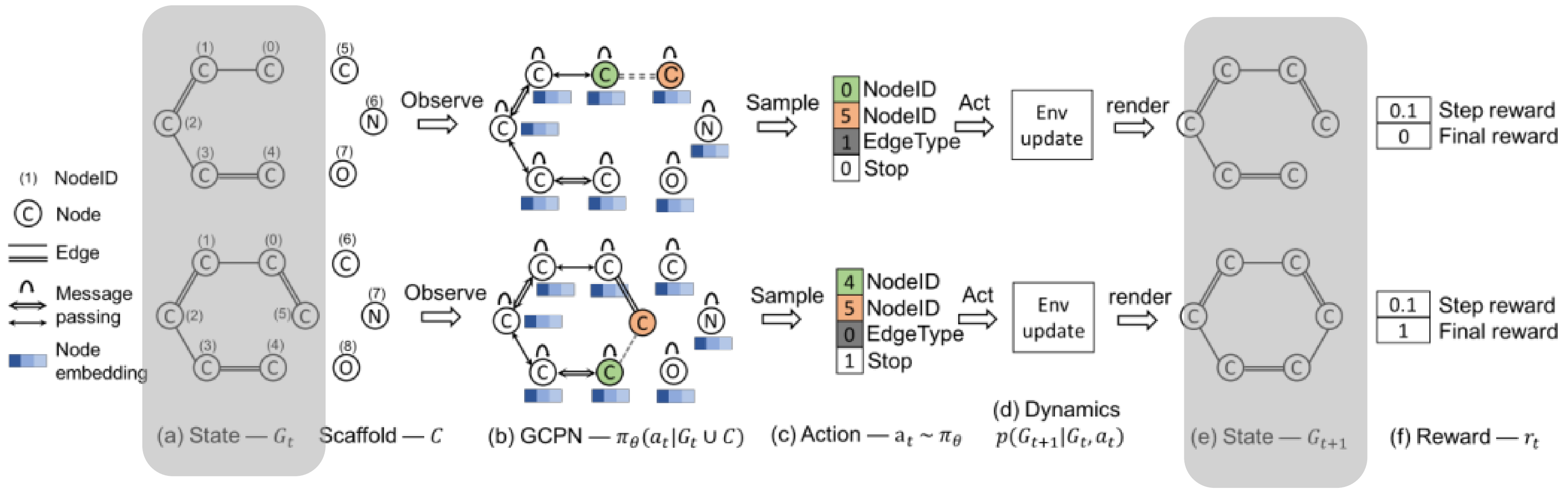
- $M = (S, A, P, R, \gamma)$
  - States $S = \{s_i\}$ consists of all possible intermediate and final graphs
  - Action $A = \{a_i\}$ modification made to the current graph at each step
  - State transitional dynamics P
  - Reward function R
  - Discount factor γ



MDP based Graph Generative Model

GCPN, You et al. 2018, State space

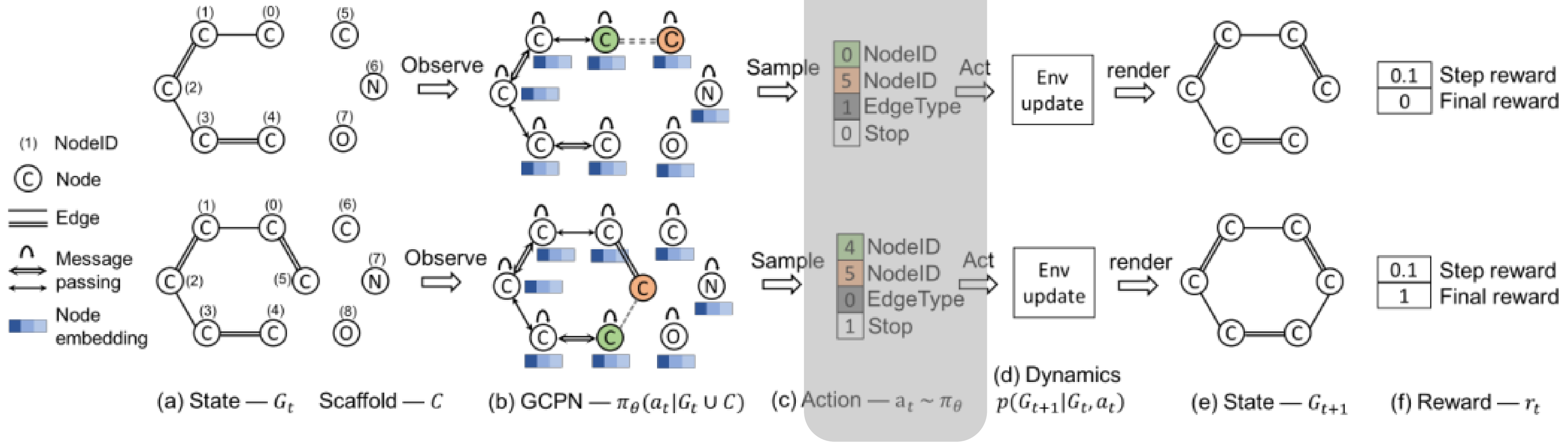
- s_t as the intermediate generated graph G_t
- G_0 contains a single node that represents a carbon atom



MDP based Graph Generative Model

GCPN, You et al. 2018, Action Space

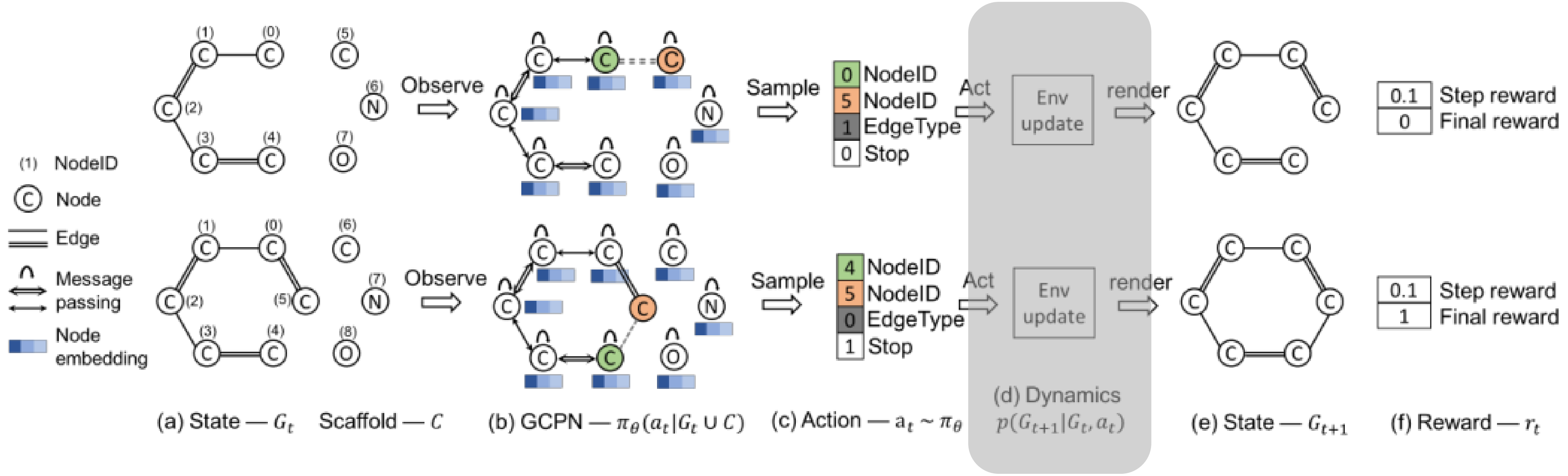
- A set of atoms $C = \cup_{i=1}^{S} C_i$ to be added during each step
- Actions
  - Connecting a new atom C_i to a node in G_t
  - Connecting existing nodes within G_t



MDP based Graph Generative Model

GCPN, You et al. 2018, State Transition Dynamic

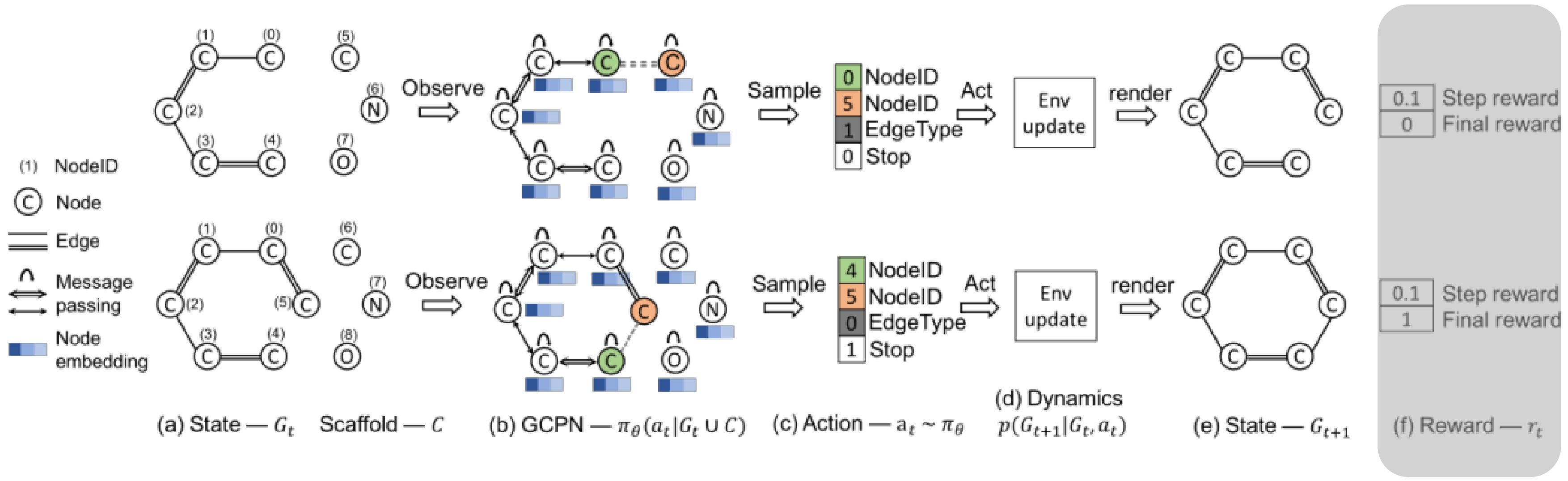
- Incorporate domain-specific rules in the state transition dynamics. Only carry out actions that obey the **given rules**
- Infeasible actions by the policy network are rejected and state remains same



MDP based Graph Generative Model

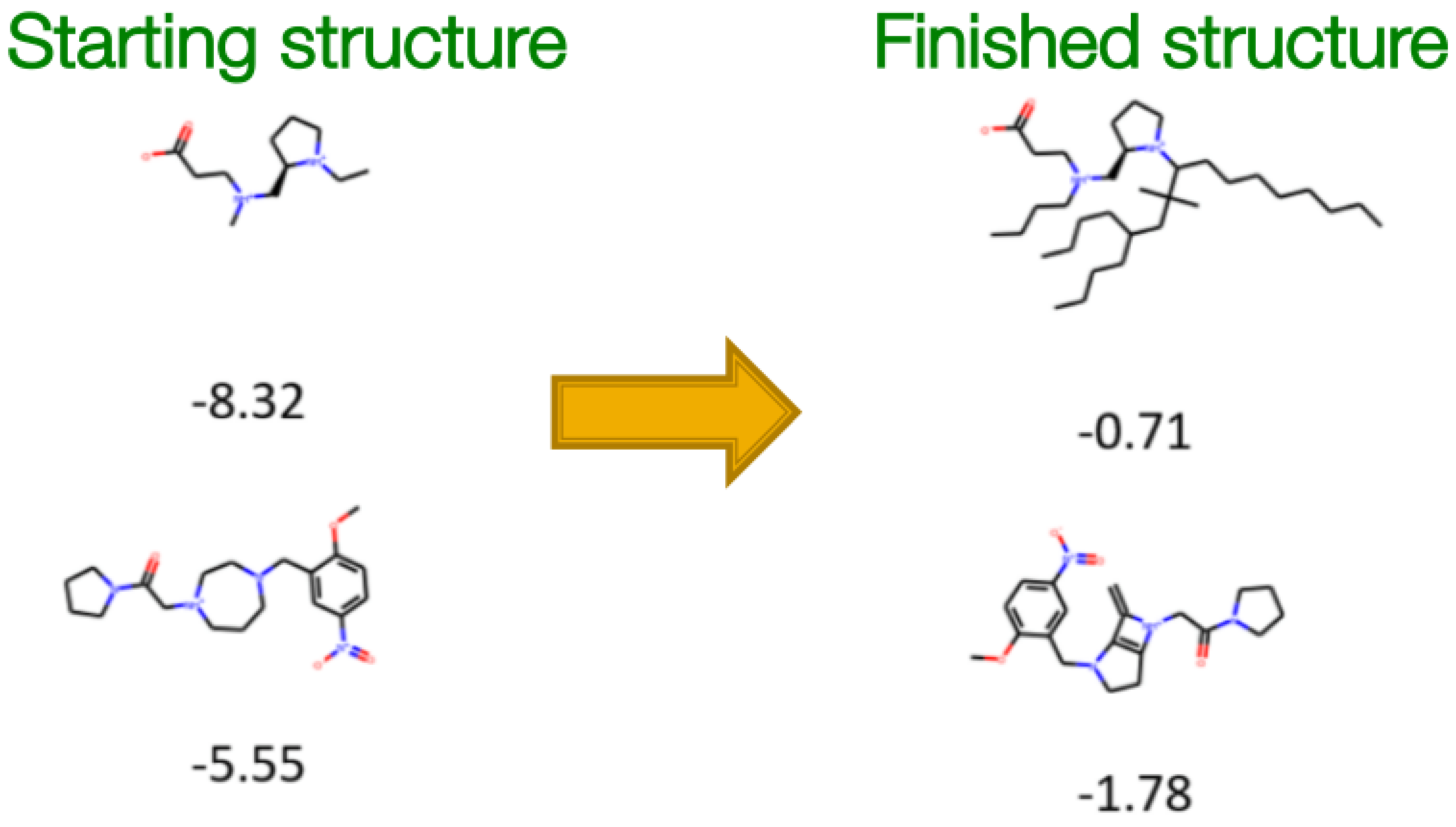
GCPN, You et al. 2018, Reward Design

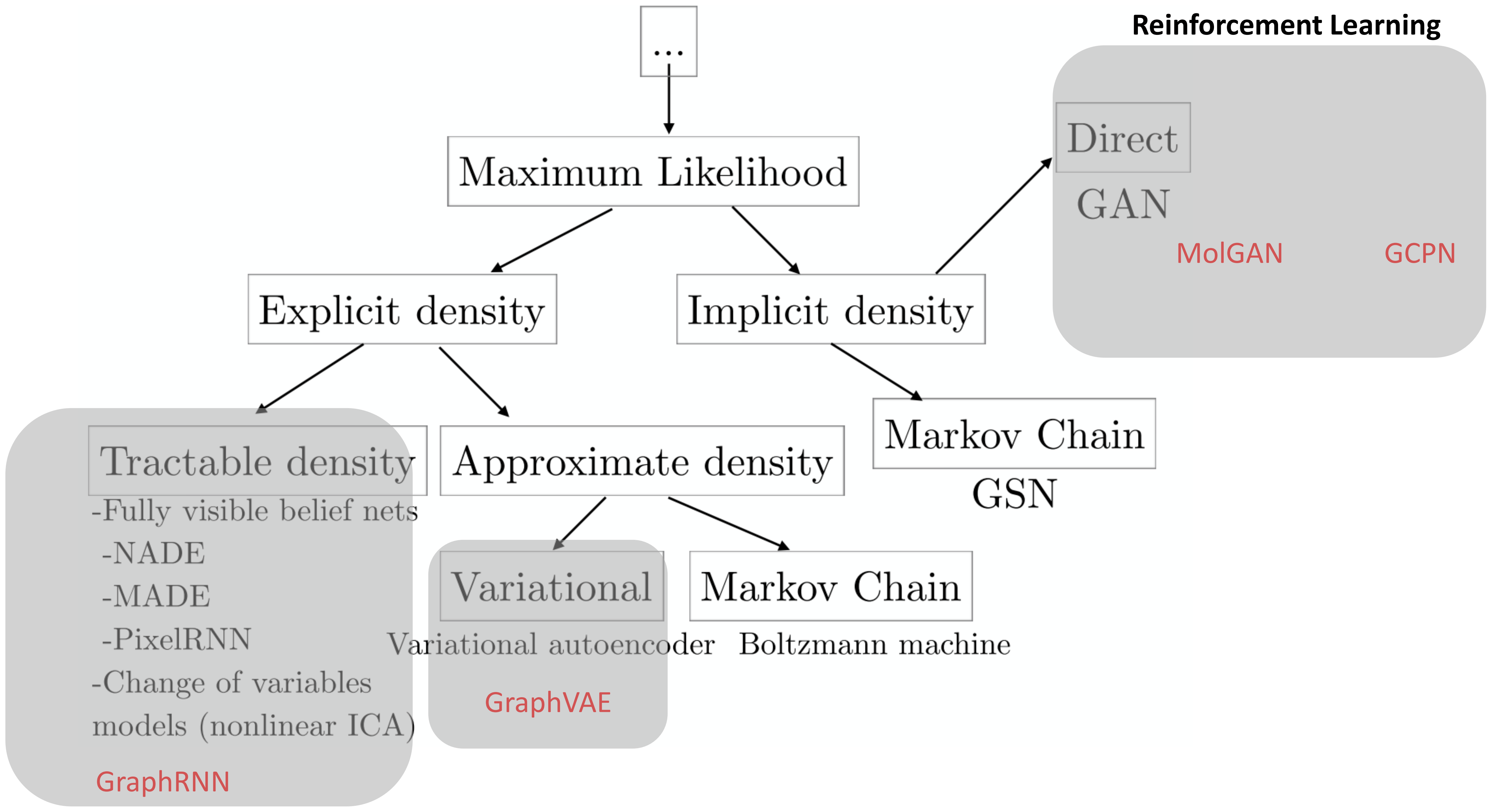
- Step rewards: step-wise validity rewards and adversarial rewards
- Final rewards: a sum over domain-specific reward
  - Final property scores, penalization of unrealistic molecules, adversarial rewards



MDP based Graph Generative Model

GCPN, You et al. 2018, Results





Reinforcement Learning
...
Maximum Likelihood
Explicit density
Implicit density
Direct
GAN
MolGAN
GCPN
Tractable density
-Fully visible belief nets
-NADE
-MADE
-PixelRNN
-Change of variables models (nonlinear ICA)
GraphRNN
Approximate density
Variational
Variational autoencoder
GraphVAE
Markov Chain
Boltzmann machine
Markov Chain
GSN

More

- Generating graphs in other domain
  - 3D mesh reconstruction, scene graphs, knowledge graphs, etc.
- Scale up to large graphs
  - Hierarchical action space, allowing high-level action like adding a structure at a time
  - Leverage the sparse structure of graphs (Dai et al. 2020)
- More on **graph neural networks + reinforcement learning**
  - Relational deep reinforcement learning (Zambaldi et al. 2018)
- New discrepancy on graphs
  - Gromov Wasserstain distance (Bunne et al. 2019)

References

- CS 224W – Machine Learning with Graphs, Stanford University
- Tutorial on Graph Representation Learning , AAAI 2019
- Simonovsky, Komodakis, *GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders,* ICANN'18
- You et al. *GraphRNN: Generating Realistic Graphs with Deep Auto-regressive models,* ICML'18
- *You et al. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation,* NIPS'18
- Cao, Kipt. *MolGAN: An implicit generative model for small molecular graphs,* 2018
- Dai et al. *Scalable Deep Generative Modeling for Sparse Graphs,* ICML'20
- Li et al. *Learning Deep Generative Models of Graphs,* ICML'18