Project in Machine Learning, Advanced - DD2434
Stockholm, Sweden 2023

Network Representation Learning

Large Project - Group 46

Adhithyan Kalaivanan - adhkal@kth.se
Aishwarya Ganesan - aganesan@kth.se
Daniel Richards Ravi Arputharaj - drra@kth.se
Vishal Nedungadi - vishaln@kth.se

Abstract

Unsupervised Network Representation Learning (UNRL) has evolved from simple techniques to more advanced ones that rely on neural networks. In this project, five popular UNRL approaches are evaluated over ten datasets to understand their behaviour for different graphs on various tasks. The comparative evaluation of models is done based on their performance in two downstream tasks - node classification and link prediction. The objective is to reproduce the results provided in the paper 'A Comparative Study for Unsupervised Network Representation Learning' by Khosla et al.,2019 and reason out why the the reproduced outcomes are similar or dissimilar. The code is built from scratch and respective pseudo-codes are provided for DeepWalk, Node2vec, NetMF. LINE-1 and GraphSAGE. To an extent, the reproduced tasks obtained similar results as the original paper but was hindered by the resource constraints of the system available for reproduction. Under the resource constraints, the results indicate that (1) NetMF performs consistently better than other methods for node classification tasks given a small dataset, and (2) LINE-1 efficiently runs on dataset of any size and is the best link predictor.

KTH ROYAL INSTITUTE OF TECHNOLOGY

1 Introduction

Graphs are data structures that encode key information related to links or relations between data points. They are often used in representing social network connections, protein-protein interactions or links between different citations of papers. Given the nature of these data, the graphs can be very large and have to be embedded in low dimension space for any further analysis tasks. Unsupervised Network Representation Learning (UNRL) methods have started to become significantly important as it is very expensive to obtain labelled and annotated data to perform supervised learning. The main idea behind UNRL is to have similar network embeddings for nodes that have close connections and dissimilar embeddings otherwise. The objective of this project is to successfully reproduce the results for five of the UNRL techniques evaluated in the comparative study in the paper [**khosla2019comparative**]. The paper developed a unified framework to evaluate all models under a common objective function defined on their context graph. They evaluated nine different UNRL models on eleven real-world datasets. This project will implement NetMF, Line, Node2Vec, DeepWalk and GraphSAGE. These models will be run on ten real-world datasets - nine from the original paper, and on the Facebook Page-Page dataset. The evaluations will be based on two downstream tasks - node classification and link prediction. An analysis on the reproduced results of the original paper will be given.

2 Network Embedding Methods

2.1 Algorithm

2.1.1 DeepWalk

DeepWalk [**perozzi2014deepwalk**] performs unbiased random walks of length T, r times from each node. This corpus is used to train a Word2Vec model with skip-gram architecture, using hierarchical softmax.

In this project, random walks are implemented using *random* package and the Word2Vec model is imported from *Gensim* library [**rehurek_lrec**]. The hyperparameters are modified from the paper, due to hardware limitations. The embedding dimension is maintained as $d = 128$, but the walk length is set to $T = 10$ and walks per node to $r = 20$. Correspondingly, the context window size of Word2Vec is set to $w = 2$ and the model is trained for 20 epochs. These hyperparameters are the same across data sets and the node classification and link prediction tasks.

Pseudo code for DeepWalk

```
FOR $n = 1, 2, ..., r$
    FOR each node $v$ in graph $G$
        Get node sequence of length $T$ with unbiased random walk from $v$
        Add node sequence to corpus $C$
TRAIN Word2Vec on $C$ with Hierarchical softmax
RETURN embedding
```

2.1.2 Node2Vec

Node2Vec [**grover2016node2vec**] is similar to DeepWalk except it performs a biased random walk parameterized by p and q. A Word2Vec model with skip-gram architecture is trained on the corpus using negative sampling with n samples. Random walks are implemented using *numpy* package and the Word2Vec model is imported from *Gensim* library. Performing this biased random walk is a lot more computationally expensive than an unbiased one, so the hyperparameters of the walk were reduced to meet the hardware limitations. The embedding dimension is still $d = 128$, but the walk length is set to $T = 5$ and walks per node to $r = 10$. The context window of Word2Vec is reduced to $w = 2$ and the model is trained for 20 epochs with negative samples $n = 5$. These hyperparameters are again the same across data sets and both the prediction tasks. p and q used different data sets are the same as what is reported in the paper.

Pseudo code for Node2Vec

```
FOR $n = 1, 2, ..., r$
    FOR each node $v$ in graph $G$
        Get node sequence of length $T$ with biased random walk from $v$
        Add node sequence to corpus $C$
TRAIN Word2Vec on $C$ with negative samples $n$
RETURN embedding
```

2.1.3 NetMF

NetMF [**qiu2018network**] is a matrix factorization framework built on the theoretical analysis of DeepWalk matrix M. It is implemented using *scipy.sparse* package for generating the normalization graph laplacian and eigen-decomposition, and *sklearn.decomposition* package for performing Truncated SVD. Number of negative sampling b is varied between 1 to 5, and $b = 1$ is picked as it showed best performance on all datasets. Similarly, window size $T = 1$ for small window and $T = 5$ for large window yields the best results. Embedding dimension of $d = 128$ was kept same as in the original paper.

Pseudo code for NetMF

```
CREATE adjacency matrix $A$ from graph $G$
IF window size is small
   CALCULATE $vol(G), D^{-1}, P$ from graph $G$
   COMPUTE $M = \frac{vol(G)}{bT} \times \left(\sum_{r=1}^{T} P^r\right) D^{-1}$
IF window size is large
   CALCULATE $vol(G), D^{-1/2}$ from graph $G$
   COMPUTE eigen-decompostion $\Lambda_h, U_h$ from $D^{-1/2}AD^{-1/2}$
   COMPUTE $M = \frac{vol(G)}{b} \times D^{-1/2}U_h \left(\frac{1}{T}\sum_{r=1}^{T} \Lambda_h^r\right) U_h^T D^{-1/2}$
COMPUTE $\log M' = \log max(M, 1)$
OBTAIN embedding using TruncatedSVD of $\log M'$ for $d$ components
RETURN embedding
```

2.1.4 LINE-1

LINE-1 [**line**] generates node embedding based on first order connections. The theoretical objective function to learn embedding is

$$O \quad = \quad - \sum_{(u,v)\epsilon \mathbf{E}} w_{uv} \log\left(\frac{1}{1 + \exp -\vec{\mathbf{u}}_u \cdot \vec{\mathbf{u}}_v}\right) \tag{1}$$

where w_{uv} is the edge weight between u to v and vectors $\vec{\mathbf{u}}_u$, $\vec{\mathbf{u}}_v$ are node embedding of u, v. To avoid the exact computation over all nodes, negative sampling is used to approximate this (number of samples $k = 5$). Alias table method is used to sample wherever needed.

The original paper for LINE uses a SGD optimiser with mini batch size as 1 and a decaying learning rate. Due to lack of high computing resources, here batch size of 1024 and Adam optimiser is used. The number of training epochs is varied based on the dataset size. 50 epochs for Cora and PubMed, 15 epochs for BlogCatalog, Epinion, DBLP-Ci and Twitter and 10 epochs for large dataset like Reddit, YouTube and Flickr.

Pseudo code for LINE-1

```
CREATE alias tables for nodes and edges distribution
INITIALISE embedding
LOOP number of epoch times
     LOOP batch in batches of size $1024$ until NUM_EDGES
          SAMPLE $1024$ edges using edge alias table
          SAMPLE $k = 5$ negative samples using node alias table for sampled edges
          COMPUTE Objective function for each batch
          PERFORM Learning based on the optimiser
RETURN embedding
```

2.1.5 GraphSAGE

GraphSAGE [**inductive**], unlike most algorithms, does not directly learn the node embedding but instead learns the weights of an aggregator function. The aggregator function can be thought of as a method to obtain information from neighboring nodes. So when given a new node, this function represents it as an aggregation of neighbours without any need for re-training. This aggregation is done in different ways, from a simple "mean" to something complex like using a neural network. In this project, only the mean aggregator for GraphSAGE is used.

The algorithm involves, aggregating information from each node's neighbors through every node in the graph. The process is repeated a certain number of iterations and in each iteration certain number of neighbours (hop or search depth) are considered. Moreover, for unsupervised GraphSAGE, random walks are used to obtain the node pairs required to compute the loss function.

The implementation uses *Tensorflow* for the neural network.

Pseudo code for GraphSAGE

```
READ node features $x$ (here identity matrices),
     aggregator function $AGGREGATE$,
     weight matrix $W$,
     depth $K$ (number of hops)
INITIALIZE $h_v^0 = x_v$ for all nodes $v$
LOOP $k$ from 1..$K$
     FOR all nodes $v$
         compute $AGGREGATE(h_u^{k-1})$ where $u$ belongs to neighbors of $v$
         compute sigmoid($W.CONCAT(h_u^{k-1}, h_u^k)$)
     NORMALIZE $h_v^k$ for all nodes $v$
RETURN $h_v^k$ for all nodes $v$, which are the embeddings
```

Hyperparameters: Embedding dimension is 128, dropout is set to 0.4, identity dimension is 64, learning rate is 0.0001 and maximum number of iterations is 1000. The parameters for random walk are walk length of 10 and number of walks as 20.

2.2 Datasets

Social Networks: Labelled graphs BlogCatalog, Flickr and Youtube and unlabelled graphs Twitter and Epinion used in the original paper are trained and tested on. Additionally, Facebook Page-Page graph data [**rozemberczki2019multiscale**] ($|E| = 171002, |V| = 22470$) where nodes are official Facebook pages with edges based on mutual likes is used for predictive tasks, here nodes are labelled by the page-type.

Citation and Collaboration Networks: DBLP-Ci, Cora and PubMed dataset with nodes as paper and edges as citations networks used in the original paper is also tested here. However, DBLP-Au and CoCit could not be obtained.

Based on these datasets, *Networkx* [**hagberg2008exploring**] is used for creating graphs to feed to the respective UNRL methods.

2.3 Downstream Tasks

Node Classification: In the node classification task, for dataset with node labels, node embedding are used to train a logistic regression model in a one vs all manner. 5-fold cross validation from *sklearn* is used to get the micro-F1 and macro-F1 scores.

Link Prediction: For link prediction, given a list of test edges and node embedding, dot product of the embedding for every node pair in the test list is obtained. Applying a threshold, this is used to get predictions which are compared with the ground truth labels and the ROC-AUC scores are reported. To get the test edge list, *StellarGraph.EdgeSplitter* –[**StellarGraph**] is used and for directed graphs, 0%, 50%, 100% of negative samples in test set are replaced by flipped positive samples to check if the algorithm predicts edge direction.

3 Results

DeepWalk: In node classification, DeepWalk's performance is comparable to the best performance for any given data set and produces the best results for Cora. But it performs much worse on link prediction and barely better than random guess. While this trend is observed in the original paper, the scores in this project are significantly less than that in the paper. This could be due to the much smaller walk length, walks per path and window size than what the paper used. DeepWalk also generalises well to the Facebook data set this project extends to, and produces competitive results for node classification. With the hardware limitations, DeepWalk fails on very large data sets like Reddit, YouTube and Twitter.

	BlogCatalog		PubMed		Cora		Reddit		Flickr		Youtube		Facebook	
method	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.	mic.	mac.
DeepWalk	32.06	19.82	80.45	79.03	**63.97**	**57.52**	✗	✗	**34.29**	**22.02**	✗	✗	85.32	83.90
Node2vec	29.91	15.97	78.67	77.06	62.95	56.64	✗	✗	32.23	16.98	✗	✗	84.40	83.02
NetMF	**36.70**	**24.72**	**81.12**	**79.66**	62.95	55.87	✗	✗	✗	✗	✗	✗	**87.79**	**87.18**
LINE-1	31.00	18.00	67.64	63.95	49.35	43.99	**89.26**	**84.56**	26.00	11.00	**32.30**	**18.38**	63.33	59.15
GraphSAGE	15.50	3.00	46.37	37.96	16.63	10.70	✗	✗	17.46	1.29	✗	✗	42.48	32.70

Table 3.1: Reproduced results for Multilabel Node Classification in terms of Micro-F1 and Macro-F1 means from 5-fold cross validations. [1]

Node2Vec: Node2Vec's scores are sightly lower than that of DeepWalk across data sets and tasks. But they are much worse than what the original paper reports. The latter could once again be attributed to the choice of hyperparameters which were set to be compatible with the reduced hardware resources. While Node2Vec is computationally expensive in generating the corpus, it does successfully run on all data sets where DeepWalk does and fails in the same large data sets like Reddit, YouTube and Twitter. In link prediction, Node2Vec performs only as well as a random guess and this is the drastically different from the paper. While searching for better hyperparameters could have improved this, it wasn't feasible to do with reduced hardware resources.

NetMF: NetMF provides consistently overall best reproduced results and they are comparable to

method	BlogCat.	Youtube	Reddit	Flickr	Facebook
DeepWalk	0.54	✗	✗	0.56	0.53
Node2vec	0.50	✗	✗	0.50	0.50
NetMF	0.57	✗	✗	✗	0.72
LINE-1	**0.57**	**0.60**	**0.73**	**0.64**	**0.73**
GraphSAGE	0.53	✗	✗	0.58	0.62

Table 3.2: Reproduced results for Link predicted for undirected graphs using 50% edges as training data. [1]

the results of original paper. Given the RAM resources, it was not possible to run neither the small NetMF nor the large NetMF for large datasets like Reddit, Flickr and Youtube but the performance for NetMF was better than all other methods. As observed in the paper, large window size provids better results than small for node classification tasks. NetMF requires symmetric adjacency matrix for matrix factorization hence only undirected link prediction is performed. Here, small window size gave better performance on the link prediction. For the datasets, NetMF finished link prediction gave reasonably similar performance and was close to the reproduced best performer LINE-1.

LINE-1: With the reduced hardware resources, only LINE-1 was able to run for large datasets. In comparison to the results reported by the original paper, the scores were in general lower. For undirected link prediction, in BlogCatalog, a higher score than the paper is obtained. And LINE-1 had the highest score across datasets. In directed graphs, higher scores than the paper is obtained for Twitter and DBLP-Ci.

	Cora			Twitter			DBLP-Ci			Epinion		
method	0%	50%	100%	0%	50%	100%	0%	50%	100%	0%	50%	100%
DeepWalk	0.53	0.51	0.50	✗	✗	✗	0.53	0.51	0.50	0.54	0.52	0.51
Node2vec	0.50	0.50	0.50	✗	✗	✗	0.50	0.50	0.50	0.50	0.50	0.50
LINE-1	**0.70**	**0.60**	**0.51**	**0.55**	**0.52**	**0.50**	**0.67**	**0.59**	**0.50**	**0.64**	**0.59**	**0.59**
GraphSAGE	0.58	0.54	0.51	✗	✗	✗	0.55	0.53	0.51	0.62	0.56	0.54

Table 3.3: Reproduced results for Link predicted for directed graphs. [1]

GraphSAGE: The obtained results are lower than the original results due to lack of high performance hardware. For random walks, the paper recommends using walk length 40 and number of walks 8, but this was computationally prohibitive. Hence the implementation makes use of walk length 10 and number of walks 20. This results is missed higher neighbourhood context. Also the number of iterations is limited to 1000, whereas the original paper no limit. As the datasets have few features, 64 identity features are used for every dataset. More identity features could have been used to improve the results, but this would require more compute.

4 Discussion

From the results, we observe that NetMF outperforms other models in the node classification task. We believe this is due to the low computational resources as random walk based methods like DeepWalk and Node2Vec require large walk lengths to capture higher order neighbourhood topology, and GraphSAGE

[1]Datasets for which the corresponding method failed to finish is indicated with ✗

contains a neural network which requires a lot of compute to train.

For link prediction, the original paper reports ROC-AUC score of 0.007 for the LINE-1 model on the Twitter dataset. But this result seems suspicious for a binary classification task where even a random guess would have resulted in a score of 0.5. In our results, on the other hand, the LINE-1 model gets a score of 0.55 for link prediction on the Twitter dataset.

We agree that the original paper provides a good framework for comparing different methods by introducing a common objective function which operates on their corresponding context graphs. It concludes that there is no one model that outperforms the others for every dataset and a model's performance depends highly on the properties of the graph. In our project we note, NetMF and LINE-1 are the best for node classification and link prediction respectively under resource constraints. So, we conclude that in low resource settings, lighter and simpler models can outperform deeper and complex models.