NLP - Graph Embedding

A Brief Introduction into GE Models utilizing Random Walk

Supervisor: M.Sc. Matthias Assenmacher

Noah Hurmer

tbd

Abstract

TODO

Contents

1 Introduction **1**
1.1 Orginization . . . 1
1.2 Data structure . . . 1
1.3 Graph types . . . 1

2 Embedding **2**
2.1 Motivation . . . 2
2.2 Types and Applications . . . 2
2.3 Techniques . . . 3
2.3.1 Matrix Factorization . . . 3
2.3.2 Deep Learning . . . 4
2.3.3 Others . . . 4
2.4 Performance Evaluation . . . 4

3 node2vec **4**
3.1 Random Walk . . . 4
3.1.1 RW in node2vec . . . 5
3.2 SkipGram . . . 6
3.2.1 Negative Sampling . . . 7
3.3 Extensions and Applications . . . 7

4 Outlook into other RW models **7**

5 References **7**

List of Tables

List of Figures

1 Random example Graph 6

List of Abbreviations

Term	Abbreviation
DL	Deep Learning
GE	Graph Embedding
RW	Random Walk

1 Introduction

Dont really know what to put here yet tbh.

1.1 Orginization

TODO write out

- written table of context
- section 1 describes graphs their types and applications
- section 2 embedding motivation, types and associated downstream tasks
- section 3 shows the use of a Random Walk in order to use a DL model to embed with the example of node2vec
- section 4 tries to tie back into NLP, showing RW principles at work in order to perform cQA related tasks

1.2 Data structure

Graphs are a type of data structure that consist of so called nodes (or vertices) and edges. Therefore, a graph G is usually noted as $G = (V, E)$. Here, V is the set of nodes and E the set of edges. source
Nodes typically represent some form of object or entity. This can range from a physical object, component or location to theoretical concepts. The data form of a node is therefore generally not limited to any specific form, and can be anything from text to images, videos, lists and many more. However, most embedding and downstream tasks only deal with a label per node, as the data it represents is often not necessarily required.
Edges describe the relation between a pair of nodes, which can equivalently portray physical properties such as proximity and connectivity or other links such as effects, associations or relationships. An apparent example of these abstract or complex concepts a graph can embody is a social media network. In such a graph, people or their representative profiles would be the nodes and friendships, follows, comments, likes, etc. the edges.

1.3 Graph types

These pairwise relations of a graph can also be directed in what is known as a directed or directional Graph, where the given relation only applies from one node to the other, yet not (necessarily) the other way around. The relation is therefore not symmetric and also generally does not need to be reflexive either. For instance in a social media network, the edges may be a 'following' status or something equivalent. This relation between two people is directional.
In addition, an edge between two nodes can be weighted, where it receives a numerical label. This can represent concepts such as a cost of traversal, distance or capacity. Generally, it gives the relation of the graph a (metric) scale. This type of graph is then called a weighted Graph. A simple example of this might be a road network. Here nodes can represent intersections or cities,

dependent on the scale and edges connecting roads or highways. The roads can then be attributed with a time cost to use them. sources

Graphs may also be of heterogeneous mode, meaning different types or classes of nodes as well as edges can exist in a single graph. This is useful to embody networks of different interactions or effects from objects to one another. A very basic example of a heterogeneous graph is a simple semantic graph, which represents semantic relationships such as "is a" or "has" and so forth. sources, maybe example graph plot

To return to the example of social networks however, these are also often heterogeneous. An example we will return to later is a so-called cQA or community Question and Answer forum. These typically have Objects of the classes "Question," "Answer" and "Profile," as well as edges such as "asked," "answered," "follows," "upvoted" and possibly more. source

2 Embedding

Embedding refers to the representation of an object by another object. Typically, this representation is a two-dimensional vector.

Dunno bout this part.

2.1 Motivation

Two of the main issues with data in the form of a graph are its inherent structure and the limited applicable mathematics available to deal with that. Additionally, the computational challenge associated with any type of storage or calculation performed on it poses a problem.

The above described composition of a graph is usually stored in a so called adjacency matrix, with the dimensions of $N \times N$, where N is the number of nodes in the graph. The edges are then captured with a binary indicator (or a value for a weighted edge), whether or not two nodes are connected via an edge. The sheer size of such matrices can quickly become a problem both in form of (dynamic) storage space but also computational expense.

Therefore we aim to compress the information of a graph down to lower dimensions and into a form that lets us better apply analysis tools. Usually a two dimensional vector space is selected as the embedding dimension. This projects the graph into an euclidean space and enables the application of a distance metric between objects in that space, enabling downstream tasks that rely on this.

Moreover, as long as the graph does not change, once a graph has been embedded, multiple tasks and calculations can be performed without the need to embed anew, resulting in a performance gain.

source

2.2 Types and Applications

There are different forms of graph embedding, each with their own uses and specific tasks. These are usually coupled with the structure that is preserved.

Entire graphs can be embedded into a low dimensional vector space in order to compare different graphs to each other. Similar graphs are embedded close to each other. This can for example be useful for biologists to compare proteins or predict their functional labels. Here, a complex protein can be represented structurally by a single graph which will then be embedded to a single vector. source

The most common way of graph embedding is to embed the nodes of a graph. Here, each node is represented as a vector in a low dimensional space, where embeddings of similar or close nodes are located close to each other. This proximity can be defined in different ways. A typical metric for this is a proximity of order n, which describes the similarity of the n-order neighbourhoods of the respective nodes.

Node embedding can then be used for various downstream tasks such as node classification or clustering, in which one aims to group, categorize or label the objects of a network. Data compression or dimensionality reduction and visualization are other applications for node embedded graphs. source

A different approach is to embed the edges between nodes. The so called edge- or knowledge embedding aims to preserve relations between node pairs. This is particularly useful to predict missing or validate existing links in a graph, in tern to possibly predict missing relations between entities. Typically, this is done by embedding the tupel $< a, b >$ of nodes a and b respectively, or the triple $< a, t, b >$ if the graph is heterogeneous, where t represents the edge between the two nodes. Applications of link predictions range from suggestions on social media sites to predicting interactions of proteins or drugs. sources

It is also possible to embed subgraphs or groups (communities) of a graph separately. This technique mostly finds its uses in community detection of a network, question answering and semantic searches. Often it is paired with a form of node embedding. multiple sources here

2.3 Techniques

There are several different approaches used to embed a graph. As mentioned above, different tasks require different information to be preserved about the graph. So mostly, a given model is designed for a specific task or goal and thus the type of input graph it can accept is often limited. missing something here

Embedding models are generally summarized into the following categories.

2.3.1 Matrix Factorization

This technique mostly represents the beginnings of GE and has the disadvantage of high computational and storage cost, as a Matrix of pairwise node similarity is constructed, that is then factorized to obtain the embedding of each node. explain this more

Due to the fact that all node-pairs are considered (unlike other techniques), it can be quite performant, however the cost disadvantages make this largely infeasible for larger graphs.

2.3.2 Deep Learning

The use of DL models in GE has recently become more popular, as they usually carry the promise of great efficiency and performance. Simply put, existing (or purpose-built) DL models are applied in order to embed information into vector form. These models typically use Autoencoder or CNN based methods. (Cai, Zheng, and Chang (2018)) However, the input to such models can also be paths sampled from a graph. These sampled paths are strings of nodes visited during a sampling strategy. This method is called Random Walk and can be seen as its own subcategory of DL embedding models. (Goyal and Ferrara (2018))

2.3.3 Others

While other methods exists, there are considerably less applications of such and tend to be summarized as "Other Methods" or similar. (Goyal and Ferrara (2018)) Cai, Zheng, and Chang (2018) however, divide them into three main categories: Edge Reconstruction, Graph Kernel based methods and Generative Models.

2.4 Performance Evaluation

TODO short section on micro and macro F1 scoring

3 node2vec

node2vec (Grover and Leskovec (2016)) is an example of such a Random Walk based DL approach to Graph Embedding. It usually represents nodes of a Graph in the feature space and is therefore a Node embedding model.

3.1 Random Walk

The Random Walk principle is to sample paths from a graph in order to estimate the proximity of a node pair.
Similar to *DeepWalk* (Perozzi, Al-Rfou, and Skiena (2014)), the *node2vec* algorithm has its origins in Natural Language Processing, as it adapts the idea of *Word2vec* (Mikolov et al. (2013)). In NLP, tasks include to predict the surrounding words in a context window given a specific word. Or put simpler, prediction of context given a word. These context widows, often part-sentences or whole sentences, are input into a DL Model called *skipgram*. From this, a vector representation of a word is gained or a so called word-embedding. todo source for skipgram
In *node2vec* and other RW based Node embedding models, artificial "sentence"-equivalents are created by sampling paths or walks within a graph. These paths are our context windows for the *skipgram* model and can be directly input as such.
Each node is used as a starting point, from which neighbouring nodes are sampled in succession to obtain a path of desired length l. The probability to reach a node x from a previous node v is

therefore given by the number of edges at the node v and is 0 if x and v are not directly connected by an edge. More generally the probabilities are given by

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{v,x}}{Z}, & \text{if } (v,x) \in E \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where c_i denotes the ith node of a RW, $\pi_{v,x}$ the transition probability from node v to node x and Z a nomalizing constant. In the basic RW form, the transition probability is 1 if the graph is unweighted and Z is the number of edges originating from node v. (Grover and Leskovec (2016)) In comparison to other strict searching algorithms to create neighbourhoods of nodes, there are clear spatial and time cost advantages of RW, as they can easily be parallelised and depending on the order and length of the RW do not need a lot of memory capacity. In addition, this technique usually does a decent job of capturing the structure of a graph, as the power-law remains. In simple terms, a well connected node will be visited more often. (Perozzi, Al-Rfou, and Skiena (2014))

3.1.1 RW in node2vec

The *node2vec* model is a slight modification of a basic Random Walk. Here, two parameters are introduced to modify a RW. These modifiers, typically denoted p and q, add a bias to the sampling of nodes during the RW in order to control mostly the locality of the walk. This results in a mixture of BFS and DFS and can therefore represent different types of network neighbourhoods and node equivalences (Grover and Leskovec (2016)). The random walk is of 2^{nd} order, meaning that the previous node visited has an impact on the sampling of the next node of a RW traverse. In this case, the probability of a node x to become the next node in the RW $\{..., t, v\}$, is biased not only by the weight of the edge (v, x) ($w_{i,j} = 1; \forall i, j \in V$ if the graph is unweighted), but also by the distance to the previous node $d_{t,x}$, where

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p}, & \text{if } d_{t,x} = 0 \\ 1, & \text{if } d_{t,x} = 1 \\ \frac{1}{q}, & \text{if } d_{t,x} = 2 \end{cases}$$

This bias α is then multiplied by the weight of the edge $w_{v,x}$ to obtain $\pi_{v,x}$ of equation 1. p, also called the Return Parameter (Grover and Leskovec (2016)), defines the probaility of backtracing a step, while q, also called the In-Out Parameter (Grover and Leskovec (2016)), controls the exploratory nature of the walk. Setting p low and q high therefore keeps the walk very local, whilst a high p almost acts like a small self-avoiding-walk (Madras and Slade (2013)) and a low q ensures that the walk travels further outward from the starting node.

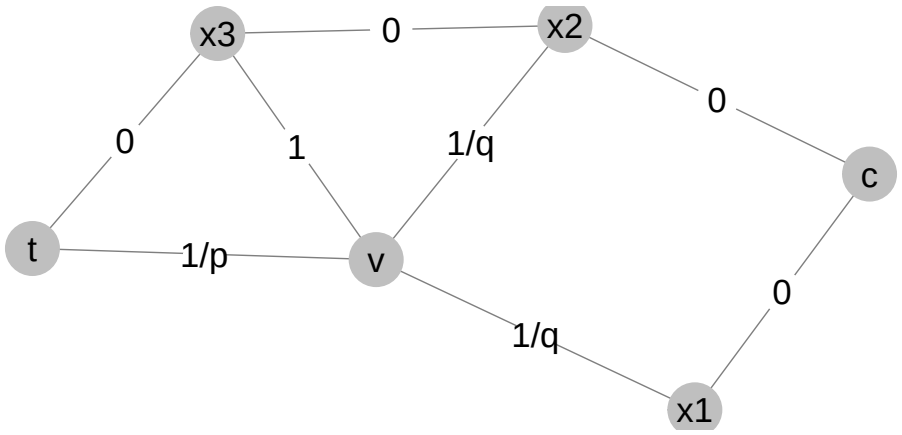

Figure 1: *Random example Graph*

3.2 SkipGram

node2vec like many other RW-based DL embedding methods, utilizes a variation of skipGram as the DL model (Cai, Zheng, and Chang (2018)). SkipGram originates in NLP with word embeddings. The model attempts to predict surrounding words of a certain range in a sentence, given a current word. The current word is used as an input to a log-linear classifier. Projecting this method to the task of graph prediction, a RW is to be interpreted as a sentence and a node as a word. SkipGram can then be applied on the sampled paths in order to maximize the probability to oberserve a neighbourhood of a node v, based on the feature representation of that node. So, let $N(u)$ be the neighborhood of the node u and f the projecting function to our feature space: $f : V \to \mathbb{R}^d$, where d denotes the number of dimensions of the desired feature space. The objective function results in

$$\max_f \sum_{u \in V} log(P(N(u)|f(u))) \tag{2}$$

To simplify, we assume conditional indepence in a neighbourhood so that the likelihood to observe a neighbourhood can be defined by factorizing over the likelihoods to observe each neighbourhood node.

$$log(P(N(u)|f(u))) = \prod_{n_i \in N(u)} P(n_i|f(u))$$

Also, we define the conditional likelihood between two nodes as a softmax equation.

$$P(n_i|f(u)) = \frac{exp(f(ni) * f(u))}{\sum_{v \in V} exp(f(v) * f(u))}$$

These assumptions transform the objective function 2 to

$$\max_f \sum_{u \in V} -log\left(\sum_{v \in V} exp(f(v) * f(u))\right) + \sum_{n_i \in N(u)} f(n_i) * f(u)$$

dunno bout this following paragraph The node-neighbourhoods in *node2vec* are defined by the Random Walks, so one can observe, how the walk defining parameters p and q directly effect the embedding, as nodes with similar neighbourhoods will end up with similar feature representations.

3.2.1 Negative Sampling

TODO

However, the first sum, also called the per-node partition function $\sum_{v \in V} exp(f(v) * f(u))$ is too expense to compute directly, so *node2vec* utilizes negative sampling to estimate this function.

3.3 Extensions and Applications

TODO

Grover and Leskovec (2016) have also shown, how when adapting the *node2vec* model to represent a pair of nodes (an edge) in the feature space, it then also lends itself to tasks such as link prediction.
LSTM node2vec combination
Walklets
RW with skips

4 Outlook into other RW models

TODO

- elaborate on cQA
- introduce LSTM
- explain tasks associated

5 References

Alicia Frame, PhD. 2019. "Graph Embeddings." 2019. https://www.youtube.com/watch?v=oQPCxwmBiWo&ab_channel=Neo4j.

Cai, HongYun, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2018. "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications." *IEEE Transactions on Knowledge and Data Engineering* 30 (9): 1616–37. https://doi.org/10.1109/TKDE.2018.2807452.

Cohen, Elior. 2018. "Node2vec: Embeddings for Graph Data." *Towardsdatascience*. https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef.

Godec, Primož. 2018. "Graph Embeddings — the Summary." https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007.

Goyal, Palash, and Emilio Ferrara. 2018. "Graph Embedding Techniques, Applications, and Performance: A Survey." *Knowledge-Based Systems* 151: 78–94. https://doi.org/https://doi.org/10.1016/j.knosys.2018.03.022.

Grover, Aditya, and Jure Leskovec. 2016. "Node2vec: Scalable Feature Learning for Networks." *CoRR* abs/1607.00653. http://arxiv.org/abs/1607.00653.

Hong, Shanon. 2020. "An Introduction to Graph Neural Network(GNN) for Analysing Structured Data." https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc.

Khoshraftar, Shima, Sedigheh Mahdavi, Aijun An, Yonggang Hu, and Junfeng Liu. 2019. "Dynamic Graph Embedding via LSTM History Tracking." *CoRR* abs/1911.01551. http://arxiv.org/abs/1911.01551.

Madras, N., and G. Slade. 2013. *The Self-Avoiding Walk*. Probability and Its Applications. Birkhäuser Boston. https://books.google.de/books?id=JsoFCAAAQBAJ.

Mikolov, Tomás, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Efficient Estimation of Word Representations in Vector Space." In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, edited by Yoshua Bengio and Yann LeCun. http://arxiv.org/abs/1301.3781.

Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. 2014. "DeepWalk: Online Learning of Social Representations." *CoRR* abs/1403.6652. http://arxiv.org/abs/1403.6652.

Pilehvar, Mohammad Taher, and Jose Camacho-Collados. 2020. "Embeddings in Natural Language Processing: Theory and Advances in Vector Representations of Meaning." *Synthesis Lectures on Human Language Technologies* 13 (4): 1–175. https://doi.org/10.2200/S01057ED1V01Y202009HLT047.