Graph Generative Model

Yujia Zhu

We can generate realistic graphs. By generating graphs, we can understand the formulation of graphs, predict how will the graph further evolve, use the same process to general novel graph instances and decide if a graph is normal.

Properties

We will characterize graphs by **Degree distribution**, **Clustering coefficient** and **Path length**. The generative graph should have similar characters with the original graph.

Degree Distribution Degree distribution is the probability that a randomly chosen node has degree k. Let N_k is the number of nodes with degree k. Then the degree distribution has the probability mass function $P(x)$:

$$P(k) = \frac{N_k}{N}, \quad k = 1, 2, ...$$

Clustering Coefficient Clustering coefficient measures how a node's neighbors are connected to each other. Suppose a node i has degree k_i and their exist e_i edges between the nodes of i's neighbours. The clustering coefficient of node i is

$$C_i = \frac{2e_i}{k_i(k_i - 1)}$$

The graph clustering coefficient is the average clustering coefficients of all the nodes in the graph, $C = \frac{1}{N}\sum_{i=1}^{N} C_i$.

Path Length We usually use **Diameter** and **Average Path Length** to reflect path length. Diameter is the maximum (shortest path) distance between any pair of nodes in a graph. Average path length is the average of path length in the graph.

Random Graphs

Random Graph G_{np} is an undirected graph with n nodes and each edge appears i.i.d with probability p. It can also be defined as graph $G(n, m)$, which means the graph has n nodes and m edges which will appear on the graph randomly.

The degree distribution of random graph will obey a binomial distribution. Let $P(k)$ denote the fraction of nodes with degree k, then

$$P(k) = C_{n-1}^{k} p^k (1-p)^{n-1-k}$$

So, the mean degree of the graph is $\bar{k} = (n-1)p$.

The expected clustering coefficiency of random graph is

$$E\left[C_i\right] = E\left[\frac{2e_i}{k_i(k_i-1)}\right] = \frac{2}{k_i(k_i-1)}E[e_i] = \frac{2}{k_i(k_i-1)}\frac{k_i(k_i-1)p}{2} = p = \frac{\bar{k}}{n-1} \approx \frac{\bar{k}}{n}$$

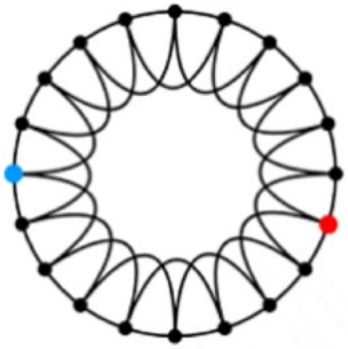
With the grow of p, the diameter of the random graph will first increase and then decrease. The turning point of the probability p is called *phase transition point*. We can prove that when the average degree $\bar{k} = 1$ or $p = \frac{1}{n-1}$, the turning point emerges. The average path length of the random graph is $l \approx \frac{\ln|V|}{\ln \bar{k}}$.

The problems of random graph include:

- The degree distribution may differ from that of real networks. The real network's degree distribution often obeys to the power law (that means $P(k) = ak^{-b}$) but random graph's degree distribution obeys to the binomial distribution.
- Clustering coefficient of random graph is relatively low so the local structure of the graph can not be reflected in the simulated graph.

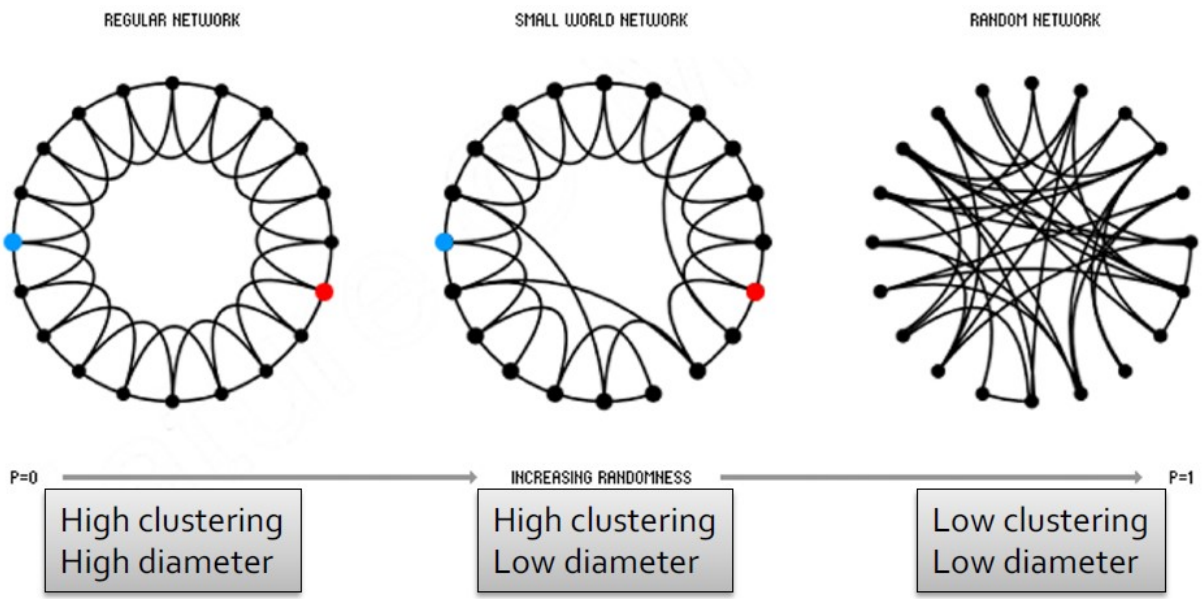
The Small World Model

Small world model has high clustering coefficients and short paths. Before introducing small world model, we need to have a knowledge of ring lattice network. As is shown in the following figure, ring lattice network is a tightly-connected network. Every node in the graph will be only connected to the nearest c nodes (c is an even number). In other words, if we sort nodes in a certian order, every node will only connect to the past $c/2$ nodes and following $c/2$ nodes.



For the ring lattice network, the degree distribution is a sigle point distribution with $P(k = c) = 1$. The clustering coefficient of the ring lattice network is $\frac{3(c-2)}{4(c-1)}$. So, the clustering coefficient will converge to $\frac{3}{4}$ when c is large. The average path length of ring lattice network is $\frac{n}{2c}$. Compared with random graph, the clustering coefficient and average path length of ring lattice network are much larger.

The small world model adds some randomness to the ring lattice network. For each time, we random choose an edge in the graph, with probability p, we move the endpoint to a random node. By performing this operation many times, we will get a small world network. The small world network has high clustering coefficient and low diameter. The relationship among ring lattice network, small world network and random network is shown in the following figure.



Kronecker Graph Model

Many graphs have a character that they have many similar subparts/subgraphs. **Kronecker product** is a way of producing self-similar metrics. Suppose we have two matrixs A and B. The dimension of A is $n \times m$, while B's dimension is $k \times l$. The Kronecker product of A and B is defined as

$$C = A \bigotimes B = \begin{pmatrix} a_{11}B & a_{12}B & ... & a_{1m}B \\ a_{21}B & a_{22}B & ... & a_{2m}B \\ ... & ... & ... & ... \\ a_{n1}B & a_{n2}B & ... & a_{nm}B \end{pmatrix}$$

C is a $nk \times ml$-dimensional matrix. Then we can define a kronecker product of two graphs as a kronecker product of their adjacency matrixs. Kronecker graph obtained by growing sequence of graphs by iterating the kronecker product over the initiator matrix K_1. That means the adjacency matrix of the m-th order kronecker graph is

$$K_1^{[m]} = K_m = K_1 \bigotimes K_1 \bigotimes ... \bigotimes K_1 = K_{m-1} \bigotimes K_1$$

We can generate directed stochastic kronecker graph by the following steps:

1. Create $N_1 \times N_1$ **probability matrix** Θ_1
2. Compute the k-th kronecker product of Θ_1, namely Θ_k
3. For each element/probability p_{uv} of Θ_k, add a edge (u, v) with probability p_{uv}

GraphRNN

Before we have talked about traditional graph generative models, we will dive into deep graph generative model now. GraphRNN is a representative of deep graph generative models.

Assume we want to learn a generative model from a set of data points (i.e. graphs) $\{x_i\}$. We have two probability distributions:

- $p_{data}(x)$: the data distribution, which is never known to us, but we have sampled $x_i \sim p_{data}(x)$.

- $p_{model}(x, \theta)$: the model, which is use to approximate $p_{data}(x)$. θ is the parameter of the distribution. We need to make sure that $p_{model}(x, \theta)$ is close to $p_{data}(x)$ and we can sample from $p_{model}(x, \theta)$.

We can use maximum likelihood to make $p_{data}(x)$ close to $p_{model}(x, \theta)$. Then, we can find the best θ^* by

$$\theta^* = arg \max_{\theta} E_{x \sim p_{data}} \log p_{model}(x|\theta)$$

We can sample from the model by the following approachs:

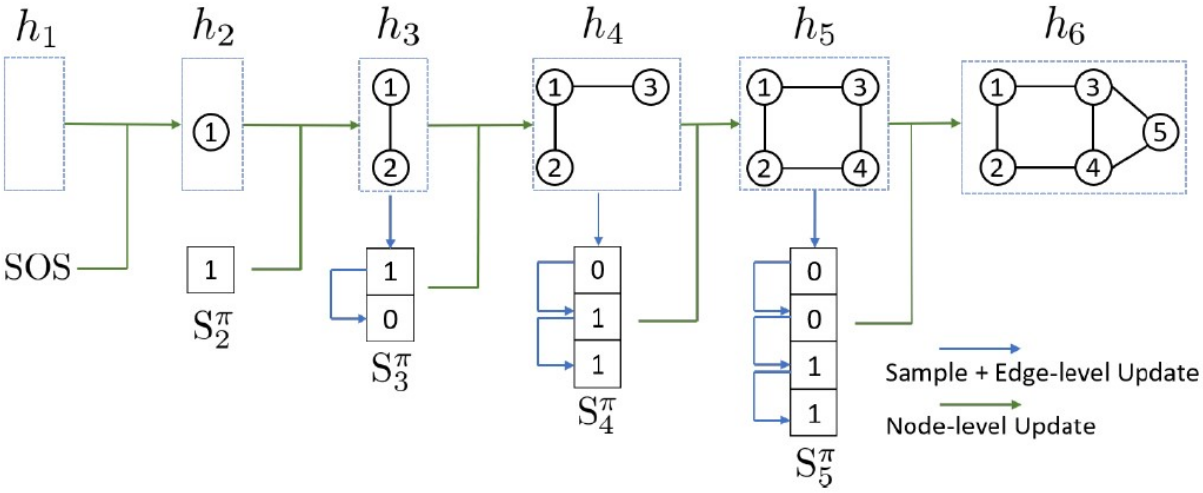
1. Sample from a simple noise distribution: $z_i \sim N(0, 1)$
2. Transform z_i to a complex graph generative distribution x_i by function $f(.)$: $x_i = f(z_i)$

GraphRNN uses a Recurrent Neural Network to design $f(.)$. GraphRNN generates graphs via sequentially adding nodes and edges. So, we can model the process of graph generation as a sequence. For an example, $S^\pi = (S_1^\pi, S_2^\pi, S_3^\pi)$, then S_1^π may be adding node 1, S_2^π may be adding node 2 and connecting node 1 and node 2 and S_3^π may be adding node 3 and connecting node 1 and node 3. Then a graph with three nodes and two edges is generated. So, the sequence S^π has two levels:

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

GraphRNN uses two RNNs to model node level and edge level processes.

1. Use **Node-level RNN** to generate a new node and the initial state for edge-level RNN
2. Use **Edge-level RNN** to sequentially generate edges for the new node based on its state



As is shown in the above figure, if we have already trained the model, we can generate graphs by the following process.

1. Give a start of sequence token (SOS) to the Node-level RNN to generate a hidden state.
2. Give the hidden state produced by Node-level RNN to the Edge-level RNN. Then use Edge-level RNN to sequentially predict the probability of whether the new node will connect to the following nodes.(For example, suppose we just add node 5, the edge-level rnn will sequentially output the probability of whether node 5 will connect to node 1,2,3,4). By the output probabilties, we simulate the edges.
3. Give the output of Edge-level RNN to the input of Node-level RNN. And then continue to generate nodes and edges by two RNNs.
4. End the graph generation until the Node-level RNN outputs the end of sequence token (EOS).

We can train GraphRNN easily by binary cross entropy loss function and back propagation.

Summary

In this file, we cover traditional graph generative methods and deep generative methods.

For traditional generative methods, we use some graph properties to supervise/evaluate the generation of new graphs. Critical traditional methods include Random Graph, Small World Graph and Kronecker Graph.

For deep generative methods, a representative is GraphRNN. It use two RNNs to sequentially generate nodes and edges.