DeepWalk: Online Learning of Social Representations

Bryan Perozzi
Stony Brook University
Department of Computer Science
Rami Al-Rfou
Stony Brook University
Department of Computer Science
Steven Skiena
Stony Brook University
Department of Computer Science

{bperozzi, ralrfou, skiena}@cs.stonybrook.edu

ABSTRACT

We present DeepWalk, a novel approach for learning latent representations of vertices in a network. These latent representations encode social relations in a continuous vector space, which is easily exploited by statistical models. DeepWalk generalizes recent advancements in language modeling and unsupervised feature learning (or deep learning) from sequences of words to graphs.

DeepWalk uses local information obtained from truncated random walks to learn latent representations by treating walks as the equivalent of sentences. We demonstrate DeepWalk’s latent representations on several multi-label network classification tasks for social networks such as BlogCatalog, Flickr, and YouTube. Our results show that DeepWalk outperforms challenging baselines which are allowed a global view of the network, especially in the presence of missing information. DeepWalk’s representations can provide F1 scores up to 10% higher than competing methods when labeled data is sparse. In some experiments, DeepWalk’s representations are able to outperform all baseline methods while using 60% less training data.

DeepWalk is also scalable. It is an online learning algorithm which builds useful incremental results, and is trivially parallelizable. These qualities make it suitable for a broad class of real world applications such as network classification, and anomaly detection.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications - Data Mining; I.2.6 [Artificial Intelligence]: Learning; I.5.1 [Pattern Recognition]: Model - Statistical

Keywords

social networks; deep learning; latent representations; learning with partial labels; network classification; online learning

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD’14, August 24–27, 2014, New York, NY, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623732

1. INTRODUCTION

The sparsity of a network representation is both a strength and a weakness. Sparsity enables the design of efficient discrete algorithms, but can make it harder to generalize in statistical learning. Machine learning applications in networks (such as network classification [16, 37], content recommendation [12], anomaly detection [6], and missing link prediction [23]) must be able to deal with this sparsity in order to survive.

In this paper we introduce deep learning (unsupervised feature learning) [3] techniques, which have proven successful in natural language processing, into network analysis for the first time. We develop an algorithm (DeepWalk) that learns social representations of a graph’s vertices, by modeling a stream of short random walks. Social representations are latent features of the vertices that capture neighborhood similarity and community membership. These latent representations encode social relations in a continuous vector space with a relatively small number of dimensions. DeepWalk generalizes neural language models to process a special language composed of a set of randomly-generated walks. These neural language models have been used to capture the semantic and syntactic structure of human language [7], and even logical analogies [29].

DeepWalk takes a graph as input and produces a latent representation as an output. The result of applying our method to the well-studied Karate network is shown in Figure 1. The graph, as typically presented by force-directed
layouts, is shown in Figure 1a. Figure 1b shows the output of our method with 2 latent dimensions. Beyond the striking similarity, we note that linearly separable portions of 1b correspond to clusters found through modularity maximization in the input graph 1a (shown as vertex colors).

To demonstrate DEEPWALK’s potential in real world scenarios, we evaluate its performance on challenging multi-label network classification problems in large heterogeneous graphs. In the relational classification problem, the links between feature vectors violate the traditional i.i.d. assumption. Techniques to address this problem typically use approximate inference techniques [32] to leverage the dependency information to improve classification results. We distance ourselves from these approaches by learning label-independent representations of the graph. Our representation quality is not influenced by the choice of labeled vertices, so they can be shared among tasks.

DEEPWALK outperforms other latent representation methods for creating social dimensions [39, 41], especially when labeled nodes are scarce. Strong performance with our representations is possible with very simple linear classifiers (e.g.

logistic regression). Our representations are general, and can be combined with any classification method (including iterative inference methods). DEEPWALK achieves all of that while being an online algorithm that is trivially parallelizable.

Our contributions are as follows:

- We introduce deep learning as a tool to analyze graphs, to build robust representations that are suitable for statistical modeling. DEEPWALK learns structural regularities present within short random walks.
- We extensively evaluate our representations on multi-label classification tasks on several social networks. We show significantly increased classification performance in the presence of label sparsity, getting improvements 5%-10% of Micro F1, on the sparsest problems we consider. In some cases, DEEPWALK’s representations can outperform its competitors even when given 60% less training data.
- We demonstrate the scalability of our algorithm by building representations of web-scale graphs, (such as YouTube) using a parallel implementation. Moreover, we describe the minimal changes necessary to build a streaming version of our approach.

3. LEARNING SOCIAL REPRESENTATIONS

We seek to learn social representations with the following characteristics:

- **Adaptability** - Real social networks are constantly evolving; new social relations should not require repeating the learning process all over again.
- **Community aware** - The distance between latent dimensions should represent a metric for evaluating social similarity between the corresponding members of the network. This allows generalization in networks with homophily.
- **Low dimensional** - When labeled data is scarce low-dimensional models generalize better, and speed up convergence and inference.
- **Continuous** - We require latent representations to model partial community membership in continuous space. In addition to providing a nuanced view of community membership, a continuous representation has smooth decision boundaries between communities which allows more robust classification.

Our method satisfies these requirements by learning representation for vertices from a stream of short random walks, using optimization techniques originally designed for language modeling. Here, we review the basics of both random walks and language modeling, and describe how their combination satisfies our requirements.
3.1 Random Walks

We denote a random walk rooted at vertex $v_i$ as $W_{v_i}$. It is a stochastic process with random variables $W_{v_i}^1, W_{v_i}^2, \ldots, W_{v_i}^k$ such that $W_{v_i}^{k+1}$ is a vertex chosen at random from the neighbors of vertex $v_k$. Random walks have been used as a similarity measure for a variety of problems in content recommendation [12] and community detection [2]. They are also the foundation of a class of output sensitive algorithms which use them to compute local community structure information in time sublinear to the size of the input graph [35].

It is this connection to local structure that motivates us to use a stream of short random walks as our basic tool for extracting information from a network. In addition to capturing community information, using random walks as the basis for our algorithm gives us two other desirable properties. First, local exploration is easy to parallelize. Several random walkers (in different threads, processes, or machines) can simultaneously explore different parts of the same graph. Secondly, relying on information obtained from short random walks make it possible to accommodate small changes in the graph structure without the need for global recomputation. We can iteratively update the learned model with new random walks from the changed region in time sub-linear to the entire graph.

3.2 Connection: Power laws

Having chosen online random walks as our primitive for capturing graph structure, we now need a suitable method to capture this information. If the degree distribution of a connected graph follows a power law (i.e. scale-free), we observe that the frequency which vertices appear in the short random walks will also follow a power-law distribution.

Word frequency in natural language follows a similar distribution, and techniques from language modeling account for this distributional behavior. To emphasize this similarity we show two different power-law distributions in Figure 2. The first comes from a series of short random walks on a scale-free graph, and the second comes from the text of the 100,000 articles from the English Wikipedia.

A core contribution of our work is the idea that techniques which have been used to model natural language (where the symbol frequency follows a power law distribution (or Zipf’s law)) can be re-purposed to model community structure in networks. We spend the rest of this section reviewing the growing work in language modeling, and transforming it to learn representations of vertices which satisfy our criteria.

3.3 Language Modeling

The goal of language modeling is to estimate the likelihood of a specific sequence of words appearing in a corpus. More formally, given a sequence of words $W^t_1 = (w_0, w_1, \ldots, w_t)$, where $w_i \in V$ (the vocabulary), we would like to maximize the $\Pr(w_n|w_0, w_1, \ldots, w_{n-1})$ over all the training corpus. Recent work in representation learning has focused on using probabilistic neural networks to build general representations of words which extend the scope of language modeling beyond its original goals.

In this work, we present a generalization of language modeling to explore the graph through a stream of short random walks. These walks can be thought of as short sentences and phrases in a special language; the direct analog is to estimate the likelihood of observing vertex $v_i$ given all the previous vertices visited so far in the random walk, i.e.

$$\Pr(v_i | (v_1, v_2, \ldots, v_{i-1}))$$

Our goal is to learn a latent representation, not only a probability distribution of node co-occurrences, and so we introduce a mapping function $\Phi: v \in V \mapsto \mathbb{R}^{|V| \times d}$. This mapping $\Phi$ represents the latent social representation associated with each vertex $v$ in the graph. (In practice, we represent $\Phi$ by a $|V| \times d$ matrix of free parameters, which will serve later on as our $X_E$). The problem then, is to estimate the likelihood:

$$\Pr(v_i | (\Phi(v_1), \Phi(v_2), \cdots, \Phi(v_{i-1})))$$

However, as the walk length grows, computing this conditional probability becomes unfeasible.

A recent relaxation in language modeling [27, 28] turns the prediction problem on its head. First, instead of using the context to predict a missing word, it uses one word to predict the context. Secondly, the context is composed of the words appearing to both the right and left of the given word. Finally, it removes the ordering constraint on the problem, instead, requiring the model to maximize the probability of any word appearing in the context without the knowledge of its offset from the given word. In terms of vertex representation modeling, this yields the optimization problem:

$$\min_{\Phi} - \log \Pr \left( \{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i | \Phi(v_i) \right)$$

We find these relaxations are particularly desirable for social representation learning. First, the order independence assumption better captures a sense of ‘nearness’ that is provided by random walks. Moreover, this relaxation is quite useful for speeding up the training time by building small models as one vertex is given at a time.

Solving the optimization problem from Eq. [3] builds representations that capture the shared similarities in local graph structure between vertices. Vertices which have similar neighborhoods will acquire similar representations (encoding co-citation similarity), allowing generalization on machine learning tasks.

By combining both truncated random walks and language models we formulate a method which satisfies all of our desired properties. This method generates representations of social networks that are low-dimensional, and exist in a continuous vector space. Its representations encode latent forms of community membership, and because the method
Algorithm 1: DeepWalk(G, w, d, γ, t)
Input: graph G(V, E)
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations Φ ∈ R |V|×d
1: Initialization: Sample Φ from U |V|×d
2: Build a binary Tree T from V
3: for i = 0 to γ do
4:   γ = Shuffle(V)
5:   for each vi ∈ O do
6:     Wvi = RandomWalk(G, vi, t)
7:     SkipGram(Φ, Wvi, w)
8: end for
9: end for

outputs useful intermediate representations, it can adapt to changing network topology.

4. METHOD
In this section we discuss the main components of our algorithm. We also present several variants of our approach and discuss their merits.

4.1 Overview
As in any language modeling algorithm, the only required input is a corpus and a vocabulary V. DeepWalk considers a set of short truncated random walks its own corpus, and the graph vertices as its own vocabulary (|V| = |V|). While it is beneficial to know V and the frequency distribution of vertices in the random walks ahead of the training, it is not necessary for the algorithm to work as we will show in 4.2.2.

4.2 Algorithm: DeepWalk
The algorithm consists of two main components; first a random walk generator, and second, an update procedure. The random walk generator takes a graph G and samples uniformly a random vertex vi as the root of the random walk Wvi. A walk samples uniformly from the neighbors of the last vertex visited until the maximum length (t) is reached. While we set the length of our random walks in the experiments to be fixed, there is no restriction for the random walks to be of the same length. These walks could have restarts (i.e. a teleport probability of returning back to their root), but our preliminary results did not show any advantage of using restarts. In practice, our implementation specifies a number of random walks γ of length t to start at each vertex.

Lines 3-9 in Algorithm 1 shows the core of our approach. The outer loop specifies the number of times, γ, which we should start random walks at each vertex. We think of each iteration as making a ‘pass’ over the data and sample one walk per node during this pass. At the start of each pass we generate a random ordering to traverse the vertices. This is not strictly required, but is well-known to speed up the convergence of stochastic gradient descent.

In the inner loop, we iterate over all the vertices of the graph. For each vertex vi we generate a random walk |Wvi| = t, and then use it to update our representations (Line 7). We use the SkipGram algorithm 27 to update these representations in accordance with our objective function in Eq. 3.

4.2.1 SkipGram
SkipGram is a language model that maximizes the co-occurrence probability among the words that appear within a window, w, in a sentence. It approximates the conditional probability in Equation 3 using an independence assumption as follows

\[
\Pr\{\{v_{i-w}, \ldots, v_{i+w}\} \mid v_i \mid \Phi(v_i)\} = \prod_{j=i-w, j \neq i}^{i+w} \Pr(v_j \mid \Phi(v_i))
\]

Algorithm 2 iterates over all possible collocations in random walk that appear within the window w (lines 1-2). For each, we map each vertex vj to its current representation vector Φ(vj) ∈ R d (See Figure 3b). Given the representation of vj, we would like to maximize the probability of its neighbors in the walk (line 3). We can learn such a posterior distribution using several choices of classifiers. For example, modeling the previous problem using logistic regression would result in a huge number of labels (that is equal to |V|) which could be in millions or billions. Such models require vast computational resources which could span a whole cluster of computers 4. To avoid this necessity and speed up the training time, we instead use the Hierarchical Softmax 30,31 to approximate the probability distribution.

4.2.2 Hierarchical Softmax
Given that uk ∈ V, calculating \Pr(uk \mid \Phi(vj)) in line 3 is not feasible. Computing the partition function (normalization factor) is expensive, so instead we will factorize the conditional probability using Hierarchical softmax. We assign the vertices to the leaves of a binary tree, turning the prediction problem into maximizing the probability of a specific path in the hierarchy (See Figure 3c). If the path to vertex uk is identified by a sequence of node trees (b0, b1, ..., b[log |V|]), (b0 = root, b[log |V|] = uk) then

\[
\Pr(uk \mid \Phi(vj)) = \prod_{l=1}^{[\log |V|]} \Pr(b_l \mid \Phi(vj))
\]

Now, \Pr(b_l \mid \Phi(vj)) could be modeled by a binary classifier that is assigned to the parent of the node bl as Equation 6 shows,

\[
\Pr(b_l \mid \Phi(vj)) = 1/1 + e^{-\Phi(vj) \cdot \Phi(b_l)}
\]

where \Psi(b_l) ∈ R d is the representation assigned to tree node bl’s parent. This reduces the computational complexity of calculating \Pr(uk \mid \Phi(vj)) from O(|V|) to O(log |V|).

We can speed up the training process further, by assigning shorter paths to the frequent vertices in the random walks. Huffman coding is used to reduce the access time of frequent elements in the tree.
4.2.3 Optimization

The model parameter set is $\theta = \{\Phi, \Psi\}$ where the size of each is $O(d|V|)$. Stochastic gradient descent (SGD) is used to optimize these parameters (Line 4, Algorithm 2). The derivatives are estimated using back-propagation. The learning rate $\alpha$ for SGD is initially set to 2.5% at the beginning of the training and then decreased linearly with the number of vertices that are seen so far.

4.3 Parallelizability

As shown in Figure 2, the frequency distribution of vertices in random walks of social network and words in a language both follow a power law. This results in a long tail of infrequent vertices, therefore, the updates that affect $\Phi$ will be sparse in nature. This allows us to use asynchronous version of stochastic gradient descent (ASGD), in the multi-worker case. Given that our updates are sparse and we do not acquire a lock to access the model shared parameters, ASGD will achieve an optimal rate of convergence. While we run experiments on one machine using multiple threads, it has been demonstrated that this technique is highly scalable, and can be used in very large scale machine learning.

Figure 4 presents the effects of parallelizing DeepWalk. It shows the speed up in processing BlogCatalog and Flickr networks is consistent as we increase the number of workers from 1 to 8 (Figure 4a). It also shows that there is no loss of predictive performance relative to the running DeepWalk serially (Figure 4b).

4.4 Algorithm Variants

Here we discuss some variants of our proposed method, which we believe may be of interest.

4.4.1 Streaming

One interesting variant of this method is a streaming approach, which could be implemented without knowledge of the entire graph. In this variant small walks from the graph are passed directly to the representation learning code, and the model is updated directly. Some modifications to the learning process will also be necessary. First, using a decaying learning rate may no longer be desirable as it assumes the knowledge of the total corpus size. Instead, we can initialize the learning rate $\alpha$ to a small constant value. This will take longer to learn, but may be worth it in some applications. Second, we cannot necessarily build a tree of parameters any more. If the cardinality of $V$ is known (or can be bounded), we can build the Hierarchical Softmax tree for that maximum value. Vertices can be assigned to one of the remaining leaves when they are first seen. If we have the ability to estimate the vertex frequency a priori, we can also still use Huffman coding to decrease frequent element access times.

4.4.2 Non-random walks

Some graphs are created as a by-product of agents interacting with a sequence of elements (e.g., users’ navigation of pages on a website). When a graph is created by such a stream of non-random walks, we can use this process to feed the modeling phase directly. Graphs sampled in this way will not only capture information related to network structure, but also to the frequency at which paths are traversed.

In our view, this variant also encompasses language modeling. Sentences can be viewed as purposed walks through an appropriately designed language network, and language models like SkipGram are designed to capture this behavior.
5. EXPERIMENTAL DESIGN

In this section we provide an overview of the datasets and methods which we will use in our experiments. Code and data to reproduce our results will be available at the first author’s website[

5.1 Datasets

An overview of the graphs we consider in our experiments is given in Figure[1]

- **BlogCatalog**[39] is a network of social relationships provided by blogger authors. The labels represent the topic categories provided by the authors.
- **Flickr**[39] is a network of the contacts between users of the photo sharing website. The labels represent the interest groups of the users such as ‘black and white photos’.
- **YouTube**[40] is a social network between users of the popular video sharing website. The labels here represent groups of viewers that enjoy common video genres (e.g. anime and wrestling).

5.2 Baseline Methods

To validate the performance of our approach we compare it against a number of baselines:

- **SpectralClustering**[41]: This method generates a representation in $\mathbb{R}^d$ from the $d$-smallest eigenvectors of $L$, the normalized graph Laplacian of $G$. Utilizing the eigenvectors of $L$ implicitly assumes that graph cuts will be useful for classification.
- **Modularity**[39]: This method generates a representation in $\mathbb{R}^r$ from the top-$d$ eigenvectors of $B$, the Modularity matrix of $G$. The eigenvectors of $B$ encode information about modular graph partitions of $G$[35]. Using them as features assumes that modular graph partitions will be useful for classification.
- **EdgeCluster**[40]: This method uses $k$-means clustering to cluster the adjacency matrix of $G$. Its has been shown to perform comparably to the Modularity method, with the added advantage of scaling to graphs which are too large for spectral decomposition.
- **wvRN**[25]: The weighted-vote Relational Neighbor is a relational classifier. Given the neighborhood $N_i$ of vertex $v_i$, wvRN estimates $Pr(y_i|N_i)$ with the (appropriately normalized) weighted mean of its neighbors (i.e $Pr(y_i|N_i) = \frac{1}{|N_i|} \sum_{v_j \in N_i} w_{ij} Pr(y_j | N_j)$). It has shown surprisingly good performance in real networks, and has been advocated as a sensible relational classification baseline[26].
- **Majority**: This naïve method simply chooses the most frequent labels in the training set.

<table>
<thead>
<tr>
<th>Name</th>
<th>BlogCatalog</th>
<th>Flickr</th>
<th>YouTube</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10,312</td>
<td>80,513</td>
<td>1,183,499</td>
</tr>
<tr>
<td>$</td>
<td>E</td>
<td>$</td>
<td>333,983</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{Y}</td>
<td>$</td>
<td>39</td>
</tr>
<tr>
<td>Labels</td>
<td>Interests</td>
<td>Groups</td>
<td>Groups</td>
</tr>
</tbody>
</table>

Table 1: Graphs used in our experiments.

6. EXPERIMENTS

In this section we present an experimental analysis of our method. We thoroughly evaluate it on a number of multi-label classification tasks, and analyze its sensitivity across several parameters.

6.1 Multi-Label Classification

To facilitate the comparison between our method and the relevant baselines, we use the exact same datasets and experimental procedure as in [39,40]. Specifically, we randomly sample a portion ($T_R$) of the labeled nodes, and use them as training data. The rest of the nodes are used as test. We repeat this process 10 times, and report the average performance in terms of both Macro-$F_1$ and Micro-$F_1$. When possible we report the original results [39,40] here directly.

For all models we use a one-vs-rest logistic regression implemented by LibLinear[11] extended to return the most probable labels as in [39]. We present results for DeepWalk with ($\gamma = 80$, $w = 10$, $d = 128$). The results for (SpectralClustering, Modularity, EdgeCluster) use Tang and Liu’s preferred dimensionality, $d = 500$.

6.1.1 BlogCatalog

In this experiment we increase the training ratio ($T_R$) on the BlogCatalog network from 10% to 90%. Our results are presented in Table 2. Numbers in bold represent the highest performance in each column.

DeepWalk performs consistently better than EdgeCluster, Modularity, and wvRN. In fact, when trained with only 20% of the nodes labeled, DeepWalk performs better than these approaches when they are given 90% of the data. The performance of SpectralClustering proves much more competitive, but DeepWalk still outperforms when labeled data is sparse on both Macro-$F_1$ ($T_R \leq 20\%$) and Micro-$F_1$ ($T_R \leq 60\%$).

This strong performance when only small fractions of the graph are labeled is a core strength of our approach. In the following experiments, we investigate the performance of our representations on even more sparsely labeled graphs.

6.1.2 Flickr

In this experiment we vary the training ratio ($T_R$) on the Flickr network from 1% to 10%. This corresponds to having approximately 800 to 8,000 nodes labeled for classification in the entire network. Table 3 presents our results, which are consistent with the previous experiment. DeepWalk outperforms all baselines by at least 3% with respect to Micro-$F_1$. Additionally, its Micro-$F_1$ performance when only 3% of the graph is labeled beats all other methods even when they have been given 10% of the data. In other words, DeepWalk can outperform the baselines with 60% less training data. It also performs quite well in Macro-$F_1$, initially performing close to SpectralClustering, but distancing itself to a 1% improvement.

6.1.3 YouTube

The YouTube network is considerably larger than the previous ones we have experimented on, and its size prevents two of our baseline methods (SpectralClustering and Modularity) from running on it. It is much closer to a real world graph than those we have previously considered.

The results of varying the training ratio ($T_R$) from 1% to 10% are presented in Table 4. They show that DeepWalk significantly outperforms the scalable baseline for creating
<table>
<thead>
<tr>
<th>% Labeled Nodes</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Micro-F1(%)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepWalk</td>
<td>36.00</td>
<td>38.20</td>
<td>39.60</td>
<td>40.30</td>
<td>41.00</td>
<td>41.30</td>
<td>41.50</td>
<td>41.50</td>
<td>42.00</td>
</tr>
<tr>
<td>SpectralClustering</td>
<td>31.06</td>
<td>34.95</td>
<td>37.27</td>
<td>38.93</td>
<td>39.97</td>
<td>40.99</td>
<td>41.66</td>
<td>42.42</td>
<td>42.62</td>
</tr>
<tr>
<td>EdgeCluster</td>
<td>27.94</td>
<td>30.76</td>
<td>31.85</td>
<td>32.99</td>
<td>34.12</td>
<td>35.00</td>
<td>34.63</td>
<td>35.99</td>
<td>36.29</td>
</tr>
<tr>
<td>Modularity</td>
<td>27.35</td>
<td>30.74</td>
<td>31.77</td>
<td>32.97</td>
<td>34.09</td>
<td>36.13</td>
<td>36.08</td>
<td>37.23</td>
<td>38.18</td>
</tr>
<tr>
<td>wvRN</td>
<td>19.51</td>
<td>24.34</td>
<td>25.62</td>
<td>28.82</td>
<td>30.37</td>
<td>31.81</td>
<td>32.19</td>
<td>33.33</td>
<td>34.28</td>
</tr>
<tr>
<td><strong>Macro-F1(%)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepWalk</td>
<td>31.30</td>
<td>23.80</td>
<td>25.30</td>
<td>26.30</td>
<td>27.30</td>
<td>27.60</td>
<td>27.90</td>
<td>28.20</td>
<td>28.90</td>
</tr>
<tr>
<td>SpectralClustering</td>
<td>19.14</td>
<td>23.57</td>
<td>25.97</td>
<td>27.46</td>
<td>28.31</td>
<td>29.46</td>
<td>30.13</td>
<td>31.38</td>
<td>31.78</td>
</tr>
<tr>
<td>EdgeCluster</td>
<td>16.16</td>
<td>19.16</td>
<td>20.48</td>
<td>22.00</td>
<td>23.00</td>
<td>23.64</td>
<td>23.82</td>
<td>24.61</td>
<td>24.92</td>
</tr>
<tr>
<td>Modularity</td>
<td>17.36</td>
<td>20.00</td>
<td>21.65</td>
<td>22.65</td>
<td>23.41</td>
<td>23.89</td>
<td>24.20</td>
<td>24.97</td>
<td>25.28</td>
</tr>
<tr>
<td>wvRN</td>
<td>6.25</td>
<td>10.13</td>
<td>11.64</td>
<td>14.24</td>
<td>15.86</td>
<td>17.18</td>
<td>17.98</td>
<td>18.86</td>
<td>19.57</td>
</tr>
<tr>
<td>Majority</td>
<td>2.52</td>
<td>2.55</td>
<td>2.52</td>
<td>2.58</td>
<td>2.63</td>
<td>2.61</td>
<td>2.48</td>
<td>2.62</td>
<td>2.62</td>
</tr>
</tbody>
</table>

Table 2: Multi-label classification results in BlogCatalog

<table>
<thead>
<tr>
<th>% Labeled Nodes</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
<th>4%</th>
<th>5%</th>
<th>6%</th>
<th>7%</th>
<th>8%</th>
<th>9%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Micro-F1(%)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepWalk</td>
<td>32.4</td>
<td>34.6</td>
<td>35.9</td>
<td>36.7</td>
<td>37.2</td>
<td>37.7</td>
<td>38.1</td>
<td>38.3</td>
<td>38.5</td>
<td>38.7</td>
</tr>
<tr>
<td>SpectralClustering</td>
<td>27.43</td>
<td>30.11</td>
<td>31.63</td>
<td>32.69</td>
<td>33.31</td>
<td>33.95</td>
<td>34.46</td>
<td>34.81</td>
<td>35.14</td>
<td>35.41</td>
</tr>
<tr>
<td>EdgeCluster</td>
<td>25.75</td>
<td>28.53</td>
<td>29.14</td>
<td>30.31</td>
<td>30.85</td>
<td>31.53</td>
<td>31.75</td>
<td>31.76</td>
<td>32.19</td>
<td>32.84</td>
</tr>
<tr>
<td>Modularity</td>
<td>22.75</td>
<td>25.29</td>
<td>27.3</td>
<td>27.6</td>
<td>28.05</td>
<td>29.33</td>
<td>29.43</td>
<td>28.89</td>
<td>29.17</td>
<td>29.2</td>
</tr>
<tr>
<td><strong>Macro-F1(%)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepWalk</td>
<td>14.0</td>
<td>17.3</td>
<td>19.6</td>
<td>21.1</td>
<td>22.1</td>
<td>22.9</td>
<td>23.6</td>
<td>24.1</td>
<td>24.6</td>
<td>25.0</td>
</tr>
<tr>
<td>SpectralClustering</td>
<td>13.84</td>
<td>17.49</td>
<td>19.44</td>
<td>20.75</td>
<td>21.60</td>
<td>22.36</td>
<td>23.01</td>
<td>23.36</td>
<td>23.82</td>
<td>24.05</td>
</tr>
<tr>
<td>EdgeCluster</td>
<td>10.52</td>
<td>14.10</td>
<td>15.91</td>
<td>16.72</td>
<td>18.01</td>
<td>18.54</td>
<td>19.54</td>
<td>20.18</td>
<td>20.78</td>
<td>20.85</td>
</tr>
<tr>
<td>Modularity</td>
<td>10.21</td>
<td>13.37</td>
<td>15.24</td>
<td>15.11</td>
<td>16.14</td>
<td>16.64</td>
<td>17.02</td>
<td>17.1</td>
<td>17.14</td>
<td>17.12</td>
</tr>
<tr>
<td>wvRN</td>
<td>1.53</td>
<td>2.46</td>
<td>2.91</td>
<td>3.47</td>
<td>4.95</td>
<td>5.56</td>
<td>5.82</td>
<td>6.59</td>
<td>8.00</td>
<td>7.26</td>
</tr>
<tr>
<td>Majority</td>
<td>0.45</td>
<td>0.44</td>
<td>0.45</td>
<td>0.46</td>
<td>0.47</td>
<td>0.44</td>
<td>0.45</td>
<td>0.47</td>
<td>0.47</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Table 3: Multi-label classification results in Flickr

<table>
<thead>
<tr>
<th>% Labeled Nodes</th>
<th>1%</th>
<th>2%</th>
<th>3%</th>
<th>4%</th>
<th>5%</th>
<th>6%</th>
<th>7%</th>
<th>8%</th>
<th>9%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Micro-F1(%)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeepWalk</td>
<td>37.95</td>
<td>39.28</td>
<td>40.08</td>
<td>40.78</td>
<td>41.32</td>
<td>41.72</td>
<td>42.12</td>
<td>42.48</td>
<td>42.78</td>
<td>43.05</td>
</tr>
<tr>
<td>SpectralClustering</td>
<td>23.90</td>
<td>31.68</td>
<td>35.53</td>
<td>36.76</td>
<td>37.81</td>
<td>38.63</td>
<td>38.94</td>
<td>39.46</td>
<td>39.92</td>
<td>40.07</td>
</tr>
<tr>
<td>EdgeCluster</td>
<td>23.01</td>
<td>28.18</td>
<td>33.1</td>
<td>32.88</td>
<td>35.76</td>
<td>37.38</td>
<td>38.21</td>
<td>37.75</td>
<td>38.68</td>
<td>39.42</td>
</tr>
<tr>
<td>Modularity</td>
<td>26.79</td>
<td>24.84</td>
<td>25.25</td>
<td>25.23</td>
<td>25.32</td>
<td>25.33</td>
<td>25.31</td>
<td>25.34</td>
<td>25.38</td>
<td>25.38</td>
</tr>
</tbody>
</table>

Table 4: Multi-label classification results in YouTube
graph representations, EdgeCluster. When 1% of the labeled nodes are used for test, the Micro-F1 improves by 14%. The Macro-F1 shows a corresponding 10% increase. This lead narrows as the training data increases, but DeepWalk ends with a 3% lead in Micro-F1, and an impressive 5% improvement in Macro-F1.

This experiment showcases the performance benefits that can occur from using social representation learning for multi-label classification. DeepWalk, can scale to large graphs, and performs exceedingly well in such a sparsely labeled environment.

6.2 Parameter Sensitivity

In order to evaluate how changes to the parameterization of DeepWalk effect its performance on classification tasks, we conducted experiments on two multi-label classification tasks (Flickr and BlogCatalog). In the interest of brevity, we have fixed the window size and the walk length to emphasize local structure (\( w = 10, r = 40 \)). We then vary the number of latent dimensions (\( d \)), the number of walks started per vertex (\( \gamma \)), and the amount of training data available (\( R \)) to determine their impact on the network classification performance.

6.2.1 Effect of Dimensionality

Figure 5a shows the effects of increasing the number of latent dimensions available to our model.

Figures 5a1 and 5a2 examine the effects of varying the dimensionality and training ratio. The performance is quite consistent between both Flickr and BlogCatalog and show that the optimal dimensionality for a model is dependent on the number of training examples. (Note that 1% of Flickr has approximately as many labeled examples as 10% of BlogCatalog).

Figures 5a3 and 5a4 examine the effects of varying the dimensionality and number of walks per vertex. The relative performance between dimensions is relatively stable across different values of \( \gamma \). These charts have two interesting observations. The first is that there is most of the benefit is accomplished by starting \( \gamma = 30 \) walks per node in both graphs. The second is that the relative difference between different values of \( \gamma \) is quite consistent between the two graphs. Flickr has an order of magnitude more edges than BlogCatalog, and we find this behavior interesting.

These experiments show that our method can make useful models of various sizes. They also show that the performance of the model depends on the number of random walks it has seen, and the appropriate dimensionality of the model depends on the training examples available.

6.2.2 Effect of sampling frequency

Figure 5b shows the effects of increasing \( \gamma \), the number of random walks that we start from each vertex.

The results are very consistent for different dimensions (Fig. 5b1 Fig. 5b3, and the amount of training data (Fig. 5b2 Fig. 5b4). Initially, increasing \( \gamma \) has a big effect in the results, but this effect quickly slows (\( \gamma > 10 \)). These results demonstrate that we are able to learn meaningful latent representations for vertices after only a small number of random walks.

7. RELATED WORK

The main differences between our proposed method and previous work can be summarized as follows:

1. We learn our latent social representations, instead of computing statistics related to centrality or partitioning.

2. We do not attempt to extend the classification procedure itself (through collective inference or graph kernels).

3. We propose a scalable online method which uses only local information. Most methods require global information and are offline.

4. We apply unsupervised representation learning to graphs. In this section we discuss related work in network classification and unsupervised feature learning.

7.1 Relational Learning

Relational classification (or collective classification) methods use links between data items as part of the
classification process. Exact inference in the collective classification problem is NP-hard, and solutions have focused on the use of approximate inference algorithm which may not be guaranteed to converge.

The most relevant relational classification algorithms to our work incorporate community information by learning clusters, by adding edges between nearby nodes, by using PageRank, or by extending relational classification to take additional features into account. Our work takes a substantially different approach. Instead of a new approximation inference algorithm, we propose a procedure which learns representations of network structure which can then be used by existing inference procedure (including iterative ones).

A number of techniques for generating features from graphs have also been proposed. In contrast to these methods, we frame the feature creation procedure as a representation learning problem.

Graph Kernels have been proposed as a way to use relational data as part of the classification process, but are quite slow unless approximated. Our approach is complementary; instead of encoding the structure as part of a kernel function, we learn a representation which allows them to be used directly as features for any classification method.

7.2 Unsupervised Feature Learning

Distributed representations have been proposed to model structural relationship between concepts. These representations are trained by the back-propagation and gradient descent. Computational costs and numerical instability led to these techniques to be abandoned for almost a decade. Recently, distributed computing allowed for larger models to be trained, and the growth of data for unsupervised learning algorithms to emerge. Distributed representations usually are trained through neural networks, these networks have made advancements in diverse fields such as computer vision, speech recognition, and natural language processing.

8. CONCLUSIONS

We propose DeepWalk, a novel approach for learning latent social representations of vertices. Using local information from truncated random walks as input, our method learns a representation which encodes structural regularities. Experiments on a variety of different graphs illustrate the effectiveness of our approach on challenging multi-label classification tasks.

As an online algorithm, DeepWalk is also scalable. Our results show that we can create meaningful representations for graphs which are too large for standard spectral methods. On such large graphs, our method significantly outperforms other methods designed to operate for sparsity. We also show that our approach is parallelizable, allowing workers to update different parts of the model concurrently.

In addition to being effective and scalable, our approach is also an appealing generalization of language modeling. This connection is mutually beneficial. Advances in language modeling may continue to generate improved latent representations for networks. In our view, language modeling is actually sampling from an unobservable language graph. We believe that insights obtained from modeling observable graphs may in turn yield improvements to modeling unobservable ones.

Our future work in the area will focus on investigating this duality further, using our results to improve language modeling, and strengthening the theoretical justifications of the method.

Acknowledgements

The authors thank the reviewers for their helpful comments. This research was partially supported by NSF Grants DBI-1060572 and IIS-1017181, and a Google Faculty Research Award.

9. REFERENCES


