Graph Representation Learning

Abhishek Yadav
160040

Aditya Prakash
160051

Ahsan Barkati
160055

Akanksha Makkar
160062

Harshit Sharma
160283

Naman Jain
160427

1 Problem Description:

Machine Learning on graph based data is has one of the finest application of ML on real-life problems ranging from drug design, community detection, recommendation systems and many more. The main challenge in this domain is finding a way to represent or encode the graph so that it can be fed to some ML algorithm. Traditionally many techniques existed which relied on user defined description of graph for example using degree of nodes as their representation. But in recent years, many techniques have been developed which automatically learns low-dimensional embedding which captures the structural information of the graph. In this project, we did a comparative study of some of the embedding methods by comparing their performances on the *Cora-Citation Network*.

2 Embedding methods:

We have done a comparative study of embedding methods like GraphSAGE, Graph Convolutional Networks and Node2Vec for solving the problem of node classification and clustering on graph i.e finding community in a graph. Clustering on graph has several applications such as finding a community of friends in a social network, protein-protein interactions in biological networks etc.

2.1 GraphSAGE

The main idea behind GraphSAGE is to learn how to aggregate feature information from a node's local neighborhood. It uses an agregator function to aggregate the encoding of the neighbouring nodes for each node. The neighbour of a node is defined by the function $\mathcal{N}(x)$. The encoding of each node is updated by the concatenating the aggregated encoding of the neighboring nodes multiplied by a Weight matrix and applying non-linearity by using sigmoid function. The algorithm for this update is mentioned below. The weight parameters are learned by defining a loss, which makes the nodes encoding similar for nodes close to each other in a random walk of fixed length.

The two main components of GraphSAGE are the **aggregate** and the **concat** functions, which can be explained in the following manner :

Aggregate : This function can be simply explained by viewing it analogous to pooling in convolutional neural networks as used in computer vision, in a way more specific to graphs we can say that it looks at the embeddings of neighbours of a node and combines them. A suitable example of aggregate function can be a mean of all the embeddings, which is analogous to mean pooling in CNN.

Concat : After we have a suitable sample embeddings for the surrounding of nearby nodes, essentially a sample embedding of adjacent neighbour, intuitively we make the embedding of this node closer to its surrounding, for this we may use a concat function that can be as simple as taking average of current node embedding and the embedding learned from the surrounding.

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

```
Input  : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth $K$; weight matrices
         $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity $\sigma$; differentiable aggregator functions
         $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \to 2^{\mathcal{V}}$
Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$
1 $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for $k = 1...K$ do
3     for $v \in \mathcal{V}$ do
4         $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;
5         $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$
6     end
7     $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$
8 end
9 $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$
```

The loss function is defined as :

$$J_G(\check{z}_u) = -log(\sigma(\check{z}_u^T \check{z}_v)) - Q * E_{v_n \sim P_n(v)} log(\sigma(-\check{z}_u^T \check{z}_{v_n})) \quad (1)$$

where v is node that exists in a random walk of fiexd length, intuitively we can observe that this loss function not only tries to make those node embeddings closer to each other which are nearby but also it tries to make those embeddings far-off which do not lie closer to each other in the graph, this closeness of nodes is derived from random walks of fixed length on the graph.

2.2 GCN

Graph convolutional networks are a very powerful neural network architecture for machine learning on graphs. GCN is similar to convenctional CNNs on images, however more robust than other methods since the weight parameters are shared so it is capable of extracting more general features from the given data of the graph than the other methods.

GCN can be thought of a normal encoder-decoder network. It is to be noted that genrally all the weight parameters are contained in the encoder part only and there are not weight parameters in the decoder network.

A decoder can be as simple as computing the cosine similarity between embedding of a pair of noodes, we also construct a degree normalized matrix such that no node is given a starting bias in terms of number of nodes connected to it, i.e we make the sum of entire row same for all the rows by normalizing it by the degree matrix.

A degree matrix **D** is a diagonal matrix such that the diagonal entry D_{ij} is sum of incoming degree in a directed graph. However in order to this we pre-multiply and post-multiply the adjacency matrix **A** with $D^{-\frac{1}{2}}$ because that makes the operation symmetric and gives better results that pre-multiplying by D^{-1}
Following equations define the generic structure of GCN layer-by-layer:

$$H^{l+1} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{2}$$

$$Loss = -\Sigma_{l\in y_l}\Sigma_{f=1}^{F}Y_{lf}Z_{lf} \tag{3}$$

The figure briefly explains the overall structure of GCN, i.e there is an input layer which is same as the input graph with the features of each nodes and we have subsequent hidden layers governed by the above equations.We assume a semi-supervised approach to the problem, we would be given few ground truth labels using which we would optimize our loss function which is described as above.

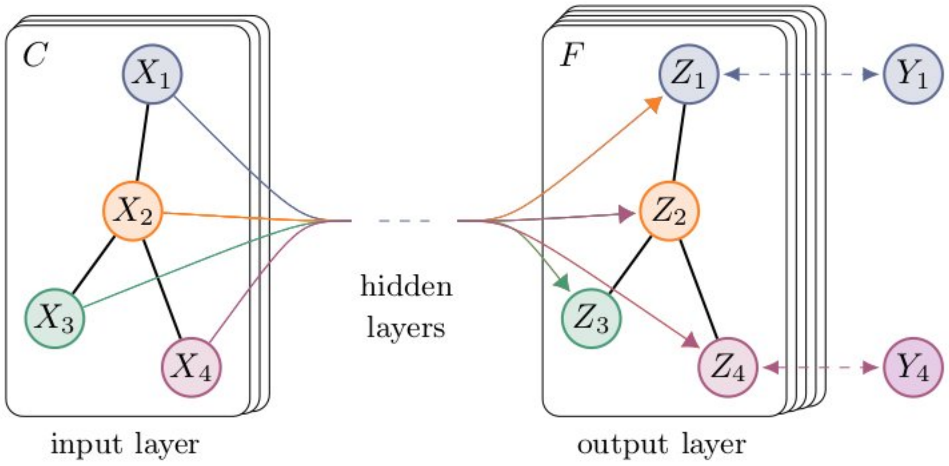


Figure 1: Diagrammatic representation of Graph Convolutional Networks

2.3 Node2Vec

Node2vec generates node embeddings in a similar way as the word2vec does. It generates corpus by doing random walks over the graph. These walks(sentences) are fed to word embedding methods. Node2vec's sampling strategy has following major parameters:

- **Number of walks:** Number of random walks to be generated from each node in the graph
- **Walk length:** Number of nodes in each random walk
- **P:** Controls the likelihood of immediately revisiting the node. Set it to high value to ensure sampling of recently visited node less probable.
- **Q:** Controls the inwards and outwards likelihood of next node. Setting q low captures local neighbourhood while high value captures local structure of graph.

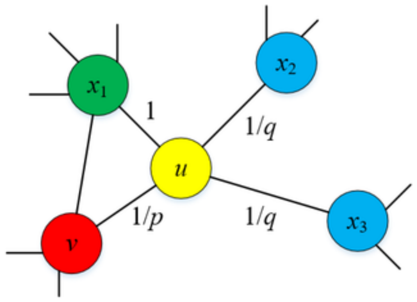


Figure 2: Node2Vec: Walks sampling

Intuitively it can be observed that the parameters **P** and **Q** control the extent of **DFS** and **BFS** from each node in a random walk.

3 Results:

We evaluated various algorithms listed above on Cora-Citation Network. Description of dataset is given below.

Cora Dataset

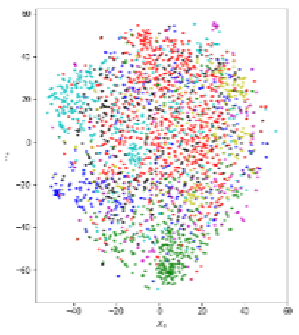
The Cora dataset consists of 2708 scientific publications classified into one of seven classes. The citation network consists of 5429 links. Each publication in the dataset is described by a binary 1433-dimensional word vector indicating the absence/presence of the corresponding word from the dictionary.
These papers are classified into one of the following seven classes:

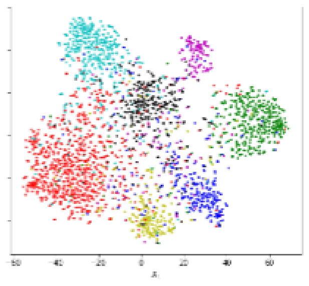
S.no	Labels
1	Case Based
2	Genetic Algorithms
3	Neural Networks
4	Probabilistic Methods
5	Reinforcement Learning
6	Rule Learning
7	Theory

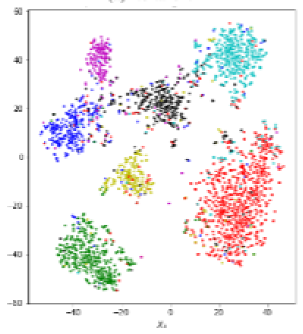
Link for the dataset :- `http://www.research.whizbang.com/data`

GraphSAGE: Following images show results of successive iterations of graphSAGE, the embeddings are visualized using t-SNE, the points with same color represents the low dimensional embedding of those points which belong to same category. It is clearly observed that the results become better as the algorithm proceeds. Mean aggregator was used during the training and cross-entropy loss function was minimized using Adam optimizer. Rand-index for GCN came out to be **98%**.

Graph Convolutions Networks: Following images depicts the layer-wise t-SNE embeddings. Training was done on 2-layer GCN with one hidden unit using ReLU activated hidden unit and softmax activation on output layer. Semi-supervised node classification using GCN trained on 50%


(a) Iteration: 2


(b) Iteration: 4


(c) Iteration: 13

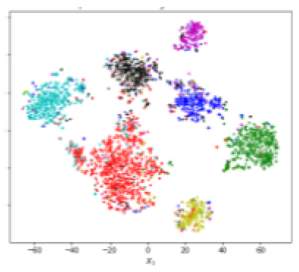
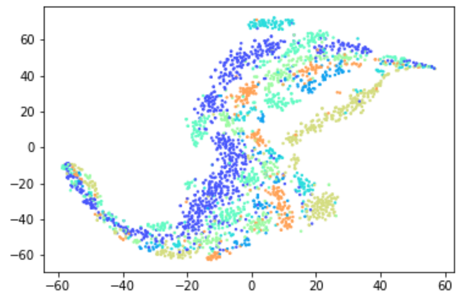

(d) Iteration: 21

Figure 3: GraphSage: Node Classification

data resulted in test accuracy of **91%**. When trained with full dataset, it resulted in **99%** training accuracy. Rand-index for GCN came out to be **97%**.


(a) Hidden layer embeddings visualized using t-SNE

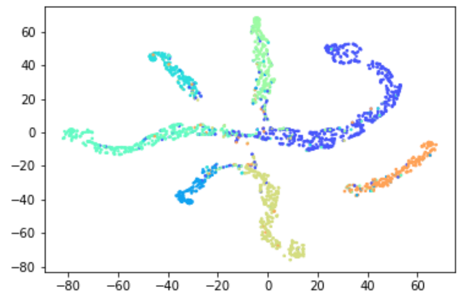

(b) Final layer activation visualization using t-SNE

Figure 4: GCN layer-wise embedding

Node2vec: Tuned hyperparameters of node2vec came out to be Walk Length = 10, p = 0.3 and q = 0.4. 20-dimensional embedding learned when projected to 2-D space using t-SNE produced poor results compared to above 2 methods. The visualizations are shown below. Rand-index for GCN came out to be **69%**.

It can be observed that GCN and graphSAGE outperformed node2vec. This is in accordance with what was expected because node2vec relies on Graph structure only while other two methods utilize node features also.

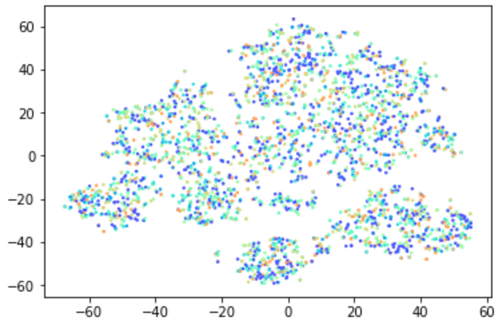


Figure 5: Node2vec embeddings

4 Conclusion:

Graph embedding methods can be broadly categorized into Matrix Factorization, Deep Learning (with or without random walks), Convolutional based and Generative Modelling. We studied the Deep learning and Convolutional based graph embedding methods. Node2vec and graphSage are among deep learning based graph embedding methods that use random walks. GraphSage uses node features to learn which enhances its capability. Convolutional based method GCN performed pretty well and is much more flexible.

GraphSAGE is kind of extension to node2vec model. Whatever node2vec learns seems to be captured in graphSAGE model. Instead of learning the embeddings of nodes fed during training (as what node2vec does), it learns a functions that can predict embeddings of unseen nodes.

Code can be accessed from our **Github repo**.

References

[1] Aditya Grover, Jure Leskovec. *node2vec: Scalable Feature Learning for Networks*.

[2] William L. Hamilton, Rex Ying, Jure Leskovec. *Inductive Representation Learning on Large Graphs*.

[3] Thomas N. Kipf, Max Welling. *Semi-Supervised Classification with Graph Convolutional Netorks*

[4] `https://tkipf.github.io/graph-convolutional-networks/`