Master Thesis

Gaining Customer Insights using Machine Learning on Graphs

University of Basel

Author:
Michael von Siebenthal

Supervisor:
Prof. Dr. Dietmar Maringer

May 25, 2021

Abstract

This is the abstract.

Declaration

"I hereby declare - that I have written this master thesis without any help from others and without the use of documents and aids other than those stated in the references, - that I have mentioned all the sources used and that I have cited them correctly according to the established academic citation rules, - that the topic or part of it are not already the object of any work or examination of another course unless explicitly stated,- that I am aware of the consequences of plagiarism at the Business and Economics Faculty of University of Basel."

Michael von Siebenthal, Martikel-Nr.: 2015-256-837, Date: May 25, 2021

Contents

List of Figures

List of Tables

1 Introduction

The aim of this thesis is to explore the relatively new field of machine learning on graphs. In particular, this thesis will focus on graph machine learning applications for the purpose of gaining customer insights. Graph machine learning is the current frontier in machine learning and has vast applications in many areas as recently shown by the success of AlphaFold (Senior et al. 2020). AlphaFold made a breakthrough for predicting protein structures where they made use of the observation that a folded protein can be considered as a spatial graph (AlphaFold 2020). In addition, there is a vast range of applications for graphs in the fields of natural science, social science and many more as shown by the excellent overview given by Zhou et al. (2020). In principle, graphs are useful whenever one wants to model interactions or relationships.

1.1 Relevance to Economics

From a business & economics perspective, graphs are particularly interesting if one wants to for instance model the interactions between institutions. An example for this is a study published by Schweitzer et al. (2009) which created a graph showing the interdependencies of international banks as a network, see figure 1. This is a useful representation of interdependencies and is an important basis for making systems more robust.

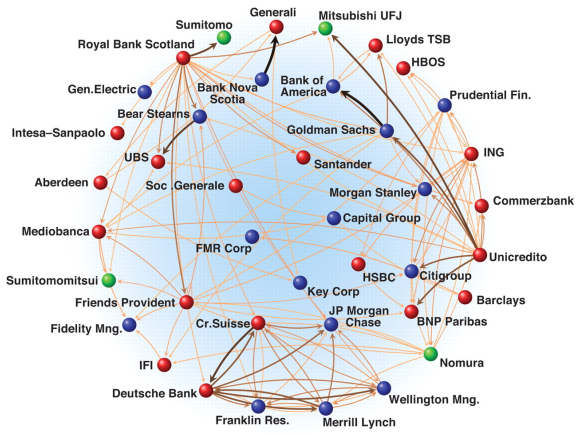

Figure 1: Bank Network
(Schweitzer et al. 2009, p. 424)

Another interesting application of graphs for business & economics are social interactions. While there are many different types of social interactions of interest,

social interactions for marketing purposes have been among the most popular. Indeed, this is one of the main areas where social networks such as Facebook or search providers such as Google make their revenue by selling advertising (Facebook 2021, Alphabet 2021). Both Facebook and Google have the advantage, that their businesses naturally capture relational or more generally network data which can be represented as graphs. Most researchers or companies however do not have access to such data. Companies for instance may have access to large amounts of customer data, however they typically would not have access to relational information (e.g. which client is connected with which other clients). The same is true for researchers, where social scientists often collect data via anonymous surveys. It is important to highlight, that there is a lot of social network data available online. This network data however typically only contains the network structure. The feature data of the people present in the network is however typically not provided. This is an issue in terms of data access.

1.2 Research Topic

Given the difficult access to graph data, this thesis will explore to what extent synthetic graphs can be generated using real-world cross-sectional data. An example for such data could be the client database of a company. The aim is then not only to generate graphs but to test whether the resulting graph can be used for meaningful machine learning tasks. In order to test this, appropriate datasets will be selected. It would of course be best, if one would have access to real graph data. If graph data had been available for an appropriate application, this would have been preferred for this thesis. The absence of such available graph data and the difficulty of generating / collecting real graph data sparked the interest and research topic for exploring synthetic graph generation for subsequent machine learning tasks. If this application proves to be successful, this could provide an alternative and hopefully successful approach for analyzing cross-sectional data. In order to assess, whether synthetic graphs based on real cross-sectional data is a viable approach, the results of graph machine learning methods will be compared to "standard" machine learning methods. This leads to the research question formally presented in the following section.

1.2.1 Research Question

The research question for this thesis is defined as follows:

To what extent are synthetic graphs based on real cross-sectional data useful for machine learning tasks?

In terms of machine learning task, this thesis will focus on a classification task. This ensures that the results of the graph machine learning methods versus the "standard" machine learning tasks can be compared.

1.3 Literature Review

An extensive literature review was conducted. To the best knowledge of the author, there are no other published studies which investigate the applicability of synthetic graphs based on real cross-sectional data for the purpose of machine learning. The literature review revealed the following related literature.

A large related research area is regarding synthetic graph generation. Classical models include the famous Erdös-Rényi graph generation model (Erdös & Rényi 2011), the small-world model by Watts and Strogatz (1998) or the model by Barabási & Albert (1999). More recent approaches include Kronecker Graphs (Leskovec et al. 2010) and its genralization Multiplicative Attribute Graphs (MAG) (Kim & Leskovec 2012). MAG work well in that it follows commonly observed network properties observed in real networks as outlined in the paper by Kim & Leskovec (2012). For the purpose of this thesis the MAG model will be used for transforming cross-sectional data into a graph. A more detailed introduction of the MAG model will be given in the theory section. Recently, synthetic graph generation models have shifted to deep generative models. Prominent models include GraphRNN by You et al. (2018) or the deep generative model presented by Li et al. (2018). These models differ to the traditional graph generation models in that they learn to create synthetic graphs based on real graph samples. These newer approaches could for instance be used for drug molecule discovery among many other approaches. These newer models are however not purposeful for this thesis.

2 Theory

This chapter will cover the necessary theoretical background for this master thesis and consists of following parts:

1. Graph Theory
2. Machine Learning on Graphs
3. Graph Generation

2.1 Graph Theory

This section provides a brief introduction to graph theory, with a focus on the relevant aspects for this master thesis. The theory presented is primarily taken from the book "Networks: An Introduction" by Mark Newman (2010).

Graph theory is an old field of mathematics and can be traced back to Leonhard Euler and the famous "Königsberg Bridge Problem" (Euler 1736). The study of graphs has had a recent revival thanks to its useful applications in machine learning. Graphs are special data structures as shown in Figure 2. The terms graph and network are often used interchangeably and have identical meaning for the purpose of this master thesis. Typically, the term graph is used more commonly when referring to mathematical analysis of graphs and the term network is more commonly used for data science purposes.

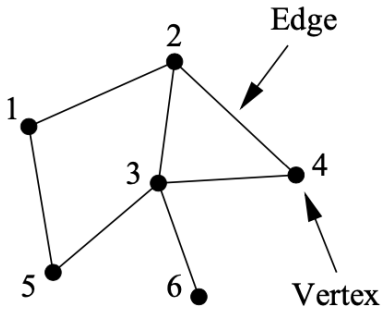

Figure 2: Example of a Graph
(Newman 2010, p. 111)

The graph shown in Figure 2 corresponds to an undirected graph in which the connections between the vertexes are mutual. In a directed graph for instance, vertex A

could be connected to vertex B, however vertex B need not be connected to vertex A. For the purpose of this thesis, only undirected graphs are considered. Vertexes are often referred to as nodes and the terms will be used interchangeably. Edges refer to the connections between the vertices. Edges are often also referred to as links and the terms will be used interchangeably as well. Graphs may have additional elements such as multi-edges or self-edges. Self-edges refer to nodes which have a looped link to itself. This can be considered as a feedback loop of a node on to itself. Lastly, multi-edges refer to direct node connections with multiple paths.

In terms of mathematical notation, graphs are typically defined as follows:

$$G(V, E) \tag{1}$$

G refers to the graph as an output. V refers to the set of vertices present in the graph and E refers to edges present between the vertices.

2.1.1 Adjacency Matrix

The adjacency matrix A is defined as a $n \times n$ matrix, where n refers to the number of vertices present in the graph. Each vertex is therefore recorded by a column and a row in the adjacency matrix. The elements in the adjacency matrix are further typically defined as follows:

$$A_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ and } j \text{ are connected by an edge} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

For illustration, the adjacency matrix of the graph shown in Figure 2 is shown as follows:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

As one can see, if vertex i and j are connected, this is recorded with 1 and 0 otherwise. Note, that all the elements on the $diag(A)$ are equal to 0. This is because there are no self-edges present in figure 2. Nodes with self-loops would have

a 1 recorded on the corresponding diagonal element of the adjacency matrix. As this is an undirected network, the adjacency matrix is symmetric. There are many additional aspects one could mention with regard to the adjacency matrix, they are however not relevant for this thesis. For additional information regarding the adjacency matrix, the book by Mark Newman (2010) is highly recommended.

2.1.2 Degree Measures

An important measure for graphs are the degrees denoted by k of the vertices. Degrees refer to the number of edges connected to a vertex. The degrees of vertex denoted by i can be formulated as (Newman 2010, p. 133):

$$k_i = \sum_{j=1}^{n} A_{ij} \tag{3}$$

For an undirected graph, edges have two ends. This is due to the fact that vertices connected by an edge are mutually connected. In terms of the sum of the degrees of all vertices, we can therefore write for a graph with m edges (Newman 2010, p. 133):

$$2m = \sum_{i=1}^{n} k_i \tag{4}$$

The sum of all degrees is therefore just the number of edges m multiplied by 2. In terms of statistical measures, the mean degree c of a vertex is defined as follows (Newman 2010, p. 134):

$$c = \frac{1}{n} \sum_{i=1}^{n} k_i = \frac{2m}{n} \tag{5}$$

In order to define the connectance or density of a graph, we must first observe, that the maximum number of edges is given by (Newman 2010, p. 134):

$$\binom{n}{2} = \frac{1}{2} n(n-1) \tag{6}$$

The density ρ can therefore be written as (Newman 2010, p. 134):

$$\rho = \frac{m}{\binom{n}{2}} = \frac{2m}{n(n-1)} = \frac{c}{n-1} \tag{7}$$

Note, that the density ρ lies strictly between $0 \leqslant \rho \leqslant 1$. In addition, for sufficiently large graphs, one can approximate $\rho = \frac{c}{n}$.

2.1.3 Eigenvector Centrality

The degrees of a vertex shown in the previous section already correspond to the simplest form of centrality measures. The issue with this measure however is, that the every neighbor of vertex i are valued the same. This is a problem, as not all neighbors are of equal importance due to:

1. Number of neighbors
2. Importance of neighbor
3. both

There are many different alternative centrality measures which can consider the factors listed above such as eigenvector centrality, Katz centrality or PageRank (Katz 1953, Page et al. 1999, Landau 1895, Newman 2010). As we are only dealing with simple undirected graphs, eigenvector centrality will suffice, where the other mentioned methods are adaptations to the eigenvector centrality.

Eigenvector centrality gives all vertices a score proportional to the sum of the scores of its neighbors. This is a procedure in which typically the initial centrality x_i of vertex i is guessed to be 1 $\forall$ i. This can be used to calculate the centralities of the neighbors of i which is denoted as x'_i. We can thus write (Newman 2010, p. 169):

$$x'_i = \sum_j A_{ij} x_j \tag{8}$$

In matrix form:

$$x' = Ax \tag{9}$$

This process is repeated t times to provide better estimates (Newman 2010, p. 170):

$$x(t) = A^t x(0) \tag{10}$$

Where $x(0)$ is a linear combination of (Newman 2010, p. 170):

$$x(0) = \sum_i c_i v_i \tag{11}$$

v_i correspond to the eigenvectors of the adjacency matrix A and c_i corresponds to an appropriately chosen constant. Therefore we can write (Newman 2010, p. 170):

$$x(t) = A^t \sum_i c_i v_i = \sum_i c_i k_i^t v_i = k_i^t \sum_i c_i \left[\frac{k_i}{k_1}\right]^t v_i \tag{12}$$

In the above equation, k_i correspond to the eigenvalues of A, the adjacency matrix. k_1 corresponds to the largest eigenvalue of A. As $\frac{k_i}{k_1} < 1, \ \forall \ i \neq 1$, the term is decaying as $t \to \infty$. The centralities x can therefore be written as fulfilling following condition (Newman 2010, p. 170):

$$Ax = k_1 x \tag{13}$$

Lastly, the eigenvector centrality is defined as (Newman 2010, p. 170):

$$x_i = k_1^{-1} \sum_j A_{ij} x_j \tag{14}$$

2.1.4 Closeness Centrality

Closeness centrality, C_i, is defined as the average distance from a vertex to the other vertices. This centrality measure is defined as follows (Newman 2010, p. 182):

$$C_i = \frac{1}{l_i} = \frac{n}{\sum_j d_{ij}} \tag{15}$$

In this measure, central vertices exhibit high closeness centrality and are therefore closer connected to other vertices compared to vertices with low closeness centrality. l_i refers to the average of the geodesic distances d_{ij} of vertex i.

2.1.5 Betweenness Centrality

This centrality measures to which extent a vertex lies on paths between other vertices. For instance, a bottle neck vertex would exhibit a large betweenness centrality as many, if not all nodes must pass through it. More formally, betweenness centrality, x_i, is defined as (Newman 2010, p. 187):

$$x_i = \sum_{st} \frac{\eta_{st}^i}{g_{st}} \tag{16}$$

In the above equation, η_{st}^i refers to the number of geodesic paths from s to t which pass through vertex i. Further, g_{st} is defined as the number of geodesic paths between vertex s and t.

In order to allow for better comparison of betweenness centrality, it is often standardized by the number of connected vertex pairs s and t denoted as η^2. The betweenness centrality can therefore be expressed as (Newman 2010, p.190):

$$x_i = \frac{1}{\eta^2} \sum_{st} \frac{\eta_{st}^i}{g_{st}} \tag{17}$$

With this measure, the betweenness centrality is within the range $0 \leqslant 1$

2.2 Machine Learning on Graphs

Graph structures are significantly different compared to traditional types of data usually used for machine learning tasks or regression analyses. Graph structures are special in that the data points in a graph have connections with each other. A practical example for this are social networks. In a social network, the profiles of "Peter" and "Paul" might be connected because Peter and Paul are friends. In addition, "Paul" and "Peter" can only ever reach each-other if they are directly or perhaps indirectly connected. This aspect is unique to graph or network data and provides both additional information and additional complexity to graph data. This property does not allow for comparing nodes in a graph in terms of euclidean distances as only connected nodes can reach each-other. For the purpose of this thesis, machine learning on graphs will be categorized into two categories:

1. Graph representation learning with subsequent downstream machine learning
2. Graph neural networks

In the following subsections, methods for both graph machine learning categories will be introduced.

2.2.1 Graph Representation Learning

The aim of graph representation learning is to embed nodes into a d-dimensional vector representation. The resulting vector embeddings can then be used for "standard" machine learning applications. A graphical representation of this task is shown in figure 3.

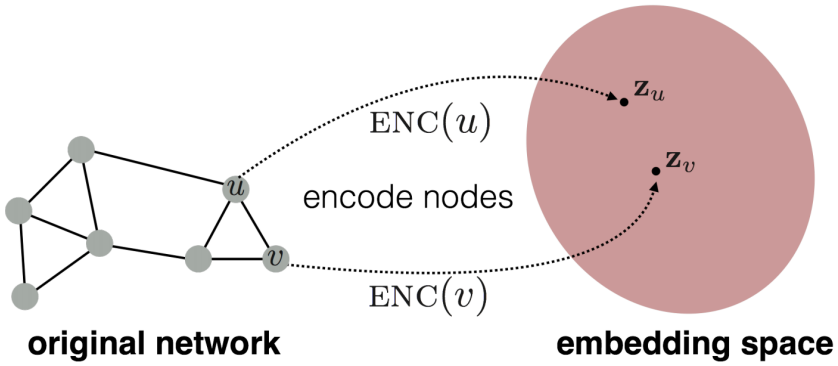

Figure 3: Network Embedding
Leskovec (2021)

Figure 3 outlines the task of finding an encoder such that nodes which are similar are also embedded close in the embedding space for which distance measures such as Euclidean distances can be calculated. A common measure for similarity is to find vector embeddings z of nodes u and node v such that (Leskovec 2021):

$$z_u^T z_v \approx similarity(u, v) \tag{18}$$

The dot product of two node vector embedding should thus approximately equal the similarity of the corresponding nodes in the network. There are different approaches for defining similarity of two nodes which can be applied. An early solution for instance was a graph factorization approach introduced by Ahmed et al. (2013). More recent and successful approaches include methods which make use of random walks. In this context, similarity is defined as follows (Leskovec 2021):

$$z_u^T z_v \approx \text{Prob. that node } u \text{ and } v \text{ co-occur on a random walk over the network} \tag{19}$$

This thesis will focus on the methods Deepwalk by Perozzi et al. (2014) and its generalization Node2Vec by Grover & Leskovec (2016). There are other noteworthy methods such as LINE by Tang et al. (2015) which could be considered. In order to remain focused these additional methods are not considered. The selected methods are however well suited for the tasks given in this thesis.

DeepWalk and Node2Vec make use of methods which originated in natural language processing (NLP). Specifically they makes use of the Skip-Gram model introduced by Mikolov et al. (2013a,b). As the Skip-Gram model is a core component of DeepWalk and Node2Vec, the Skip-Gram framework is explained in detail in the following paragraph as it relates to graph representation learning.

In NLP words are one-hot encoded for the Skip-Gram model in which vector representations for the words are learned. Similarly, nodes in a network can be thought of as words in a NLP context. A basic overview of the Skip-Gram model is provided in figure 4. The input defined as X corresponds to a matrix with dimension $N \times N$ of one-hot encoded words or rather nodes. N corresponds to the number of nodes. The projection H is calculated using a simple forward propagation of the input times the weight matrix Φ with dimension $N \times D$. Here D defines the number of dimensions for the vector embedding of the words or nodes. The projection H is thereafter forward propagated again by multiplying it with a second weight matrix Ψ with dimensions $D \times V$. V refers to the context size a node is being compared to. For graph representation learning context size is defined by the number of neighboring

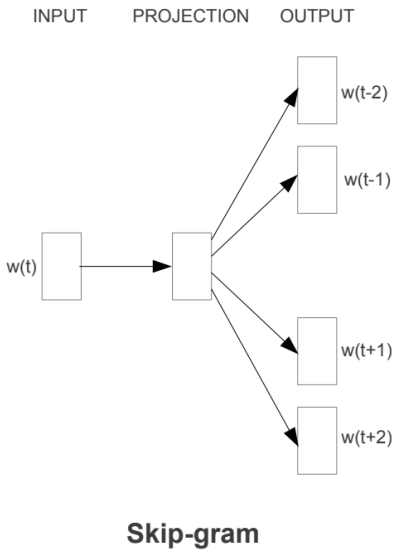

Figure 4: Skip-Gram Architecture
(Mikolov et al. 2013a, p. 5)

nodes a node is being compared to. The output is thereafter activated using the Softmax activation function and the loss is calculated by comparing the activated output with the actual context node. The aim of the Skip-Gram model is to accurately predict the context of an input node, the neighbors of a given input node. The weights Ψ and Φ are thereafter updated using stochastic gradient descent. This procedure is repeated for several iterations and the Skip-Gram model is therefore a special form of a artificial neural network. The weight matrix Φ lastly corresponds to the node vector embeddings.

Based on this logic, the DeepWalk algorithm was developed which was a big breakthrough. The DeepWalk algorithm builds on top of the Skip-Gram model and uses random walks for learning the node vector embeddings. To provide a better overview of the DeepWalk algorithm, the pseudo-code is presented in algorithm 1 & 2 as defined in its original paper.

The DeepWalk algorithm generates fixed length random walks for every node. The Skip Gram model is then run for every node on the random walk. In terms of context, the window size determines the neighboring nodes on the random walk the currently selected node is trying to predict in terms of context. The DeepWalk algorithm is then repeated for a defined number of random walks. In terms of training, the DeepWalk algorithm can lastly be run for a defined number of epochs. Lastly, the DeepWalk algorithm often uses more efficient approximation methods such as hierarchical Softmax which makes use of a binary tree or negative sampling. Both

Algorithm 1: DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

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

Algorithm 2: SkipGram$(\Phi, \mathcal{W}_{vi}, w)$

foreach $v_j \in \mathcal{W}_{vi}$ **do**
    foreach $u_k \in \mathcal{W}_{vi}[j - w : j + w]$ **do**
        $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
        $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
    end
end

approximation methods are outlined in the paper by Mikolov et al. (2013b).

This is in principle the model which will be used to find the node embeddings. In the application, the Node2Vec algorithm by will be employed. Node2Vec is a generalization of the DeepWalk algorithm and allows for the deployment of biased random walks. In particular, it allows to set probabilities as to whether the random walk is biased towards breadth-first (BFS) search or depth-first search (DFS). Depending on the network structure, setting an appropriate bias can greatly improve the quality of the embeddings. If no bias towards BFS or DFS is set, an unbiased random walk is employed which is when the Node2Vec algorithm corresponds to the DeepWalk algorithm. More precisely, this occurs when the search bias $\alpha = 1$ with $p = q = 1$ as outlined in the Node2Vec paper (Grover & Leskovec 2016, p. 860). The results revealed, that an unbiased random walk embedded the nodes very well. For that reason the Node2Vec algorithm is not explained in further detail as the relevant parts are covered by the simpler and reader friendlier DeepWalk algorithm.

The resulting node vector embeddings can then be used for downstream machine learning tasks. An additional benefit of graph representation learning is that the nodes can be encoded into an arbitrary number of dimensions. In this sense, graph representation learning can be used as a powerful dimensionality reduction strategy. Further, the node embedding vectors correspond to the features used in the downstream machine learning tasks. The values of the features were learned automatically using the DeepWalk or Node2Vec algorithm. This approach directly takes

care of the otherwise tedious feature selection process. With this approach, only the number of features needs to be defined in terms of feature selection. This is a large advantage and can save a lot of time when working with graphs.

2.2.2 Graph Neural Networks

This section will provide an overview regarding the theory about Graph Neural Networks (GNN). Within the family of GNNs, there are a myriad of different methods available and every few months new methods are being published. GNNs currently enjoy a large popularity and benefit from a large research output. This thesis will focus on two popular and established GNN approaches which are:

1. Graph Convolutional Networks
2. GraphSage

Before presenting the two above mentioned methods, a general overview of the GNN framework is given. First the required setup is defined (Leskovec 2021):

- $G(V, E)$ is a graph with a set of vertices and edge connections
- V is a set of vertices
- A is the adjacency matrix graph G
- $X \in \mathbb{R}^{|V| \times F}$ is the matrix of node features
- v is a node $\in V$ and $\mathcal{N}(v)$ is the set of neighbors of v

If there are no node features present, X can for instance be defined as a one-hot encoded vector. A naïve approach would be to join the adjacency matrix with the feature matrix as the input for a artificial neural network. The problem with this approach is, that the input is not order invariant and the model cannot be applied to graphs of different sizes (Leskovec 2021).

Modern GNNs have overcome this problem by drawing inspiration from Convolutional Neural Networks (CNN) and its famous filtering mechanism as outlined by Krizhevsky et al. (2012). CNNs typically work with grid structured input data such as pixels of images. The convolutional filter then samples the input grid using a filter with a specified size (e.g. 3×3 grid filter). Similarly GNN sample a graph using the node neighbors $\mathcal{N}(v)$ of node v as a filter. The filter can then be fine tuned in the sense of how many k-hops of neighbors to consider (e.g. 1-hop: immediate neighbors of v, 2-hop: include neighbors of v's neighbors etc.). In terms of implementation,

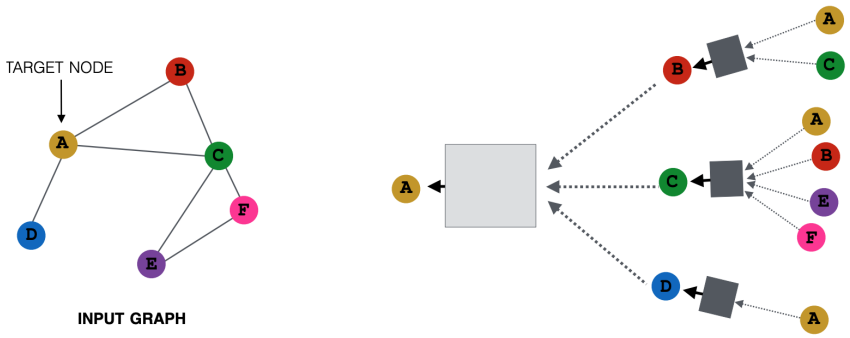

Figure 5: GNN Structure
Leskovec (2021)

the number of k-hops is set by the number of graph convolutional layers included in the GNN model. A graphical illustration of this mechanism is shown in figure 5.

The GNN Structure outlined shows an example of a 2-hop or 2 layer GNN. The 1-hop convolutional layer considers the neighboring nodes of the target node A. The 2-hop layer considers the neighbors of node A's neighbors. Note, that the target node A is included as an input node in the 2-hop layer. This is reasonable as node A itself is also a neighboring node to it neighbors. Further, node A is also a determining factor for its neighbors. Taking the example shown in figure 5, the challenge for the GNN is to find node embeddings based on local network neighborhoods (Leskovec 2021). The node embeddings at layer 0 correspond to the features of the input nodes where $X = H^{(0)}$. A typical procedure for a GNN model is outlined in algorithm 3 (Hamilton et al. 2017, Leskovec 2021, You et al. 2020).
Algorithm 3 is not meant to be considered as a complete overview and should be rather regarded as an example of a typical GNN structure. GNNs are flexible in that a myriad of modifications can be added to the GNN layers similar to the possibilities of CNNs or ANNs. The defining features of different GNN methods usually involve the selection of different message passing methods and aggregation strategies. An excellent overview regarding the design space for GNNs is provided in the articles by You et al. (2020) and Zhou et al. (2020). Of course there are exceptions and alternative procedures exist. The GNN methods evaluated in this thesis and most successful GNNs however tend to follow a variation of the structure shown in algorithm 3.

In terms of interpretation, the output of the first GNN layers in figure 5 (gray boxes) corresponds to the hidden layer representations of the direct neighbors of target node A and the output of the final GNN layer corresponds to the node embedding of the

Algorithm 3: Typical GNN algorithm

Input: Graph $G(V, E)$;
input features $\{x_v, \forall v \in V\}$;
node labels $\{y_v, \forall v \in V\}$;
training set $\mathcal{T} \subset V$;
depth/layers K with parameters $\Theta^k = \{W^k, B^k\}, \forall k \in \{1 \dots K\}$
non-linearity σ;
differentiable aggregator functions $AGGREGATE_k, \forall k \in \{1, \dots, K\}$;
neighborhood function $\mathcal{N} : v \rightarrow 2^V$;
loss function $\mathcal{L}$ such as cross entropy CE;
learning rate α

Output: Vector representations $z_v, \forall v \in V$

Initialize layer specific W^k from $\mathcal{U}, \forall k \in \{1, \dots, K\}$;
Initialize layer specific B^k from $\mathcal{U}, \forall k \in \{1, \dots, K\}$;
$h_v^0 = x_v, \forall v \in V$
for *Number of epochs* **do**
  for $k = 1 \dots K$ **do**
    foreach $v \in V$ **do**
      $h_v^k = \sigma\left(CONCAT\left(AGGREGATE_k(\{W^k \cdot h_u^{k-1}, u \in \mathcal{N}(v)\}), B^k \cdot h_v^{k-1}\right)\right)$
    end
  end
  $z_v = h_v^K, \forall v \in V$;
  $\mathcal{L}(\Theta) = \sum_{(z_v, y_v) \in \mathcal{T}} CE(y_v, z_v)$;
  $\Theta = \Theta - \alpha \cdot \frac{\partial \mathcal{L}}{\partial \Theta}$
end

target node A. This should sound familiar when comparing this approach to the graph representation learning approach outlined in the previous section. GNNs can indeed be used for unsupervised learning tasks such as finding node embeddings such that similar nodes are embedded close together. Good examples for this is shown in the papers regarding Graph Convolutional Networks by Kipf & Welling (2016) and GraphSage by Hamilton et al. (2017). As shown in algorithm 3, GNNs can be applied directly to a supervised learning task. In order to avoid redundancy with the graph representation learning strategies introduced previously, the GNNs will be applied in a supervised learning context. GNNs are powerful tools for unsupervised learning for tasks such as clustering or graph representation learning (Zhou et al. 2020). While GNNs are possibly more efficient for graph representation learning (Kipf & Welling 2016, p. 12-13), the results are unlikely to differ significantly and is not the focus of this thesis. The aim is to use GNNs in a supervised learning setting as it hopefully provides superior results for a given classification task. In the following paragraph the first GNN method of interest is presented.

Graph Convolutional Networks

The Graph Convolutional Network (GCN) was introduced by Kipf & Welling (2016) and makes use of simplified spectral graph convolutions. The author Thomas Kipf (2016) provides excellent explanations on his website which is used as inspiration for presenting the theory. As outlined, GNNs typically differ with regards to the type of message passing and aggregation strategy. GCN make us of the following

forward propagation method (Kipf & Welling 2016, p. 2):

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{20}$$

The variables in equation 20 are defined as follows:

- $H^l \in \mathbb{R}^{N\times D}$ refers to the embedding matrix at layer l where N refers to the number of nodes $\mid V \mid$ and D refers to the number of embedding dimensions. The input embedding matrix is set equal to the feature matrix, $H^{(0)} = X$.
- W^l refers to the trainable and layer specific weight matrix for the linear message passing employed in the GCN model.
- $\tilde{A} = A + I_N$ where A is the adjacency matrix of the input graph G. The identity matrix is added so that self-loops are considered. This is necessary as the target node of every layer is considered in the aggregation process as previously outlined.
- $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is a diagonal matrix containing the degree distributions of the modified adjacency matrix $\tilde{A}$.
- $\sigma(\cdot)$ refers to an activation function such as ReLu or Softmax.

To provide a better overview, the compact notation shown in equation 20 is expanded in the following equation for one GCN layer (Dubois 2019):

$$h_{ij}^{(l)} = \sigma\left(\sum_{(i,j)\in\mathcal{N}(v)} \frac{\tilde{a}_{ik}h_{kj}^{(l-1)}}{\sqrt{\tilde{d}_{k,k}\tilde{d}_{i,i}}}W^{(l)}\right) \tag{21}$$

In Equation 21 $h_{ij}^{(l)}$ refers to the hidden layer representation of node i at layer l considering the set of neighbors j. $h_{kj}^{(l-1)}$ corresponds to the hidden layer representation of node k at layer $l-1$ which is part of the set of neighbors j. In terms of filtering strategy, $(i,j) \in \mathcal{N}(v)$. This means that both set of nodes i and j are neighbors of the target node v at layers l and $(l-1)$ respectively. Linear message passing is then performed for every neighbor in j at layer $l-1$. The node in the graph G at position $\tilde{a}_{ik}$ of the modified adjacency matrix selects the nodes for which a connection between $h_{ij}^{(l)}$ and $h_{kj}^{(l-1)}$ exists. The embeddings $h_{kj}^{(l-1)}$ of the selected nodes are normalized by the symmetric degree distributions of the previous hidden layer node $\tilde{d}_{k,k}$ and the new hidden layer node $\tilde{d}_{i,i}$. Afterwards the normalized embeddings are message passed by multiplying it with the shared weight matrix $W^{(l)}$. Lastly, the sum of the received messages is taken in terms of aggregation strategy and the resulting aggregate is passed through the activation function to yield $h_{ij}^{(l)}$.

Note, that the aggregation strategy involves taking a weighted sum thanks to the symmetric normalization.

The detailed explanations given above show the procedure for one layer of a GCN. Returning now to compact notation, this procedure can be expanded for two or more GCN layers. First, the notation is further simplified by defining $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$. An example of a two-layer GCN is then given as follows where Z refers to the embedding or output of the target node (Kipf & Welling 2016, p. 3):

$$Z = f(X, A) = \text{softmax}\left(\hat{A}\ \text{ReLu}\left(\hat{A}XW^{(0)}\right)W^{(1)}\right) \tag{22}$$

Finally, the model parameters are updated analogues to the procedure outlined in algorithm 3. The main distinctive feature for the GCN is the differing forward propagation procedure.

GraphSage

This paragraph introduces GraphSage by Hamilton et al. (2017) which can be thought of as the inductive counterpart and generalization of the GCN presented in the previous paragraph. Inductive refers to the capability of not only performing machine learning tasks on the graph used for training but to apply the trained GNN model to new and unseen graphs. This is a large leap as GCN for instance can only be used to predict unseen nodes on the graph which was used for training. This is very limiting for the application of GNN in a practical setting. GraphSage achieves this by applying different aggregation strategies. In addition, GraphSage allows for efficiency increases when learning on large graphs by only considering a sample of the neighborhood $\mathcal{N}(v)$. In the GraphSage setting only a fixed number of uniformly sampled neighbors $\in \mathcal{N}(v)$ with depth K are considered. A graphical example of this procedure is shown in figure 6:

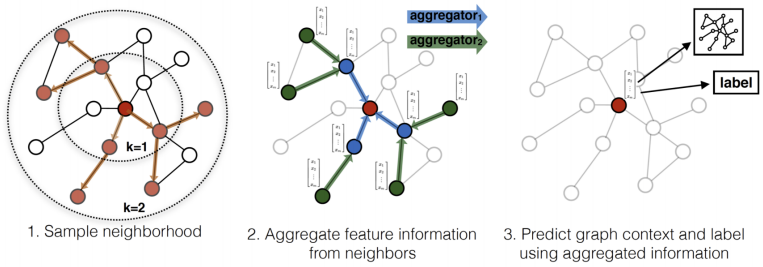

Figure 6: GraphSage Sampling
(Hamilton et al. 2017, p. 2)

The steps shown in figure 6 are similar to the procedures for GCNs. The pseudo code for the GraphSage forward propagation layers is given in algorithm 4 (Hamilton et al. 2017, p. 12).

Algorithm 4: GraphSAGE minibatch forward propagation algorithm

Input: Graph $G(V, E)$;
input features $\{x_v, \forall v \in \mathcal{B}\}$;
depth K;
weight matrices $W^k, \forall k \in \{1, \dots, K\}$;
non-linearity σ;
differentiable aggregator functions $AGGREGATE_k, \forall k \in \{1, \dots, K\}$;
neighborhood sampling functions, $\mathcal{N}_k : v \to 2^v, \forall k \in \{1, \dots, K\}$
Output: Vector representations z_v for all $v \in \mathcal{B}$

$\mathcal{B}^K \in \mathcal{B}$
for $k = K \dots 1$ **do**
  $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^k$
  for $u \in \mathcal{B}^k$ **do**
    $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^{k-1} \cup \mathcal{N}_k(u)$
  end
end
$h_u^0 \leftarrow x_v, \forall v \in \mathcal{B}^0$
for $k = 1 \dots K$ **do**
  for $u \in \mathcal{B}^k$ **do**
    $h_{\mathcal{N}(u)}^k \leftarrow AGGREGATE_k(\{h_{u'}^{k-1}, \forall u' \in \mathcal{N}_k(u)\})$;
    $h_u^k \leftarrow \sigma\left(W^k \cdot CONCAT(h_u^{k-1}, h_{\mathcal{N}(u)}^k)\right)$;
    $h_u^k \leftarrow h_u^k / \|h_u^k\|_2$
  end
end
$z_v \leftarrow h_u^K, \forall u \in \mathcal{B}$

Note that algorithm 4 assumes that the parameters of the K aggregator functions and the weight matrices W^k are known. Algorithm 4 thus shows the forward propagation of a trained GraphSage model. The model is trained by optimizing the model parameters for every batch trained during a specified number of epochs. The training procedure can be implemented by using an adaptation of the general procedure shown in algorithm 3. In algorithm 4 $\mathcal{B}$ refers to the mini-batch training for the set of vertices V of the graph $G(V, E)$.

There are three aggregator types proposed for GraphSage (Hamilton et al. 2017):

1. Mean aggregation
2. Max pooling
3. LSTM (Long short-term memory) aggregation

The three proposed aggregation strategies are briefly introduced as follows:

Mean Aggregation:
This type of aggregation is similar to the GCN and takes the average of the received

messages from the message passing process. The difference to GCN is that mean aggregation does not rely on the full graph Laplacian and makes use of a slightly different normalization approach. The aggregation process differs to the one shown in algorithm 4 and replaces the procedures in line 9 and 10 with (Hamilton et al. 2017, p. 5):

$$h_v^k \leftarrow \sigma\left(W \cdot \text{MEAN}(\{h_v^{k-1}\} \cup \{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})\right) \tag{23}$$

Max Pooling:
Max Pooling aggregation refers to the application of an element-wise max operator. This means that of the neighbors, only the largest features are considered. More formally, max pooling aggregation is defined as follows and is used for the aggregation shown in line 9 of algorithm 4 (Hamilton et al. 2017, p. 6):

$$AGGREGATE_k^{pool} = \max\left(\sigma(\{W_{pool} h_{u_i}^k + b), \forall u_i \in \mathcal{N}(v)\}\right) \tag{24}$$

Note, that W_{pool} refers to a separate weight matrix for the one layer ANN of the max pooling aggregation. In principle, an arbitrary number of layers could be added for max pooling. The authors Hamilton et al. (2017) however focus on the case with one layer. The parameters of max pooling are learned analogues to the other GraphSage model parameters.

LSTM:
LSTM is the last aggregation strategy proposed for GraphSage and uses the LSTM recurrent neural network (RNN) first introduced by Hochreiter & Schmidhuber (1997) as an aggregation strategy. LSTMs are not permutation invariant which is a requirement for the aggregation strategy. The authors propose using a random permutation of the set of node neighbors to counter this problem (Hamilton et al. 2017, p. 5). The LSTM parameters are trained along with the other GraphSage model parameters.

2.3 Graph Generation

As mentioned in the introduction, gathering network data including features is rather difficult task. This is especially true, as most databases do not capture relationships. Further, it is very difficult if not impossible to collect network data using standard surveys. This is a problem for this master thesis for which no perfect solution exists. As outlined, this problem sparked the research question as to whether network data can be generated from cross-sectional data for subsequent and meaningful machine learning tasks. Among the many graph generation algorithms researched, the Mul-

tiplicative Attribute Graph (MAG) model by Kim & Leskovec (2012) appears to provide a feasible solution and will be used for this thesis.

The MAG model makes use of two main inputs which are:

1. Node attribute vectors a_i
2. Link-affinity matrices Θ_i

The node attribute vector a_i is part of the graph node-attribute generation matrix $B^{N\times K}$. For the purpose of graph generation, only attributes can be considered for which reasonable link-affinity probabilities can be defined. Therefore $B \subseteq X$ where $X^{N\times F}$ is the full feature matrix with N observations and F features. The feature matrix X could for instance be data collected in a survey or a client database with absent relational information. F could for instance refer to the number of questions in a survey for which responses were collected. Every respondent in the survey subsequently is considered as a node in graph setting and the responses are the corresponding node features. Note, that the authors refer to node features as attributes. The terms are used interchangeably for the purpose of this thesis. The K appropriate features which can be included in B are variables for which reasonable link-affinity probabilities can be defined. A typical example for this could be age. One could imagine an appropriate setting in which people of the same age group are more likely to form a connection / become friends in a social network setting versus people which are in different age groups. Another example for this could be gender in terms of biological sex which is a classical binary setting for a link-affinity matrix. An example for binary attribute link-affinity matrices Θ_i is given in figure 7.

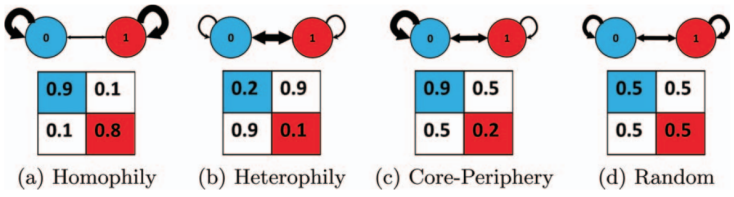

Figure 7: Node-attribute link-affinities
(Kim & Leskovec 2012, p. 118)

Figure 7 shows 4 types of link-affinity matrices depending on the relationship between nodes one wants to model. Homophily refers to love of the same which would make a connection between two nodes more likely if they have the same features. Similarly, Heterophily refers to the love of the different where nodes which do not

have the same features are more likely form a connection. Core-periphery is a special case which can be used to generate realistic social-networks in terms of network properties (Kim & Leskovec 2012, p. 139). For instance one could model a group of people in terms of whether they are members in a football club or not. It would then be reasonable to assume, that members of a football club are very likely to be connected while non-members have a significantly lower probability of forming a connections. Lastly, random graphs can be generated by setting the link-affinity probabilities to 0.5. Given the available data sets, graphs will be generated using homophily structures. The node-attribute link-affinity matrices Θ_i are defined for every attribute and can be set for an arbitrary size of categories within an attribute (extends beyond the binary setting). More formally for each node $u \in V$ with K categorical attributes of cardinality d_i for $i = 1, 2, \ldots, K$ and corresponding link-affinity matrices $\Theta_i \in d_i \times d_i$ for $i = 1, 2, \ldots, K$, the probability $P[u, v]$ of an edge between nodes (u, v) is defined as (Kim & Leskovec 2012, p. 119):

$$P[u, v] = \prod_{i=1}^{K} \Theta_i[a_i(u), a_i(v)] \tag{25}$$

In equation 25 $a_i(u)$ refers to the value of the ith attribute of node u. A schematic representation of the procedure for a binary link-affinity matrix is shown in figure 8.

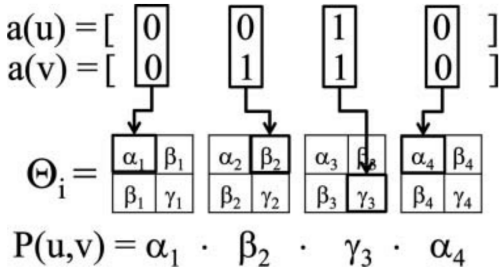

Figure 8: Schematic representation of the multiplicative attribute graphs (MAG) model

(Kim & Leskovec 2012, p. 120)

Based on this procedure the graph was generated using algorithm 5.

Algorithm 5: Multiplicative Attribute Graph Model

Input: graph node-attribute generation matrix $B^{N\times K}$, where $B \subseteq X^{N\times F}$;
node attribute vector $\{a_i, \forall i \in B\}$ with cardinalities d_i for $i = 1, 2, \ldots, K$;
link affinity matrices $\Theta_i \in d_i \times d_i$ for $i = 1, 2, \ldots, K$;
Output: adjacency Matrix $A^{N\times N}$ for Graph $G(V, E)$

$B^{K\times N} = B^T$
for $j = 1, 2, \ldots, N$ **do**
  $u = B[:, j]$
  for $k = 1, 2, \ldots, N$ **do**
    $v = B[:, k]$
    for $i = 1, 2, \ldots, K$ **do**
      $P_{j,k} = \prod_{i=1}^{K} \Theta_i[a_i(u), a_i(v)]$
    end
  end
end
$U^{N\times N} = uppertriangular(P)$ and with $diag(P) = 0$
for $i = 1, 2, \ldots, N$ **do**
  for $j = 1, 2, \ldots, N$ **do**
    if $U_{i,j} > \mathcal{U}(0, 1)$ **then**
      $\hat{A}_{i,j} = 1$
    end
    else
      $\hat{A}_{i,j} = 0$
    end
  end
end
$A = \hat{A} + \hat{A}^T$

References

Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V. & Smola, A. J. (2013), Distributed large-scale natural graph factorization, *in* 'Proceedings of the 22nd international conference on World Wide Web', pp. 37–48.
URL: *https://doi.org/10.1145/2488388.2488393*

Alphabet (2021), 'Alphabet investor relations'. (accessed: 04.05.2021).
URL: *https://abc.xyz/investor/*

AlphaFold (2020), 'Alphafold: a solution to a 50-year-old grand challenge in biology'. (accessed: 05.05.2021).
URL: *https://deepmind.com/blog/article/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology*

Barabási, A.-L. & Albert, R. (1999), 'Emergence of scaling in random networks', *science* **286**(5439), 509–512.

Dubois, Y. (2019), 'Interpretation of symmetric normalised graph adjacency matrix?', Mathematics Stack Exchange.
URL: *https://math.stackexchange.com/q/3284214*

Erdös, P. & Rényi, A. (2011), On the evolution of random graphs, *in* 'The structure and dynamics of networks', Princeton University Press, pp. 38–82.
URL: *https://doi.org/10.1515/9781400841356.38*

Euler, L. (1736), 'Solutio problematis ad geometriam situs pertinentis', *Commentarii academiae scientiarum Petropolitanae* **8**, 128–140.

Facebook (2021), 'Facebook investor relations'. (accessed: 04.05.2021).
URL: *https://investor.fb.com/investor-events/default.aspx*

Grover, A. & Leskovec, J. (2016), node2vec: Scalable feature learning for networks, *in* 'Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining', pp. 855–864.
URL: *https://doi.org/10.1145/2939672.2939754*

Hamilton, W. L., Ying, R. & Leskovec, J. (2017), Inductive representation learning on large graphs, *in* 'Proceedings of the 31st International Conference on Neural Information Processing Systems', NIPS'17, Curran Associates Inc., Red Hook, NY, USA, p. 1025–1035.
URL: *https://dl.acm.org/doi/abs/10.5555/3294771.3294869*

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.
URL: *https://doi.org/10.1162/neco.1997.9.8.1735*

Katz, L. (1953), 'A new status index derived from sociometric analysis', *Psychometrika* **18**(1), 39–43.
URL: *https://doi.org/10.1007/BF02289026*

Kim, M. & Leskovec, J. (2012), 'Multiplicative attribute graph model of real-world networks', *Internet mathematics* **8**(1-2), 113–160.
URL: *https://doi.org/10.1080/15427951.2012.625257*

Kipf, T. (2016), 'Graph convolutional networks'. (accessed: 15.05.2021).
URL: *https://tkipf.github.io/graph-convolutional-networks/*

Kipf, T. N. & Welling, M. (2016), 'Semi-supervised classification with graph convolutional networks', *arXiv preprint arXiv:1609.02907* .
URL: *https://arxiv.org/abs/1609.02907*

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), 'Imagenet classification with deep convolutional neural networks', *Advances in neural information processing systems* **25**, 1097–1105.
URL: *https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf*

Landau, E. (1895), 'Zur relativen wertbemessung der turnierresultate', *Deutsches Wochenschach* **11**, 366–369.

Leskovec, J. (2021), 'Cs224w lecture at stanford'. (accessed: 15.05.2021).
URL: *http://web.stanford.edu/class/cs224w/*

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C. & Ghahramani, Z. (2010), 'Kronecker graphs: an approach to modeling networks.', *Journal of Machine Learning Research* **11**(2).
URL: *https://www.jmlr.org/papers/volume11/leskovec10a/leskovec10a.pdf*

Li, Y., Vinyals, O., Dyer, C., Pascanu, R. & Battaglia, P. (2018), 'Learning deep generative models of graphs', *arXiv preprint arXiv:1803.03324* .
URL: *https://arxiv.org/abs/1803.03324*

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013a), 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781* .
URL: *https://arxiv.org/abs/1301.3781*

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. (2013b), 'Distributed representations of words and phrases and their compositionality', *arXiv preprint arXiv:1310.4546* .
URL: *https://arxiv.org/abs/1310.4546*

Newman, M. (2010), *Networks: An Introduction*, Oxford University Press, Inc.

Page, L., Brin, S., Motwani, R. & Winograd, T. (1999), The pagerank citation ranking: Bringing order to the web., Technical report, Stanford InfoLab.
URL: *http://ilpubs.stanford.edu:8090/422/*

Perozzi, B., Al-Rfou, R. & Skiena, S. (2014), Deepwalk: Online learning of social representations, *in* 'Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining', pp. 701–710.
URL: *https://doi.org/10.1145/2623330.2623732*

Schweitzer, F., Fagiolo, G., Sornette, D., Vega-Redondo, F., Vespignani, A. & White, D. R. (2009), 'Economic networks: The new challenges', *science* **325**(5939), 422–425.

Senior, A. W., Evans, R., Jumper, J., Kirkpatrick, J., Sifre, L., Green, T., Qin, C., Žídek, A., Nelson, A. W., Bridgland, A. et al. (2020), 'Improved protein structure prediction using potentials from deep learning', *Nature* **577**(7792), 706–710.
URL: *https://doi.org/10.1038/s41586-019-1923-7*

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. & Mei, Q. (2015), Line: Large-scale information network embedding, *in* 'Proceedings of the 24th international conference on world wide web', pp. 1067–1077.
URL: *https://doi.org/10.1145/2736277.2741093*

Watts, D. J. & Strogatz, S. H. (1998), 'Collective dynamics of 'small-world'networks', *nature* **393**(6684), 440–442.
URL: *https://doi.org/10.1038/30918*

You, J., Ying, R., Ren, X., Hamilton, W. & Leskovec, J. (2018), Graphrnn: Generating realistic graphs with deep auto-regressive models, *in* 'International Conference on Machine Learning', PMLR, pp. 5708–5717.
URL: *http://proceedings.mlr.press/v80/you18a.html*

You, J., Ying, Z. & Leskovec, J. (2020), 'Design space for graph neural networks', *Advances in Neural Information Processing Systems* **33**.
URL: *https://proceedings.neurips.cc/paper/2020/file/c5c3d4fe6b2cc463c7d7ecba17cc9de7-Paper.pdf*

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. & Sun, M. (2020), 'Graph neural networks: A review of methods and applications', *AI Open* **1**, 57–81.
URL: *https://doi.org/10.1016/j.aiopen.2021.01.001*