Machine Learning with Graphs (MLG)

Homework 1 Report

吳承霖 成功大學電機所己組 N26090693

1 INTRODUCTION

1.1 Motivation

In modern society, concepts behinds human activities in the internet world can be abstracted as a graph. For example, people's relationship in social media websites and the relationship between consumers and products in eCommerce platforms can be viewed as a large graph, each person / product stands for a node, which contains information including self information like user's age or product price. The links between nodes in social media graph can be applied on representing the friendship or post sharing between users.

Thus, analysis on graph became key to get insight from the content of these human activities. Thanks to the widespread of the internet and computing resource, today it's getting easier and easier to collect data from the internet platform's database or IoT sensing network.

1.2 Why betweeness centrality is important

In graph theory, betweenness centrality is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex.

Betweenness centrality was devised as a general measure of centrality, it applies to a wide range of problems in network theory, including problems related to social networks, biology, transport and scientific cooperation.

Betweenness centrality finds wide application in network theory; it represents the degree to which nodes stand between each other. For example, in a telecommunications network, a node with higher betweenness centrality would have more control over the network, because more information will pass through that node.

1.3 Current challenge of evaluating BC of a node

With the information exploding accelerated by internet technologies, the graph behind people's activities become extremely huge. Some old algorithms for analyzing graph centrality and routes will cost expensively even though the performance of computing hardwares has been doubled every year.

For example, according to the statistics of Facebook 2020, there are over 2.4 billion monthly active users on Facebook platform, that means that over 2.4 billion nodes in Facebook graph may be highly dynamic.

These nodes in the graph will change their degrees (build relationship with others or unfriend others) frequently and it's hard to globally compute the exact value of between centrality of nodes.

In order to compute between centrality of nodes, we have to calculate the shortest paths of all node pairs. A typical algorithm to calculate is Floyd–Warshall algorithm, which takes time complexity of $\Theta(|V|^3)$. It is very impractical to applied on large graph like graph behind Facebook. Additionally, because that the algorithm calculate shortest paths based on whole graph information (globally), once links between some nodes become disconnected, we have to take another $\Theta(|V|^3)$ time to recalculate the shortest paths and compute betweenness centrality of nodes.

1.4 DrBC: A Deep Learning Approach

Instead of using global algorithm to calculate the betweenness centrality of graph nodes, DrBC is an approach for predicting betweenness centrality of target node based on self information of the target node and information of the node's neighborhood. In comparison to traditional algorithms which calculate globally, DrBC model predicts locally and thus takes much fewer time to compute. The advantage of efficiency makes DrBC is suitable to be applied on large graphs behind internet platforms which has content changing frequently.

2 ALGORITHM AND ARCHITECTURE

DrBC using network of Deep Learning to predict betweenness centrality of nodes, the following sections will illustrate the model architecture and objective function for training as well as other implementation details.

2.1 Network Architecture

DrBC network is mainly made up of encoder part and decoder part. The encoder is responsible to gather node's self information, such as degree, as input features, then combine the input features with neighboring nodes' information to encode into an embedding vector. The decoder is a multilayer perceptron(MLP), it uses the embedding vector generated from encoder to generate final node value. The final value returned by the decoder is a scalar which represent the node's predicted betweenness centrality. Figure 1 illustrates the conceptual architecture of DrBC network.

The initial input vector is create by node's degree, which has dimension of [1, 3]. For example, if a node has degree d, then the input vector of the node will be [[d, 1, 1]]. The two "1" values in the input vector doesn't stand for any reason, and it can be replace by other node's information like price of product (if the node represents a product in eCommerce graph) or age of user (if the node represents a human in social media graph).

The Input vector then will be converted to an vector with length 128 by a fully-connected layer, after going through 2 GRU Cells, the vector will be combined with neighborhood information and be encoded into two embedded vector. In the element-wise max pooling layer, it operates $max(h_v^{(1)}, h_v^{(2)}, h_v^{(3)})$ to select the most informative (using max operator) layer for each feature coordinate and return the selected layer-aggregated vector to the decoder part.

The decoder is simply made up of two fully-connected layer to enable the model has the ability to fit the model weights to generate more correct result.

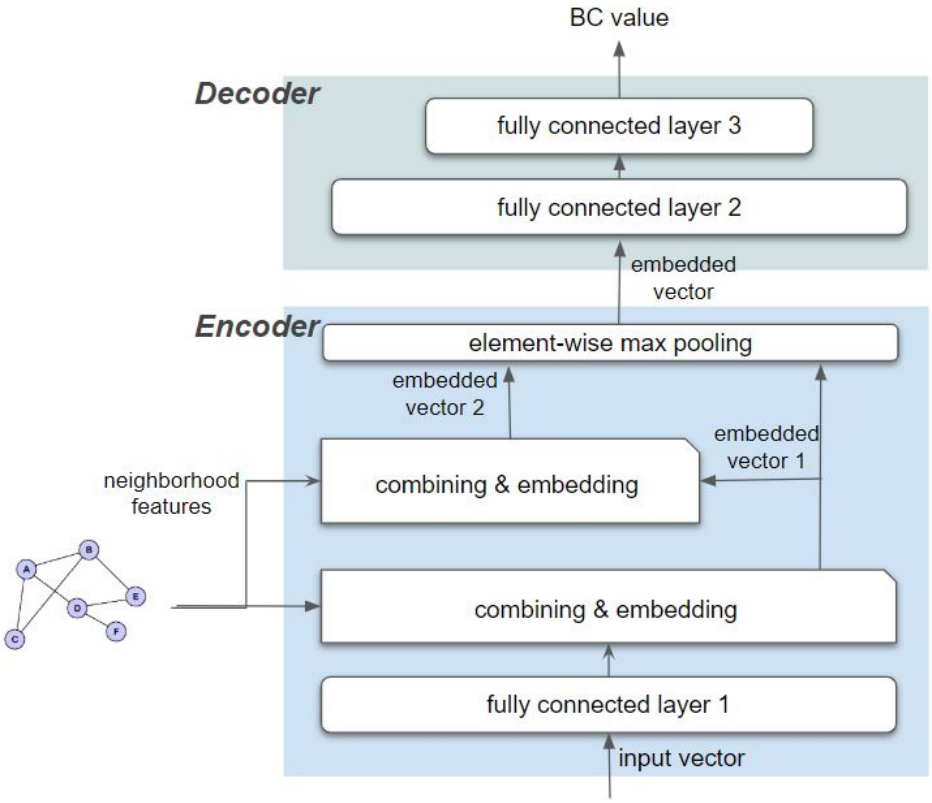

Figure 1: Illustrate conceptual architecture of DrBC network model, which consists of 2 component: decoder and endoder.

2.1.1 Neighborhood Aggregation and Combination

In the layers of combining and embedding illustrated in Figure 1, the input vector of nodes will be aggregation will node's neighboring information including its direct neighboring node's input vector. After summing up all neighboring information and normalized by neighboring nodes degree, the summed neighboring information will be combined with our target node's input vector and generated a new embedded vector.

The procedure of neighborhood aggregation is recurrent, thus we applied RNN (GRUCell) techniques to build a series of neighborhood aggregation layers. GRUCells(Gated Recurrent Unit Cell) in the model connected sequentially and make use of different layer of neighboring information aggregation. The mathmatic operations inside a GRUCell is:

$$
\begin{aligned}
r &= \sigma(W_{ir}x + b_{ir} + W_{hr}h + b_{hr}) \\
z &= \sigma(W_{iz}x + b_{iz} + W_{hz}h + b_{hz}) \\
n &= \tanh(W_{in}x + b_{in} + r * (W_{hn}h + b_{hn})) \\
h' &= (1 - z) * n + z * h
\end{aligned}
$$

where σ is the sigmoid function, and ∗ is the Hadamard product.

2.2 Data Synthesizing

Because we only take node degree as input feature, we can use synthetic graph generated by ourself to train the DrBC model and predict betweenness centrality of node in read world graph. This type of training method is called Inductive Learning, which is a very efficient way to build usable Deep Learning model because it the work of collecting real world dataset unnecessary.

In the DrBC project, we use Python package 'networkx' to generate synthetic graph to train our DrBC model. To reach and simulate the graph in real world, the generated graphs follow power-law distribution, at least asymptotically. That is, the fraction P(k) of nodes in the network having k connections to other nodes goes for large values of k as:

$$P(k) \sim k^{-\gamma}$$

where γ is a parameter whose value is typically in the range $2 < \gamma < 3$.

2.3 Optimization Objective

Unlike many other prediction algorithm which are responsible to predict a accurate value, predicting betweenness centrality of a node in graph doesn't need the predicted score to be as close to target score as possible. In real-world application, the requirement of knowing nodes betweenness centrality is arised from the need of knowing which nodes have higher BC values than others, says, "more important" than other nodes. These information is useful in applying recommendation system in graph network and other plafforms have graph-like contents.

Due to the reason of meeting real-world applications' needs, the optimization objective is focused on comparing the 'relative' ranking of BC values among nodes instead of calculating the actual distance of score between predicted betweenness centrality values and target betweenness centrality values.

2.3.1 Pairwise Loss

The first approach to calculate loss is pairwise loss: Given a node pair (i, j), suppose the ground truth BC values are bi and bj, respectively, our model predicts two corresponding BC ranking scores yi and yj. Given bij ≡ bi − bj, since we seek to preserve the relative rank order specified by the ground truth, our model learn to infer yij ≡ yi − yj based on the following binary cross-entropy cost function:

$$Ci,j = -g(b_{i,j}) * log\, g(y_{i,j}) - (1 - g(b_{i,j})) * log(1 - g(y_{i,j}))$$

where $g(x)$ is sigmoid function:

$$g(x) = \frac{1}{1 + e^{-x}}$$

And the loss is defined as:

$$Loss = \sum_{i,j \in V} C_{i,j}$$

The pair number of whole nodes will be $|V|^2$. If we use all pairs of node to calculate loss will lead to expensive cost, thus we use sampling method to pick pairs from the node.

2.3.2 Softmax Loss

Beside pairwise loss, we adapted softmax as another method to calculate the relative ranking and get error between predicted BC values and target BC value.

In mathematics, the softmax function is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1, but after applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.

The standard softmax function $\sigma : R^k \rightarrow R^k$ is defined by the formula:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

And the loss function adapted by DrBC project is:

$$\sum_i \left(\left| \sigma(z)_i - \sigma(y)_i \right| \right)$$

Whcih $\sigma(z)_i$ represents the ranking score of predicted BC value of node i, and $\sigma(y)_i$ represents the ranking score of target (ground truth) value of node i.

2.4 Evaluation Metric

Like the optimization object mentioned in previous sections, the evaluation is focused on the relative ranking of nodes' betweenness centrality values rather than absolute distance between the predicted and ground truth.

Furthermore, because people making strategies on graphs often care about the most important nodes more, we use another approach to calculate 'Top-N% Accuracy' (N can be 1, 5, 10), which is the accuracy of correctly predicting the top N% nodes.

2.4.1 Kendall tau distance

The Kendall tau rank distance is a metric that counts the number of pairwise disagreements between two ranking lists. The larger the distance, the more dissimilar the two lists are. Kendall tau distance is also called bubble-sort distance since it is equivalent to the number of swaps that the bubble sort algorithm would take to place one list in the same order as the other list.

In the project we use package 'scipy' to help calculating Kendall tau distance, the formula is:

$$Kendall\ tau = \frac{(P - Q)}{\sqrt{(P+Q+T) * (P+Q+U)}}$$

where P is the number of concordant pairs, Q the number of discordant pairs, T the number of ties only in x, and U the number of ties only in y. If a tie occurs for the same pair in both x and y, it is not added to either T or U.

2.4.2 Top-N% Accuracy

Top-N% accuracy is defined as the percentage of overlap between the top-N% nodes as returned by our DrBC model and the top-N% nodes of ground truth:

$$\text{Top-}N\% = \frac{|\{\text{returned top-}N\%\text{nodes}\} \cap \{\text{true top-}N\%\text{nodes}\}|}{\lceil |V| \times N\% \rceil}$$

where $|V|$ is the number of nodes in graph. In this project we evaluate on Top-1%, 5% and 10% accuracy.

2.5 Validation

Beside the evaluation of final prediction on testing data, we also implement validation in the training procedures. Like the synthetic test data fed in DrBC model, we used the same parameters of distribution to generate graphs which follow power-law.

The purpose of doing validation during training is to find the proper training settings to avoid overfitting on training data without touching the testing data.

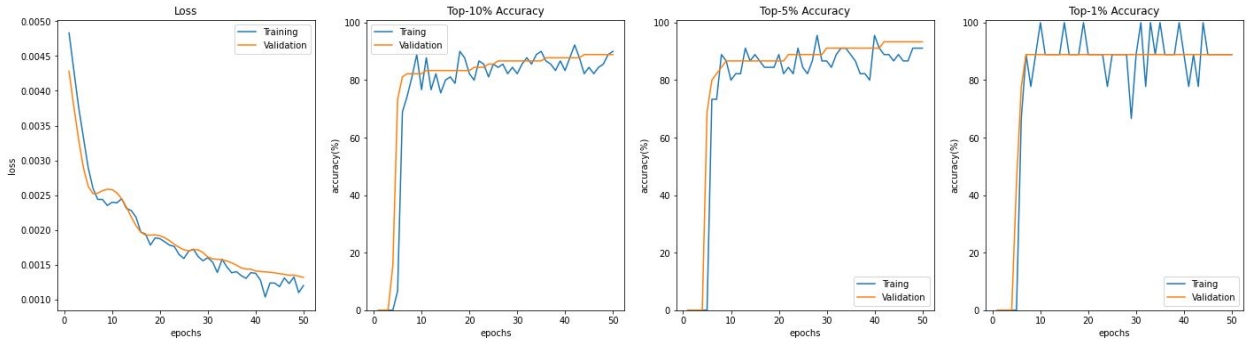

Figure 2: The plot of training and validation's loss / accuracy.

3 EXPERIMENTS

During DrBC project development, we applied multiple different sets of parameter to test the model's ability and feature. The following sections will illustrate the detailed comparison results.

3.1 Environment

The project is developed on Google Colab, it allows you to write and execute Python in your browser, with:

- Zero configuration required
- Free access to GPUs
- Easy sharing

The types of GPUs that are available in Colab vary over time. The GPUs available in Colab often include Nvidia K80s, T4s, P4s and P100s. And the memory size of Colab VM is approximately 32GB.

3.2 Number of Epoch per Training Graph

In training process of DrBC model, the training time and number of graph fed into model to train are positively related. Before a new graph being fed into model to train, the adjacency look-up matrix and aggregation weights of all node pairs look-up matrix.

These calculation steps are highly time-consuming, and we conducted a series of experiments using different epochs per training graph and different graph numbers, and try to get insight of the relationship between these setting and the model's ability of generalization.

The following two testing used the fixed graph node number, which set to 500.

First, we set training graph number to 10, and set the epoch number per training graph to 10, so the total training epoch will be $10 * 10 = 100$, the loss along epochs are plotted in following Figure 3:

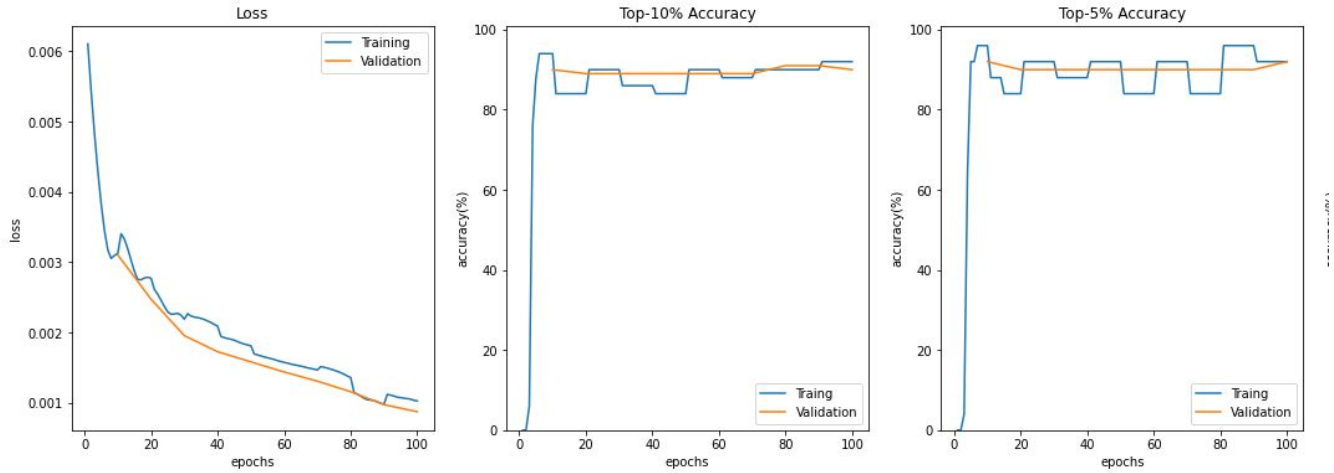

Figure 3: Loss of model training, using 10 graphs to train and 10 training epochs per graph

Then we use another setting to train a new model: training graph number = 100, train 1 epoch per graph. The total training epoch will be $100 * 1 = 100$, which is the same as previous test.

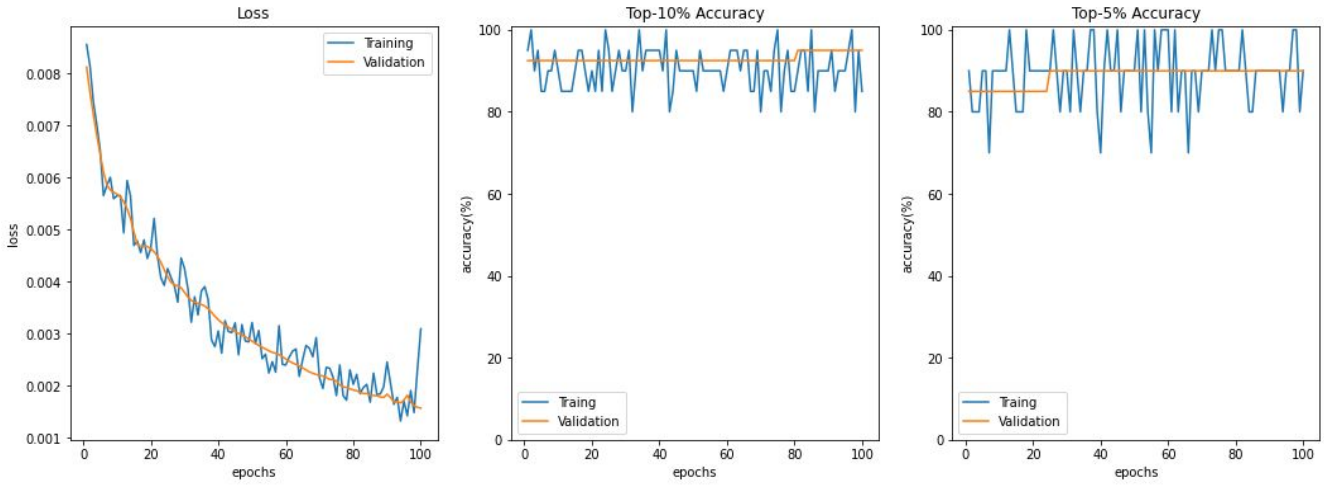

Figure 4: Loss of model training, using 100 graphs to train and 1 training epoch per graph

From the test result listed in Figure 3 and Figure 4, we noticed that the ability of generalization is not proportional to the number of training graph. Beside the loss, the accuracy of top-10% and top-5% on validation are similar among these two tests.

The reason behinds the testing result is because all the training graphs and validation graph are generated from the same power-law distribution parameters, so the degree distribution of nodes in each graph is similar, so that adding more graph into training will not power up the ability of generalization of the model.

The experiment results also support our knowledge that the approach of Inductive Learning can be applied on Deep Learning problems of graphs. We can use relatively small amount of graph to train our model and have a decent accuracy on predicting real-world graph nodes.

3.3 Top-N% accuracy of synthetic graphs of different scales

To test if our model have ability to be applied on large scale of graph, we used different size of graph for prediction and measure the accuracy of top-1%, top-5% and top-10%

Scale	Top-1%	Top-5%	Top-10%
5K	96.00	81.6	80.6
100K	92.00	82.4	82.8

From the table of result, we can see that the accuracy of top-1% is decrease in larger scale of graph, however, the accuracy of top-5% and top-10% is slightly higher than those in small scale. The possible reason behind it may be caused by the number of 100K test graph. Although the degree distribution among many social network graph are similar, but putting more graphs into testing and use mean value of the prediction accuracies will lead to more accurate result.

3.4 Kendall tau distance of synthetic graph of different scales

Unlike Top-N% accuracy which focused on the most important only. Kendall tau distance calculate the whole prediction loss comparing with ground truth.

Scale	Kendall tau Distance
5K	60.70
100K	70.07

Higher value of Kendall tau distance means that the correlation between prediction and ground truth is higher (if prediction result is the same as ground truth, the Kendall tau distance will be 1). We might expect that the Kendall tau distance of prediction on large scale graph will be lower, but, according the experiment result, the Kendall tau distance of prediction on graph with 100K nodes have higher Kendall

tau distance. We infer that that the possible reason is also caused by the number of 100K graph is too low to get the mean Kendall tau distance.

Although we can use package ‘networkx’ to generate graph with over 100K nodes, but the computation of graph’s exact betweenness centrality will become a heavy task.

3.5 Running Time comparison on synthetic networks

The prediction test is conducted on Google Colab environment, which provides a virtual machine with GPU, here is the running time running on different scale of graphs:

Scale	Running Time (s)
5K	0.018
100K	6.234

The running time is less than the testing result on original paper. The main reason is that our computing is powered by Google’s GPU, which have much more ability to handle the matrix computation. And the testing in the original paper is conducted by CPU only.

3.6 Kendall tau distance generalization results on different scales

We are curious about the ability of models trained using different size of graph, here we uses graphs with node size 200, 300, 1000, 2000, 4000 to train respectively, then conduct testing on small graph (node size 5000) and large graph (node size 100K) to get Kendall tau distance as model ability:

Train \ Test	5K	100K
200	57.42	58.54
300	60.70	61.66
1000	58.65	60.29
2000	69.09	70.07
4000	67.49	68.49

According to the experiment result, which is not surprising, the models trained by smaller graphs tend to have lower Kendall tau distance, and the models trained by larger graphs can predict better on both small and large graphs.

3.7 Top-N% accuracy & running time on real-word data

Here we used provided com-youtube data to conducting testing:

Scale	Top-1%	Top-5%	Top-10%	Running Time (s)
Youtube	92.00	82.4	82.8	6.234

We can notice that the accuracy in Top-5% and Top-10% is lower. One possible answer we can infer is that the error is not distributed uniformly, so that the ranking of 'least important' nodes will have more loss.

4 CONCLUSION

In this project we implemented the DrBC network to predict betweenness centrality of nodes in a graph. After training with different set of setting, we found that Deep Learning is really an efficient approach to analyzing information of a graph, which is a significant problem essential to many application but the older algorithms compute expensively.

In addition to the efficiency, DrBC network is also highly scalable. We can apply this algorithm into any scale of graph and completed the prediction task in an acceptable range of time. In the experiment section we also demonstrated that models trained with small graphs could also be used at predicting BC value in large scale graph with acceptable accuracy.

4.1 Future Work: Apply on Real-World Applications

Before a DrBC model start to train BC values of nodes in graph, a adjacency look-up matrix and a aggregation weights look-up matrix have to be calculated in order to proceed the neighborhood aggregation. However, nowadays links of graphs behind the internet applications are often change frequently. We have to develop a efficient way to dynamically adjust those look-up matrix once the system detects some changes happening in the graph. This is the key to enable the internet applications to perform graph analysis in real-time.

6 REFERENCES

Changjun Fan, Li Zeng, Yuhui Ding, Muhao Chen, Yizhou Sun, Zhong Liu. 2019. Learning to Identify high Betweenness Centrality Nodes from Scratch: A novel Graph Neural network Approach. In CIKM'19