Large Scale Graph Analytics for Communities Using Graph Neural Networks

Asif Ali Banka[1,2(✉)] and Roohie Naaz[1]

[1] National Institute of Technology, Srinagar, India
asifbanka@nitsri.net
[2] IUST Awantipora, Awantipora, India

Abstract. One of the challenging research areas in modern day computing is to understand, analyze and model massively connected complex graphs resulting due to highly connected networks because of newly accepted paradigm of Internet of Things. Patterns of interaction between nodes reflect a lot of information about nature of underlying network graph. The connectedness of nodes has been studied by several researchers to provide near optimal solution about topological structure of the graphs. This is more commonly known as community detection, which in mathematical and algorithmic terms is often referred to as graph partitioning. The study is broadly based on clustering of nodes, which share similar properties. Lower order connection patterns that detect communities at node and edge level are extensively studied. A wide range of algorithms has been studied to identify communities in large-scale networks. Spectral clustering, hierarchical clustering, Markov models, modularity maximization methods, etc have shown promising results in context to application domains under consideration. In this paper, the authors propose a neural network based method to identify the communities in large-scale networks. The study is broadly based on clustering of nodes, which share similar properties. This work is devoted to identify the efficacy of neural networks in community detection for large and complex networks in comparison to existing methods. The approach is motivated by neural network pipeline for data embedding into lower dimensional space, which is expected to simplify the task of clustering data into communities with inherent ability to learn between mapping and predicted communities.

Keywords: Community detection · Graph neural networks · Deep leaning · Hyper-parameter optimization

1 Introduction

Identifying group of nodes sharing similar properties is active research area in many disciplines like social science, telecommunication, computer networks, semantic web, protein networks, social networks etc. [2,4]. Community detection

S. Chellappan et al. (Eds.): CSoNet 2020, LNCS 12575, pp. 39–47, 2020.
https://doi.org/10.1007/978-3-030-66046-8_4

algorithms accept the graph as input dataset and outputs the community label for each node. Nodes in same community share similar properties with each other than with nodes outside the community. The community detection term is best fit for social networks rather than other domains where community is referred to as clusters or modules.

On the other hand, computer researchers are working day in and day out to incorporate deep learning into every field of study due to its efficient and accurate results. Deep learning has shown successful results in domains like image processing, computer vision and natural language processing etc. Relating deep learning to graphs seems challenging as graphs are sparse in nature and while deep learning shows outstanding results with dense matrices. Graphs depicting complicated relationships among objects have seeked attention from machine learning researchers to fit deep learning models over graphs that inherently are not natural fit for such models.

Dynamic nature of graphs with undefined structure and complex interconnections are challenging factors for deep learning engineers. Various supervised and unsupervised architectures have been proposed and adopted. Graph Neural Networks (GNN), Graph Convolutional Networks (GCN), Graph Auto-Encoders (GAE), Graph Recurrent Neural Networks (GRNN) and Graph Reinforcement Learning are architectures that have evolved in past few years to address graph related problems. Figure 1 presents a broad classification of deep learning methods implemented on graphs that have evolved over time. A detailed survey of these architectures is presented by Ziwei Zhang et al. in their work Deep Learning on Graphs: A Survey [14].

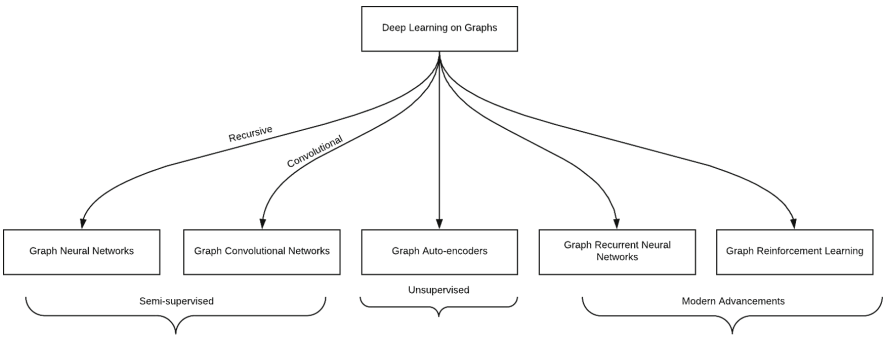

Fig. 1. GNN Classification

GNN and GCN adopt semi supervised model of machine learning where as GAE are unsupervised and GRNN and Graph Reinforcement Learning are more recent advancements in neural networks towards graph based problems. Graphs are usually studied from graph centric or node centric point of view and this distinction is emphasized more rigorously by different deep learning models. Node classification, link prediction and node recommendation are examples of

node centric classification while as graph centric tasks are associated with the whole graph. Examples include graph classification, estimating certain properties of the graph and community detection [12].

2 Graph Neural Networks and Graph Convolutional Networks

Graphs are high dimensional in nature and a simple underlying principle of Graph Neural Networks (GNNs) is to represent each node as a lower dimensional state vector s_i. The state of a node may be represented to assign label to each node as $s : V \rightarrow \{1, k\}$ where k is number of labels.

Recursively the states can be represented as

$$\mathbf{s}_i = \sum_{j \in \mathcal{N}(i)} \mathcal{F}\left(\mathbf{s}_i, \mathbf{s}_j, \mathbf{F}_i^V, \mathbf{F}_j^V, \mathbf{F}_{i,j}^E\right)$$

To obtain the objective function which is to minimize loss and improve accuracy between ground-truth and predicted values, an iterative algorithm like the Jacobi method [9] followed by gradient descend is performed [1,8] until convergence. GNN unifies the recursive nature neural networks and markov chains to take advantage of states in learning parameters. GNN formalizes the basis for GCN wherein each layer reaches a stable state. Since many iterations are required for gradient descent step and all previous states are maintained, GNN becomes computationally expensive. GCN replaces the recursive nature of neural network by convolution operation. GCN is an advancement of GNN focusing on training based on learning parameter. Underlying principle of GCN is convolution, which cannot be directly used in graph due to lack of dense matrix representing graphs [11]. Convolution on graphs Laplacian $\mathcal{L}$ for first time was introduced by Bruna et al. [3,10]. Typical convolution over a graph can be defined as

$$\mathbf{u}_1 *_G \mathbf{u}_2 = \mathbf{Q}\left(\left(\mathbf{Q}^T \mathbf{u}_1\right) \odot \left(\mathbf{Q}^T \mathbf{u}_2\right)\right) \tag{1}$$

where Q are eigen vectors of $\mathcal{L}$ and u_1 and u_2 are signals defined on nodes. Filter operation $\mathbf{u}'$ on signal can therefore be defined as $\mathbf{u}' = \mathbf{Q}\Theta\mathbf{Q}^T\mathbf{u}$ where $\mathbf{u}'$ is output signal and Θ is diagonal matrix of learnable filters. Convolution can thus be defined by applying different filters on different input and output signals. Passing the input through filters that can learn and aggregate the information after transformation is underlying idea of convolution. By using node features as the input layer and stacking multiple convolutional layers, the overall architecture becomes similar to CNNs.

3 Methodology

In this study, the authors are interested in employing graph convolutional networks to detect communities using neural networks, compare the overlap with

ground truth (or true community structures) and improve the accuracy. Since the neural networks are data driven and provide flexibility of learning by gradient descent and hyper-parameter optimization which make model efficient and robust against wrong heuristics. More formally, a GCN model is a neural network that operates on graphs. Given a graph $G = (V, E)$, a GCN takes as input, an input feature matrix X and an adjacency matrix A. X is a $N \times \mathbf{F}_i^V$ feature matrix, where N is the number of nodes and $\mathbf{F}^V$ represents the features for each node. Adjacency matrix A is an $N \times N$ matrix representation of the graph structure of graph G. At each layer input β is transformed via convolution applied to the array of operators[15]. The hidden layer of a neural network is represented as function of previous layer and adjacency matrix governed by a propagation rule. $H_i = f(H_{i-1}, A)$ [6].

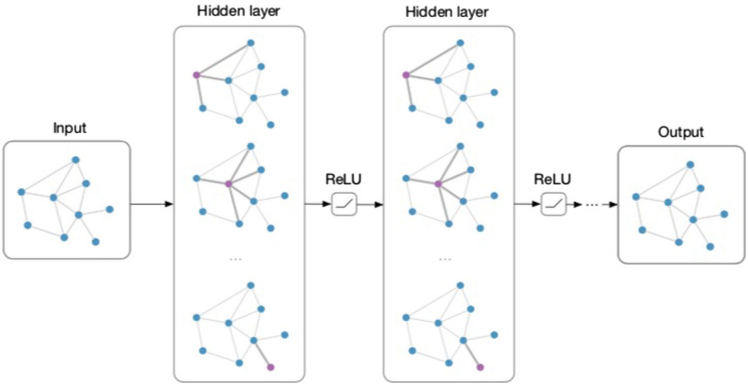

Fig. 2. GNN architecture [6]

Figure 2 is a pictorial representation of algorithm proposed in [6] for graph neural networks. Input to the first layer of model is a feature matrix X and adjacency matrix A. Features are aggregated using propagation rule f which enhances the abstraction in consecutive layers. The propagation rule may vary and one of simplest is one where weighted matrix and activation function is considered. This can be written as $f(H_i, A) = \rho(AH_iW_j)$ where W_i is the weight matrix for layer i and ρ is an activation function. This is similar to filtering operation as these weights are shared across nodes in the network. Nodes with higher degree will have larger values for their feature representation. This will explode the gradient which is typically used to train networks as they are sensitive to scale of input features. Similarly, nodes with smaller degree have diminishing effects on gradient descent which is typically used to train networks. Each node in network is represented as aggregate of features of neighbouring nodes, however, its own features are not considered if there is not any self loop. Identity matrix I is used to address self loops, as the features of node get summed to its own features before applying propagation rule. Since the feature space gets

exploded, multiplying adjacency matrix with inverse degree matrix can be used to normalize the features as the weights in each row are divided by degree of each node [5]. The propagation rule is modified as $f(X, A) = D^{-1}AX$.

Algorithm 1. Graph Neural Network for Community Detection

Input: Graph $G = (V, E)$
Output: A clustering of the vertices v into k clusters. This can be encoded as a function $F : V \rightarrow \{1, 2, ..., k\}$.

1: **procedure** GNN
2: Input feature matrix X and an adjacency matrix A. X is a $N \times \mathbf{F}_i^V$ feature matrix, where N is the number of nodes and $\mathbf{F}^V$ represents the features for each node. Adjacency matrix A is an $N \times N$ matrix representation of the graph structure of graph G
3: Hidden layer is $H_i = f(H_{i-1}, A)$
4: $f(H_i, A) = \rho(AH_iW_j)$ where W_i is the weight matrix for layer i and ρ is an activation function.
5: Identity matrix I is used address self loops
6: The propagation rule is modified as $f(X, A) = D^{-1}AX$

If $\beta \in \mathbb{R}^{v \times k}$ is input where V is the number of vertices and k is the number of communities we want to detect. $\mathbb{R}^{n \times k}$ is a one-hot encoding of the clustering. The output of final layer is one hot encoding of the community labels. Finally, we divide the train and test sets by enforcing test examples to contain disjoint communities from those in the training set. Input at each layer can be represented as

$$\beta^{V+1} = \beta_1^{V+1} + \beta_2^{V+2}$$

where

$$\beta_1^{V+1} = \mu(I \cdot \beta^V, A \cdot \beta^V, D^{-1} \cdot \beta^V)$$

and

$$\beta_2^{V+1} = \rho \circ \mu(I \cdot \beta^V, A \cdot \beta^V, D^{-1} \cdot \beta^V)$$

4 Experimentation and Results

Our performance measure is the overlap between predicted and true labels, which quantifies how much better than random guessing, a predicted labelling is. The GNNs were all trained with 10, 20, 30 and 40 layers, 10 feature maps and J = 3 in the middle layers. We varied the optimization parameter and evaluated the performance for Adamax and rmsprop [5]. The learning rate was changed from 0.001 to 0.00001 with varying decay between 0 and 0.01. The effect of variation of epoches was also measured. The experimentation was performed on intel i7 processor with 32 GB RAM and 1070 GPU. All the trainings were performed on GPU.

Diverse datasets from SNAP were used to train the GNN with community labels provided. These datasets used vary from social networks to hierarchical co-purchasing networks [7]. Different dataset with known community structures enable us to understand how well the model behaves. Top 5000 quality communities provided in dataset were used. These were used to identify those edges (i, j) that cross at least two different communities. For each of such edges, we consider the two largest communities C1,C2 such that i $\notin$ C2 and j $\notin$ C1, i $\in$ C1,j $\in$ C2, and extract the subgraph determined by C1 $\cup$ C2, which is connected since all the communities are connected. Finally, we divide the train and test sets by enforcing test examples to contain disjoint communities from those in the training set. Table 1 lists the datasets used for experimentation.

Table 1. Datasets used for Graph Neural Networks.

Dataset	Number of vertices	Number of edges	Diameter	Average Degree
Amazon	334,863	925,872	44	3.4576
DBLP	317,080	1,049,866	21	6.6221
Youtube	1,134,890	2,987,624	20	5.8167

Multiple experiments were performed to realize the performance of model and performance achieved was compared with the Community-Affiliation Graph Model (AGM). The AGM is a generative model defined in [13] that allows for overlapping communities where over-lapping area have higher density. This is a statistical property observed in many real datasets with ground truth communities. The hyper-parameters like learning rate, number of layers, number of ephocs were studied. Effect of changing optimizer and batch size was also explored. Table 2 list the results for AGM experiment carried over datasets. Among the three datasets Amazon dataset showed the best results with 73.32% overlap.

Table 2. Accuracy of Community-Affiliation Graph Model

Dataset	Training size	Test size	AGMFIT overlap
Amazon	359	34	72.32
DBLP	7751	1274	64.01
Youtube	211678	33290	57.01

Figure 3(a) depicts the results for ten layer hundred ephocs and ten layer two hundred ephocs carried over datasets. DBLP dataset showed best results with 83.0724% overlap compared to Amazon and YouTube dataset on 10 layer hundred ephoc configuration, however, the accuracy of Amazon and YouTube

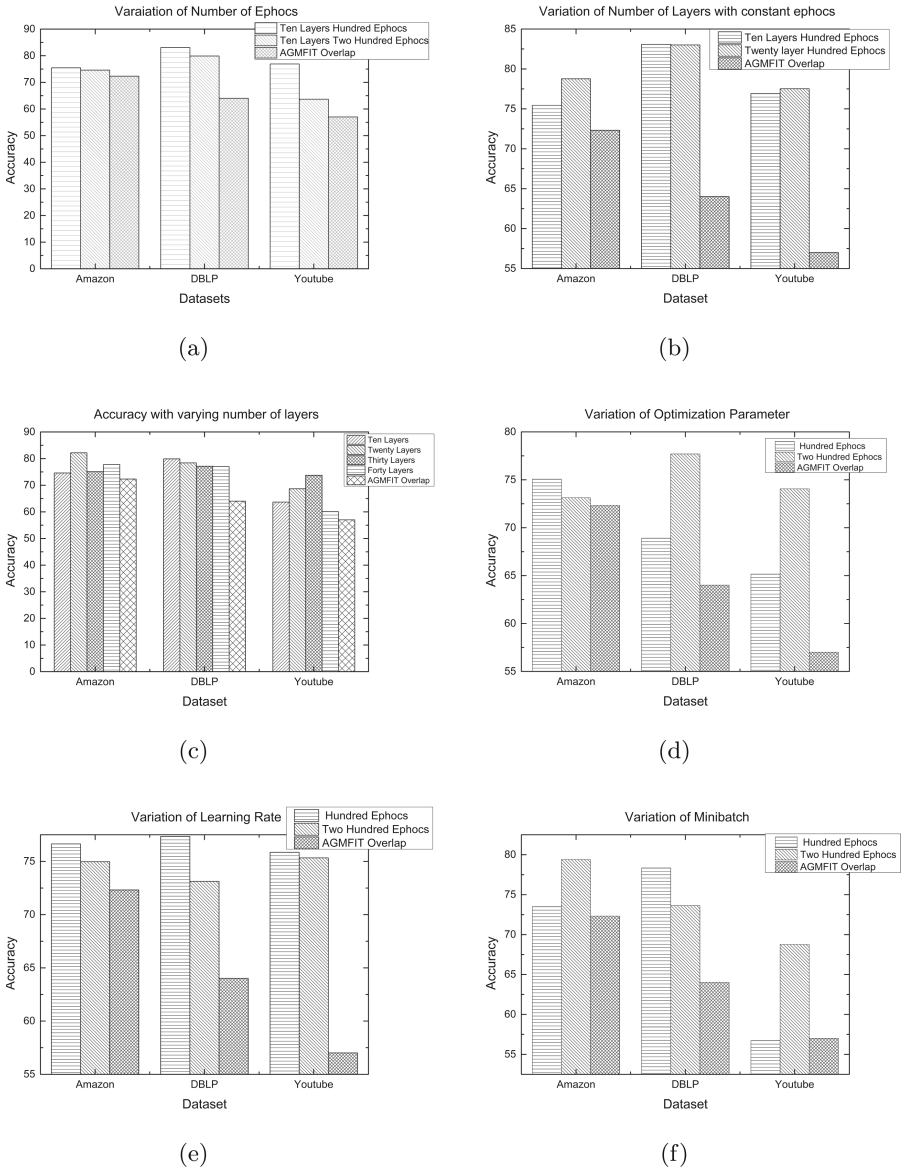

Fig. 3. Effect of variation of hyper-parameters on Community Detection.

dataset decreased on ten layer hundred ephoc configuration. DBLP dataset showed best results for twenty layers hundred configuration with 79.9042% overlap. The effect of variation of number of layers with constant ephocs was also studied. The results of variation of ten layers and twenty layers with hundred ephocs is depicted in Fig. 3(b). The overlap for ten layer configuration showed

best results for DBLP dataset whereas, for twenty layer configuration Amazon dataset showed better results compared to DBLP and Youtube datasets.

Again, the experimentation was carried to study the effect of variation of number of layers on accuracy. It was observed Amazon dataset showed accuracy of 82.1791% for twenty layer whereas, it showed accuracy of 75.0693 and 77.7470% for thirty layer and forty layer neural networks respectively. For twenty layer configuration best results were obtained for DBLP dataset with 79.9042%. For thirty and forty layer configuration best results were obtained for DBLP (877.1397) and Amazon (77.7470) datasets respectively. Best accuracy for Youtube dataset was achieved with thirty layer neural network configuration. The reults are shown in Fig. 3(c).

Results for variation is change of gradient optimization technique were also studied. Among the datasets rmsprop with two hundred ephocs showed best results with 77.6961% accuracy for DBLP. Amazon dataset showed better results with rmsprop among datasets when trained with hundred ephocs only. Figure 3(d) list the results for variation in optimization technique.

The learning rate was modified and its effect was studied. The effect was studied with hundred and two hundred ephocs configurations. In general modifying learning rate with hundred ephocs showed better results that modified learning rate over two hundred ephocs. Figure 3(e) depicts the results for varying learning rate hyper-parameter.

Figure 3(f) shows the results for variarion in minibatch size hyper-parameter. The results were best obtained for Amazon dataset with an accuracy of 79.4090% in two hundred ephocs configuration.

5 Discussion

A significant contribution of this work is the study of multiple hyper-parameters and their effect on accuracy. We began the experimentation with 10 layers and model was studied with 20, 30 and 40 layers as well. The effect of variation of number of layers was also studied along with the effect of selection of optimizer. The combined effect of variation of ephocs and variation in number of layers was also studied. Learning rate and batch size were also changed to study the corresponding effect. The complexity of running time varied exponentially with respect to depth of neural network which is a function of addition and multiplication operations in neural network. The DBLP dataset showed promising results for 10 layered 100 ephoc configuration with 83.0724 overlap percentage, however it decreased as we increased the number of ephocs to 200. The decrease in accuracy is result of over-fitting of data. Amazon dataset showed 82.1791% overlap for 20 layer 200 ephoc configuration and Youtube dataset showed 77.5156% overlap with 20 layer 200 ephoc configuration. Among 10, 20, 30 and 40 layers with 200 ephocs our model showed 82.1791 % over lap for Amazon dataset for 20 layer 200 ephocs, DBLP showed 79.9042 % overlap for 10 layers and Youtube showed 73.7482% overlap for 30 layers. When the choice of optimizer was varied to rmsprop with 100 and 200 ephocs the Amazon dataset

showed overlap of 75.0693 on 100 ephocs and DBLP and Youtube showed overlap of 77.6951 and 74.0590% respectively. Change of learning rate performed better with 100 ephocs only with overlap of 76.6390, 77.3466 and 75.8534 for Amazon, DBLP and Youtube datasets respectively. Only DBLP showed overlap of 78.3441 on 100 ephocs when batch size was varied. Amazon and Youtube showed overlap of 79.4090 and 68.7747 respectively on 200 ephocs settings. The results obtained with the proposed model in all configurations outperformed the accuracy achieved by AGM model.

References

1. Almeida, L.B.: A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In: Proceedings, 1st First International Conference on Neural Network, Vol. 2, pp. 609–618. IEEE (1987)
2. Aridhi, S., Nguifo, E.M.: Big graph mining: frameworks and techniques. Big Data Res. **6**, 1–10 (2016)
3. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs (2013). arXiv preprint: arXiv:1312.6203
4. Han, M., Daudjee, K., Ammar, K., Özsu, M.T., Wang, X., Jin, T.: An experimental comparison of pregel-like graph processing systems. Proc. VLDB Endow. **7**(12), 1047–1058 (2014)
5. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization (2014). arXiv preprint: arXiv:1412.6980
6. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2016). arXiv preprint: arXiv:1609.02907
7. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data (Jun 2014)
8. Pineda, F.J.: Generalization of back-propagation to recurrent neural networks. Phys. Rev. Lett. **59**(19), 2229 (1987)
9. Powell, M.J.: An efficient method for finding the minimum of a function of several variables without calculating derivatives. Comput. J. **7**(2), 155–162 (1964)
10. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Netw. **20**(1), 61–80 (2009)
11. Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A., Vandergheynst, P.: The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains (2012). arXiv preprint: arXiv:1211.0053
12. Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W., Yang, S.: Community preserving network embedding. In: Thirty-First AAAI Conference on Artificial Intelligence (2017)
13. Yang, J., Leskovec, J.: Community-affiliation graph model for overlapping network community detection. In: 2012 IEEE 12th International Conference on Data Mining, pp. 1170–1175. IEEE (2012)
14. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: a survey (2018). CoRR abs/1812.04202: http://arxiv.org/abs/1812.04202
15. Zhou, Z., Li, X.: Graph convolution: a high-order and adaptive approach (2017). arXiv preprint: arXiv:1706.09916