FAST NODE EMBEDDINGS: LEARNING EGO-CENTRIC REPRESENTATIONS

Tiago Pimentel[1], **Adriano Veloso**[1] **& Nivio Ziviani**[1,2]
[1] CS Dept., UFMG
Belo Horizonte, Brazil
{tpimentel,adrianov,nivio}@dcc.ufmg.br

[2] Kunumi
Belo Horizonte, Brazil
nivio@kunumi.com.br

ABSTRACT

Representation learning is one of the foundations of Deep Learning and allowed important improvements on several Machine Learning tasks, such as Neural Machine Translation, Question Answering and Speech Recognition. Recent works have proposed new methods for learning representations for nodes and edges in graphs. Several of these methods are based on the SkipGram algorithm, and they usually process a large number of multi-hop neighbors in order to produce the context from which node representations are learned. In this paper, we propose an effective and also efficient method for generating node embeddings in graphs that employs a restricted number of permutations over the immediate neighborhood of a node as context to generate its representation, thus ego-centric representations. We present a thorough evaluation showing that our method outperforms state-of-the-art methods in six different datasets related to the problems of link prediction and node classification, being one to three orders of magnitude faster than baselines when generating node embeddings for very large graphs.

1 INTRODUCTION

Many important problems involving graphs require the use of learning algorithms to make predictions about nodes and edges. These predictions and inferences on nodes and edges from a graph are typically done using classifiers with carefully engineered features (Grover & Leskovec, 2016). These features, besides taking time and manual labor to be developed and acquired, usually do not generalize well to other problems or contexts.

The field of Natural Language Processing (NLP) has had many advances due to the use of algorithms that learn word representations, instead of manually extracted features. Originally proposed by Bengio et al. (2003) and commonly used with Word2Vec algorithms like CBOW and SkipGram (Mikolov et al., 2013a), word embeddings are used in many state-of-the-art solutions for neural machine translation (Luong & Manning, 2016; Firat et al., 2016), question answering (Andreas et al., 2016) and natural language generation (Wen et al., 2015). Recent works have proposed new methods for learning representations for nodes and edges in graphs, based on random walks (Perozzi et al., 2014; Grover & Leskovec, 2016) or auto-encoding adjacency vectors (Wang et al., 2016).

In this paper, we propose a new general purpose method for generating node embeddings in very large graphs, which we call Neighborhood Based Node Embeddings (or simply NBNE). NBNE is based on the SkipGram algorithm and uses nodes neighborhoods as contexts. NBNE outperforms state-of-the-art DeepWalk (Perozzi et al., 2014) and Node2Vec (Grover & Leskovec, 2016) for the tasks of link prediction and node classification on six collections, being one to three orders of magnitude faster. In this work, we considered DeepWalk and Node2Vec as baselines.

The main reason for this improvement on effectiveness and efficiency is that we concentrate learning on the "predictable" parts of the graph. A study by Facebook research (Edunov et al., 2016) found that each person in the world (at least among the people active on Facebook) is connected to every other person by an average 3.57 other people. In a graph of this magnitude and connectedness, learning node embeddings by maximizing the log-probability of predicting nearby nodes in a random walk (with a window size of 5) can be highly inefficient and make it 'harder' for the embeddings to be constructed, even if these random walks are biased like in Node2Vec. We suspect this can also make them more unstable, which would explain why they need more iterations before embedding convergence.

The main contributions of this work are:

- We present a **new general purpose method** that is more effective and more efficient than state-of-the-art methods for generating node embeddings in graphs. NBNE has a single tunable parameter (number of permutations, which will be explained later) that can be tuned at once on the training set to avoid overfitting the representations.
- Experimental results in solving the link prediction and node classification problems for six graph sets show that **our method outperforms, in terms of effectiveness and efficiency**, the methods DeepWalk and Node2Vec.
- In spite of the fact that our method has the same time complexity of the two baselines DeepWalk and Node2Vec, we were able to **improve the training time by one to three orders of magnitude**, which is important when dealing with very large graphs. For instance, to learn node embeddings for a graph containing 317,080 nodes and 1,049,866 edges, collected from the DBLP[1] repository (Yang & Leskovec, 2012), NBNE took approximately 14m30s minutes, DeepWalk approximately 164m34s and Node2Vec approximately 3,285m59s (more than 54 hours).

2 Related Work

The definition of node similarity and finding general purpose node and/or edge representations are non-trivial challenges (Lü & Zhou, 2011). Many definitions of similarity in graphs use the notion of first and second order proximity. First-order proximity is the concept that connected nodes in a graph should have similar properties, while the second-order proximity indicates that nodes with similar neighborhoods should have common characteristics.

Some earlier works on finding these embeddings use various matrix representations of the graph, together with dimensionality reduction techniques, to obtain the nodes' representations (Roweis & Saul, 2000; Tenenbaum et al., 2000). A problem with these approaches is that they usually depend on obtaining the matrix' eigenvectors, which is infeasible for large graphs ($O(n^{2.376})$) with the Coppersmith-Winograd algorithm (Coppersmith & Winograd, 1987)). Recent techniques attempt to solve this problem by dynamically learning representations for nodes in a graph using non-linear techniques based either on first and second order proximities (Tang et al., 2015; Wang et al., 2016) or random walks (Perozzi et al., 2014; Grover & Leskovec, 2016).

Other recent works focus on finding representations for specific types of graphs. TriDNR (Pan et al., 2016) uses a graph structure together with node content and labels to learn node representations in two citation networks. Their work can be directly applied to any graph where nodes have labels and/or text contents. TEKE (Wang & Li, 2016) and KR-EAR (Lin et al., 2016) find representations for entities in knowledge graphs and metapath2vec (Dong et al., 2017) finds node representations in heterogeneous networks. The method LINE (Tang et al., 2015) finds a d dimensional representation for each node based on first and second-order graph proximities, not being feasible for large graphs, because its cost function depends on the whole adjacency matrix ($O(|V|^2)$).

Another method, Structural Deep Network Embedding (SDNE) (Wang et al., 2016), is also based on first and second order proximities. It uses autoencoders to learn a compact representation for nodes based on their adjacency matrix (second-order proximity), while forcing representations of connected nodes to be similar (first-order proximity) by using an hybrid cost function. SDNE is also not feasible for large graphs, since the autoenconders are trained on the complete adjacency vectors. Each vector has size $O(|V|)$ and is created at least once, creating a lower bound on time complexity $O(|V|^2)$.

The method DeepWalk (Perozzi et al., 2014) generates k random walks starting on each vertex in the graph to create sentences where each "word" is a node. These sentences are then trained using the SkipGram algorithm to generate node embeddings. This method has a time complexity bounded by $O(|V| \log |V|)$.

[1]http://dblp.uni-trier.de

Node2Vec (Grover & Leskovec, 2016) also uses random walks with SkipGram and can be seen as a generalization of DeepWalk. The difference between the two methods is that Node2Vec's walks are random, but biased by two pre-assigned parameters p and q. During the creation of the walks, these parameters are used to increase the chance of the walk returning to a parent node or going farther from it. This method uses a semi-supervised approach which requires several models to be generated and a small sample of labeled nodes to be used so that the best parameters p and q can be chosen. Node2Vec is not efficient for densely connected graphs, since its time and memory dependencies on the graph's branching factor b are $O(b^2)$.

In this work, we considered DeepWalk (Perozzi et al., 2014) and Node2Vec (Grover & Leskovec, 2016) as baselines, since they are scalable, having a time complexity ($O(|V| \log |V|)$). The main differences between NBNE and the two baselines are: (i) we use a different sentence sampling strategy which is based in a node's neighborhood instead of random walks, (ii) NBNE is more effective than both Node2Vec and DeepWalk, as supported by our experiments in six different datasets, and (iii) NBNE is efficient for both dense and sparse graphs and scalable for very large applications, having a faster training time than both Node2Vec and DeepWalk.

3 Neighborhood Based Node Embeddings

The context of a word is not a straightforward concept, but it is usually approximated by the words surrounding it. In graphs, a node's context is an even more complex concept. As explained above, DeepWalk and Node2Vec use random walks as sentences and consequently as contexts in which nodes appear.

In this work, the contexts are based solely on the neighborhoods of nodes, defined here as the nodes directly connected to it, focusing mainly on the second-order proximities. Consequently, nodes' representations will be mainly defined by their neighborhoods and nodes with similar neighborhoods (contexts) will be associated with similar representations.

3.1 Sentence Generation

In our Neighborhood Based Node Embedding's (NBNE) method, as the name implies, sentences are created based on the neighborhoods of nodes. There are two main challenges in forming sentences from neighborhoods, as follows:

- A sentence containing all the neighbors from a specific highly connected root node might be of little use. Most neighbors would be distant from each other in the sentence, not influencing each other's representations, and not directly influencing the root node.
- There is no explicit order in the nodes in a neighborhood. So there is no clear way to choose the order in which they would appear in a sentence.

In this work, the solution is to form small sentences, with only k neighbors in each, using random permutations of these neighborhoods. Algorithm 1 presents the code for generating sentences. As a trade-off between training time and increasing the training dataset the user can select the number of permutations n. Selecting a higher value for n creates a more uniform distribution on possible neighborhood sentences, but also increases training time.

Algorithm 1 Sentence Sampling

```
1: procedure GETSENTENCES(graph, $n$)
2:     $sentences \leftarrow [\emptyset]$
3:     for $j$ in $0 : n$ do
4:         for $node$ in $graph$.nodes() do
5:             $neighbors \leftarrow$ random_permutation($node$.neighbors())
6:             for $i$ in $0 :$ len($neighbors$)$/k$ do
7:                 $sentence \leftarrow [node] + neighbors[i \cdot k : i \cdot (k+1)]$
8:                 $sentences$.append($sentence$)
```

3.2 Learning Representations

As described in Section 3.1, Algorithm 1 forms a set of sentences S, where each word is actually a node from the graph. We train the vector representations of nodes by maximizing the log probability of predicting a node given another node in a sentence and given a set of representations r. We use a window of size k which is equal to the size of the generated sentences, so that each node in a sentence predicts all the others. The log probability maximized by NBNE is given by:

$$\max_{r} \quad \frac{1}{|S|} \sum_{s \in S} \left(\log \left(p \left(s | r \right) \right) \right) \tag{1}$$

where $p\left(s|r\right)$ is the probability of each sentence, which is given by:

$$\log \left(p \left(s | r \right) \right) = \frac{1}{|s|} \sum_{i \in s} \left(\sum_{j \in s, j \neq i} \left(\log \left(p \left(v_j | v_i, r \right) \right) \right) \right) \tag{2}$$

where v_i is a vertex in the graph and v_j are the other vertices in the same sentence. The probabilities in this model are learned using the feature vectors r_{v_i}, which are then used as the vertex representations. The probability $p\left(v_j|v_i, r\right)$ is given by:

$$p \left(v_o | v_i, r \right) = \frac{\exp \left(r\prime_{v_o}^{T} \times r_{v_i} \right)}{\sum_{v \in V} \left(\exp \left(r\prime_{v}^{T} \times r_{v_i} \right) \right)} \tag{3}$$

where $r\prime_{v_j}^{T}$ is the transposed output feature vector of vertex j, used to make predictions. The representations $r\prime_v$ and r_v are learned simultaneously by optimizing Equation (1). This is done using stochastic gradient ascent with negative sampling (Mikolov et al., 2013b).

By optimizing this log probability, the algorithm maximizes the predictability of a neighbor given a node, creating node embeddings where nodes with similar neighborhoods have similar representations. Since there is more than one neighbor in each sentence, this model also makes connected nodes have similar representations, because they will both predict each others neighbors, resulting in representations also having some first order similarities. A trade-off between first and second order proximity can be achieved by changing the parameter k, which simultaneously controls both the size of sentences generated and the size of the window used in the SkipGram algorithm.

3.3 Avoiding Overfitting Representations

When using large values of n (i.e., number of permutations) on graphs with few edges per node, some overfitting can be seen on the representations, as shown in details in Section 5.1. This overfitting can be avoided by sequentially training on increasing values of n and testing the embeddings on a validation set every few iterations, stopping when performance stops improving, as shown in Algorithm 2.

Algorithm 2 NBNE without Overfitting

```
1: procedure TRAINNBNE(graph, max_n)
2:     $sentences \leftarrow$ get_sentences($graph, max\_n$)
3:     $model \leftarrow$ [initialize_model()]
4:     for $j$ in $0 : log_2(max\_n)$ do
5:         $model \leftarrow$ train($model, sentences[2^j : 2^{j+1}]$)
6:         $error \leftarrow$ test($new\_model, validation\_set$)
7:         if $error > old\_error$ then
8:             break
```

4 Experiments

NBNE was evaluated on two different tasks: link prediction, and node classification.[2] We used a total of six graph datasets to evaluate NBNE and Table 1 presents details about these datasets.

[2]NBNE code is available at `https://github.com/tiagopms/nbne`.

Table 1: Statistics on the first six graph datasets

	Nodes	Edges	Edges/Node	# Classes
Facebook[1] (McAuley & Leskovec, 2012)	4,039	88,234	21.84	-
Astro[1] (Leskovec et al., 2007)	18,772	198,110	10.55	-
PPI[1,2] (Breitkreutz et al., 2008)	3,890	38,739	9.95	49
Wikipedia[1,2] (Mahoney, 2011)	4,777	92,517	19.36	39
Blog[1,2] (Zafarani & Liu, 2009)	10,312	333,983	32.38	39
DBLP[1] (Yang & Leskovec, 2012)	317,080	1,049,866	3.31	-

[1] used in Link Prediction
[2] used in Node Classification

A brief description of each dataset can be found in Appendix A. We present results for the link prediction problem in Section 4.1 and for the node classification problem in Section 4.2. For all experiments we used sentences of size $k = 5$ and embeddings of size $d = 128$, while the number of permutations was run for $n \in \{1, 5, 10\}$. The best value of n was chosen according to the precision on the validation set and we used early stopping, as described in Section 3.3.

On both these tasks, DeepWalk and Node2Vec were used as baselines, having been trained and tested under the same conditions as NBNE and using the parameters as proposed in (Grover & Leskovec, 2016). More specifically, we trained them with the same training, validation and test sets as NBNE and used a window size of 10 (k), walk length (l) of 80 and 10 runs per node (r). For Node2Vec, which is a semi-supervised algorithm, we tuned p and q on the validation set, doing a grid search on values $p, q \in \{0.25; 0.5; 1; 2; 4\}$.

4.1 Link Prediction

Setup. Link prediction attempts to estimate the likelihood of the existence of a link between two nodes, based on observed links and the nodes' attributes (Lü & Zhou, 2011). Typical approaches to this task use similarity metrics, such as Common Neighbors or Adamic-Adar (Adamic & Adar, 2003). Instead of these hand made similarity metrics, we propose to train a logistic classifier based on the concatenation of the embeddings from both nodes that possibly form an edge and predict the existence or not of the edge.

To train NBNE on this task, we first obtained a sub-graph with 90% randomly select edges from each dataset, and obtained the node embeddings by training NBNE on this sub-graph. We, then, separated a small part of these sub-graph edges as a validation set, using the rest to train a logistic regression with the learned embeddings as features.

After the training was completed, the unused 10% of the edges were used as a test set to predict new links. 10-fold cross-validation was used on the entire training process to access the statistical significance of the results, analyzing statistical difference between the baselines and NBNE. To evaluate the results on this task, we used as metrics: AUC (area under the ROC curve) (Baeza-Yates & Ribeiro-Neto, 2011), and training time.[3] The logistic regressions were all trained and tested using all available edges (respectively in the training or test set), and an equal sized sample of negative samples, which, during training, included part of the 10% removed edges.

Results. Table 2 presents results for this task. Considering AUC scores on the Link Prediction task, NBNE was statistically better[4] than both DeepWalk and Node2Vec on the Astro and PPI datasets, with more than 7% improvement, also showing a 4.67% performance gain in Wikipedia and a small, but statistically significant, gain on Blog. Only losing by a small percentage on Facebook, with a difference that was not statistically significant.

In DBLP, NBNE again presents the best AUC score, although this difference was small and its statistical significance could not be verified due to the large training times of the baselines. This dataset contains the largest graph analyzed in this work (317,080 nodes and 1,049,866 edges) and

[3]Training times were all obtained using 16 core processors, running NBNE, Node2Vec or DeepWalk on 12 threads, with all algorithms having been implemented using gensim (Řehůřek & Sojka, 2010).

[4]In all experiments we performed Welch's t-tests with $p = 0.01$. The symbol * marks results which are statistically different from NBNE.

Table 2: Link prediction results

	Facebook		Astro		PPI	
	AUC	Training Time	AUC	Training Time	AUC	Training Time
NBNE	0.9688	**0m11s**	**0.8328**	**0m07s**	**0.8462**	**0m02s**
DeepWalk	0.9730	2m26s	0.7548*	6m55s	0.7741*	2m30s
Node2vec	**0.9762**	69m33s	0.7738*	182m16s	0.7841*	66m37s
Gain	-0.76%	12.96x 369.85x	7.62%	59.06x 1555.80x	7.91%	77.43x 2061.67x

	Wikipedia		Blog		DBLP	
	AUC	Training Time	AUC	Training Time	AUC	Training Time
NBNE	**0.6853**	**0m02s**	**0.9375**	**1m11s**	**0.9335**†	**14m30s**
DeepWalk	0.6534*	7m38s	0.9098*	28m13s	0.9242‡	164m34s
Node2Vec	0.6547*	236m60s	0.9202*	838m41s	0.9322‡	3,285m59s
Gain	4.67%	194.86x 6049.77x	1.88%	23.86x 709.24x	0.13%	11.34x 226.52x

† average of 10 fold results
‡ no statistical tests were run, due to the time necessary to run a single fold

in it, to train a single fold, Node2Vec took 3,285m59s (more than 54 hours) and DeepWalk took 164m34s (approximately 2 hours and 44 minutes), while NBNE took only 14m30s, which represents a 226/11 times improvement over Node2Vec and DeepWalk, respectively.

Considering training time for this task, NBNE presents the biggest improvements on sparser networks of medium size, like Astro, PPI and Wikipedia datasets. On these graphs, the best results are for $n = 1$, resulting in more than 50x faster training than DeepWalk and more than $1{,}500$ times faster than Node2Vec, achieving a 6,049 times faster training than Node2Vec on Wikipedia. For the Blog and Facebook datasets the best results are for $n = 5$, resulting in larger training times, but still more than one order of magnitude faster than DeepWalk and more than 350 times faster than Node2Vec. For the DBLP dataset, the best results were achieved with $n = 10$, still much faster than the baselines.

4.2 Node Classification

Setup. Given a partially labeled graph, node classification is the task of inferring the classification of the unknown nodes, using the structure of the graph and/or the properties of the nodes. In this task, the node embeddings were trained using NBNE on the complete graph. After obtaining the node embeddings, 80% of the labeled nodes in the graph were used to train a logistic classifier that predicted the class of each node, while 5% of the nodes were used for validation and the remaining 15% nodes were used as a test set. This test was repeated for 10 different random seed initializations to access the statistical relevance of the results.

Results. Results on the Blog, PPI and Wikipedia datasets are shown in Table 3 and are presented in terms of Macro F1 scores and training times. NBNE produces statistically similar results to its baselines, in terms of Macro F1, on both PPI and Wikipedia, while showing a statistically significant 22.45% gain in the Blog dataset, indicating that NBNE's embeddings did not only get a better accuracy on Blog, but also that correct answers were better distributed across the 39 possible classes.

Considering training times, NBNE is more than 10 times faster than DeepWalk on these three datasets and is $[300 \sim 600]$ times faster than Node2Vec. NBNE didn't show statistically worse result in any dataset analyzed here, while having an order of magnitude faster training time than DeepWalk and more than two orders of magnitude faster training time than Node2Vec.

5 Further Analysis

NBNE is further analyzed in this section, with Section 5.1 presenting a detailed evaluation of hyper-parameter n, while Section 5.2 shows an analysis of NBNE's computational complexity.

Table 3: Node classification results

	Blog		PPI		Wikipedia	
	Macro F1	Training Time	Macro F1	Training Time	Macro F1	Training Time
NBNE	**0.2004**	**1m57s**	0.0978	**0m16s**	**0.0727**	**0m41s**
DeepWalk	0.1451*	31m31s	**0.0991**	3m04s	0.0679	13m04s
Node2vec	0.1637*	959m12s	0.0971	83m02s	0.0689	408m00s
Gain	22.45%	16.18x 492.57x	-1.35%	11.82x 319.78x	5.56%	19.04x 594.62x

5.1 Number of Permutations (n)

The quality of NBNE's embeddings depends on both the size of the embeddings (d) and the number of permutations (n). For highly connected graphs, larger numbers of permutations should be chosen ($n \in [10, 1000]$) to better represent distributions, while for sparser graphs, smaller values can be used ($n \in [1, 10]$).

Figure 1 shows AUC scores versus embedding sizes for several values of n on the Facebook link prediction task. Quadratic functions approximating $\log(auc\_score)$ were plotted to allow for a better understanding of the results. It can be seen that for larger numbers of permutations ($n > 100$) results improve with embedding size, while for small ones ($n = 1$) they decrease. The plot also shows that $n = 10$ gives fairly robust values for all tested embedding sizes.

A further analysis can be seen in Table 4, which shows that graphs with more edges per node tend to get better results with larger values of n, while graphs with a smaller branching factor have better results with only one permutation ($n = 1$). Other factors also enter into account when choosing n, like graph size. For example, link prediction on the DBLP graph had its best results for $n = 10$, although its branching size was only 3.31.

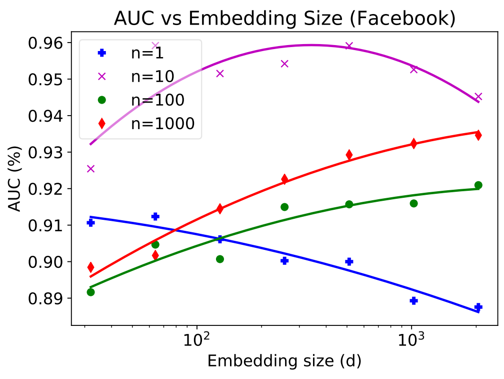

Figure 1: (Color online) NBNE AUC scores vs embedding sizes on Facebook dataset with 50% edges removes

Table 4: Link Prediction results for varying n with NBNE

n	PPI (9.95†)			Facebook (21.84†)			Blog (32.38†)		
	Precision		AUC	Precision		AUC	Precision		AUC
	Train	Test		Train	Test		Train	Test	
10	0.7071	0.7108	0.7795	**0.8453**‡	0.9061	0.9642	**0.8771**‡	0.8627	**0.9348**‡
5	0.7280	0.7305	0.8071	**0.8408**‡	**0.9070**	**0.9688**	**0.8775**‡	**0.8681**	**0.9375**‡
1	**0.7822**	**0.7751**	**0.8462**	0.8036	0.8410	0.9150	0.8115	0.8374	0.9146

† Edges per node ‡ No statistical difference

5.2 Time Complexity

SkipGram's time complexity is linear on the number of sentences, embedding size (d) and logarithmic on the size of the vocabulary (Mikolov et al., 2013a). Since the number of sentences is linear on the number of permutations (n), branching factor of the graph (b) and on the number of nodes, which is the size of the vocabulary ($|V|$), the algorithm will take a time bounded by:

$$O\left(d \cdot n \cdot b \cdot |V| \cdot log(|V|)\right)$$

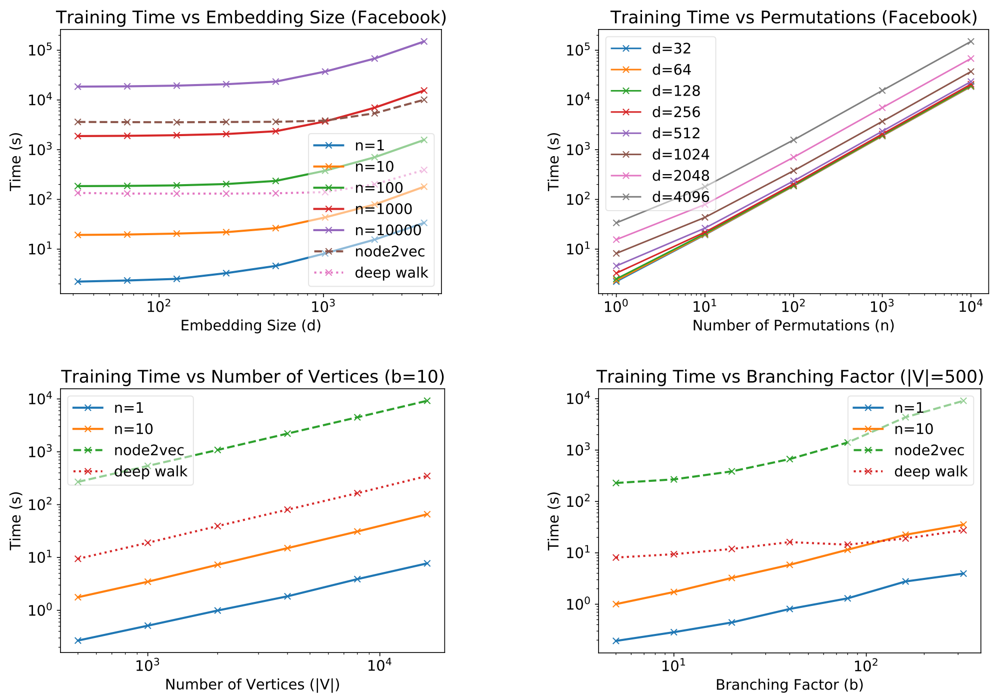

Figure 2: (Color online) **(Top)** Facebook dataset with 30% edges removed: Training times vs **(Left)** embedding size **(Right)** number of permutations. **(Bottom)** Randomly generated graphs: Training times vs **(Left)** number of vertices with $b = 10$ **(Right)** branching factor with $|V| = 500$.

Figure 2 (Top-Left and Top-Right) show training time is indeed linear on both embedding size and number of permutations. It also shows that Node2Vec is considerably slower than DeepWalk, and only has a similar training time to running NBNE with at least $n = 1000$. NBNE with $n < 10$ was by far the fastest algorithm.

NBNE, Node2Vec and DeepWalk run in a time bounded by $O(|V| \log |V|)$, as can be seen in Figure 2 (Bottom-Left). Figure 2 (Bottom-Right) shows that NBNE's time complexity is linear in the branching factor b, while Node2Vec's is quadratic. DeepWalk's running time is constant in this parameter, but for a graph with a larger branching factor, a higher number of walks per node should be used to train this algorithm, which would make it indirectly dependent on this factor.

6 Conclusions

The proposed node embedding method NBNE shows results similar or better than the state-of-the-art algorithms Node2Vec and DeepWalk on several different datasets. It shows promising results in two application scenarios: link prediction and node classification, while being efficient and easy to compute for large graphs, differently from other node embedding algorithms, such as LINE (Tang et al., 2015) or SDNE (Wang et al., 2016).

NBNE focuses learning on node's immediate neighbors, creating more ego-centric representations, which we suspect makes them more stable and faster to learn. Empirical results show that, although it has a similar time complexity, NBNE can be trained in a fraction of the time taken by DeepWalk (10 to 190 times faster) or Node2Vec (200 to 6,000 times faster), giving fairly robust results. Since embeddings are learned using only a node's immediate neighbors, we suspect it should also be easier to implement more stable asynchronous distributed algorithms to train them, and we leave this as future work.

ACKNOWLEDGMENTS

This work was partially funded by projects InWeb (grant MCT/CNPq 573871/2008- 6) and MASWeb (grant FAPEMIG/PRONEX APQ-01400-14), and by the authors individual grants from CNPq, FAPEMIG and Kunumi.

REFERENCES

Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Learning to compose neural networks for question answering. *NAACL-HLT*, pp. 1545–1554, 2016.

Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search*. Pearson Education Ltd., Harlow, England, 2011.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(2):1137–1155, 2003.

Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael S. Livstone, Rose Oughtred, Daniel H. Lackner, Jürg Bähler, Valerie Wood, Kara Dolinski, and Mike Tyers. The biogrid interaction database: 2008 update. *Nucleic Acids Research*, 36(Database-Issue):637–640, 2008.

Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *STOC*, pp. 1–6, 1987.

Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pp. 135–144, 2017.

Sergey Edunov, Carlos Diuk, Ismail Onur Filiz, Smriti Bhagat, and Moira Burke. Three and a half degrees of separation. *Research at Facebook*, 2016.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. Multi-way, multilingual neural machine translation with a shared attention mechanism. *NAACL-HLT*, pp. 866–875, 2016.

Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *KDD*, pp. 855–864, 2016.

Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *Transactions on Knowledge Discovery from Data*, 1(1):2, 2007.

Arthur Liberzon, Aravind Subramanian, Reid Pinchback, Helga Thorvaldsdóttir, Pablo Tamayo, and Jill P. Mesirov. Molecular signatures database 3.0. *Bioinformatics*, 27(12):1739–1740, 2011.

Yankai Lin, Zhiyuan Liu, and Maosong Sun. Knowledge representation learning with entities, attributes and relations. In *IJCAI*, pp. 2866–2872, 2016.

Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.

Minh-Thang Luong and Christopher D Manning. Achieving open vocabulary neural machine translation with hybrid word-character models. *ACL*, pp. 1054–1063, 2016.

Matt Mahoney. Large text compression benchmark, 2011.

Julian J McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *NIPS*, volume 2012, pp. 548–56, 2012.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pp. 3111–3119, 2013b.

Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. Tri-party deep network representation. In *IJCAI*, pp. 1895–1901, 2016.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pp. 701–710, 2014.

Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *LREC 2010 Workshop on New Challenges for NLP Frameworks*, pp. 45–50, 2010.

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pp. 1067–1077, 2015.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD*, pp. 1225–1234, 2016.

Zhigang Wang and Juan-Zi Li. Text-enhanced representation learning for knowledge graph. In *IJCAI*, pp. 1293–1299, 2016.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *EMNLP*, pp. 1711–1721, 2015.

Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, pp. 745–754, 2012.

R. Zafarani and H. Liu. Social computing data repository at ASU, 2009.

A DATASETS

We used a total of six graph datasets to evaluate NBNE, with DeepWalk and Node2Vec being used as baselines. Next we briefly describe these datasets:

1. Facebook (McAuley & Leskovec, 2012): A snapshot of a subgraph of Facebook, where nodes represent users and edges represent friendships.
2. Astro (Leskovec et al., 2007): A network that covers scientific collaborations between authors whose papers were submitted to the Astrophysics category in Arxiv.
3. Protein-Protein Interactions (PPI) (Breitkreutz et al., 2008): We use the same subgraph of the PPI network for Homo Sapiens as in (Grover & Leskovec, 2016). This subgraph contains nodes with labels from the hallmark gene sets (Liberzon et al., 2011) and represent biological states. Nodes represent proteins, and edges indicate biological interactions between pairs of proteins.
4. Wikipedia (Mahoney, 2011): A co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. Labels represent Part-of-Speech (POS) tags.
5. Blog (Zafarani & Liu, 2009): A friendship network, where nodes are bloggers and edges are friendships between them. Each node in this dataset has one class which is referent to the blogger's group.
6. DBLP (Yang & Leskovec, 2012): A co-authorship network where two authors are connected if they published at least one paper together.