LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

NLP - Graph Embedding

A Brief Introduction into GE Models utilizing Random Walk

Supervisor: M.Sc. Matthias Assenmacher

Noah Hurmer

tbd

Abstract

TODO

Contents

List of Tables

List of Figures

List of Abbreviations

Term	Abbreviation
DL	Deep Learning
GE	Graph Embedding
RW	Random Walk

1 Introduction

1.1 Data structure

1.2 Motivation

2 Embedding

2.1 Types and Applications

2.2 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 gereally summarized into the following categories.

2.2.1 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.

2.2.2 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))

2.2.3 Others

- Edge Reconstruction
- Graph Kernel
- Generative Model

2.3 Performance Evaluation

3 node2vec

3.1 Random Walk

One category of DL based graph embedding utilizes the Random Walk principle to sample paths from a graph in order to estimate the proximity of a node pair. 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.

3.2 Skipgram

3.3 Negative Sampling

3.4 Extensions and Applications

4 Outlook into other RW models

5 Introduction

5.1 What is a graph

Graphs are a type of data structure that consist of so called nodes (or vertices) and edges. Often times, the nodes describe entities and the edges the pairwise relation to one another. Therefore, a graph G is usually noted as $G = (V, E)$. Here, V is the set of nodes and E the set of edges.
In essence, this can be thought of as a network of entities. One of the simpler realizations to imagine is perhaps a social media platform. Where a person (or their profile) is represented as a node and an edge indicates that two people are befriended on the platform. As apparent in the above mentioned example, graphs can have edges that embody abstract concepts, that do not intuitively fit into any conventional coordinate system, as they cannot be mapped into an euclidean space. That being said, one can also imagine more conventional observed data of a large feature space as a graph, where the edges are weighted and represent a similarity between objects.

5.2 Graph types

The type of entity that a node represents can be in any form of data, be that words or text, images perhaps omit some examples as to not overcrowd or overcomplicate this passage.

5.3 Graph types

- What is a Graph?
- Nodes and edges

- Nodes are the entities
- Edges are the relations between nodes
- nodes can consist of all sort of Data, images, text, etc even mixed
- nodes can have additional attributes

5.3.1 Graph modes

- homogeneous, heterogeneous
- graphs can be directional, weighted, semantik, knowledge based

5.4 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 that, and the computational challenge associated with any type of storage or calculation performed on it.
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. Not only does this become a problem if the relationship the edge represents becomes more complex, but 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 vector space is selected as the embedding dimension.

5.5 Applications

As stated, graph embedding simply lets us efficiently work with the data type of a graph. So applications are just uses, where data is in the form of a graph.

- uses in social networks
- visualization
- Network compression
- Network Partitioning
- Node Classification
- Link Prediction
- fake news detection

5.6 Embedding types

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 a graph which will then be embedded to a single vector.

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.

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 or to predict missing relations between entities.

It is also possible to embed subgraphs or groups (communities) of a graph separately. This is often referred to as hybrid embedding, as it combines aspects of node- and edge embedding to support tasks such as clustering or semantic searches.

6 Problem and Dataset

introduce the Problem and the associated Dataset on which the algorithms below should be run on.

7 Techniques

There are several techniques that can be used to embed a graph, however these are mostly coupled with the task or goal, the type of input graph and each have their own disadvantages.

- 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. 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.

- Deep Learning

  Recently more popular is the use of DL models. Simply put, existing (or purpose-built) DL models are applied on the input

- Edge Reconstruction

- Graph Kernel

- Generative Model

Short Paragraph explaining how one Approach to (node)Graph embedding utilizes Deep Learning methods. List the other approaches not using DL with a brief explanation, and especially mention how the computational advantage of the Random Walk approach can be preferable to MF.

7.1 Random Walk

One category of DL based graph embedding utilizes the Random Walk principle to sample paths from a graph in order to estimate the proximity of a node pair. 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 parallelized and depending on the order and length of the RW do not need a lot of memmory 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))

7.1.1 Random Walk in node2vec

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$^{\text{nd}}$ 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.

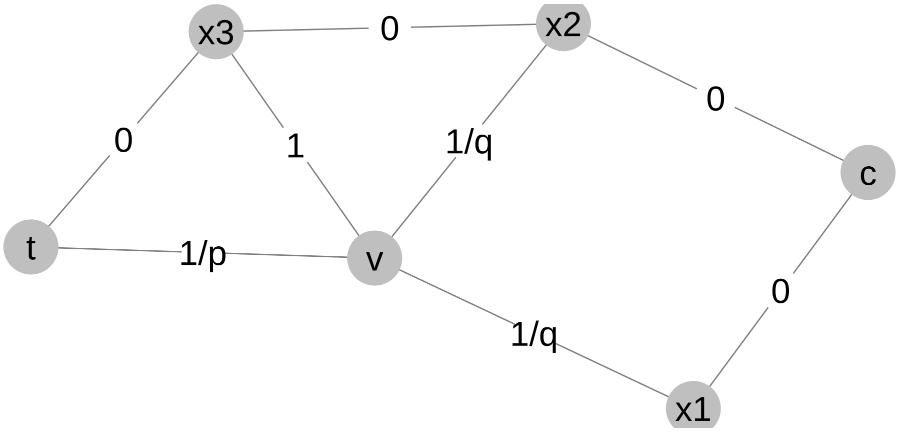

Figure 1: *Random example Graph*

7.2 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: $f : V \rightarrow \mathbb{R}^d$, 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

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

Maybe put this somewhere else
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.

7.2.1 Negative Sampling

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

7.3 (LSTM)

Optional Section considered as an excursion to another Random Walk method using a different Model and heterogeneous Graphs as Inputs. Perhaps using HSNL as the algorithm example.

8 Performance Evaluation

Discuss how to evaluate Graph Embedding methods and algorithms and then evaluate the above used ones.

9 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.

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.