Efficient Estimation of Node Representations in Large Graphs using Linear Contexts

Tiago Pimentel
CS Dept., UFMG & Kunumi
Brazil
tiago.pimentel@kunumi.com

Rafael Castro
CS Dept., UFMG
Brazil
rafael.castro@dcc.ufmg.br

Adriano Veloso
CS Dept., UFMG
Brazil
adrianov@dcc.ufmg.br

Nivio Ziviani
CS Dept., UFMG & Kunumi
Brazil
nivio@kunumi.com

***Abstract*—Learning distributed representations in graphs has a rising interest in the neural network community. Recent works have proposed new methods for learning low dimensional embeddings of nodes and edges in graphs and networks. Several of these methods rely on the SkipGram algorithm to learn distributed representations, and they usually process a large number of multi-hop neighbors in order to produce the context from which node representations are learned. This is a limiting factor for these methods as graphs and networks keep growing in size. In this paper, we propose a simple alternate method which is as effective as previous methods, but being much faster at learning node representations. Our proposed method employs a restricted number of permutations over the immediate neighborhood of a node as context to generate its representation, thus avoiding long walks and large contexts while learning the 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.**

***Index Terms*—Network Analysis, Representation Learning**

I. Introduction

Many important problems involving graphs require the use of learning algorithms to make predictions about nodes and edges [1], such as link prediction [2]–[4] and node/edge classification [5]. These predictions and inferences on nodes and edges from a graph are typically done using classifiers with carefully engineered features [6], which takes time and manual labor to be acquired and usually do not generalize well to other problems or contexts.

Instead of manually extracted features, Word2Vec [7] learns word embeddings from raw text by predicting a word from its neighbors. This method is called SkipGram and a words' context is determined by sliding a window in each sentence. Word embeddings are used in many state-of-the-art solutions for neural machine translation [8], question answering [9] and natural language generation [10].

Learning representations for nodes and edges in graphs can be more complex, though, in some senses. While text can be seen as one dimensional, each node has a different number of connections. At the same time, while a word might appear several times in the text, each node appears in only one place in the graph. New algorithms have been proposed to learn representations for nodes and edges in graphs and three representatives are DeepWalk [11], Node2Vec [6] and SDNE (Structural Deep Network Embedding) [12].

DeepWalk uses information obtained from random walks as the equivalent of text sentences. It generates the random walks starting on each node to create sentences where each word is a node. The context is represented by these sentences and the SkipGram method is used to generate node embeddings. Node2Vec learns a mapping of nodes to a low dimensional space of features that is based on the notion of a node's graph neighborhood. The context is defined by a second order random walk to generate node neighborhoods. SDNE also deals with the highly non-linear graph structure by exploiting first-order and second-order node proximity to capture both the local and the global graph structure [13].

In this work, we propose an efficient and effective algorithm to generate graph embeddings in very large graphs, which we call NBNE (Neighborhood Based Node Embeddings). NBNE is based on the SkipGram idea to define context as the nodes directly connected to each node of the graph. NBNE separates neighbors of nodes in small groups using random permutations, and then it maximizes the log likelihood of predicting a node given another node in a group.

NBNE is efficient because it concentrates learning on the most predictable parts of the graph, which is obtained by one-hop neighborhood of each node, thus forcing SkipGram to work with small contexts. In this case, a representation of a node is obtained by predicting its neighbors and nodes with similar neighborhoods (or contexts) are also associated with similar representations.

The quality of these representations is compared with DeepWalk, Node2Vec and SDNE, considered here as strong baselines. We observe improvements in accuracy at much lower computational costs. Experimental results for six graphs obtained from different datasets show that NBNE is as effective as the three baselines but at one to three orders of magnitude faster. For instance, to learn node embeddings for graphs as large as $+300\,000$ nodes and $+1\,000\,000$ edges, NBNE took approximately 14 minutes, DeepWalk approximately 164 minutes and Node2Vec approximately $3\,285$ minutes.

We thank the partial support given by the Project: Models, Algorithms and Systems for the Web (grant FAPEMIG / PRONEX / MASWeb APQ-01400-14), and authors' individual grants and scholarships from CNPq and Fapemig. We gratefully acknowledge the support of Kunumi with the donation of the GPUs used for this research.

The main contributions of this work are:

- We propose an efficient and effective algorithm to generate graph embeddings in very large graphs.
- We provide a thorough evaluation of our algorithm in real and synthetic graphs, motivating our design choices.
- Experimental results in solving the node classification and link prediction problems for six graphs show that our algorithm is as effective as DeepWalk, Node2Vec and SDNE at a much lower computational cost.

II. Related Work

The definition of node similarity and finding general purpose node and/or edge representations are non-trivial challenges [4]. 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 node representations [14]. 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 [15]). 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 [12], [16] or random walks [6], [11]. Other recent works focus on finding representations for specific types of graphs. TriDNR [17] uses a graph structure coupled with node content and labels to learn node representations in citation networks. Their work can be directly applied to any graph where nodes have labels and/or text contents. TEKE [18] and KR-EAR [19] find representations for entities in knowledge graphs, and metapath2vec [20] finds representations in heterogeneous networks where edges can have different types.

The SDNE algorithm [12] is 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 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 DeepWalk algorithm [11] 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. DeepWalk has a time complexity bounded by $O(|V|)$. The Node2Vec algorithm [6] also uses random walks with SkipGram and can be seen as a generalization of DeepWalk. The difference between the two algorithms is that Node2Vec's random walks are 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. Node2Vec 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 [11] and Node2Vec [6] as strong baselines, since they are highly effective. The main differences between NBNE and the two baselines are: (i) NBNE uses a different context 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 graphs, and (iii) NBNE is efficient for both dense and sparse graphs and scalable for very large applications, having a much faster training time than both Node2Vec and DeepWalk. We also compared NBNE with SDNE [12], which is not scalable to large graphs, having a time complexity $O(|V|^2)$. Although having a large time complexity, SDNE can be highly parallelized in modern GPUs, so we ran SDNE on both CPU and GPU to compare its performance with NBNE.

III. NBNE: 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 in Section II, DeepWalk and Node2Vec use random walks as sentences and consequently as contexts in which nodes appear. In constrast, in this work the contexts are based solely on the neighborhoods of nodes, defined here as the nodes directly connected to it. Consequently, nodes' representations will be mainly defined by their neighborhoods and nodes with similar neighborhoods (contexts) will be associated with similar representations. This results in embeddings focused mainly on the second-order proximities. More specifically, NBNE separates a nodes' neighborhood in small groups and then maximizes the log likelihood of predicting a node given another in such a group. Group sampling is defined in Section III-A and the method to learn the representations from the groups is further explained in Section III-B. Section III-C describes how to reduce overfitting while learning the representations.

A. Groups Generation

NBNE groups nodes based on their neighborhoods. There are two main challenges in forming groups from neighborhoods, as follows:

- Nodes have different degrees, so groups containing all the neighbors from a node are difficult to treat.
- 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 group.

To deal with these challenges, NBNE forms small groups with only k neighbors in each, using random permutations of their neighborhoods. The number of permutations n is specified and controls the trade-off between training time and

increasing the training dataset. Selecting a higher value for n creates a more uniform distribution on possible neighborhood groups, but also increases training time.

Algorithm 1 Groups Sampling

```
procedure GETGROUPS(graph, n, k)
    groups ← [∅]
    for j in 0 : n do
        for node in graph.nodes() do
            neighbors ← permutation(node.neighbors())
            for i in 0 : len(neighbors)/k do
                group ← [node]+neighbors[i·k : (i+1)·k]
                groups.append(group)
```

B. Learning Representations

As described in Section III-A, we first create a set of groups S, where each member of S is a node from the graph. We then train the vector representations of nodes by maximizing the log likelihood of predicting a node given another node in a group and given a set of representations r, making each node in a group predict all the others. The log likelihood 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 group, 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 group. 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:

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

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 the same mathematical formulation of SkipGram [7], and is performed using stochastic gradient ascent with negative sampling [21].

By optimizing this log probability, the algorithm maximizes the likelihood of predicting a neighbor given a node, creating node embeddings so that nodes with similar neighborhoods have similar representations. Since there is more than one neighbor in each group, this model also makes connected nodes having similar representations, because they will both predict each others neighbors, resulting in representations also with first order similarities. A trade-off between first and second order proximities can be achieved by changing the parameter k, which controls the number of nodes within each generated group.

TABLE I
STATISTICS ON THE FIRST SIX GRAPH DATASETS

	Nodes	Edges	Edges/Node	# Classes
Facebook[1]	4 039	88 234	21.84	–
Astro[1]	18 772	198 110	10.55	–
PPI[1,2]	3 890	38 739	9.95	49
Wikipedia[1,2]	4 777	92 517	19.36	39
Blog[1,2]	10 312	333 983	32.38	39
DBLP[1]	317 080	1 049 866	3.31	–

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

C. Avoiding Overfitting while Learning Representations

When using large values of n (i.e., number of permutations) on graphs with few edges per node, overfitting may be seen on the representations, as shown in details in Section V-A. We can prevent overfitting by sequentially training on increasing values of n and testing performance on a validation set every few iterations, stopping when performance stops improving.

IV. EXPERIMENTS

NBNE was evaluated on two different tasks: link prediction and node classification.[1] We used six graphs, for which we present results for the link prediction problem in Section IV-A and for the node classification problem in Section IV-B. For all experiments we used groups 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 III-C.

DeepWalk, Node2Vec and SDNE were used as baselines, having been trained and tested under the same conditions as NBNE and using the parameters as proposed in [6] and [12]. 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, we tuned p and q on the validation set, doing a grid search on values $p, q \in \{0.25; 0.5; 1; 2; 4\}$. The SDNE algorithm has a time complexity of $O(|V|^2)$, but its main computation, which is calculating the gradients of its cost function and updating model parameters, can be highly parallelized in modern GPUs. Thus, SDNE was implemented using Tensorflow [22] and trained using a dedicated K40 GPU with CUDA 8.0 and a dedicated 16 core linux server. We fixed $\alpha = 0.2$ and $\beta = 10$, since it was stated that these values commonly give the best results. We used a SDNE architecture with $[10\,300; 1\,000; 128]$ nodes on each layer. We train these embeddings using $\nu \in \{0.1, 0.01, 0.001\}$ and choose the best value on the same validation sets used to tune n for NBNE.

Datasets We used six graphs in order to evaluate NBNE, which are described next:

1) Facebook [23]: A subgraph of Facebook, where nodes represent users and edges represent friendships.
2) Astro [24]: A network that covers collaborations between authors whose papers were submitted to the Astrophysics category in Arxiv.

[1]The code for NBNE is available at https://github.com/tiagopms/nbne

3) Protein-Protein Interactions (PPI) [25]: We use the same subgraph of the PPI network for Homo Sapiens as in [6]. This subgraph contains nodes with labels from the hallmark gene sets [26] and represent biological states. Nodes represent proteins, and edges indicate biological interactions between pairs of proteins.
4) Wikipedia [27]: A co-occurrence network of words appearing in the first million bytes of the Wikipedia dump. Labels represent Part-of-Speech tags.
5) Blog [28]: A friendship network, where nodes are bloggers and edges indicate friendships. Each node in this dataset has one class which refers to the blogger's group.
6) DBLP [29]: A co-authorship network where two authors are connected if they published a paper together [30].

Table I presents details about these graphs. We can see that Blog is the graph with the largest branching factor, 32.38, and DBLP is the one with the smallest, 3.31. At the same time, PPI is the smallest graph, with only 3 890 nodes, while DBLP, the largest, as it contains 317 080 nodes and 1 049 866 edges.

A. Link Prediction

Link prediction attempts to estimate the likelihood of the existence of a link between two nodes, based on observed links [4]. Typical approaches to this task are based on similarity metrics, such as Common Neighbors or Adamic-Adar [31]. [32] presents theoretical justifications for the performance of these similarity metrics, while formalizing the link prediction problem as one of estimating distances between nodes in latent spaces. Instead of these heuristic-based 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 of the edge.

To train the algorithms on this task, we first obtained a sub-graph with 90% of the edges from each graph uniformly selected at random, and obtained the node embeddings by training the algorithms on this sub-graph. Then, we 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 training is completed, the remaining 10% of the edges are used as a test set to predict new links. 10-fold cross-validation was used on the entire training process to assess the statistical significance of the results, enabling us to compare NBNE with the baselines.[2] To evaluate the results on this task, we used as metrics: AUC (area under the ROC curve) [33] 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: As shown in Table II, NBNE presented statistically higher AUC scores than both DeepWalk and Node2Vec on all graphs except Facebook, in which there was no statistically significant difference. NBNE was better than the baselines 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. The difference in performance on Facebook is not statistically significant.

In DBLP, NBNE again presents the best AUC score, although the difference was small and its statistical significance could not be verified due to the large training times of the baselines. 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/DeepWalk, respectively.

In terms of training time, NBNE shows the biggest improvements on sparser networks of medium size, like Astro, PPI and Wikipedia graphs. On these graphs, the best results are obtained with $n = 1$, resulting in more than 50 times 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 graphs the best results are obtained with $n = 5$, 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 graph, the best results were achieved with $n = 10$, still much faster than the baselines.

Table II also shows the results obtained with SDNE. As we can see, both NBNE and SDNE algorithms are very competitive in terms of AUC scores. NBNE outperformed SDNE in Facebook and Astro graphs, while SDNE outperformed NBNE in the PPI graph. It is clear that even when training SDNE using a K40 GPU, NBNE still has more than an order of magnitude faster training time on all graphs, being more than two orders of magnitude faster on most of the graphs. When comparing with SDNE trained on a CPU, NBNE has more than three orders of magnitude faster training time. On Astro, the graph with the largest number of nodes analyzed here, NBNE had a 2 009 times faster training time compared with SDNE on a GPU and 44 896 times faster compared to SDNE on CPU.[4]

B. Node Classification

Given a partially labeled graph, node classification is the task of inferring the class of the unknown nodes, using the structure of the graph and/or the properties of the nodes. In this section, we again compare NBNE with our baselines, now analyzing node classification tasks on the Blog, PPI and Wikipedia graphs.

For each algorithm, the node embeddings were trained using 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

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

[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 [34].

[4]We tried running SDNE with the DBLP dataset, but after five days it had not reached half of the training, so we stopped it.

TABLE II
LINK PREDICTION RESULTS

	Facebook		Astro		PPI	
	AUC	Training Time	AUC	Training Time	AUC	Training Time
NBNE	0.969	0m11s	0.833	0m07s	0.846	0m02s
DeepWalk	0.973	2m26s	0.755*	6m55s	0.774*	2m30s
Node2vec	0.976	69m33s	0.774*	182m16s	0.784*	66m37s
SDNE	0.951*	20m34s† 242m10s	0.816*	234m24s† 5 237m59s	0.875*	16m10s† 232m01s

	Wikipedia		Blog		DBLP	
	AUC	Training Time	AUC	Training Time	AUC	Training Time
NBNE	0.685	0m02s	0.937	1m11s	0.933	14m30s
DeepWalk	0.653*	7m38s	0.910*	28m13s	0.924	164m34s
Node2Vec	0.655*	236m60s	0.920*	838m41s	0.932	3 285m59s
SDNE	0.678	22m23s† 337m47s	0.946	81m33s† 1 492m47s	–	–

* Statistically different from NBNE. † Training time on GPU.

TABLE III
NODE CLASSIFICATION RESULTS

	Blog		PPI		Wikipedia	
	Macro F1	Training Time	Macro F1	Training Time	Macro F1	Training Time
NBNE	0.200	1m57s	0.098	0m16s	0.073	0m41s
DeepWalk	0.145*	31m31s	0.099	3m04s	0.068	13m04s
Node2vec	0.164*	959m12s	0.097	83m02s	0.069	408m00s
SDNE	0.1364*	96m48s† 1 476m33s	0.076*	16m52s† 231m04s	0.091*	19m60s† 338m40s

* Statistically different from NBNE. † Training time on GPU.

15% nodes were used as a test set. This entire process was repeated for 10 different random seed initializations to access the statistical relevance of the results.

Results: Results on the Blog, PPI and Wikipedia graphs are shown in Table III and are presented in terms of Macro F1 scores and training times. NBNE achieved results that are statistically similar to Node2Vec and DeepWalk, in terms of Macro F1, on both PPI and Wikipedia, while showing a statistically significant gain of 22.45% in the Blog graph, 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 to 600 times faster than Node2Vec, while not showing statistically worse AUC score results in any graph analyzed here.

Table III also shows the results obtained with SDNE. As can be seen, NBNE achieved superior results on two graphs, with a gain of 29.27% on PPI and 46.94% on Blog, but losing on Wikipedia by a margin of −20.20%. Still, NBNE is more than an order of magnitude faster than SDNE on GPU in this graph, being more than two orders of magnitude faster when SDNE is trained on CPU.

V. GRAPH ANALYSIS AND MODEL PERFORMANCE

In this section we investigate the extent to which model performance is impacted by graph properties, such as assortativity, size and sparseness.

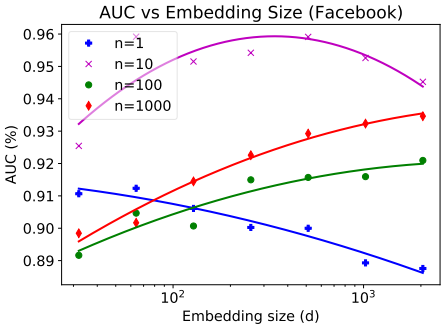

Fig. 1. (Color online) NBNE AUC scores vs embedding sizes on Facebook graph with 50% edges removed.

A. Number of Permutations (n) and Embedding Sizes (d)

The quality of NBNE 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 for several values of n and d on the Facebook link prediction task. For larger numbers of permutations ($n > 100$) results improve with embedding size, while for small values ($n = 1$) 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 IV, which indicates 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, the best result on the DBLP graph was obtained with $n = 10$, despite its small branching size of 3.31.

Figure 2 shows training time is indeed linear on both embedding size and number of permutations. The initial "flatness" implies the algorithm has a large constant in its computational time, and linearity is implied by the line inclination. This figure 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.

TABLE IV
LINK PREDICTION RESULTS FOR VARYING n WITH NBNE

	PPI (9.95†)		Facebook (21.84†)		Blog (32.38†)	
n	Precision	AUC	Precision	AUC	Precision	AUC
10	0.711	0.779	0.906‡	0.964‡	0.863‡	0.935‡
5	0.730	0.807	0.907‡	0.969‡	0.868‡	0.937‡
1	0.775	0.846	0.841	0.915	0.837	0.915

† Edges per node ‡ No statistical difference

B. Assortativity Analysis

Assortativity, also referred to as homophily in social network analysis, is a preference of nodes to attach themselves

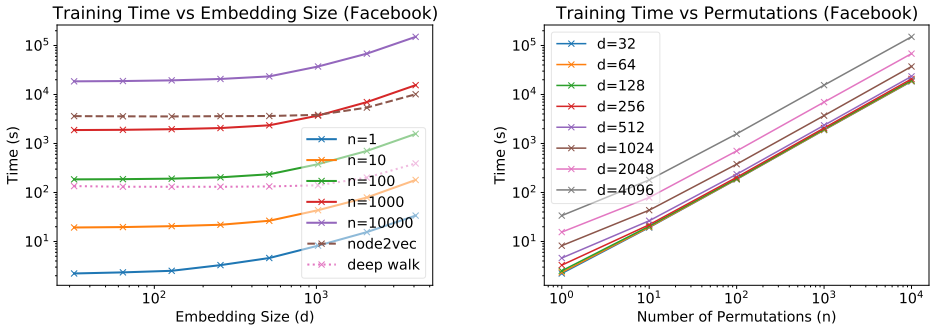

Fig. 2. (Color online) Facebook dataset with 30% edges removed. Left: Training times vs embedding size. Right: Training times vs number of permutations.

to others which are similar in some sense. Next, we investigate assortativity properties of the NBNE representations and of the graphs themselves.

Graph Homophily: There are several ways to assess homophily in a graph. Table V presents both degree and label assortativity properties of the six graphs analyzed here, calculated using the assortativity coefficient, as defined in [35]. As we can see, these graphs cover a broad spectrum of assortativity properties. PPI, Wikipedia and Blog graphs present negative degree assortativity, which means nodes in these graphs are more likely to connect with nodes of different connectivity degrees. At the same time, Facebook, Astro and DBLP present positive degree assortativity, which indicates that their nodes tend to connect to others with similar degrees.

TABLE V
DATASETS HOMOPHILY INFORMATION

	Assortativity	
	Degree[1]	Label[1]
Facebook	0.0635	–
Astro	0.2051	–
PPI	-0.0930	0.0533
Wikipedia	-0.2372	-0.0252
Blog	-0.2541	0.0515
DBLP	0.2665	–

We also analyze graphs with both positive and negative label assortativity in the label classification task. While PPI and Blog datasets present positive label assortativity, with connected nodes more frequently sharing classes, Wikipedia has a negative assortativity, with its connected nodes being more likely to have different classes.

Number of Permutations (n): We analyze how the number of permutations (n) influences both homophily and overfitting in our learned representations. We qualitatively measure homophily by comparing either cosine or euclidean distances between adjacent nodes to the distances of non-adjacent nodes.

The cosine distances for the PPI graph, shown by the box plots in Figure 3 (top-left), clearly show for larger values of n how the embeddings overfit to the specific graph structure, with the learned similarity on edges not generalizing to the links which were previously removed. In this graph, for larger numbers of permutations the removed edges have a distribution more similar to the non-edges than to the edges used during training, which is a tendency that can be observed in the other graphs, although in a smaller scale.

The box plots in Figure 3 (top-right) show the cosine distance for Facebook nodes. We can see that for $n = 5$ there is a larger separation between removed edges and non edges, which justifies the algorithm's choice of this value. For larger values of n we can again see an overlap between the distributions, caused by the embeddings overfitting. On the other hand, the cosine distances for the DBLP in Figure 3 (bottom-left) show the largest separation for $n = 10$. Finally, the box plots in Figure 3 (bottom-right) show cosine distances for the Blog graph. We can see that for $n = 1$ and $n = 5$ there is actually a larger cosine distance between nodes in removed edges than in non-edges, with this situation only inverting for $n \geq 10$. This happens due to this graph's negative degree homophily. This is also observed for similar graphs in the PPI and Wikipedia graphs, though with a smaller intensity in the PPI graph, which has a smaller absolute value of degree assortativity and where only embeddings for $n = 1$ present this property.

The box plots in Figure 3 further support our intuition that graphs with larger branching factors should have larger values of n. At the same time, this choice also depends on the graph size and structure, as shown by the algorithms choice of $n = 10$ for the DBLP graph, which contains the largest degree assortativity. The best choice of n depends on the task in hand, but we believe that, at least for link prediction, this choice is both directly proportional to a graph's branching size and degree assortativity. Nonetheless, the difficulty in analyzing these graphs supports our choice of choosing n on a per graph basis.

To better understand the results of the experiment on the PPI graph with $n = 1$, shown in Figure 3 (top-left), we present in Figure 4 a detail of the euclidean distances between nodes that share or not an edge for this number of permutations. We can see that the distribution of removed edges is a lot closer to the edges used for training than to the non edges.

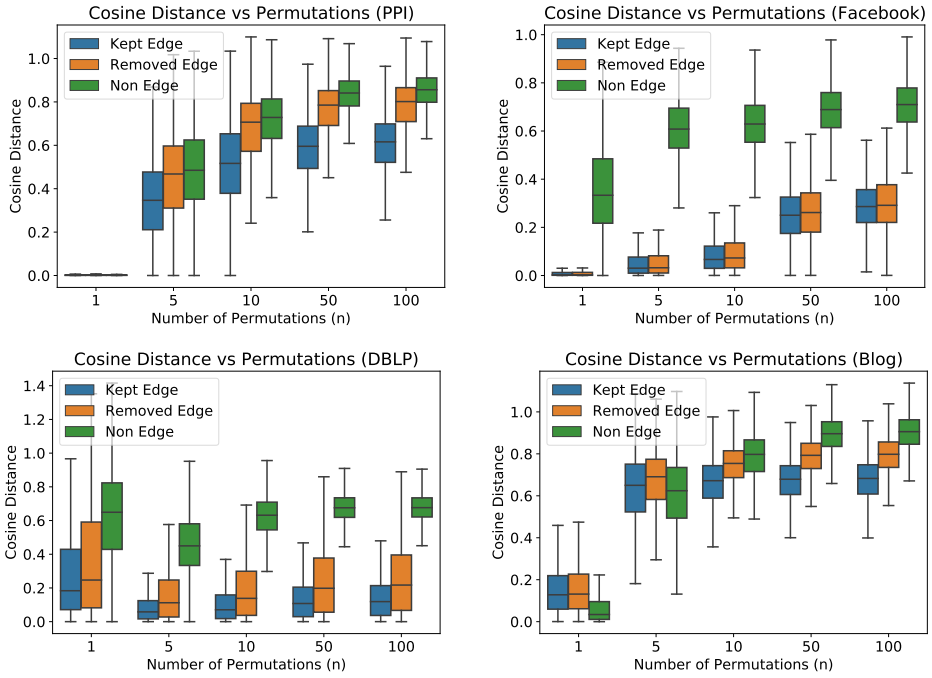

Fig. 3. (Color online) Cosine distances. Top-Left: PPI graph. Top-Right: Facebook graph. Bottom-Left: DBLP graph. Bottom-Right: Blog graph. All graphs had 10% of edges removed.

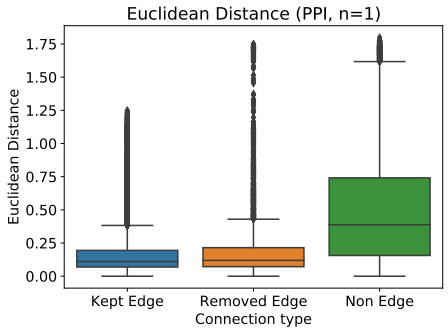

Fig. 4. (Color online) Euclidean distances on PPI graph for $n = 1$.

Window size (w) for Node2Vec and DeepWalk: The window size (w) controls the distance around a node which Node2Vec and DeepWalk use to learn their embeddings. With larger window sizes, then, the algorithm will try to predict more distant vertex while learning representations, we suspect this makes these baselines less stable and efficient. Figure 5 shows the effect of w on the AUC scores obtained by Node2Vec on four of the analyzed graphs (DeepWalk's results follow the same trend, so we omitted it). This figure shows us that for these 4 graphs (Astro, PPI, Blog and Facebook), reducing the window size to $w = 1$ actually increases their AUC scores, as we expected. The same trend does not occur when we change the group size (k) of NBNE. For some graphs (PPI, Facebook) NBNE improves with smaller groups ($k = 1$), while for others it is better for larger groups (Astro, Blog).

VI. Conclusions

The proposed node embedding algorithm, NBNE, shows results similar or better than the state-of-the-art algorithms Node2Vec, DeepWalk and SDNE on several different graphs. It shows promising results in two application scenarios: link prediction and node classification, while being fast and scalable for large graphs. The proposed NBNE algorithm focuses learning on the node immediate neighbors, creating representations from small contexts, which make them more stable and faster to learn. Empirical results show that NBNE can be trained in a fraction of the time taken by DeepWalk (10 to 190 times faster), Node2Vec (200 to 6 000 times faster) or SDNE (29 to 2 000 times faster), giving fairly robust results. Since embeddings are learned using only the node immediate neighbors, we expect it to also be easier to implement more stable asynchronous distributed algorithms to learn them, and we leave this as future work.

References

[1] L. Ribeiro, P. Saverese, and D. Figueiredo, "*struc2vec*: Learning node representations from structural identity," in *KDD*, 2017, pp. 385–394.
[2] T. Pimentel, A. Veloso, and N. Ziviani, "Fast node embeddings: Learning ego-centric representations," in *ICLR*, 2018.
[3] T. Pimentel, N. Ziviani, and A. Veloso, "Unsupervised and scalable algorithm for learning node representations," in *ICLR*, 2017.
[4] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 6, pp. 1150–1170, 2011.
[5] S. Yang, B. Long, A. Smola, N. Sadagopan, Z. Zheng, and H. Zha, "Like like alike: joint friendship and interest propagation in social networks," in *WWW*, 2011, pp. 537–546.
[6] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *KDD*, 2016, pp. 855–864.

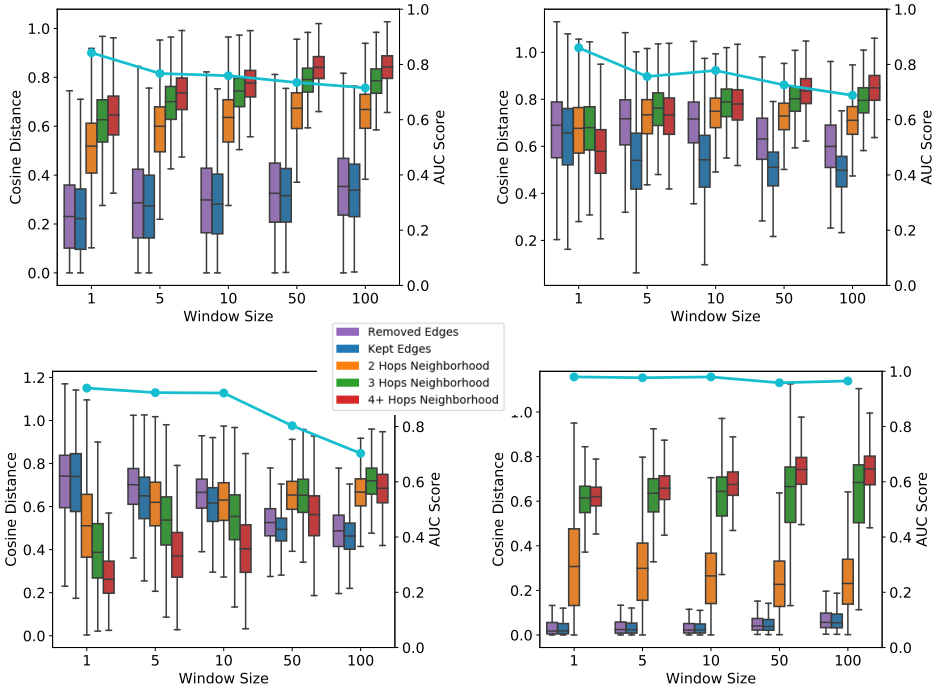

Fig. 5. (Color online) Node2Vec embeddings' cosine similarities between nodes versus Graph Distance for different values of w for the graphs: Astro (top-left); PPI (top-right); Blog (bottom-left); and Facebook (bottom-right). All graphs had 10% of edges removed. Lines show AUC scores on Link Prediction.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013.

[8] T. Pimentel, J. Viana, A. Veloso, and N. Ziviani, "Fast and effective neural networks for translating natural language into denotations," in *SPIRE*, 2018, pp. 334–347.

[9] G. L. Zuin, L. Chaimowicz, and A. Veloso, "Learning transferable features for open-domain question answering," in *IJCNN*, 2018, pp. 1–8.

[10] T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young, "Semantically conditioned LSTM-based natural language generation for spoken dialogue systems," *EMNLP*, pp. 1711–1721, 2015.

[11] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014, pp. 701–710.

[12] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *KDD*, 2016, pp. 1225–1234.

[13] A. Costa, R. Nalon, W. M. Jr., and A. Veloso, "Ego-centric analysis of supportive networks," in *ACM WebSci*, 2018, pp. 281–285.

[14] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[15] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," in *STOC*, 1987, pp. 1–6.

[16] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *WWW*, 2015, pp. 1067–1077.

[17] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *IJCAI*, 2016, pp. 1895–1901.

[18] Z. Wang and J. Li, "Text-enhanced representation learning for knowledge graph," in *IJCAI*, 2016, pp. 1293–1299.

[19] Y. Lin, Z. Liu, and M. Sun, "Knowledge representation learning with entities, attributes and relations," in *IJCAI*, 2016, pp. 2866–2872.

[20] Y. Dong, N. V. Chawla, and A. Swami, "metapath2vec: Scalable representation learning for heterogeneous networks," in *KDD*, 2017, pp. 135–144.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[22] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *USENIX OSDI*, 2016, pp. 265–283.

[23] J. J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks." in *NIPS*, vol. 2012, 2012, pp. 548–56.

[24] J. Leskovec, J. M. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *Transactions on Knowledge Discovery from Data*, vol. 1, no. 1, p. 2, 2007.

[25] B. Breitkreutz, C. Stark, T. Reguly, L. Boucher, A. Breitkreutz, M. S. Livstone, R. Oughtred, D. H. Lackner, J. Bähler, V. Wood, K. Dolinski, and M. Tyers, "The biogrid interaction database: 2008 update," *Nucleic Acids Research*, vol. 36, no. Database-Issue, pp. 637–640, 2008.

[26] A. Liberzon, A. Subramanian, R. Pinchback, H. Thorvaldsdóttir, P. Tamayo, and J. P. Mesirov, "Molecular signatures database 3.0," *Bioinformatics*, vol. 27, no. 12, pp. 1739–1740, 2011.

[27] M. Mahoney, "Large text compression benchmark," 2011.

[28] R. Zafarani and H. Liu, "Social computing data repository at ASU," 2009.

[29] A. Ferreira, A. Veloso, M. Gonçalves, and A. Laender, "Self-training author name disambiguation for information scarce scenarios," *JASIST*, vol. 65, no. 6, pp. 1257–1278, 2014.

[30] A. Veloso, A. Ferreira, M. Gonçalves, A. Laender, and W. M. Jr., "Cost-effective on-demand associative author name disambiguation," *Inf. Process. Manage.*, vol. 48, no. 4, pp. 680–697, 2012.

[31] L. A. Adamic and E. Adar, "Friends and neighbors on the web," *Social Networks*, vol. 25, no. 3, pp. 211–230, 2003.

[32] P. Sarkar, D. Chakrabarti, and A. W. Moore, "Theoretical justification of popular link prediction heuristics." in *IJCAI*, vol. 22, 2011, p. 2722.

[33] R. A. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval - the concepts and technology behind search*. Pearson Education Ltd., Harlow, England, 2011.

[34] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *LREC 2010 Workshop on New Challenges for NLP*, 2010, pp. 45–50.

[35] M. E. Newman, "Mixing patterns in networks," *Physical Review E*, vol. 67, no. 2, p. 026126, 2003.