University of
Cambridge

Mathematics Tripos

Part III Essay

Walking Deeper on Dynamic Graphs

December 16, 2019

Written by
Joshua Snyder

Contents

1 Motivation

The motivation for studying nodes in graphs and their representations comes from the desire to understand networks of people or objects and their relations. The motivating question for this essay is

> **Question** (Motivating Question)**.** Given a large network of people how can we quantify the relationships between them?

The natural way to go about this is to let nodes represent persons and let edges between them represent connections from which we can induce some understanding of relationship or trust between two people. This task is difficult; even with a relatively small number of people it is not a task on which humans perform very well and the task rapidly becomes difficult as the number of nodes in the graph increases.
Many of the networks which we wish to study are dynamic; that is they change with time. In a large network it is common that small changes occur during each epoch of time that over time cause larger changes to the network structure. How do we quantify these changes and their impact on the relationships in the network without having to completely re-analyse the network after ever epoch of time, losing the information that we already learned. A frequent example upon which we can draw similarity is that of social networks. A large social network has new users (nodes) being added and new relationships (edges) formed in each epoch of time, however in any given short time period the graph representing the social network does not change substantially. It would be both foolish and costly to re-analyse the graph after each epoch however most of the previous literature has focused on static graphs. In the latter half of this essay I will give an account of recent progress into the application of DeepWalk and similar algorithms to dynamic graphs.

2 Summary

The section on DeepWalk explores the fundamental connection between DeepWalk and matrix factorisation as presented in the paper by Qiu et al.[1]. In this section, we demonstrate the conditions under which DeepWalk is factorising an appropriate matrix that stores some measure of similarity between nodes in the graph. In the section "Why DeepWalk sucks" we look at some of the criticism that followed the original DeepWalk paper and how these criticisms were addressed by future authors, the section is indicative of the shortcomings of DeepWalk but is by no means exhaustive.
After this, the section "Dynamic DeepWalking" focusses on applying a variant of the DeepWalk algorithm, proposed by Sajjad et al. [2], for dynamic graphs. This is especially important in the context of social graphs, which usually contain information about the interactions of individuals over time. Learning social representations for dynamic graphs can allow for a better understanding of how communities move and change over time.
Finally, to exhibit Dynamic DeepWalking in practice, the final section of this essay applies the Unbiased Update algorithm introduced by Sajjad et al.[2] to data from users on the social network Gab to look at how user interaction on the network changes with time.

3 DeepWalk

This section gives an outline of the social representation learning algorithm DeepWalk, first introduced in the seminal paper DeepWalk: Online Learning of Social Representations by B. Perozzi et. al. [3]. The method proposed in this paper not only demonstrated performance improvements from previous methodologies but also motivated an entirely different approach. At the time of the paper being written, significant advancements were being made in natural languade processing (NLP) and the idea of word embeddings was becoming popular through an embedding algorithm known as word2vec [4, 5]. DeepWalk implements this algoritm but replaces the idea of the context of a word in a sentence with the context of a node in a random walk on a graph. This is the crucial concept of DeepWalk from which the remaining details naturally follow.

The original paper on DeepWalk is lacking in a mathematical underpinning and in this section we will model the algorithm mathematically. It is suggested that the reader is familiar with the concepts outlined in the paper by Perozzi et. al. prior to reading this (more) mathematical exposition. I have endevoured to use similar notation to the original paper to ease cross-referencing. Without further ado, let us begin our journey.

> **Definition.** Let $G = (V, E)$ be an undirected graph (representing a network). V represents the members of the network, commonly referred to as the nodes and $E \subset V \times V$ represents their connections.
>
> The nodes and edges have lables and $G_L = (V, E, X, Y)$ represents the partially labelled network. $X \in \mathbb{R}^{|V| \times S}$ where S is the size of the feature space for each attribute vector and $Y \in \mathbb{R}^{|V| \times |\mathcal{Y}|}$, where $\mathcal{Y}$ is the set of labels.

Our goal is to learn $X_E \in \mathbb{R}^{|V| \times d}$ where d is a small number of latent dimensions. The idea is that each latent dimension contributes a dimensional...

*** Explain the DeepWalk algorithm here ***

3.1 Introduction to SkipGram

To understand DeepWalk mathematically, we first need to understand what the SkipGram model is doing, since this is the model underpinning DeepWalk, which took it from NLP and applied it to graphs. In the context of graph networks, SkipGram trains a neural network to do the following task:
Given an input vertex v and a random walk, W_v, of length $2t + 1$ with v at it's centre. Pick a nearby vertex at random. The task of the neural network is to predict the probability that each vertex in V will be the randomly chosen vertex. Therefore, verticies far away on the graph that correspond to unfamiliar nodes are unlikely to co-occur on the same random walk and will be assigned a low probability. Conversely, nearby and well connected vertices are likely to co-occur on a random walk with input v and thus will be assigned higher probabilities. This allows us to train a network with weights that represent the connectedness between nodes on the graph. The neural network is trained by feeding it pairs of nodes (v, c) where v represents the input node and c is a

context node, which lies within distance t of the vertex v.

To formalise this, each of the nodes $v \in V$ are represented by a one-hot encoding vector $e_v \in \mathbb{R}^{|V|}$ allowing us to feed e_v into the neural network. When e_v is fed into the network, a single linear hidden layer with d neurons is used, where d is the desired dimension of the latent representations, which is then passed to a softmax classifier for output. The output of the network is a vector $o \in \mathbb{R}^d$ containing the estimated probabilities that a randomly selected nearby word is that vocabulary word.

The idea behind having a linear hidden layer, which does not use an activation function, is to use the resulting weight matrix $W \in \mathbb{R}^{|V| \times d}$ as the embedding vectors for the nodes in the graph. This is intuitive as the hidden layer acts as a bottleneck that tries to represent as much information as possible to distinguish the nodes, but is only allowed d neurons to do so. Since $d \ll |V|$ there is a low risk of overfitting.

*** Is our vocabulary the whole graph, or is it just the random walk that we feed to the SkipGram model, perhaps it is just the random walk, be careful with this! ***

*** Create here an image resembling the one found in the blog post on SkipGram but for graphs and with d dimensions ***

The algorithm used in DeepWalk varies slightly from the SkipGram algorithm discussed. Calculating the normalization factor in the Softmax layer requires a computational complexity of $O(|V|)$, in the original DeepWalk paper this is reduced by using Hierarchical Softmax to approximate the softmax probabilities, requiring a complexity of only $O(log|V|)$. In particular, a Huffman coding is used to reduce the access time of frequent elements in the tree, as suggested by Mikolov et al. in the original Word2Vec papers.[5, 4]

In later adaptations of DeepWalk, SkipGram with Negative Sampling (SGNS) is used instead of Hierarchical Softmax. In the remainder of this essay, when referring to DeepWalk, it will be implicitly assumed that Negative Sampling is used as appose to Hierarchical Softmax. This is because SGNS has been found to be more efficient and therefore has been adopted by much of the further literature. This convention will also serve us well when we look at applying DeepWalk to dynamic graphs since here Negative Sampling is also applied.

*** Can give a summary of NS here, if so just summarise what is said in Blog Post 2 on SkipGram *** *** Has NS been shown, or just found empirically, to be more efficient then Hierarchical Softmax? Check node2vec paper for information on this. ***

3.2 SkipGram as matrix factorisation

In this section we will exhibit a proof that SGNS is equivalent to factorising a certain matrix M into two smaller matricies W and C where the rows in W correspond to the learned embedding of each vertex. This result was first proved by Levy and Goldberg[6] in the context of word embeddings.

3.2.1 Objective of SGNS

Given an arbitrary input-context pair (v, c) the objective is to determine if the pair comes from the random-walk corpus $\mathcal{D}$.
Let $P(\mathcal{D} = 1|v, c)$ denote the probability that (v, c) comes from a random walk on the graph and $P(\mathcal{D} = 0|v, c)$ the probability it does not. Then the distribution is modelled by a signoid function

$$P(D = 1|v, c) = \sigma(\vec{v} \cdot \vec{c}) = \frac{1}{1 + e^{-\vec{v} \cdot \vec{c}}}$$

where $\vec{v}$ and $\vec{c}$ are d-dimensional vectors to be learned. SGNS attempts to maximise $P(\mathcal{D} = 1|v, c)$ for observed pairs (v, c) whilst simultaneously maximising $P(\mathcal{D} = 0|v, c)$ for randomly sampled negative examples.
It assumes that randomly selecting a context c for a given node v is likely to result in an unobserved pair (v, c). In the context of social networks, this assumption is reasonable since social networks are almost always sparse (The number of edges is usually $O(|V|)$). However in a different context, if the network is dense, then this may be an unreasonable assumption.

According to this assumption, the objective function of SGNS for a single observation (v, c) is:

$$\log \sigma(\vec{v} \cdot \vec{c}) + b \cdot \mathbb{E}_{c_N \sim P_D} \log \sigma(-\vec{v} \cdot \vec{c})$$

where the minus sign comes from the fact that $1 - \sigma(x) = \sigma(-x)$, b is the number of negative samples and c_N is the sampled context node, drawn according to $P_D(c) = \frac{\#(c)}{|\mathcal{D}|}$ which is known as the unigram distribution.

Notation. $\#(v, c)$, $\#(v)$ and $\#(c)$ denote the number of times vertex-context pair (v, c), vertex v and context c appear in the generated random-walk corpus $\mathcal{D}$ respectively.

This objective function is trained using stochastic gradient descent with updates after each observed pair in the random-walk corpus $\mathcal{D}$. The resulting global objective becomes

$$l = \sum_{v \in V} \sum_{c \in V} \#(v, c) \log \sigma(\vec{v} \cdot \vec{c}) + b \cdot \mathbb{E}_{c_N \sim P_D} \log \sigma(-\vec{v} \cdot \vec{c}) \tag{1}$$

3.2.2 Finding the similarity function learned by SkipGram

If we let W be the matrix with rows v_i (The matrix W is used to highlight that it is the weight matrix in the neural network previosly described) and C the matrix with columns c_i then SGNS can be interpreted as factorising a matrix $M = WC^T$. An entry in the matrix M_{ij} corresponds to the dot product $\vec{v_i} \cdot \vec{c_j}$. Therefore SGNS is factorising a matrix in which each row corresponds to an input node $v_i \in |V|$ and each column to a context node $c_j \in |V|$ and the value of M_{ij} expresses the strength of association between the input-context pair (v_i, c_j) using some similarity function $s(v_i, c_j)$.

Theorem 3.1 (Levy, Goldberg (2014))**.** *SkipGram with Negative Sampling (SGNS) is implicitly factorising a matrix $M = WC^T$ with*

$$M_{ij} = \log \frac{\#(v_i, c_j)|\mathcal{D}|}{\#(v_i)\#(c_j)} - \log b$$

where $W, C \in \mathbb{R}^{|V| \times d}$, and b is the number of negative samples.

Proof. Firstly, for sufficiently large dimensionality d (so as to allow for a perfect reconstruction of M), each of the products $v_i \cdot c_j$ can be assumed to take their values independently of the others. *** WHY. EXPLAIN THIS. Why not sufficiently large value of $|\mathcal{D}|$, why does d matter here? ***

Due to this independence, the objective function l can be maximised with respect to each pair $v \cdot c$ individually.

The expectation term in l can be written explicitly as

$$\begin{aligned}\mathbb{E}_{\tilde{c} \sim P_{\mathcal{D}}}[\log \sigma(-\vec{v} \cdot \vec{\tilde{c}})] &= \sum_{\tilde{c} \in V} \frac{\#(\tilde{c})}{|\mathcal{D}|} \log \sigma(-\vec{v} \cdot \vec{\tilde{c}}) \\ &= \frac{\#(c)}{|\mathcal{D}|} \log \sigma(-\vec{v} \cdot \vec{c}) + \sum_{\tilde{c} \in V \setminus \{c\}} \frac{\#(\tilde{c})}{|\mathcal{D}|} \log \sigma(-\vec{v} \cdot \vec{\tilde{c}})\end{aligned}$$

and the objective function l can be expressed as

$$l = \sum_{v \in V} \sum_{c \in V} \#(v, c) \log \sigma(\vec{v} \cdot \vec{c}) + \sum_{v \in V} \#(v) \left(b \cdot \mathbb{E}_{\tilde{c} \sim P_{\mathcal{D}}}[\log \sigma(-\vec{v} \cdot \vec{\tilde{c}})] \right)$$

where the second term comes from the fact that $\#(v) = \sum_{c \in V} \#(v, c)$ by definition.
Combining these equations gives that the local objective for an input-context pair is

$$l(v, c) = \#(v, c) \log \sigma(\vec{v} \cdot \vec{c}) + b \cdot \#(v) \cdot \frac{\#(c)}{|\mathcal{D}|} \log -\sigma(-\vec{v} \cdot \vec{c}) \tag{2}$$

Now to simplify the notation let $x = \vec{v} \cdot \vec{c}$ and, since we are assuming each of the $\vec{v_i} \cdot \vec{c_j}$ to take their values independently, we take the partial derivatve with respect to x and optimise the local objective:

$$\frac{\partial l}{\partial x} = \#(v, c) \cdot \log \sigma(-x) - b \cdot \#(v) \cdot \frac{\#(c)}{|\mathcal{D}|} \cdot \sigma(x)$$

Where the derivatives are since $\frac{d}{dx}\sigma(x) = \sigma(-x)$. Setting the derivative to zero and multipying through by $\frac{-e^x}{\sigma(x)\sigma(-x)}$ gives:

$$\frac{b \cdot \#(v) \cdot \#(c)}{|\mathcal{D}|} e^{2x} + \left(\frac{b \cdot \#(v) \cdot \#(c)}{|\mathcal{D}|} - \#(v, c) \right) e^x - \#(v, c) = 0$$

This is a quadratic equation in e^x with two solutions. The first solution, $e^x = -1$ is infeasable since $x \in \mathbb{R}$ and so the appropriate solution is

$$e^x = \frac{\#(v, c) \cdot |\mathcal{D}|}{\#(v)\#(c)} \cdot \frac{1}{b}$$

and substituting $x = \vec{v} \cdot \vec{c}$ back into the equation gives

$$M_{ij} = \vec{v} \cdot \vec{c} = log\left(\frac{\#(v,c) \cdot |\mathcal{D}|}{\#(v) \cdot \#(c)}\right) - \log b$$

□

Most interestingly, the resulting expression for the similarity function s is the pointwise mutual information (PMI) shifted by a factor of $\log b$. PMI was introduced as a measure of association between words in 1990 by Church and Hanks [7] and became widely adopted for NLP tasks.

There is an equivalent theorem for SkipGram with Softmax that was proved by Yang et al.[8] and was later used in their development of text-associated DeepWalk (TADW)[9] which encorporates text features of the verticies in a social graph.

Theorem 3.2 (Yang et al. (2015))**.** *SkipGram with Softmax is implicitly factorising the matrix* $M = WC^T$ *with*

$$M_{ij} = \log \frac{\#(v_i, v_j)}{\#(v_i)}$$

The proof of this follows very similarly to the previous theorem and will not be shown here since we will only be concerned with SGNS.

*** At some point I need to actually outline these algorithms as the other papers have done, there needs to be an explicit DeepWalk algorithm ***

3.3 DeepWalk as matrix factorisation

We continue to look at DeepWalk in the context of matrix factorisation. Much of the proceeding analysis was exhibited in a recent paper by Qiu et al.[1] published in 2018 building upon work by Yang et al.[9] from 2015. The aim of the former paper was to lay the foundationds for, and unify, the SkipGram based network embedding methods.

First we give some preliminary definitions.

Definition (Adjacency Matrix (A))**.** The adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$ for a graph G is the matrix with $A_{ij} = 1$ if $(v_i, v_j) \in E(G)$ and $A_{ij} = 0$ otherwise.

Definition (Degree Matrix (D))**.** The degree matrix $D \in \mathbb{R}^{|V| \times |V|}$ for a graph G is a diagonal matrix with $D_{ii} = d_i = degree(v_i)$ for $v_i \in V$ and $D_{ij} = 0$ otherwise.

Definition (Transition Matrix (P))**.** The transition matrix $P \in \mathbb{R}^{|V| \times |V|}$ for a graph G is the matrix $P = D^{-1}A$. It has entries $P_{ij} = \frac{1}{d_i}$ if $(v_i, v_j) \in E(G)$ and $P_{ij} = 0$ otherwise. It is the transition matrix corresponding to a simple

random walk on the graph G.

In their paper, Qiu et al. gave a theoretical understanding of the DeepWalk algorithm by proving the following theorem:

Theorem 3.3 (DeepWalk as implicit matrix factorisation)**.** *As $t \to \infty$, DeepWalk is equivalent to factorising*

$$\log\left(\frac{2|E|}{w}\left(\sum_{r=1}^{w} P^r\right)D^{-1}\right) - \log b$$

where b is the negative sampling rate.

The theorem assumes that the graph is undirected and that it is connected so that P is irreducible. The theorem also assumes that the graph is non-bipartite to ensure that the random walk converges to it's invariant distribution. In the application to social network graphs, this assumption will almost certainly hold as the existence of an odd cycle is expected (Social networks usually contain a large amount of triangles, representing mutual friends or connections). It is possible however that the graph is not connected, in this case a dummy node can be introduced which contains edges to all nodes which will not affect the community structure, provided the graph sub-communities are sufficiently dense. What follows is a careful outline of this proof.

*** If want to include details on bipartite asssumption, see the folder for DeepWalk for more information. ***

Then $\pi_i = \frac{d_i}{2|E|}$ satisfies the detailed balance equations:

$$\pi_i P_{ij} = d_i \cdot \frac{1}{d_i} = d_j \cdot \frac{1}{d_j} = \pi_j P_{ji}$$

and $\sum_{v_i \in V} \pi_i = 1$.Thus π defines a distribution which is invariant for the simple random walk on the graph. Since the state space is finite and by assumption, P is irreducible, a random walk on G defines an irreducible Markov Chain X with transition matrix P. Therefore π is unique by the following theorem

Theorem 3.4. *Consider an irreducible Markov chain. Then*

(i) There exists an invariant distribution if and only if some state is positive recurrent.

(ii) If there is an invariant distribution π, then every state is positive recurrent, and

$$\pi_i = \frac{1}{\mu_i}$$

for $i \in S$, where μ_i is the mean recurrence time of i. In particular, π is unique.

This is a standard proof in any course on Markov Chains and so the proof is omitted here. See Page 31 of [10] for details of a proof.

To proceed with the first important preliminary lemma, one can proceed by partitioning the random-walk corpus as follows.

Definition. For $r = 1, \dots, t$, we define the following

$$\mathcal{D}_{\overrightarrow{r}} = \{(v, c) : (v, c) \in \mathcal{D}, v = v_j, c = v_{j+r}\}$$

$$\mathcal{D}_{\overleftarrow{r}} = \{(v, c) : (v, c) \in \mathcal{D}, v = v_{j+r}, c = v_j\}$$

*** Need to give a proper definition to clarify what I mean here *** Thus $\mathcal{D}_{\overrightarrow{r}}/\mathcal{D}_{\overleftarrow{r}}$ are sub-multisets of $\mathcal{D}$ such that the context c is r steps after or before the vertex v in random walks respectively.

As an extension of previous definitions, we let $\#(v, c)_{\overrightarrow{r}}$ and $\#(v, c)_{\overleftarrow{r}}$ denote the number of times that an input-context pair (v, c) appears in $\mathcal{D}_{\overrightarrow{r}}$ and $\mathcal{D}_{\overleftarrow{r}}$ respectively. Then the following lemma holds

Lemma 3.5. *As $t \to \infty$, we have*

$$\frac{\#(v, c)_{\overrightarrow{r}}}{|\mathcal{D}_{\overrightarrow{r}}|} \xrightarrow{p} \pi_v(P^r)_{v,c} \text{ and } \frac{\#(v, c)_{\overleftarrow{r}}}{|\mathcal{D}_{\overleftarrow{r}}|} \xrightarrow{p} \pi_v(P^r)_{v,c}$$

Proof. *** A proof of this can be found in Paper 2 but the proof is not nice, perhaps I can come up with an original proof of this using convergence of the random walk to it's invariant distribution.

Note that by reversibility the two statements should be provably equivalent and I have used detailed balance here to prove that the two limits given in Paper 2 are the same.

I will leave this to come back to for a nicer proof. ***

*** Note that Ref 3 gives info about as the length of the RWs becomes long, the singleton distribution of vertices will tend to the invariant distribution, referencing Modern Graph Theory, Bollobas *** □

From this we can show that

Lemma 3.6. *When $t \to \infty$, we have*

$$\frac{\#(w, c)}{|\mathcal{D}|} \xrightarrow{p} \frac{1}{w} \sum_{r=1}^{w} \pi_v(P^r)_{v,c}$$

Proof.

$$\frac{\#(v, c)}{|\mathcal{D}|} = \frac{\sum_{r=1}^{w}(\#(v, c)_{\overrightarrow{r}} + \#(v, c)_{\overleftarrow{r}})}{\sum_{r=1}^{w}(|\mathcal{D}_{\overrightarrow{r}}| + |\mathcal{D}_{\overleftarrow{r}}|)} = \frac{1}{2w} \sum_{r=1}^{w} \left(\frac{\#(v, c)}{|\mathcal{D}_{\overrightarrow{r}}|} + \frac{\#(v, c)}{|\mathcal{D}_{\overleftarrow{r}}|} \right) \tag{3}$$

$$\xrightarrow{p} \frac{1}{r} \sum_{r=1}^{w} \frac{d_v}{2|E|} (P^r)_{v,c} \tag{4}$$

where the second equality uses the fact that $|\mathcal{D}_{\overrightarrow{r}}| = |\mathcal{D}_{\overleftarrow{r}}| = \frac{|\mathcal{D}|}{2w}$ and the convergence comes from applying [Theorem XX], together with the continuous mapping theorem. □

This gives everything we need to prove the main theorem of this section, that allows us to understand the matrix that DeepWalk is implicitely factorising.

Theorem 3.3 (DeepWalk as implicit matrix factorisation)**.** *As $t \to \infty$, DeepWalk is equivalent to factorising*

$$\log\left(\frac{2|E|}{w}\left(\sum_{r=1}^{w} P^r\right)D^{-1}\right) - \log b$$

where b is the negative sampling rate.

Proof. Firstly, by summing the result of [Theorem XX] we get

$$\begin{aligned}\frac{\#(w)}{|\mathcal{D}|} &= \sum_{c\in V} \#(w,c) \\ &\xrightarrow{p} \sum_{c\in V} \frac{1}{w}\sum_{r=1}^{w} \pi_v (P^r)_{v,c} \\ \frac{\pi_v}{w}\sum_{r=1}^{w}\sum_{c\in V}(P^r)_{v,c} &= \frac{\pi_v}{w}\sum_{r=1}^{w} 1 = \pi_v\end{aligned}$$

since P^r is stochastic for any r.
Similarly, using the fact that π is in detailed balance with P, and thus the result of [Theorem XX] can be rewritten as

$$\frac{\#(v,c)}{|\mathcal{D}|} \xrightarrow{p} \frac{1}{w}\sum_{r=1}^{w} \pi_c (P^r)_{c,v}$$

it can be shown that $\frac{\#(c)}{|\mathcal{D}|} \xrightarrow{p} \pi_c$.
Using this and by applying the continuous mapping theorem

$$\begin{aligned}\frac{\#(v,c)\cdot|\mathcal{D}|}{\#(v)\cdot\#(c)} = \frac{\frac{\#(v,c)}{|\mathcal{D}|}}{\frac{\#(v)}{|\mathcal{D}|}\cdot\frac{\#(c)}{|\mathcal{D}|}} &\xrightarrow{p} \frac{\frac{1}{w}\sum_{r=1}^{w}\frac{d_v}{2|E|}(P^r)_{v,c}}{\frac{d_v}{2|E|}\cdot\frac{d_c}{2|E|}} \\ &= \frac{2|E|}{w}\sum_{r=1}^{w}(P^r)_{v,c}\frac{1}{d_c} = \frac{2|E|}{w}\left(\sum_{r=1}^{w}(P^r D^{-1})_{v,c}\right)\end{aligned}$$

where the last equality follows since D is diagonal.

From this, applying [Theorem XX] gives that as $t \to \infty$ DeepWalk is equivalent to factorising

$$\log\left(\frac{2|E|}{w}\left(\sum_{r=1}^{w} P^r\right)D^{-1}\right) - \log b$$

□

This proof gives a better understanding of what the DeepWalk algorithm is doing. There are many other good questions to ask about DeepWalk, for example whether the algorithm is garunteed to converge under stochastic gradient descent, but we will not explore this any further in this essay.

*** Here can talk about how long it takes to tend towards this, potentially talk briefly on convergence of stochastic gradient descent etc. Important point, how large does t need to be for this to be meaningful? ***

*** Can I try and use the above result to show that DeepWalk is garunteed to converge, in other words that the objective function is convex ***

4 DeepWalk Sucks

Whilst the original DeepWalk paper made a huge impact on the future development of social representation learning on graphs, the algorithm itself has a number of drawbacks that have been pointed out in subsequent papers. In this short section, I will detail some of the issues and criticism of the implementation of DeepWalk.

Firstly, as has briefly been mentioned, in the original paper Hierarchical Softmax was used to estimate the probabilities in the softmax layer of SkipGram. This is important, since to calculate the partition function (denominator of the softmax probability) would normally have complexity $O(|V|)$. Using Hierarchical softmax however, the nodes are asssigned to the leaves of a binary tree. This reduces the computational complexity to $O(\log |V|)$.

Negative Sampling on the other hand is developed ...

*** Talk about Node2Vec and improvements to DeepWalk etc. Talk about various shortcomings, just look at papers that came afterwards that shred DeepWalk. Point to the algorithms that improved it.

Plan: Pick 2 or 3 algorithms, state what they improved, link to the papers. ***

5 Dynamic DeepWalking

In this section we transition away from the static implementation of DeepWalk towards network embeddings for Dynamic datasets. In particular, the focus of the section is to introduce an adaptation of DeepWalk to dynamic datasets as proposed in the paper published by Sajjad et al.[2] in early 2019.

As stated in the motivation for the paper, most of the recently developed representation learning methods can only be applied to static graphs while many real-world graphs are constantly changing over time. Therefore, to apply these methods the graph must be reanalysed at regular snapshots in time. This is very inneficient since in a given epoch, it is unlikely that the graph structure will change dramatically. Sajjad et al. proposed a modified version of the DeepWalk algorithm that is better suited to dynamic graphs and utilises the fact that the graph structure is unlikely to change significantly to develop a much more efficient way of analyzing social representation on dynamic graphs. The computational complexity of the algorithm depends on the graph density and number of edges added per epoch, however in the case of social networks these are both low and so the alggorithm is well suited to this application.

Previous algorithms to be used on dynamic graphs have not effectively used what was learned about the graph in previous snapshots to efficiently calculate the node representations in the next snapshot of time. The problem of doing so can be split into two main tasks:

1. Generate a random walks corpus for a new snapshot of the graph based on a corpus from an older snapshot. In particular, we seek to find an efficent update algorithm that generates a set of random walks on the new snapshot from random walks on the old one, that is statistically indistinguishable from generating a set of random walks from scratch on the new snapshot of the graph.

2. Update the vertex representations incrementally so that the representations do not need to be generated for every time step.

5.1 Updating the Random Walk Corpus

As a first solution to the problem of generating new random walks for the random walk corpus an algorithm called Naïve Update [Algorithm XX] is introduced. This algorithm initialises random walks for the affected vertices and creates random walks of length l. Once this is complete, it updates the random walks W^t by replacing the old walks with the corresponding re-generated walks and adds the random walks initiated from new vertices.

However, the authors show by way of example that the Naïve update algorithm has biased empirical transition probabilities and that the resulting corpus of random walks is not statistically indistinguishable from generating a new random walk corpus at time $t + 1$. To solve this problem a more sophisticated algorithm must be introduced.

Algorithm 1 Naïve Update

procedure NAÏVEUPDATE($\mathcal{G}^{t+1}, W^t, \mathcal{V}^1_{\text{affected}}, r, l$)
  ▷ Initialise r new walks from each of the affected vertices in $\mathcal{V}^1_{\text{affected}}$
  $W \leftarrow initWalks(\text{V}^1_{\text{affected}}, r)$
  ▷ *** WHAT DOES THIS ACTUALLY DO? ***
  $W^{t+1} \leftarrow randomwalk(\text{W, l G}^{t+1})$
  ▷ Replace the old walks and add new ones
  $W^{t+1} \leftarrow update(\text{W}^t, W^{t+1})$
  return W^{t+1}
end procedure

*** If looking for something to put in, I can put in the example given in the paper which would take up approx 1 page, left it out in case my essay is already too long ***

5.1.1 Unbiased Update

Algorithm 2 Unbiased Update

procedure UNBIASEDUPDATE($\mathcal{G}^{t+1}, W^t, \mathcal{V}^1_{\text{affected}}, r, l$)
  ▷ Partition the affected vertices into new and existing ones
  $\mathcal{V}_n \leftarrow newVertices(\text{V}^1_{\text{affected}})$
  $V_e \leftarrow existingVertices(\text{V}^1_{\text{affected}})$
  ▷ Filter the existing random walk corpus to get only the walks that contain affected vertices (in $\mathcal{V}_e$)
  $W_{\text{affected}} \leftarrow filter(\text{W}^t, \mathcal{V}_e)$
  ▷ Trim the affected walks to the first affected vertex
  $W_e \leftarrow trim(\text{W}_{\text{affected}}, \mathcal{V}_e)$
  $\triangleright \leftarrow Initialiserwalksfromeachofthenewverticesin\text{V}_n$
  $W_n \leftarrow initWalks(\text{V}_n, r)$
  ▷ Take the union of the new walks to create the updated corpus
  $W \leftarrow \text{W}_e \cup W_n$
  $W^{t+1} \leftarrow randomwalk(\text{W, l, G}^{t+1})$
  $W^{t+1} \leftarrow update(\text{W}^t, W^{t+1})$
  return W^{t+1}
end procedure

*** In the procedures I need to be clear about what initWalks and randomwalk are doing ***

The motivation behind the unbiased update algorithm is the following: Consider a random walk that has arrived at a vertex v at any point along the walk, if this vertex is not in the set $\mathcal{V}^{t+1}_{\text{affected}}$ then there is no change to it's neighbours and thus the choice for the next vertex in the random walk remains the same. However if $v \in \mathcal{V}^{t+1}_{\text{affected}}$ then the neighbours of the vertex have changed which makes the random walk biased from this vertex onwards since it evolves with incorrect transition probabilities. Therefore only the affected walks need to be updates to retain a statistically satisfactory set of random walks at time $t+1$.

The Unbiased Update algorithm updates only those walks that contain vertices in $\mathcal{V}^{t+1}_{\text{affected}}$ and does so by re-sampling the affected walks from the first affected vertex in the walk. Searching for the affected vertices in each of the affected walks is computationally expensive so another algorithm, Fast Update, is also suggested that generates the affected walks from the begginning instead of trimming them. Whilst computationally more efficient, this generates a biased random walk corpus. This is because by applying the Fast Update routine, we have effectively generated random walks on the graph $\mathcal{G}^{t+1}$ and then regenerated all walks that hit an affected vertex, which is clearly biased towards walks that do not visit the affected vertices.

In practice, when implementing the Unbiased Update algorithm the random seed that is used to generate each of the random walks can be stored and then used to regenerate the random walks up to their first affected vertex by reusing the seed. Therefore the Unbiased Update algorithm can be implemented with the same complexity as the Fast Update algorithm.

*** Can include the complexity analysis if need be but it is not particularly informative ***

Updating the vertex Representations Efficiently

The

6 Discussion of an application

The aim of this section is to bring the theory discussed on Dynamic DeepWalking into practice.

References

[1] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," *CoRR*, vol. abs/1710.02971, 2017.

[2] H. P. Sajjad, A. Docherty, and Y. Tyshetskiy, "Efficient representation learning using random walks for dynamic graphs," *CoRR*, vol. abs/1901.01346, 2019.

[3] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 03 2014.

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

[5] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[6] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," *Advances in Neural Information Processing Systems*, vol. 3, pp. 2177–2185, 01 2014.

[7] K. W. Church and P. Hanks, "Word association norms, mutual information, and lexicography," *Comput. Linguist.*, vol. 16, pp. 22–29, Mar. 1990.

[8] C. Yang and Z. Liu, "Comprehend deepwalk as matrix factorization," *CoRR*, vol. abs/1501.00358, 2015.

[9] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pp. 2111–2117, AAAI Press, 2015.

[10] D. Chua, "Part ib - markov chains."