Learning to Identify High Betweenness Centrality Nodes from Scratch: A Novel Graph Neural Network Approach

JIA LE THIAN, CALEB

20 March 2023

1 Introduction

This project aims to implement DrBC(Deep ranker for betweenness centrality)[2] on 31 networks, including 30 networks from Synthetic data and 1 real-world data from youtube. For networks in Synthetic, each contains 5000 nodes, each with 19980-19984 edges. For the real-world data com-youtube.txt, it contains 1134890 nodes and 2987624 edges.

DrBC is a Graph Neural Network method that finds the relative betweenness centrality between nodes, making the algorithm faster by ignoring the calculation of the betweenness centrality (BC) of the nodes. The evaluation metrics used include top-N% accuracy, Kendall tau distance, and running time. The results are compared with the baseline method KADABRA[1], but because KADABRA only outputs the top-N% nodes, Kendall tau distance does not be compared.

2 Limitation

The experiments of DrBC were originally conducted on an 80-core server with 512 GB memory, and 8 16GB Tesla V100 GPUs, while my experiments are conducted on a 14-core notebook with 16 GB memory, and 1 6GB NVIDIA GeForce RTX3060 Laptop GPU. This results in differences in the implementation which will be introduced later.

3 Methodology

The DrBC model follows an encoder-decoder framework, as shown in Figure 1. The model includes 2 components. The first component is the encoder which generates node embeddings. The second component is the decoder which maps the embeddings to scalars to rank the nodes specified by BC values.

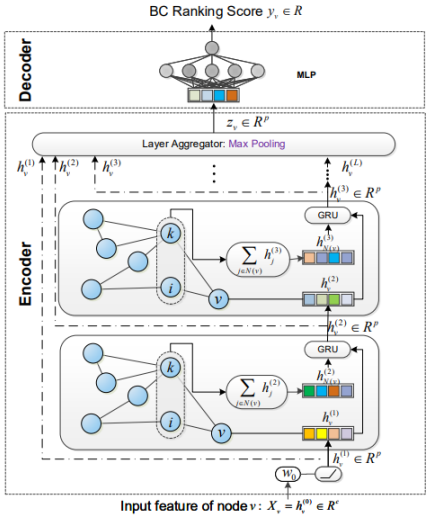

Figure 1: The encoder-decoder framework of DrBC

3.1 The Implementation of the Encoder

The encoder function takes a network graph as its input. The model implemented is according to the design.

At first, create the initial feature h_0 as a tensor with shape $(n, 3)$, each row corresponds to a node, and the entry of each row is the degree of the corresponding node. The other elements of the row are filled by 1. h_0 will undergo normalization before input into the model. This generating operation is defined in *gen_nodes_feature()*. The encoder model which is defined as *DrBCEncoder()* is constructed by a *nn.Linear* layer with *nn.ReLU*, and 5 *nn.GRUCell* taking last layer node features and corresponding neighbor nodes aggregation defined in the paper[2], which is

$$h_{N(v)}^{l} = \sum_{j \in N(v)} \frac{1}{\sqrt{d_v + 1} d_j + 1} h_j^{l-1}$$

, where l is the current layer, $N(v)$ is the neighbor nodes of node v, and d_i is the degree of the node i. Normalization by implementing *nn.BatchNorm1d* is done on the node features after completing each layer. All the node feature outputs in each layer are stored and compared after finished all the layers. The max operator (*torch.max*) is used to select the most informative layer for each feature coordinate, and output as the node embedding z_v. Notice that the hidden node size is set as 128 which remains unchanged in hidden layers, and same as the original implementation[2].

3.2 The Implementation of the Decoder

The decoder is implemented with a two-layered multilayer perceptron (MLP) by *nn.Linear* with normalization *nn.BatchNorm1d*, and a *nn.ReLU* between the two layers. This MLP maps the embedding generated by the encoder z_v to the approximate BC ranking score y_v, which is a scalar.

3.3 The Implementation of the Loss

The loss implemented is a pairwise ranking loss. To calculate this loss, the first is to sample random pairs of nodes. Totally $5|V|$ of random pairs are generated. For each node pair (i, j), calculate $b_{ij} \equiv b_i - b_j$ where b_i is the BC of node i, and $y_{ij} \equiv y_i - y_j$. Then binary cross-entropy cost function is used, note that before calculating the cost function,b_{ij} and y_{ij} undergo *nn.Sigmoid*. The cost function is then becomes $C_{i,j} = -\sigma(b_{ij}) \times \log \sigma(y_{ij}) - (1 - \sigma(b_{ij})) \times \log(1 - \sigma(y_i j))$. Thus, the loss is the sum of C_{ij}.

3.4 The Implementation of Evaluation Metrics

3.4.1 Top-N% accuracy

Top-N% accuracy is defined as the percentage of overlap between the top-N% nodes which returned by our model and the top-N% nodes according to the ground truth BC given. *torch.topk* is used to get the top-N% nodes and the number of nodes overlap is got by using *torch.unique* on the concatenation of the returned top-N% nodes and the ground truth top-N% nodes. The formula of the top-N% accuracy is

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

Top-1% accuracy, top-5% accuracy, and top-10% accuracy are used for comparison.

3.4.2 Kendall tau distance

Kendall tau distance is a metric that calculates the number of disagreements between the rankings of the compared methods.

$$K(\tau_1, \tau_2) = \frac{2(\alpha - \beta)}{n(n-1)}$$

, where α is the number of concordant pairs, and β is the number of discordant pairs. The range of K is in $[-1, 1]$, where 1 means that two rankings are in total agreement, while -1 means that two rankings are in complete disagreement.

3.4.3 Wall-clock running time

The time calculation is implemented in the training function. The difference with the original implementation is that the time to calculate each node's exact BC value is skipped because the exact BC value is provided. The time is calculated in seconds.

3.5 Other Implementation Details

Putting encoder and decoder together, *class EncoderDecoder* is defined. The training is done in function *train_and_result*. The optimizer used is Adam with the same learning rate as mentioned in the paper, i.e. 0.0001.

3.5.1 Modification

Some modification is done to the original implementation.

1. Early stopping is used. The early stop will occur when the event that absolute difference between the current loss and the last episode's loss is less than 1 occurs 5 times.

2. Because of the speed problem, the maximum number of episodes for training is 10 instead of 10000.

3. Because the initial performance is not good at all, the bias is included in each GRU Cell, and the normalization layers are set to be trainable.

4. Exact 5 GRU cells are implemented instead only 1 GRU cell is used repeatedly. This modification aims to increase the model's converging speed and makes the model generates more accurate results.

5. Originally, $h_{N(v)}$ calculation and GRUCell are done node by node, but I implemented it by matrix form, which means they are done layer by layer.

6. Mini-batch is not implemented. This is because I can't figure out how to implement it.

3.5.2 Hyper-parameters

The hyper-parameters used are listed in Table 1. Because some modifications are made, some of the hyper-parameters used are different from the original implementation.

4 Experiment

4.1 Datasets

The experiment is performed on 30 Synthetic networks and 1 large real-world network, i.e. com-youtube.txt. For Synthetic networks, each one has 5000 nodes with about 19980 edges, the average degree of these networks is about 7.99, and the average diameter is 6. For com-youtube.txt, it has 1134890 edges, with 2987624 edges. Its average degree is 5.27, and its diameter is 20.

Hyper-parameter	Value	Value in Original Implementation
learning rate	0.0001	0.0001
embedding dimension	128	128
mini-batch size	N/A	16
average node sampling times	5	5
maximum episodes	10	10000
layer iteration	1 (as 5 GRU cells is implemented)	5

Table 1: Hyper-parameters configuration for DrBC

4.2 Train

The model is trained on 1.txt which belongs to one of the Synthetic networks and will be tested on other networks. The loss during training is shown in Figure 2.

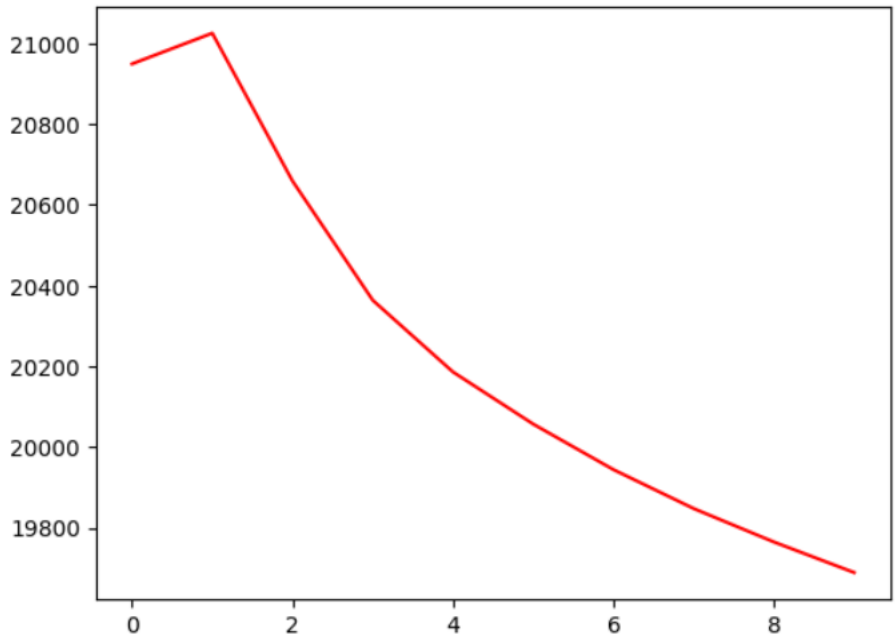

Figure 2: Loss during training

The result during training is shown in Table 2.

top-1%	0.46
top-5%	0.13
top-10%	0.11
Kendall tau Distance	-0.15
Time/s	1249.85

Table 2: Evaluation result on the training network

4.3 Test

4.3.1 Test on Synthetic Networks

Below shows the evaluation result of the model test on all the Synthetic networks. For easier discussion, The performance of KADABRA, the baseline model used, is also listed. The results are shown in Table 3 and Table 4. Notice that KADABRA did not provide Kendall tau distance, thus the comparison of the Kendall tau distance is skipped.

4.3.2 Test on com-youtube Networks

The large real-world network, the com-youtube network, is too large, making it unable to run by DrBC due to the limitation of memory, but KADABRA works on it. The result of KADABRA works on the com-youtube network is shown in Table 5.

4.4 Results on Synthetic Network

Before our discussion, notice that the training time of the DrBC model is excluded. It takes about 20 minutes for 10 episodes. Besides, there is an outlier, 0.txt, which has a different performance both by KADABRA and DrBC but still takes it into the analysis.

4.4.1 Evaluated by Top-N%

From Table 3, DrBC has better performance than KADABRA in top-1%, but worse performance in the other two top-N%. It means that DrBC actually works well for determining top-k BC nodes even with low episodes, and for more episodes, better performance for the other two top-N% is expected. For KADABRA, even if it does not have a good performance in the top-1%, it is almost fully accurate in the top-5% and top-10%.

4.4.2 Evaluated by Kendall tau Distance%

From Table 4, DrBC has an average Kendall tau distance of 0.42, which means that about half of the total pairs are concordant. DrBC achieving this performance is really surprising with only 10 episodes.

	Top-1%		Top-5%		Top-10%	
Network	KADABRA	DrBC	KADABRA	DrBC	KADABRA	DrBC
0.txt	0.90	0.98	1.00	0.88	1.00	0.83
1.txt	0.60	0.94	0.99	0.91	1.00	0.86
2.txt	0.60	0.90	0.99	0.88	1.00	0.84
3.txt	0.56	0.96	0.99	0.86	1.00	0.83
4.txt	0.59	0.96	0.99	0.87	1.00	0.85
5.txt	0.57	0.92	0.99	0.89	1.00	0.82
6.txt	0.57	0.90	0.99	0.89	1.00	0.84
7.txt	0.58	0.96	0.99	0.89	1.00	0.84
8.txt	0.58	0.90	0.99	0.85	1.00	0.88
9.txt	0.59	0.96	0.99	0.89	1.00	0.87
10.txt	0.60	0.94	0.99	0.88	1.00	0.85
11.txt	0.57	0.96	0.99	0.85	1.00	0.83
12.txt	0.60	0.92	1.00	0.85	1.00	0.85
13.txt	0.59	0.90	0.99	0.88	1.00	0.83
14.txt	0.58	0.94	0.99	0.84	1.00	0.86
15.txt	0.60	0.96	0.99	0.88	1.00	0.87
16.txt	0.59	0.92	0.99	0.86	1.00	0.88
17.txt	0.59	0.94	0.99	0.86	1.00	0.83
18.txt	0.60	0.98	0.99	0.90	1.00	0.82
19.txt	0.57	0.96	0.99	0.88	1.00	0.85
20.txt	0.59	0.96	0.99	0.87	1.00	0.85
21.txt	0.56	0.96	0.99	0.85	1.00	0.84
22.txt	0.59	0.94	0.99	0.88	1.00	0.85
23.txt	0.57	0.92	0.99	0.86	1.00	0.87
24.txt	0.59	0.92	0.99	0.88	1.00	0.85
25.txt	0.57	0.94	0.99	0.88	1.00	0.87
26.txt	0.60	0.96	0.99	0.88	1.00	0.83
27.txt	0.59	0.98	0.99	0.88	1.00	0.85
28.txt	0.59	0.94	0.99	0.89	1.00	0.82
29.txt	0.58	0.96	0.99	0.90	1.00	0.84
Average±std	0.60 ± 0.06	0.94 ± 0.02	0.99 ± 0.00	0.88 ± 0.02	1.00 ± 0.00	0.85 ± 0.02

Table 3: Top-N% on all the testing networks

	Kendall tau Distance	Time/s	
	DrBC	KADABRA	DrBC
0.txt	0.42	0.06	25.76
1.txt	0.41	0.08	9.12
2.txt	0.42	0.09	8.47
3.txt	0.42	0.08	7.07
4.txt	0.42	0.08	6.82
5.txt	0.44	0.08	6.66
6.txt	0.41	0.05	6.50
7.txt	0.41	0.08	6.60
8.txt	0.43	0.06	7.08
9.txt	0.40	0.06	6.64
10.txt	0.42	0.04	8.47
11.txt	0.46	0.05	8.41
12.txt	0.42	0.07	8.47
13.txt	0.46	0.09	9.20
14.txt	0.44	0.08	8.96
15.txt	0.39	0.07	7.94
16.txt	0.40	0.04	7.81
17.txt	0.44	0.06	6.86
18.txt	0.42	0.09	7.05
19.txt	0.44	0.09	6.71
20.txt	0.43	0.08	6.86
21.txt	0.45	0.05	6.68
22.txt	0.41	0.04	6.95
23.txt	0.42	0.05	6.61
24.txt	0.39	0.04	6.88
25.txt	0.40	0.06	6.71
26.txt	0.42	0.06	6.29
27.txt	0.43	0.05	6.28
28.txt	0.44	0.08	6.53
29.txt	0.40	0.08	6.90
Average±std	0.42 ± 0.02	0.07 ± 0.02	7.83 ± 3.43

Table 4: Kendall tau Distance and Time used on all the testing networks

	Top-1%	Top-5%	Top-10%	Time/s
com-youtube	0.17	0.13	0.12	5.00

Table 5: Performance of KADABRA on com-youtube network

4.4.3 Evaluated by Time(s)

From Table 4, KADABRA is really fast, taking no more than 0.1 seconds to finish execution, but for DrBC, even ignoring the training time, it still needs about 8 seconds to finish execution, which is much slower than KADABRA.

4.4.4 Summary based on Results on Synthetic Network

Comparing DrBC and KADABRA, even though DrBC is expected to have better performance with higher episodes, the resources required are huge and also the time execution is longer, but KADABRA can return a satisfying result with limited resources, and also faster. Thus, if considering the trade-off between accuracy and efficiency, KADABRA is better than DrBC in the Synthetic network except on the top-1% evaluation metric.

4.5 Results on Real-world Networks

Because of the limitation, DrBC cannot work on the com-youtube network. Even though KADABRA works on it, it doesn't have a performance that is as good as in the Synthetic networks. The paper[2] also states that KADABRA doesn't work well on large networks, even if it is efficient. But if resources is enough for DrBC to work on the com-youtube network, it should work well on it[2].

5 Discussion

In this session, I tried to implement DrBC according to the original implementation but in PyTorch. Some modifications are done to the network architecture, and because of the limitation, it only trained by 10 episodes, while the original implementation trained for a maximum of 10000 episodes. The performance of DrBC is compared with KADABRA on Synthetic networks and the com-youtube network. Below are some of the observations and comments.

1. Because of DrBC is an end-to-end manner, it requires label data like other deep learning applications while KADABRA doesn't. Originally DrBC requires another algorithm such as the Brandes algorithm to help it get the exact BC values so that it can start training. This is one of the cons of DrBC.

2. DrBC requires large resources to train on the large real-world network, which may be because of my modification. But even with enough resources, in the paper[2] it still mentioned that it needs 4.5 hours for training. If under the limitation mentioned, expect 33 hours to be needed to train the network.

3. DrBC has a surprising performance on the Synthetic networks even though only with low episodes. It shall have the best accuracy in identifying high BC nodes with enough episodes.

4. Even though the DrBC model only trains on 1 of the Synthetic network, it does not overfit with it, instead, it still can identify the high BC nodes of the other network.

References

[1] Michele Borassi and Emanuele Natale. *KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation*. Aug. 12, 2016. arXiv: 1604.08553[cs]. URL: http://arxiv.org/abs/1604.08553 (visited on 03/25/2023).

[2] Changjun Fan et al. *Learning to Identify High Betweenness Centrality Nodes from Scratch: A Novel Graph Neural Network Approach*. arXiv.org. May 24, 2019. URL: https://arxiv.org/abs/1905.10418v4 (visited on 03/25/2023).