Benchmarking DeepWalk and Node2Vec

Anjali, Sumanth Varambally
{csz198763,mt6170855}@.iitd.ac.in

ABSTRACT

DeepWalk and Node2Vec are were seminal papers in the field of graph representation learning, paving the way for more advanced and contemporary neural-network based architectures like GCNs. In this report, we benchmark Node2Vec and Deepwalk architecture on several prediction tasks such as pair-wise node classification, multi-class node classification, and link prediction. We observe that, as expected, these transductive architectures fail to outperform contemporary inductive architectures. Our code can be found here: https://github.com/anjaliakg17/COL868_Benchmark

ACM Reference Format:
Anjali, Sumanth Varambally. 2021. Benchmarking DeepWalk and Node2Vec. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the recent surge in popularity of Graph Neural Networks, it has become imperative to process graphs and obtain informative representations for nodes and edges. Rather than directly use the vertices and edges of the graphs for inference, we would like to extract semantically useful vector representations for them, which are then further used for downstream tasks like node classification and link prediction.

There are several aspects involved in extracting semantic features from a graph. Some methods like Node2Vec and DeepWalk aim to produce representations based on structural similarity of nodes. They do so in a transductive manner, i.e. they learn representations for nodes of particular graphs in a task-agnostic manner. However, the model has to be retrained for different graphs, i.e. the representations obtained on one graph are not useful in obtaining representations for other graphs. In comparison, modern methods like GCNs learn representations in an inductive fashion, i.e. model trained on one sets of graphs can be used to extract representations for unseen set of graphs. Typically, the representations so learnt are task-specific, but with a suitable choice of learning objective, task-agnostic representations can be learnt as well.

In this work, we aim to benchmark the Node2Vec and DeepWalk algorithms on the PPI, Protein and Bright Kite datasets. We apply Node2Vec and DeepWalk to these datasets for the tasks of Link Prediction, Pair-wise node classification and multi-class node classification and perform extensive experiments to observe the effects of different parameters on the model performance.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

The most commonly used methods for graph-related learning problems are Graph Neural Networks. The idea is to associate each node with a feature vector, and propagate or pass messages across the nodes of the network through the edges.

Different GNN architectures mainly differ in terms of the aggregation method used to collect and combine feature vectors from neighbourhood nodes [9]. GCN [6] uses max pooling, GraphSAGE [4] uses mean/max/LSTM pooled information and GAT [8] uses transferable attention weights to aggregate information from neighbourhood nodes. PGNNs [9] use sampled anchor nodes to compute the distance of target node from all of these anchor nodes and then learns a non-linear distance-weighted aggregation over the anchor nodes. GraphSAINT [5] uses minibatches that are sampled from the whole graph and then learn full GCNs on each sampled graph.

3 NODE2VEC

Node2Vec [3] uses random walks to sample neighbourhood nodes for a given node, interpolating smoothly between BFS and DFS. Formally, the likelihood of the neighbourhood $N_S(u)$ given the representation $f(u)$ of a given node u is modelled as:

$$\Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} \Pr(n_i|f(u))$$

where the node probability given $f(u)$ is modelled using a softmax function. Further, random walks are simulated with the transition matrix constructed as below:

$$\Pr(c_i = x|c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x) \in E \\ 0 & \text{otherwise} \end{cases}$$

where c_i is the i^{th} node in the walk, E is the set of edges in the graph, Z is a normalizing constant, π_{vx} is the un-normalized transition probability between nodes v and x. Specific to Node2Vec, there are hyper-parameters p and q, which are used to control how fast the walk explores and leaves the node-neighbourhood of starting node u. The return parameter p controls the likelihood of revisiting the current node in a walk, and the in-out parameter q allows the search to differentiate between inward and outward nodes. Mathematically, the transition probability π_{vx} is modelled as:

$$\pi_{vx}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

where d_{tx} denotes the shortest path distance between nodes t and x.

4 DEEPWALK

DeepWalk [7] uses a simlar idea of using random walks to learn feature embeddings. At each vertex, γ random walks of length t are simulated, where γ and t are fixed. The goal is to maximize the

probability of neighbours in the walk, given a vertex representation $f(v_j)$, drawing inspiration from the SkipGram language model. To reduce the complexity of computing the likelihood of a neighbourhood node u_k, given a vertex representation $f(v_j)$, the vertices are assigned to the leaves of a binary tree. The prediction problem then turns into maximizing the probability of a specific path in the tree.

The differences between Node2Vec and DeepWalk are subtle but important. The major difference is the way in which the nodes of the random walk are sampled. In DeepWalk, the next vertex v_{j+1} is picked randomly from among v_j's neighbours, i.e. a first-order Markovian sampling procedure is followed. In Node2Vec, the next node v_{j+1} in the random walk depend on the parameters p and q, based on the current node v_j and the previous node v_{j-1}, i.e. the sampling procedure is second-order Markovian.

5 EXPERIMENTAL FRAMEWORK

In this section, we detail the experimental setup used to compare these two methods on various tasks.

5.1 Datasets

We have used the following datasets:

(1) **PPI**: The **P**rotein-**P**rotein **I**nteraction dataset [10] consists of 24 Protein-Protein Interaction Networks, with each graph having an average of 3000 nodes each and average degree 28.8. The nodes in this network denote different proteins, while the edges represent the interactions between them. Each node may belong to several out of the 121 possible classes. We use this dataset for the task of multi-class node classification and link prediction.
(2) **Protein**: The Protein dataset [1] consists of 1113 protein graphs, with each node labelled with a functional role of the protein. Each node belongs to one out of three possible classes. We use this dataset for the task of pairwise node-classification
(3) **Brightkite**: Brightkite [2] is a former location-based social networking service where users shared their locations by checking in. We use the collected friendship network, which consists of 58,228 nodes and 214,078 edges, and use it for the task of link prediction. Note here that the task of link prediction has an easily interpretable meaning, i.e. given two people, predict whether they could be friends.

5.2 Task Description

Here, we detail the different tasks that we have performed during the course of this benchmarking exercise and the data processing steps for each.

5.2.1 Pair-wise Node Classification. Given two nodes, we try to predict if they belong to the same class, i.e. they have the same node label.

We first learn embeddings on the graphs using Node2Vec and DeepWalk in a transductive manner and obtain embeddings for each node. We then train a binary classifier on the aggregated embeddings to predict whether the pair of nodes represented by the aggregated embedding have the same label. We use the Protein dataset for this task, classifying on a random sample of 1,000,000 pairs of nodes.

5.2.2 Multi-class Node Classification. In this task, we assume that a node has multiple class labels associated with it. Given a node, we aim to predict *all* it's associated class labels.

As before, we learn embeddings on the graphs using Node2Vec and DeepWalk in a transductive manner and obtain embeddings for each node. Then, we train a multi-class classifier on the embeddings to predict all the associated class labels. We use the PPI dataset for this task, with each node being associated with multiple of 121 possible class labels.

5.2.3 Link Prediction. This task consists of predicting whether there can be an edge between two given nodes.

For this task, we perform the following procedure:

(1) Given a graph G with edge set E, sample 50% of the edges to obtain set E_1. Also sample 'negative' edges E_{neg}, i.e. edges between the nodes of G which are not present in E. The new training graph will be G' with edge set $E' = E \backslash E_1$.
(2) Learn node embeddings on G'. The edges in G' comprise of the 'positive' edges in the train set, while a portion of the set E_{neg} is designated as the 'negative' edges in the train set.
(3) To construct the test set, we sample positive edges from E_1 and negative edges from the remaining portion of E_{neg}.

We use both PPI and Brightkite datasets for this task.

5.3 Evaluation Metrics

To evaluate and compare different methods, we use the following evaluation metrics:

- **Precision:** Precision is given by $\left(\frac{\text{True Positives}}{\text{True Positives+False Positives}}\right)$. It denotes the fraction of the positives predicted that were done so accurately. A low precision score indicates a high false positive rate.
- **Recall:** Recall is given by $\left(\frac{\text{True Positives}}{\text{True Positives+False Negatives}}\right)$. It denotes the fraction of the true positives in the dataset that were accurately predicted as positives. A low recall score indicates a high false negative rate.
- **F1 Score:** A high precision or recall score on it's own does not guarantee good performance since either one of false positives or false negatives might be high. Hence, we use F1 score, which is the harmonic mean of Precision and Recall, to give a more balanced measure of the classifier performance. $\text{F1 Score} = \left(\frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}\right)$
- **ROC AUC:** The Area Under the Curve of the Receiver Operator Characteristic denotes the area under the true positive rate (vs) false positive rate curve plotted at various thresholds. Therefore, unlike Precision, Recall and F1 score, the ROC AUC score is independent of the threshold at which the classifier is evaluated, and hence can be thought of as a more comprehensive indicator of classifier performance.

5.4 Implementation Details

In all of our experiments, we used Python 3.7. The implementations of node2vec and DeepWalk that we used were from https://github.

com/eliorc/node2vec and https://github.com/phanein/deepwalk respectively. All experiments were run on CPU in Google Colab, with a single core Xeon CPU with 2 threads.

We perform extensive hyperparameter tuning to determine the effect of each parameter on model performance, and to determine the best performing set of parameters. The range of hyperparameters explored is presented in Table 1. For Pair-wise Node Classification and Link Prediction, we experimented with different aggregation functions to obtain a single embedding; the details of which we have summarized in Table 2.

Parameters	Set of values
Embedding dimension	32, 64, 128
Walk Length	10, 40, 100
Number of Walks	10, 50, 100
Return Parameter p	0.2, 1, 2
In-Out Parameter q	0.2, 1, 2
Window Size	5, 10, 20

Table 1: Hyper-Parameter Ranges for DeepWalk and Node2Vec

Operator	Symbol	Definition
Average	$\boxplus$	$f(u) \boxplus f(v) = \frac{f(u)+f(v)}{2}$
Hadamard	$\boxtimes$	$f(u) \boxtimes f(v) = \left[\frac{f_i(u)\times f_i(v)}{2}\right]_{i=1}^{n}$
L1	$\|\cdot\|_1$	$\|f(u) - f(v)\|_1 = [\|f_i(u) - f_i(v)\|]_{i=1}^{n}$
L2	$\|\cdot\|_2$	$\|f(u) - f(v)\|_2 = \left[\|f_i(u) - f_i(v)\|^2\right]_{i=1}^{n}$

Table 2: Choice of aggregation functions

We have used the Logistic Regression classifier for the Link Prediction task, following the setup detailed in [3] and the Random Forest Classifier for the Multi-Class Classification and Pair-wise Node Classification tasks. All the classifiers were from the sklearn package. For the Multi-Class classification task, we used the Multi-Output Classifier class along with Random Forest from the sklearn package. We use 5-fold cross validation for the Node Classification tasks, and a 80-20% split for Link Prediction task.

6 RESULTS

We report the result for each task and dataset here. In each case, the optimal parameters were obtained through extensive grid search over different ranges of parameter settings.

6.1 Pairwise Node Classification Task

We detail the results of the pair-wise node classification task in Table 3. As can be seen from the table, Node2Vec and DeepWalk perform very poorly, especially in comparison to GNN based methods like P-GNN and GAT. This can be attributed to the fact that these methods can exploit node features better, and also because they can be trained in a task-specific manner, whereas the node embeddings learnt by DeepWalk and Node2Vec are task-agnostic.

Method	Precision	Recall	F1 score	ROC AUC
P-GNN	0.811	0.803	0.576	0.647
GCN	0.495	0.506	0.506	0.506
GraphSAGE	0.499	0.5	0.5	0.5
GAT	**0.896**	**0.829**	**0.829**	**0.829**
Node2Vec	0.513	0.513	0.513	0.502
DeepWalk	0.509	0.509	0.509	0.504

Table 3: Performance of different methods on Pairwise Node Classification on Protein Dataset

6.2 Link Prediction

Tables 4 and 5 respectively show the results of DeepWalk and Node2Vec on PPI and BrightKite datasets. It is surprising to observe that these two simple architectures perform so well, sometimes even outperforming the GNN architectures. We attribute this to two possible reasons - (a) our testing methodology might be slightly different to ones used in other methods (b) the transductive nature of these algorithms might lend them an upper hand in this task. It is also surprising that DeepWalk outperforms Node2Vec on the Brightkite dataset, but this could be due to suboptimal parameter choices for Node2Vec.

Method	Precision	Recall	F1 score	ROC AUC
GCN	0.772	0.655	0.655	0.655
GraphSAGE	0.790	0.667	0.667	0.667
GAT	0.902	0.822	0.822	0.822
P-GNN	0.784	0.730	0.701	0.715
Node2Vec	**0.957**	0.929	0.943	0.944
DeepWalk	0.956	**0.989**	**0.972**	**0.972**

Table 4: Performance of different methods on Link Prediction on PPI Dataset

Method	Precision	Recall	F1 score	ROC AUC
P-GNN	0.795	0.751	0.694	0.721
GAT	0.891	**0.806**	0.806	0.806
GCN	0.925	-	-	-
GraphSAGE	0.934	-	-	-
Node2Vec	0.670	0.770	0.717	0.696
DeepWalk	**0.963**	0.803	**0.876**	**0.886**

Table 5: Performance of different methods on Link Prediction on Brightkite Dataset

6.3 Multi-Class Classification

The results for multi-class classification are presented in Table 6. As expected, while the two transductive methods perform decently, their results wane in comparison to the GNN based architectures, especially GAT. Again, this could be attributed to their ability to exploit node features as well as their specificity.

Method	Precision	Recall	F1 score	ROC AUC
PGNN	0.656	0.621	0.481	0.543
GCN	0.789	0.818	0.744	0.780
GraphSAGE	0.960	**0.957**	**0.953**	**0.955**
GAT	**0.97**	0.924	0.925	0.925
Node2Vec	0.634	0.463	0.536	0.673
DeepWalk	0.643	0.430	0.515	0.662

Table 6: Performance of different methods on Multi-Class Classification on PPI Dataset

6.4 Effect of Hyperparameters

We look at the effect and trends observed when different hyperparameters are varied. For this purpose, we evaluate different hyperparameter settings on the Multi-Class classification task on the PPI dataset.

6.4.1 Embedding Dimension. Table 7 shows the variation of performance of DeepWalk with Embedding dimensions, with all other parameters kept fixed. We observe no discernable pattern or trend in the performance as the embedding dimension is changed. We would like to mention here that the original paper reported a slight increase in performance with increase in embedding dimension, but our results suggest that the correlation is tenuous at best.

Embed. Dimension	ROC	Precision	Recall	F1 Score
32	0.649	0.655	0.387	0.486
64	0.646	0.650	0.383	0.482
128	0.647	0.648	0.387	0.484

Table 7: Effect of Embedding Dimension on Node2Vec performance in Multi-Class Classification

6.4.2 Walk Length. Tables 8 and 9 show the variation of performance of Node2Vec and DeepWalk with walk length, with all other parameters kept fixed. We observe no discernable pattern with Node2Vec, while DeepWalk experiences a slight increase in performance. It is worth mentioning that the original papers report a significant increase in performance with increased walk length, which we were unable to replicate.

Walk Length	ROC	Precision	Recall	F1 Score
10	0.641	0.634	0.377	0.473
40	0.646	0.650	0.382	0.482
100	0.626	0.641	0.334	0.440

Table 8: Effect of Walk Length on Node2Vec performance in Multi-Class Classification

Walk Length	ROC	Precision	Recall	F1 Score
10	0.637	0.624	0.374	0.468
80	0.648	0.639	0.394	0.487
200	0.650	0.641	0.399	0.491

Table 9: Effect of Walk Length on DeepWalk performance in Multi-Class Classification

6.4.3 Number of Walks. Tables 10 and 11 show the variation of performance of Node2Vec and DeepWalk with number of walks, with all other parameters kept fixed. A similar observation as with walk length follows, with the increase in performance reported in the original paper unable to be replicated.

#Walks	ROC	Precision	Recall	F1 Score
10	0.646	0.650	0.382	0.482
20	0.628	0.642	0.341	0.445
50	0.623	0.641	0.327	0.433

Table 10: Effect of Number of Walks on Node2Vec performance in Multi-Class Classification

# Walks	ROC	Precision	Recall	F1 Score
5	0.645	0.638	0.388	0.482
10	0.648	0.640	0.394	0.488
50	0.651	0.641	0.401	0.493

Table 11: Effect of Number of Walks on DeepWalk performance in Multi-Class Classification

6.4.4 Parameters p and q. Tables 12 and 13 represent the trends in performance of Node2Vec with change in parameters p and q respectively. Although we observe an increase in performance with increased p and decreased q, we must note here that these parameter depend heavily on the dataset itself and hence a monotonic trend as observed here might not always be reflected in practice.

p	ROC	Precision	Recall	F1 Score
0.2	0.637	0.645	0.362	0.464
1	0.646	0.650	0.382	0.481
2	0.648	0.652	0.388	0.486

Table 12: Effect of p on Node2Vec performance in Multi-Class Classification

q	ROC	Precision	Recall	F1 Score
0.2	0.648	0.646	0.391	0.487
1	0.646	0.649	0.383	0.481
2	0.643	0.649	0.376	0.476

Table 13: Effect of q on Node2Vec performance in Multi-Class Classification

6.4.5 Window Size. Table 14 represents the variation in performance of DeepWalk with change in window-size. As expected, a slight increase in performance is observed with increase in the window size.

Window Size	ROC	Precision	Recall	F1 Score
5	0.641	0.636	0.378	0.474
10	0.648	0.640	0.393	0.488
20	0.651	0.641	0.401	0.494

Table 14: Effect of Window Size on DeepWalk performance in Multi-Class Classification

6.4.6 Effect of Aggregation function on Link Prediction. Tables 15 and 16 show the performance of Node2Vec and DeepWalk respectively on the PPI dataset, while tables 17 and 18 show the performance of Node2Vec and DeepWalk respectively on the Brightkite dataset. A significant difference in performance is observed across

the different aggregation functions, especially across different architectures and datasets. The Hadamard aggregation function performs well on the PPI dataset, but performs poorly on the Brightkite dataset. However, the L1 and L2 aggregation functions show roughly the same performance, and are the better option in most scenarios.

Agg. Function	ROC	Precision	Recall	F1 Score
Average	0.725	0.711	0.761	0.735
Hadamard	0.923	0.985	0.860	0.918
L1	0.933	0.957	0.906	0.931
L2	0.944	0.957	0.929	0.943

Table 15: Performance of Node2Vec on PPI dataset in Link Prediction using different aggregation functions

Agg. Function	ROC	Precision	Recall	F1 Score
Average	0.685	0.666	0.742	0.702
Hadamard	0.972	0.956	0.989	0.972
L1	0.946	0.977	0.914	0.944
L2	0.950	0.976	0.922	0.948

Table 16: Performance of DeepWalk on PPI dataset in Link Prediction using different aggregation functions

Agg. Function	ROC	Precision	Recall	F1 Score
Average	0.696	0.670	0.770	0.717
Hadamard	0.752	0.927	0.546	0.687
L1	0.748	0.987	0.503	0.667
L2	0.749	0.987	0.504	0.667

Table 17: Performance of Node2Vec on Brightkite dataset in Link Prediction using different aggregation functions

Agg. Function	ROC	Precision	Recall	F1 Score
Average	0.750	0.707	0.856	0.774
Hadamard	0.716	0.807	0.567	0.666
L1	0.883	0.967	0.793	0.871
L2	0.886	0.963	0.803	0.876

Table 18: Performance of DeepWalk on Brightkite dataset in Link Prediction using different aggregation functions

6.5 Timing Analysis

Tables 19 and 20 show the time taken by the Node2Vec and DeepWalk architectures for learning node embeddings for the Link Prediction task on the PPI dataset. Since the most time-sensitive hyperparameter in both these methods is the number of walks, we show the train time for different number of walks and the associated performance. As the tables indicate, a higher train time need not always indicate better performance (as evidenced by the study of the impact of the number of walks on performance earlier). However, a striking observation is that DeepWalk is significantly faster than Node2Vec, with negligable difference in performance. We attribute this to the binary tree optimization performed by the DeepWalk implementation for improved runtime.

#Walks	ROC	Precision	Recall	F1 Score	Train Time (s)
2	0.965	0.943	0.990	0.966	516
10	0.968	0.953	0.985	0.969	1865
20	0.956	0.955	0.958	0.956	3682
40	0.943	0.957	0.927	0.942	7216

Table 19: Timing Analysis of Node2Vec on Link Prediction task on PPI dataset

#Walks	ROC	Precision	Recall	F1 Score	Train Time (s)
2	0.937	0.932	0.943	0.937	69
5	0.955	0.965	0.943	0.954	145
10	0.952	0.971	0.931	0.951	268
20	0.950	0.974	0.924	0.948	507
40	0.950	0.975	0.925	0.949	977

Table 20: Timing Analysis of DeepWalk on Link Prediction task on PPI dataset

7 CONCLUSIONS

We make the following concluding observations:

- In most use-cases, contemporary inductive architectures like GCNs outperform Node2Vec and DeepWalk.
- DeepWalk and Node2Vec are completely unsuited for datasets (like Proteins) where node labels are important informative features. On the other hand, they might work better in scenarios where structural properties are more important (like in link prediction).
- The trends reported in the original papers of the performance of these models with different hyperparameters, are not readily reproducible, and should be taken with a grain of salt.
- It is not necessarily the case that Node2Vec outperforms DeepWalk in all settings, however in most cases the difference in performance is marginal at best.
- In most time-sensitive applications, DeepWalk seems to be a better choice than Node2Vec. Note, however, that this could be an implementation issue. We see no reason that the tree-based optimization cannot be applied on Node2Vec.

REFERENCES

[1] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 8–pp.

[2] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1082–1090.

[3] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[4] William L. Hamilton, Zhitao Ying, and J. Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*.

[5] et.al Hanqing Zeng. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[6] Thomas Kipf and M. Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ArXiv* abs/1609.02907 (2017).

[7] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[9] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817* (2019).
[10] Marinka Zitnik and Jure Leskovec. 2017. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics* 33, 14 (2017), i190–i198.