DeepWalk: Online Learning of Social Representation

(2014)

Perozzi Bryan, Skiena Steven
Resume

December 21, 2018

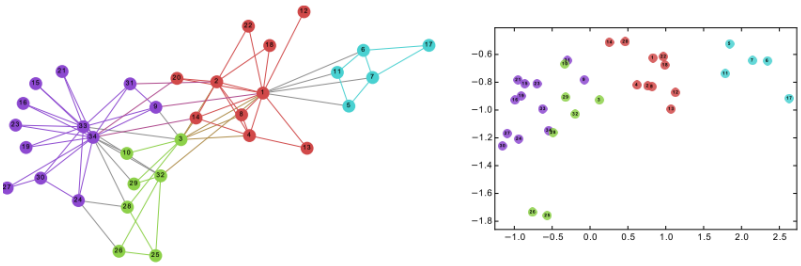
Abstract

A novel approach for learning latent representations of vertices in a network. Thus encoding these representation in a continuous vector space, which can be the input of other deep learning models. This done using truncated random walks to learn the leten representations by treating walks as the equivalent of sentences.

1 Introduction

To be able to use deep learning techniques in network analysis, they introduce the DeepWalk Alorithms that learn *social representations* of the graph's vertices by modeling a stream of short random walks, social representation are latent features of the vertices that capture neighborhood similarity and community membership.

Deep walk takes a graph as input and produces a latent representation as output.



(a) Input: Karate Graph (b) Output: Representation

Main contributions:

- DeepWalk learns structural regularities present within short random walks.
- Can outperform old approches
- The scalability of the algorithm, and being easily parallelizable.

Side note: Random Walks First introducted by Karl Pearson:
A man starts from a point 0 and walks l yards in a straight line; he then turns through any angle whatever and walks another l yards in a straight line. He repeats this process n times. I require the probability that after these n stretches he is at a distance between r and $r + \delta r$ from his starting point.
Among other issues, the following four have been investigated in one, two, and three dimensions:

- The expected position x of the drunk after n steps.
- The maximum position x that the drunk has reached after n steps.
- The expected time of the drunks last visit to 0.
- The probability that the drunk hasnt stumbled upon his own path after n steps

Suppose a drunk leaves a bar and walks aimlessly up and down the street, totally disoriented. We model the street as a line with the bar at theorig in, and assume that the drunk takes unit steps, so we may record his position withan integer. Thus, for example, if he takes 5 steps to the left, he will be at position 5. We now calculate the probability that the drunk is at position x after n steps. Assume that the drunks walk can be modeled by a Bernoulli process. Say that the ithstep is represented by the random variable X_i . A value of 1 indicates a step to theleft; a value of 1, a step to the right, so that

$$G_n = X_1 + X_2 + \ldots + X_n$$

gives the position of the drunk after n steps. We want to know the distribution of therandom variable G_n. Denote the value of G_n by x. Denote the number of steps taken to the right by r,the number to the left by l. Then :

$$x = r - l \text{ and } n = r + l$$

It follows that

$$r = \frac{1}{2}(x + n) \text{ and } l = \frac{1}{2}(n - x)$$

Now, there are $\begin{pmatrix} n \\ l \end{pmatrix}$ ways that l given steps can occur among n total steps. This is also the number of ways of arriving at the point x, and each way has probability $p^r q^l$. Notethat n and x must have the same parity because $nx = 2l$. We can therefore concludethat the probability distribution at point x is given by the formula,

$$P(G_n = x) = \begin{cases} \begin{pmatrix} n \\ l \end{pmatrix} p^r q^l, & \text{if } x = n \bmod 2 \\ 0, & \text{otherwise.} \end{cases}$$

2 Preliminaries

2.1 Definitions

Let $G = (V, E)$, where V represent the members of the network, E the connections, $E \subseteq (V \times V)$ and $G_L = (V, E, X, Y)$ is a partially labeled social network, with attributes $X \in \mathbb{R}^{|V| \times S}$ where S is the size of the feature space for each attribute vector, and $Y \in \mathbb{R}^{|V| \times |\mathcal{Y}|}$, $\mathcal{Y}$ is the set of labels.

The aim is to learn a hypothesis H that maps elements of X to the labels set $\mathcal{Y}$, in this paper they use the structure of the graph to capture the network topology information, as an unsupervised method which learns the features that capture the structure independentely of the labels distribution, and using these features as the inputs for other ML algorithms, and achieve superior performance.

The goal is to learn $X_E \in \mathbb{R}^{|V| \times d}$, where d is small number of latent dimensions.

2.2 Learning social representations

The objective is 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.
- **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.

With the combination of short random walks and optimization techniques, they are capable of satisfying these requirements.

Random walks A random walk rooted at vertex v_i is represented as $\mathcal{W}_{v_i}$, it is a stochastic process with random varibales $\mathcal{W}_{v_i}^1, \mathcal{W}_{v_i}^2, \ldots, \mathcal{W}_{v_i}^k$, such that $\mathcal{W}_{v_i}^{k+1}$ is a vertex chosen at random from the neighbors of vertex v_k.

Random walks gives two 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.

Side note: Power low & Network degrees
The power law (also called the scaling law) states that a relative change in one quantity results in a proportional relative change in another. The simplest example of the law in action is a square; if you double the length of a side then the area will quadruple. A power law distribution has the form $Y = kX^{\alpha}$, where:
- X and Y are variables of interest,
- α is the laws exponent,
- k is a constant.
Any inverse relationship like $Y = X^{-1}$ is also a power law, because a change in one quantity results in a negative change in another.
A power-law distribution is a special kind of probability distribution. There are several ways to define them mathematically. One of them, for a continuous random variable.

$$p(x) = Cx^{-\alpha} \quad \text{for } x \geq x_{\min}$$

Degree Distribution A property of the full-scale structure of a network that is typically investigated is the distribution of the network node degrees. The degree of a node is the number of neighbours of the node. For any integer $k \geq 0$, the quantity p_k is the fraction of nodes having degree k. This is also the probability that a randomly chosen node in the network has degree k. The quantities p_k, for $k \geq 0$, represent the degree distribution of the network.

Directed networks have two different degree distributions, the in-degree and the out-degree distributions. Typically, the in-degree distribution is the important one. We might observe, however, that the true degree distribution of a directed network is a joint distribution of in- and out- degrees. That is, for every $i, j \geq 0$, we have a probability $p_{i,j}$ that a randomly chosen node in the graph has i predecessor nodes and j successor nodes. By using a joint distribution in this way we can investigate the correlation of the in- and out- degrees of vertices. For instance, if vertices with high out-degree tend to have also high in-degree.

Power laws 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.

This work used the idea that that techniques which have been used to model natural language (where the symbol frequency follows a power law distribution (or Zipfs law)) can be re-purposed to model community structure in networks.

2.3 Language model

The goal of language modeling is to estimate the likelihood of a specific sequence of words appearing in a corpus, given a sequence of words $W_1^n = (w_0, w_1, \cdots, w_n)$, where $w_i \in \mathcal{V}$ (V is the vocabulary), we would like to maximize the $\Pr(w_n | w_0, w_1, \cdots, w_{n-1})$ over all the training corpus.

Similarily, random 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.

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

But the goal is to learn a latent representation and not only a probability distribution of node co-occurrences, the problem is then to estimate,

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

Given that ϕ is a mapping function representing the latent social representation as associated with each vertex of the network: $\Phi : v \in V \mapsto \mathbb{R}^{|V| \times d}$. However, as the walk length grows, computing this conditional probability becomes unfeasible.

Using the relaxation presented at word2vec. 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 thegiven 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. Translating in this context to :

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

By combining both truncated random walks and language models the proposed method satisfies all of our desired properties.

3 Proposed method

inputs Short truncated random walks (considered as the corpus in language modeling) to obtain the frequency distribution of the vertices, and the graph vertices (as the vocabulary).

Algorithm 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 v_i as the root of the random walk W_{v_i}. A walk samples uniformly from the neighbors of the last vertex visited until the maximum length (t) is reached. This is done for a number of random walks γ.

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 $\Phi \in \mathbb{R}^{|V| \times d}$
Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O}$ = Shuffle(V)
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

We can think of γ as the number of epochs, each time we shuffle the graph vertices, and go trought all the vertives of the graph, generating a random walk $|\mathcal{W}_{v_i}|$ for each one, and then use it to update the representations.

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

1: **for each** $v_j \in \mathcal{W}_{v_i}$ **do**
2: **for each** $u_k \in \mathcal{W}_{v_i}[j-w : j+w]$ **do**
3: $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
4: $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
5: **end for**
6: **end for**

SkipGram is a language model that maximizes the co-occurrence probability among the words that appear withina window, w, in a sentence. It approximates the conditional probability using an independence assumptionas the following

$$Pr\left(\{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i | \Phi\left(v_i\right)\right) = \prod_{j=i-w, j\neq i}^{i+w} Pr\left(v_j | \Phi\left(v_i\right)\right)$$

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 v_j to its current representationvector $\Phi(v_j) \in R_d$. Given the representation of v_j, we would like to maximize the probability of its neighbors in the walk. We can learn such a posterior distribution using several choices of classifiers. Like a softmax, given a center node, we'll have an ouput of size $|V|$, each one is the probability for a given vertex being in the window of the vertices vesited in the random walk starting from the node v_j, this is computationnaly expensive $O(V)$, to reduce the computation, they use hierarchical softmax to obtain logarithmic complexity.

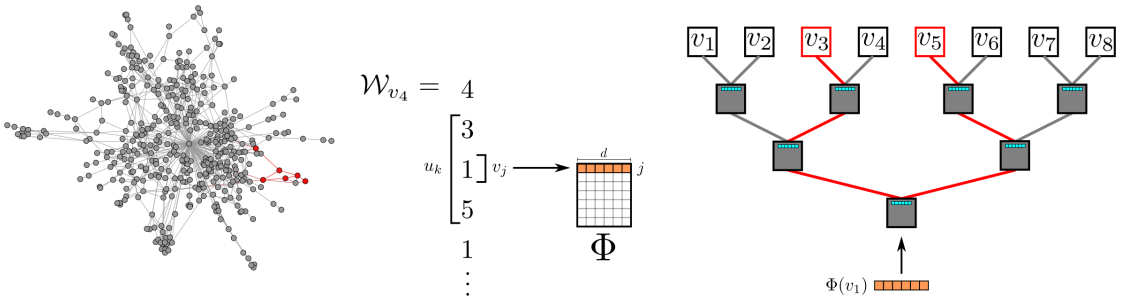
Hierarchical Softmax Given that $u_k \in V$, calculating $\Pr\left(u_k | \Phi\left(v_j\right)\right)$is not feasible. Computing the partition function (normalization factor) is expensive, so instead we will factorize the conditional probability using Hierarchical softmax. We assign the vertices to the leaves of a binary tree, turning the prediction problem into maximizing the probability of a specific path in the hierarchy. If the path to vertex u_k is identified by a sequence of tree nodes $\left(b_0, b_1, \ldots, b_{\lceil \log |V| \rceil}\right)$, $\left(b_0 = \text{ root }, b_{\lceil \log |V|} = u_k\right)$ then

$$\Pr\left(u_k | \Phi\left(v_j\right)\right) = \prod_{l=1}^{\lceil \log |V| \rceil} \Pr\left(b_l | \Phi\left(v_j\right)\right)$$

Now, $\Pr\left(b_l | \Phi\left(v_j\right)\right)$ could be modeled by a binary classifierthat is assigned to the parent of the node b_l:

$$\Pr\left(b_l \left| \Phi\left(v_j\right) = 1 / \left(1 + e^{-\Phi(v_j) \cdot \Psi(b_l)}\right)\right.\right.$$

where $\Psi\left(b_l\right) \in \mathbb{R}^d$ d is the representation assigned to tree node b_ls parent. This reduces the computational complexity of calculating $\Pr\left(u_k | \Phi\left(v_j\right)\right)$ from $O(|V|)$ to $O(log|V|)$.



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

Figure 3: Overview of DeepWalk. We slide a window of length $2w+1$ over the random walk $\mathcal{W}_{v_4}$, mapping the central vertex v_1 to its representation $\Phi(v_1)$. Hierarchical Softmax factors out $\Pr(v_3 \mid \Phi(v_1))$ and $\Pr(v_5 \mid \Phi(v_1))$ over sequences of probability distributions corresponding to the paths starting at the root and ending at v_3 and v_5. The representation Φ is updated to maximize the probability of v_1 co-occurring with its context $\{v_3, v_5\}$.

Optimisation 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. The derivatives are estimated using the back-propagation algorithm. The learning rate α 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.

Other variants **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 (but we must have fixed learning rate, and not used hirearchical softmax given that V in unknown).

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.

4 Results

	% Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1(%)	DeepWalk	**36.00**	**38.20**	**39.60**	**40.30**	**41.00**	**41.30**	41.50	41.50	42.00
	SpectralClustering	31.06	34.95	37.27	38.93	39.97	40.99	**41.66**	**42.42**	**42.62**
	EdgeCluster	27.94	30.76	31.85	32.99	34.12	35.00	34.63	35.99	36.29
	Modularity	27.35	30.74	31.77	32.97	34.09	36.13	36.08	37.23	38.18
	wvRN	19.51	24.34	25.62	28.82	30.37	31.81	32.19	33.33	34.28
	Majority	16.51	16.66	16.61	16.70	16.91	16.99	16.92	16.49	17.26
Macro-F1(%)	DeepWalk	**21.30**	**23.80**	25.30	26.30	27.30	27.60	27.90	28.20	28.90
	SpectralClustering	19.14	23.57	**25.97**	**27.46**	**28.31**	**29.46**	**30.13**	**31.38**	**31.78**
	EdgeCluster	16.16	19.16	20.48	22.00	23.00	23.64	23.82	24.61	24.92
	Modularity	17.36	20.00	20.80	21.85	22.65	23.41	23.89	24.20	24.97
	wvRN	6.25	10.13	11.64	14.24	15.86	17.18	17.98	18.86	19.57
	Majority	2.52	2.55	2.52	2.58	2.58	2.63	2.61	2.48	2.62

Table 2: Multi-label classification results in BlogCatalog

	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**32.4**	**34.6**	**35.9**	**36.7**	**37.2**	**37.7**	**38.1**	**38.3**	**38.5**	**38.7**
	SpectralClustering	27.43	30.11	31.63	32.69	33.31	33.95	34.46	34.81	35.14	35.41
	EdgeCluster	25.75	28.53	29.14	30.31	30.85	31.53	31.75	31.76	32.19	32.84
	Modularity	22.75	25.29	27.3	27.6	28.05	29.33	29.43	28.89	29.17	29.2
	wvRN	17.7	14.43	15.72	20.97	19.83	19.42	19.22	21.25	22.51	22.73
	Majority	16.34	16.31	16.34	16.46	16.65	16.44	16.38	16.62	16.67	16.71
Macro-F1(%)	DeepWalk	**14.0**	17.3	**19.6**	**21.1**	**22.1**	**22.9**	**23.6**	**24.1**	**24.6**	**25.0**
	SpectralClustering	13.84	**17.49**	19.44	20.75	21.60	22.36	23.01	23.36	23.82	24.05
	EdgeCluster	10.52	14.10	15.91	16.72	18.01	18.54	19.54	20.18	20.78	20.85
	Modularity	10.21	13.37	15.24	15.11	16.14	16.64	17.02	17.1	17.14	17.12
	wvRN	1.53	2.46	2.91	3.47	4.95	5.56	5.82	6.59	8.00	7.26
	Majority	0.45	0.44	0.45	0.46	0.47	0.44	0.45	0.47	0.47	0.47

Table 3: Multi-label classification results in Flickr

	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
	DeepWalk	**37.95**	**39.28**	**40.08**	**40.78**	**41.32**	**41.72**	**42.12**	**42.48**	**42.78**	**43.05**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
Micro-F1(%)	EdgeCluster	23.90	31.68	35.53	36.76	37.81	38.63	38.94	39.46	39.92	40.07
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	26.79	29.18	33.1	32.88	35.76	37.38	38.21	37.75	38.68	39.42
	Majority	24.90	24.84	25.25	25.23	25.22	25.33	25.31	25.34	25.38	25.38
	DeepWalk	**29.22**	**31.83**	**33.06**	**33.90**	**34.35**	**34.66**	**34.96**	**35.22**	**35.42**	**35.67**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
Macro-F1(%)	EdgeCluster	19.48	25.01	28.15	29.17	29.82	30.65	30.75	31.23	31.45	31.54
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	13.15	15.78	19.66	20.9	23.31	25.43	27.08	26.48	28.33	28.89
	Majority	6.12	5.86	6.21	6.1	6.07	6.19	6.17	6.16	6.18	6.19

Table 4: Multi-label classification results in YouTube