Graph Neural Networks

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

KAUST Academy
King Abdullah University of Science and Technology

February 1, 2023

Recap

- Convolutional filters can be generalized to graphs giving us Graph Convolutional Neural Networks (GCNNs).
- Several processes have a sequential nature. To learn from them, we need dedicated architectures.
- Recurrent Neural Networks (RNNs) are designed to work with sequential data.
- We define Graph Recurrent Neural Networks (GRNNs) as particular cases of RNNs.
- Hidden and observed state are propagated through graph filters to update the hidden states and to predict outputs.
- Time and spatial Gating is used to deal with the problem of vanishing gradients in GRNNs.

Graph Generation

- So far, we have been learning from graphs and we assume graphs are given.
- But how are these graphs generated?

Image credit: Medium

Social Networks

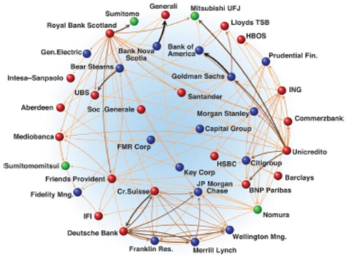
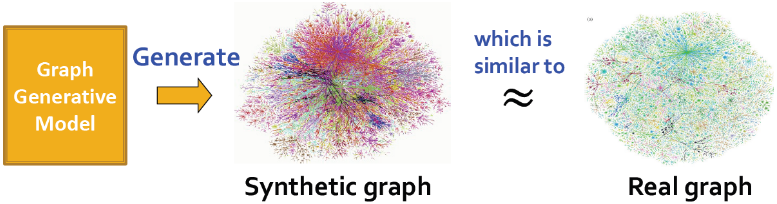


Image credit: Science

Economic Networks

Graph Generation (cont.)

- We want to generate realistic graphs, using graph generative models.
- Some applications include Drug discovery, material design, Social network modeling, etc.

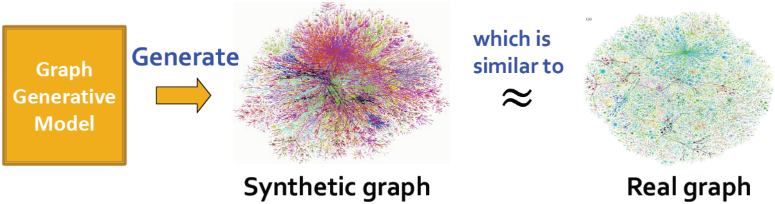


Why do we study graph generation?

- **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 general novel graph instances.
- **Anomaly detection**: We can decide if a graph is normal / abnormal.

Graph Generative Models

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

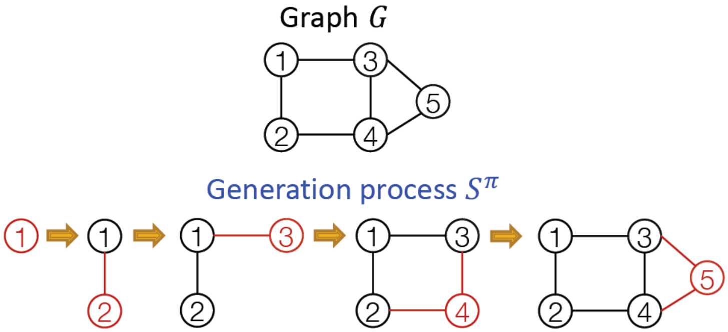


Graph Generative Models (cont.)

- We want to learn a generative model from a set of data points (i.e., graphs) $\{x_i\}$.
  - $p_{data}(x)$ is the data distribution, which is never known to us, but we have sampled $x_i \sim p_{model}(x)$.
  - $p_{model}(x; \theta)$ is the model, parameterized by θ, that we use to approximate $p_{data}(x)$.
- Goal:
  - Make $p_{model}(x; \theta)$ close to $p_{data}(x)$ (Density estimation).
  - Make sure we can sample from $p_{model}(x; \theta)$ (Sampling)

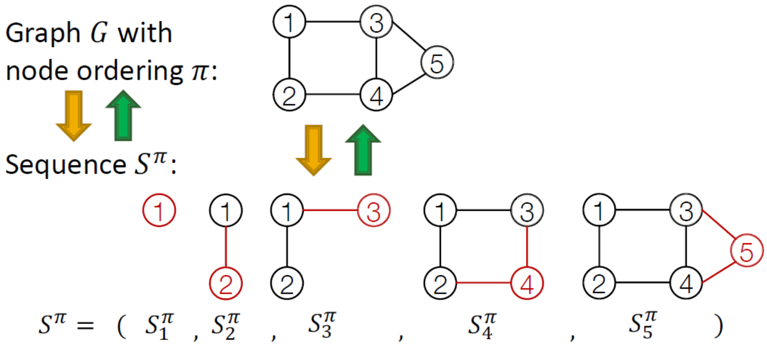
Graph Generation using GraphRNNs

- We can generate graphs via sequentially adding nodes and edges.



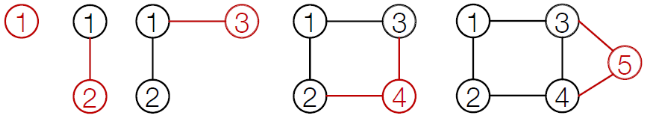
Graph Generation using GraphRNNs (cont.)

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


Graph G with node ordering π:
Sequence S^π:
1
2
3
4
5
$S^\pi = (S_1^\pi , S_2^\pi , S_3^\pi , S_4^\pi , S_5^\pi)$

Graph Generation using GraphRNNs (cont.)

- 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.

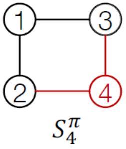


$$S^\pi = (S_1^\pi , S_2^\pi , S_3^\pi , S_4^\pi , S_5^\pi)$$

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

Graph Generation using GraphRNNs (cont.)

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



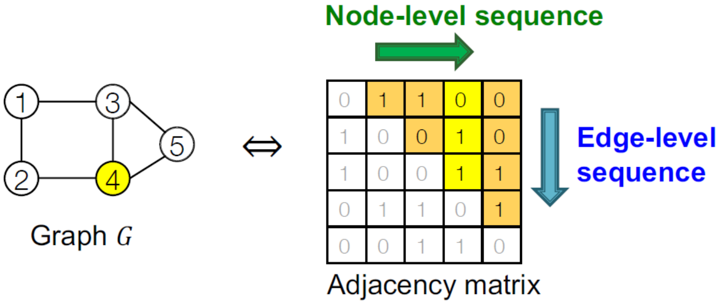
$$S_4^\pi = (\quad S_{4,1}^\pi \quad , \quad S_{4,2}^\pi \quad , \quad S_{4,3}^\pi \quad)$$

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

0 1 1

Graph Generation using GraphRNNs (cont.)

- Summary: A graph + a node ordering = A sequence of sequences
- Node ordering is randomly selected.



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

Graph Generation using GraphRNNs (cont.)

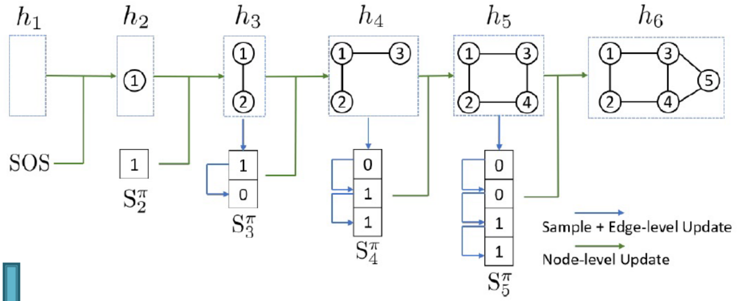
- We have transformed graph generation problem into a sequence generation problem.
- Need to model two processes:
  - Generate a state for a new node (Node-level sequence).
  - Generate edges for the new node based on its state (Edge-level sequence).
- We can use Recurrent Neural Networks (RNNs) to model these processes.

Our Plan

- **Add a new node:** We run Node RNN for a step, and use it output to initialize Edge RNN.
- **Add new edges for the new node:** We run Edge RNN to predict if the new node will connect to each of the previous node.
- **Add another new node:** We use the last hidden state of Edge RNN to run Node RNN for another step.
- **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.

Our Plan (cont.)

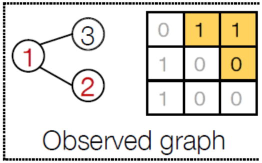
Node-level RNN generates the initial state for edge-level RNN

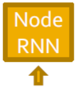


Edge-level RNN sequentially predict if the new node will connect to each of the previous node

Training

Assuming **Node 1** is in the graph
Now adding **Node 2**



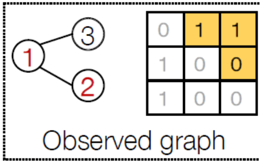


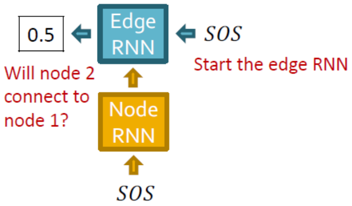
SOS

Start the node RNN

Training (cont.)

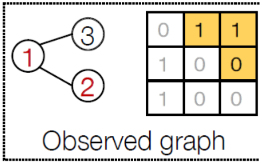
Edge RNN predicts how
Node 2 connects to **Node 1**

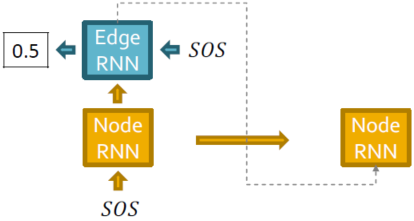




Training (cont.)

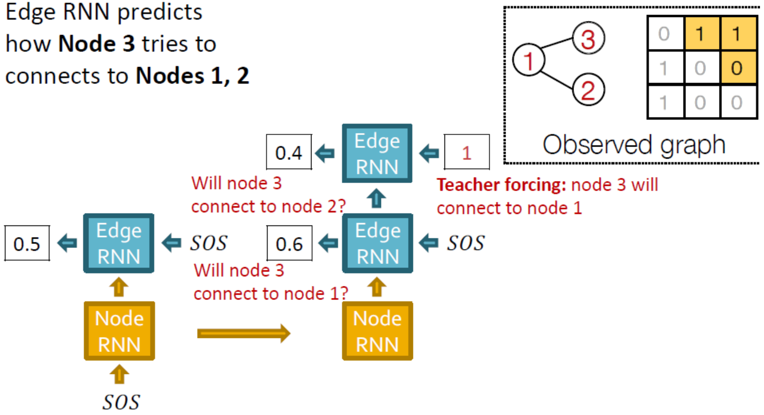
Update Node RNN using **Edge RNN's hidden state**





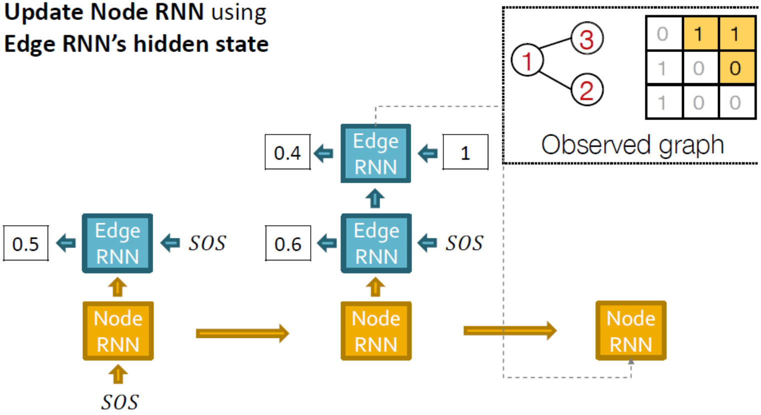
Training (cont.)

Edge RNN predicts how **Node 3** tries to connects to **Nodes 1, 2**


Observed graph
1
2
3
0 1 1
1 0 0
1 0 0
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

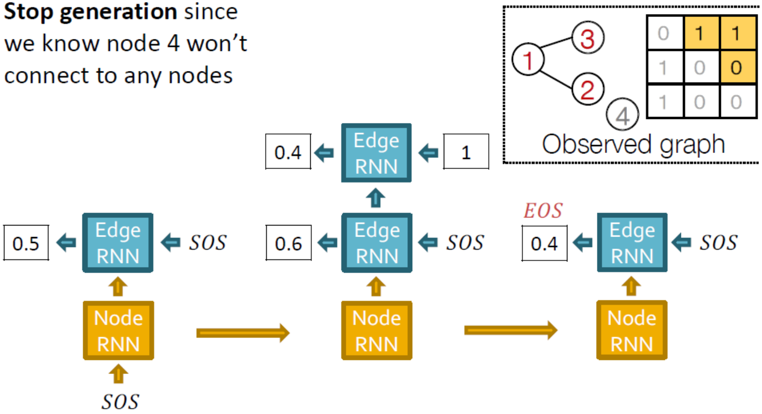
Training (cont.)

Update Node RNN using
Edge RNN's hidden state



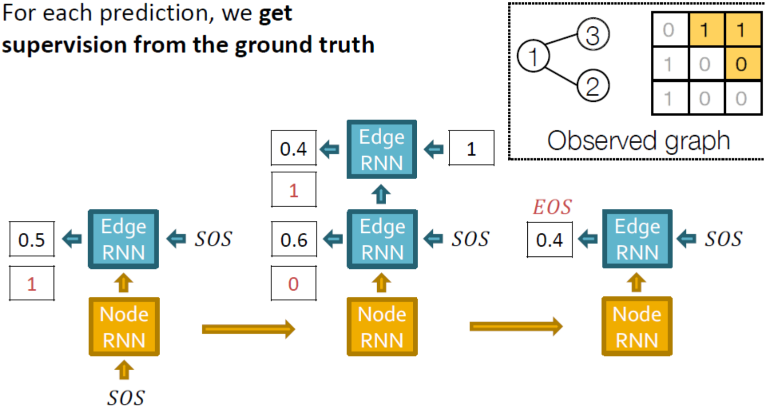
Training (cont.)

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



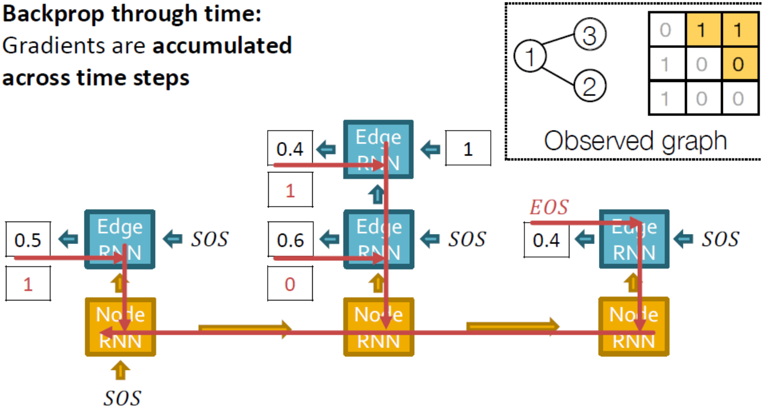
Training (cont.)

For each prediction, we **get supervision from the ground truth**



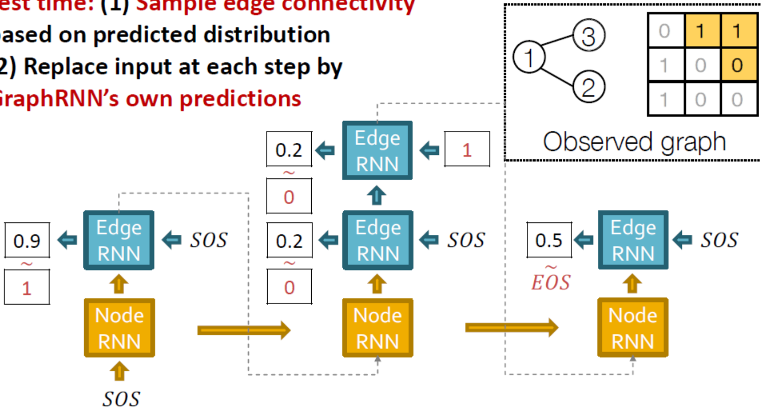
Training (cont.)

Backprop through time:
Gradients are **accumulated across time steps**



Testing

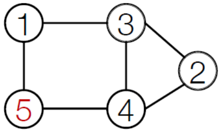
Test time: (1) Sample edge connectivity based on predicted distribution (2) Replace input at each step by GraphRNN's own predictions



0	1	1
1	0	0
1	0	0

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
- How do we limit this complexity?



Random node ordering:

"Recipe" to generate the left graph:

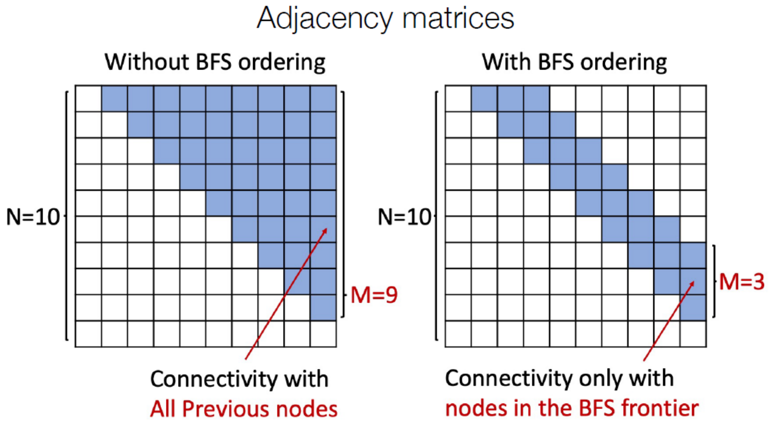
- Add node 1
- Add node 2
- Add node 3
- Connect 3 with 2 and 1
- Add node 4
- ...

Solution: Tractability

- We can employ BFS ordering for the nodes
  - 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.
- Benefits:
  - Reduces possible node orderings (From $O(n!)$ to number of distinct BFS orderings).
  - Reduces steps for edge generation (Reducing number of previous nodes to look at).

Solution: Tractability (cont.)

- BFS reduces the number of steps for edge generation.



Results

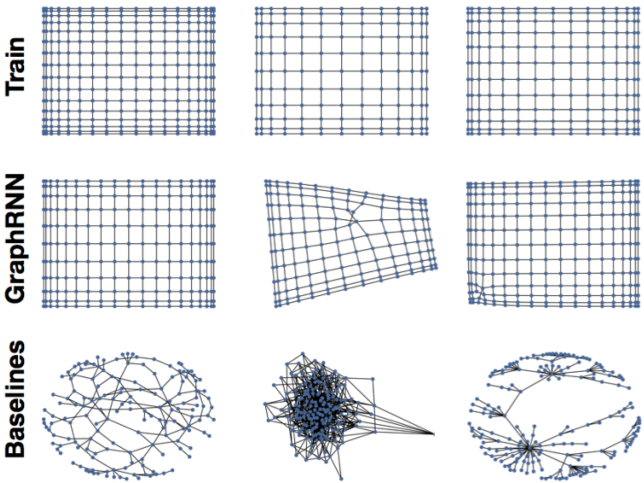


Figure 2: Visualization of graphs from Grid dataset generated by GraphRNN and some baseline methods (Kronecker, MMSB and B-A)

Results (cont.)

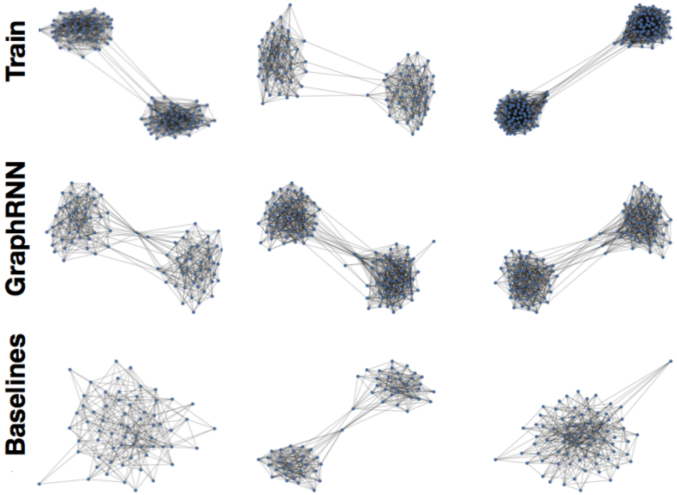


Figure 3: Visualization of graphs from Community dataset generated by GraphRNN and some baseline methods (Kronecker, MMSB and B-A)

Results (cont.)

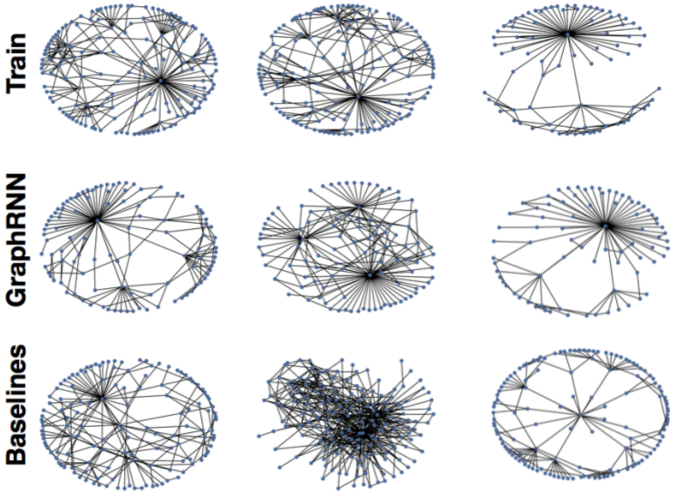
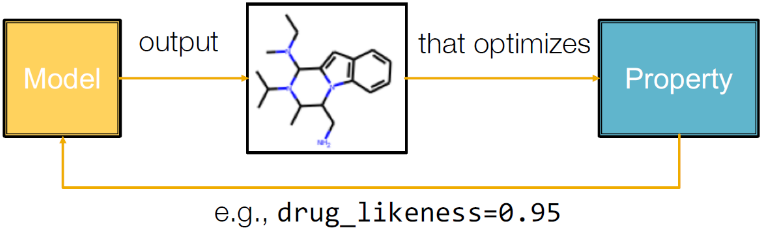


Figure 4: Visualization of graphs from Ego dataset generated by GraphRNN and some baseline methods (Kronecker, MMSB and B-A)

Goal Directed Graph Generation

- Can we learn a model that can generate valid and realistic molecules with optimized property scores?

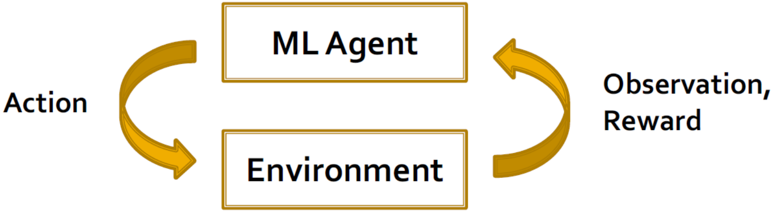


Solution: GCPN

- Graph Convolutional Policy Network (GCPN). It combines graph representation + RL.
- Key component of GCPN:
  - **Graph Neural Network** captures graph structural information.
  - **Reinforcement learning** guides the generation towards the desired objectives.
  - **Supervised training** imitates examples in given datasets.

Reinforcement Learning

- A ML agent **observes** the environment, takes an **action** to interact with the environment, and receives positive or negative **reward**.
- The agent then **learns** from this loop.
- **Key idea:** Agent can directly learn from environment, which is a blackbox to the agen

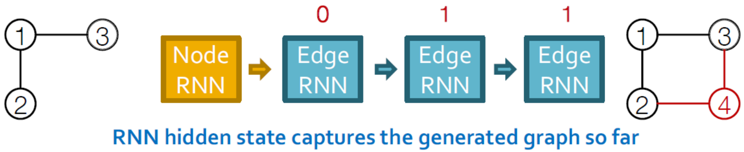


GraphRNN vs. GPCN

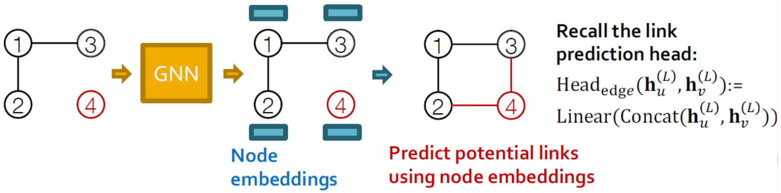
- Commonality of GCPN & GraphRNN:
  - Generate graphs sequentially.
  - Imitate a given graph dataset.
- Main Differences:
  - GCPN uses GNN to predict the generation action (GNNs are more expressive than RNNs but are more computationally expensive).
  - GCPN further uses RL to direct graph generation to our goals (this enables goal directed graph generation).

GraphRNN vs. GPCN (cont.)

- **GraphRNN**: predict action based on RNN hidden states.



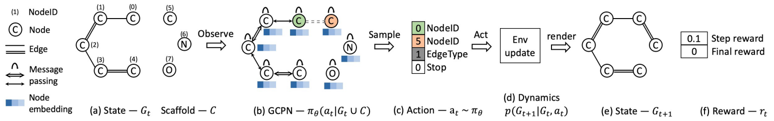
- **GCPN**: predict action based on GNN node embeddings.



Recall the link prediction head:

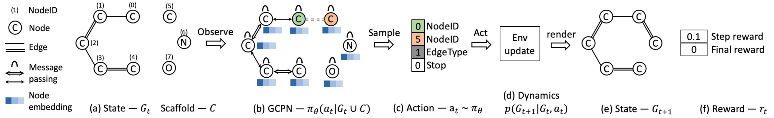
$\text{Head}_{\text{edge}}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}) :=$
$\text{Linear}(\text{Concat}(\mathbf{h}_u^{(L)}, \mathbf{h}_v^{(L)}))$

Overview of GCPN



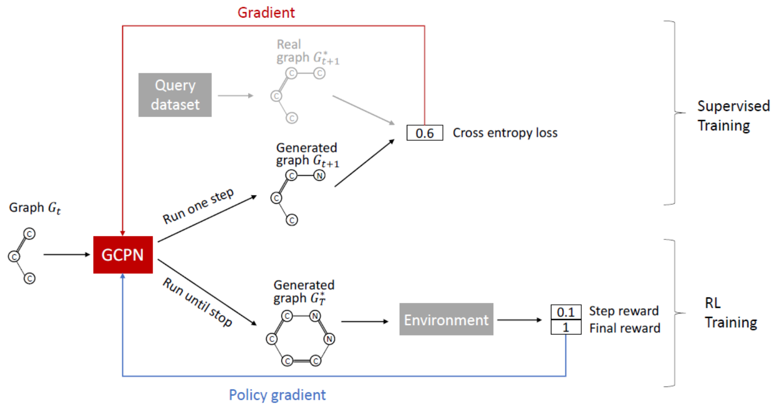
- (a) Insert nodes
- (b,c) Use GNN to predict which nodes to connect
- (d) Take an action (check chemical validity)
- (e, f) Compute reward

Rewards



- **Step reward:** Learn to take valid action.
  - At each step, assign small positive reward for valid action.
- **Final reward:** Optimize desired properties
  - At the end, assign positive reward for high desired property.
- **Reward = Final reward + Step reward**

Training GCPN



Results

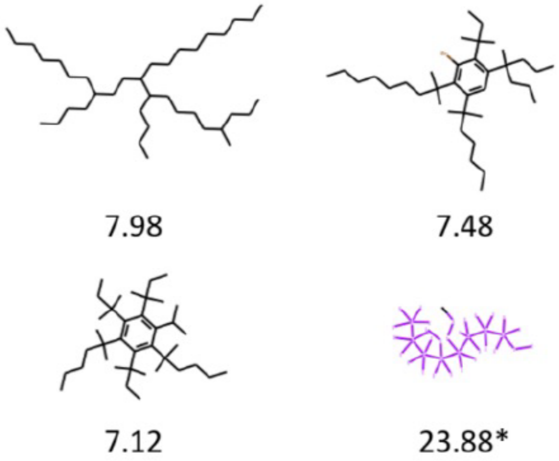


Figure 5: Property optimization: Generate molecules with high specified property score

Results (cont.)

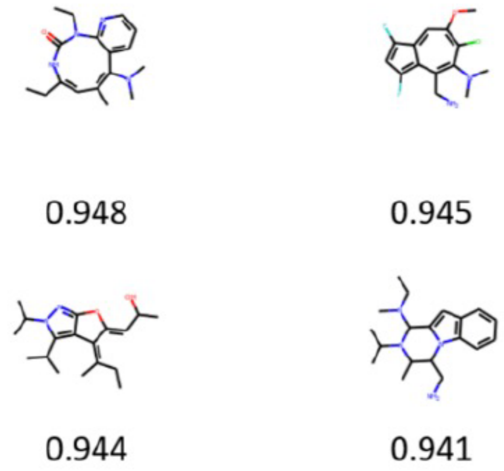


Figure 6: Property optimization: Generate molecules with high specified property score (Molecular Weight).

Results (cont.)

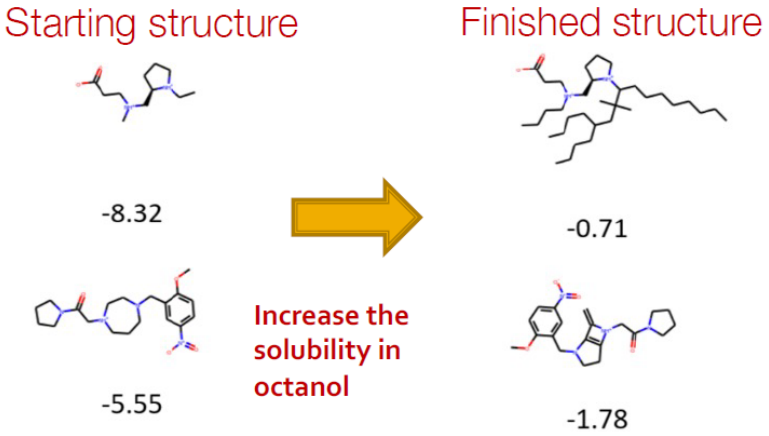


Figure 7: Constrained optimization: Edit a given molecule for a few steps to achieve higher property score (Molecular Weight).

Summary

- Complex graphs can be successfully generated via sequential generation using deep learning.
- Two of the ways that we can do this by leveraging RNNs or Reinforcement Learning.
- In GraphRNN, we sequentially predict nodes and then edge connections for these nodes.
- In GCPN (RL based method), the model predicts potential links using node embeddings.

References

These slides have been adapted from

- Jure Leskovec, Stanford CS224W: Machine Learning with Graphs
- You et al., GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models
- You et al., Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation