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
BFS	Breadth First Search
cQA	Community-Based Question Answering
DFS	Depth First Search
DL	Deep Learning
GE	Graph Embedding
LSTM	Long short-term memory
NCE	Noise Contrastive Estimation
NEG	Negative Sampling
RNN	Recurrent Neural Network
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

The performance of a graph embedding model is typically measured by scoring the performance of downstream tasks with the gained embeddings. For node classification tasks, this is generally done with a *micro-F1* and *macro-F1* score. The *F1-score* is typically an accuracy measure for binary classification tasks, defined as the harmonic mean of precision and recall.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

macro-F1 is simply the multiclass extension of this, averaging over the F1 scores for each label. Whereas *micro-F1* calculates precision and recall globally, which results in the proportion of true positive predictions to all positive predictions. While this shows the global average prediction accuracy, as every node prediction is weighted equally, the *macro-F1* score can be useful especially for class imbalances. (Lipton, Elkan, and Narayanaswamy (2014))
Goyal and Ferrara (2018) also propose to use Precision at k (Pr@k) as well as Mean average precision (MAP) for graph reconstruction and link prediction tasks.
Also, dimensionality reduction for tasks such as visualizing a network structure may be of interest, when applying graph embedding models. As performance of this cannot be measured or denoted by a numeric score, it often depends on what is the expected result or structure to be visualized. Scalability as well as required computational performance may additionally be interesting measures with which an embedding model is selected, as these may be required or limited respectively.

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 Word2vec summary

Similar to *DeepWalk* (Perozzi, Al-Rfou, and Skiena (2014)), *node2vec* has its origins in Natural Language Processing, as it adapts the idea of *Word2vec* (Mikolov, Chen, et al. (2013)). *Word2vec* is a NLP model for learning word embeddings. It does so by inputing sentences of a text corpus into a DL Model called *skipgram*, with the fake task of attempting to predict the surrounding words in a context window, given a specific word (the word in the middle of this context window). The size of a context window has to be defined but it is simply the n words surrounding the current target word in a sentence.
As an example, this will be shown with the following sentence:

'Adopting a loudly barking dog as a pet is a good strategy to annoy the neighbours.'

If we take 'dog' as the current target word and the context window size to be 4 (or ± 2) and ignore prepositions (&etc.), the context window of 'dog' consist of the words 'loud(ly),' 'bark(ing)' and 'pet.' The desired result is therefore to predict those words as context when given the word 'dog,' resulting in the embedding vectors of those context window words to be similar to that of 'dog.'

3.2 Random Walk

Adapting this *skipgram* architecture for use in node embedding poses a specific problem. In Text, there is a clear linearity and the context window is defined to be the words surrounding the current word. A network however, does not posses such a linear nature. Therefore, a context window equivalent has to be created.
This is done by replacing sentences of a text corpus with node sequences sampled from a graph. The Random Walk principle is to sample these paths traversing a graph, in order to receive a linear sequence of visited nodes. Now the same context window concept can be used, defined as a neighbourhood $N_S(u)$, where u shall be the current target node. The task of the model is then to predict the neighbourhood of a node, given this node.
To create Random Walks, 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 normalizing 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 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))

3.2.1 RW in node2vec

Similar to other RW based node embedding models, *node2vec* uses these artificial "sentence"-equivalents of paths as an input to the *skipgram* model. However, the *node2vec* RW 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, and control thereof, 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, defines the probability of backtracking a step, while q, also called the In-Out Parameter, controls the exploratory nature of the walk. (Grover and Leskovec (2016))
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. As mentioned above, choosing these parameters leads to a tuned trade-off between BFS and DFS. This behaviour can be exploited in order to represent either node proximity or structural node proximity, depending on what is desired for a downstream task. (Grover and Leskovec (2016))
Example Graph 1 shows an example of the RW bias α. Consider t to be the node last visited and v the current node. The possible next nodes in this RW are therefore $t, x1, x2$ and $x3$ with their respective bias α displayed on the edge.

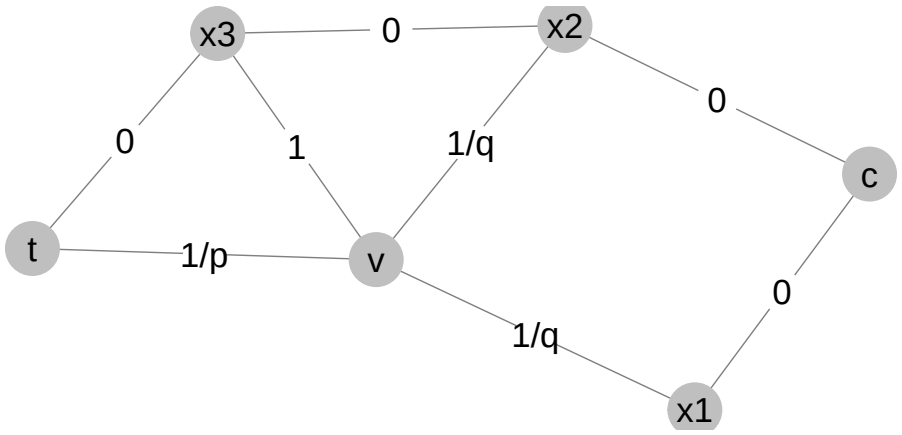

Figure 1: *Example Graph to show the pobabilty bias of the node2vec RW*

3.3 SkipGram

node2vec like some other RW-based DL embedding methods, utilizes a variation of *skipgram* as the DL model (Cai, Zheng, and Chang (2018)). As mentioned, *skipgram* originates in NLP and is used in *Word2vec* to obtain word embeddings. The model attempts to predict surrounding words of a certain range in a sentence, given a current word. In this model the hidden layer is comprised of vector representations for each of the words, so called word embeddings. These are created, because, in order to predict the context of a word, it attempts to represent words in the so called context window of a target word similarly to this specific word, resulting in a similar feature representations of those words. It does so, by maximizing the log probability to observe feature representations of words in the context window, given the feature representation of a word in what is hereinafter referred to as the *skipgram* objective function.

Projecting this method onto the task of node embedding, 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 neighbourhood 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 adapted *skipgram* objective function then results in

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

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

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

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

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

These assumptions transform the objective function 2 into

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

3.3.1 Negative Sampling

However, the first sum of the objective function of the skipgram model, also called the per-node partition function by Grover and Leskovec (2016), $\sum_{v \in V} exp(f(v) * f(u))$, is very expense to compute directly, as it necessitates calculating a $|V|$-way softmax equation every iteration. In addition, even though all weights are recalculated every step, most weights will not be altered by a relevant amount. Mikolov, Sutskever, et al. (2013) propose two different methods to optimize this for *Word2vec*. The first being a hierarchical softmax, which utilizes the structure of a binary tree to cut the amount of evaluations per iteration from $|V|$ to approximately $log_2(|V|)$.

While *DeepWalk* uses this method of softmax simplification, *node2vec* implements the second suggested method, called negative sampling (NEG), which is a simplification of Noise Contrastive Estimation (NCE). (Gutmann and Hyvärinen (2012), Mnih and Teh (2012))

NCE proposes, that, using logistic regression, data should be separable from noise. (Mikolov, Sutskever, et al. (2013)) NEG adapts this idea by constructing input data for a logistic regression, combining a context object (expected output of regression = 1) with k sampled negative objects (expected output of regression = 0). In the case of word embedding, objects here refer to words and the context word would be words of the context window for a target word, while negative words any other words in the text corpus. This easily translates into node embedding with RW, as the context nodes are simply the next nodes (dependent on context window size) of the current path.

example

Not only does this transform a $|V|$-way softmax equation into $|V|$ number of logistic regressions, which are inherently more efficient, but by limiting the k number of negative sample nodes, we limit the number of weights revalued and adjusted, drastically reducing the complexity even further. Mikolov, Sutskever, et al. (2013) suggests a k of 5-20 negative samples for smaller datasets and that 2-5 negative samples can be sufficient for larger datasets. The original python *node2vec* implementation by Grover and Leskovec (2016) keeps the default setting of $k = 5$ from the *Word2vec* implementation of the *Gensim* python library. (Řehůřek and Sojka (2010))

A free parameter of NEG is the distribution from which the negative samples are pulled, also named the noise distribution $P_n(v)$. According to Mikolov, Sutskever, et al. (2013), a Unigram distribution in which the proportional frequency of a word in the corpus (here the proportional frequency of a node appearing in all of the random walks) is considered, is recommended. Specifically, they say that a Unigram distribution raised to $\frac{3}{4}$ths power performs the best. This way, less frequent words

tend to be sampled more often. Again, this is the default for the *Word2vec* implementation used and therefore also the noise distibution used in the *node2vec* implementation.
So the probability of sampling node u is described by $P_n(u)$, where:

$$P_n(u) = \frac{f(u)^{\frac{3}{4}}}{\sum_{j=0}^{|V|} \left(f(v_j)^{\frac{3}{4}}\right)}$$

and $f(x)$ equals the number of appearances of node x in all of the random walks.
Applying NEG to the created Random Walks by *node2vec*, the new objective is therefore to maximize

$$\log \sum_{n_i \in N_S(u)} \sigma(f(n_i) * f(u)) + \sum_{j=1}^{k} \mathbb{E}_{v_j \sim P_n(v)} \left(\sigma(-f(v_j) * f(u))\right) \tag{3}$$

for every node u, with $\sigma(x) = \frac{1}{1+e^{-x}}$. Or in words, attempt to have the logistic regression return 1 for all $(f(n_i) * f(u))$ and 0 for all $(-f(v_j) * f(u))$, where v_j are the negative sampled nodes for u and n_i a neighbourhood node of u. That is to say, we optimize the embedding function f such that the feature representation of a given node is as similar as possible to those of its neighbourhood nodes, whilst those of the negative sampled nodes are as dissimilar as possible.
Generalizing equation 3 to the general objective function of the model results in:

$$\max_f \left[\sum_{u \in V} \left(\log \sum_{n_i \in N_S(u)} \sigma(f(n_i) * f(u)) + \sum_{j=1}^{k} \mathbb{E}_{v_j \sim P_n(v)} \left(\sigma(-f(v_j) * f(u))\right)\right)\right] \tag{4}$$

Parameters θ for embedding function f are then optimized using gradient ascent. (Grover and Leskovec (2016))

3.4 Extensions and Applications

There have been multiple proposals of *node2vec* extensions and add-ons for various applications. A few examples of this will be briefly named in the following.
Grover and Leskovec (2016) themselves have shown, how, when adapting the *node2vec* model to represent a pair of nodes (an edge) in the feature space as one vector, the model then also lends itself to tasks such as link prediction.
node2vec has been demonstrated to be useful in predictive modelling, such as to predict demographic information as well as likelihood of a customer to churn in a call network. (Óskarsdóttir and Baesens (2016) and Mitrovic et al. (2017))
Wilson et al. (2020) develop a *multi-node2vec* algorithm, capable of embedding a multi-layered network in order to analyse fMRI scans in schizophrenia patients.
A dynamic extension of *node2vec*, named *LSTM-node2vec*, has also been proposed, where the *node2vec* model is augmented with an *LSTM* model to capture temporal changes and enable dynamic embedding. (Khoshraftar et al. (2019a))
Additionally, there are multiple proposed models, that utilize the RW principle. Perozzi et al.

(2017) for example present *Walklets*, an algorithm that skips over steps of Random Walks, which can be applied to the RW of *node2vec* as well.

4 Outlook into other RW models

TODO

- elaborate on cQA
- introduce LSTM
- explain tasks associated

5 Summary

6 References

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.

Gutmann, Michael U, and Aapo Hyvärinen. 2012. "Noise-Contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics." *Journal of Machine Learning Research* 13 (2).

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. 2019b. "Dynamic Graph Embedding via LSTM History Tracking." *CoRR* abs/1911.01551. http://arxiv.org/abs/1911.01551.

———. 2019a. "Dynamic Graph Embedding via LSTM History Tracking." http://arxiv.org/abs/1911.01551.

Lipton, Zachary Chase, Charles Elkan, and Balakrishnan Narayanaswamy. 2014. "Thresholding Classifiers to Maximize F1 Score." http://arxiv.org/abs/1402.1892.

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.

Mikolov, Tomás, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. "Distributed Representations of Words and Phrases and Their Compositionality." *CoRR* abs/1310.4546. http://arxiv.org/abs/1310.4546.

Mitrovic, Sandra, Bart Baesens, Wilfried Lemahieu, and Jochen Weerdt. 2017. "Churn Prediction Using Dynamic RFM-Augmented Node2vec." In. https://doi.org/10.1007/978-3-319-71970-2_11.

Mnih, Andriy, and Yee Whye Teh. 2012. "A Fast and Simple Algorithm for Training Neural Probabilistic Language Models." http://arxiv.org/abs/1206.6426.

Óskarsdóttir, María, and Bart Baesens. 2016. "Node2vec in Telco: An Application of the Novel Feature Learning Method for Predictions in Call Networks." In *DataMiningApps Newsletter*. https://www.dataminingapps.com/2016/10/node2vec-in-telco-an-application-of-the-novel-feature-learning-method-for-predictions-in-call-networks/.

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.

Perozzi, Bryan, Vivek Kulkarni, Haochen Chen, and Steven Skiena. 2017. "Don't Walk, Skip! Online Learning of Multi-Scale Network Embeddings." http://arxiv.org/abs/1605.02115.

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.

Řehůřek, Radim, and Petr Sojka. 2010. "Software Framework for Topic Modelling with Large Corpora." In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45–50. Valletta, Malta: ELRA.

Wilson, James D., Melanie Baybay, Rishi Sankar, Paul Stillman, and Abbie M. Popa. 2020. "Analysis of Population Functional Connectivity Data via Multilayer Network Embeddings." http://arxiv.org/abs/1809.06437.