LEARNING REPRESENTATIONS OF GRAPH DATA: A SURVEY

A PREPRINT

Mital Kinderkhedia
Department of Statistical Science
University College London
London, W1CE 6BT
mital.kinderkhedia.10@ucl.ac.uk

June 10, 2019

ABSTRACT

Deep Neural Networks have shown tremendous success in the area of object recognition, image classification and natural language processing. However, designing optimal Neural Network architectures that can learn and output arbitrary graphs is an ongoing research problem. The objective of this survey is to summarise and discuss the latest advances in methods to Learn Representations of Graph Data. We start by identifying commonly used types of graph data and review basics of graph theory. This is followed by a discussion of the relationships between graph kernel methods and neural networks. Next we identify the major approaches used for learning representations of graph data namely: Kernel approaches, Convolutional approaches, Graph neural networks approaches, Graph embedding approaches and Probabilistic approaches. A variety of methods under each of the approaches are discussed and the survey is concluded with a brief discussion of the future of learning representation of graph data.

***K*eywords** Graph structured data, Graph representations, Graph-based neural networks, Graph embedding, Graph Convolutions.

1 Introduction

Structuring data into a graph facilitates the study of uncovering complex relationships and patterns in a systematic manner. For example, a World Wide Web graph shows complex structures given the high frequency of the links between webpages [1] and in Natural Language Processing [2] text is sometimes represented using trees to understand linkages between words to infer meaning of sentences. However, research in machine learning primarily focuses on data represented in a vectorial form. Real world data is not easily represented as vectors. Examples of real world scenarios with complex graph like structures include molecular and biological networks, computer networks, sensor networks, social networks, citation networks, power grids and transportation networks. Using a graph-based representation, it is possible to capture the sequential, topological, geometric and other relational characteristics of structured data.

Neural Networks are universal function approximators [3]. Following recent advances, deep learning models have achieved tremendous success in speech recognition [4],[5],[6] object recognition-detection [7],[8], and in learning natural language processing. Further, a confluence of ingredients: large datasets, advanced computational processing power and burgeoning research in machine learning methods has tremendously contributed to deep learning research. The main distinction we point out between neural and non-neural methods for machine learning is in the *learning representations of data* [9]. A *representation* of a datum object is defined as those pieces of information that, as a set, provide learnt information relevant to a given learning task. In machine learning terminology, we use the term features, whereas in representation learning terminology, we are concerned with learning the representations of data that make it easier to use the *learned* information for tasks such as prediction, classification.

The idea behind learning *graph representations* is to learn a mapping such that it embeds the vertices, sub-graphs or whole graphs into points in a low-dimensional vector space. These mappings are then optimised so that they reflect the geometric structure within the embedding space and the learned embeddings can then be used as vectorial inputs for machine learning tasks.

1.1 Contribution

The contribution of this survey is to propose a taxonomy of major approaches to learn graph representations identified as follows: Kernel approaches, Convolution approaches, Graph neural network approaches, Graph embedding approaches and Probabilistic approaches. This paper compares, contrasts and outlines the techniques used in these approaches. However, this survey is not exhaustive.

1.2 GRAPH DATA DOMAINS

Popular data domains that use graph-based representations are discussed, though the list is not exhaustive.

Biological Data: Typical biological data represents sequences of DNA, sequences of RNA and tertiary structure of proteins. The motivation for analysing such data is to discover new biological insights. Biological networks, such as the metabolic network [10] of bacteria *Escherichia coli* can be modelled as a graph to learn the relationships between small biomolecules (metabolites) and enzymes (proteins). Given that protein-protein interaction is crucial for majority of biological processes, Yook et al study *Saccharomyces cerevisiae* with the aim to uncover the network‘s generic large-scale properties and the impact of the protein‘s function and cellular localisation on the network topology [11].

Chemical Data: To represent chemical compounds as graph structures, the vertices can represent the atoms and the edges correspond to the bonds. In [12], graphs are used to model the the key topological and geometric characteristics of chemical structures. The motivation for modelling a set of chemical compounds or molecules as graphs is to understand their key characteristics, their toxicity and biological activity, for example. This has become a focal-point area of chemical graph mining [13]. Chemical data are unique in structure and the typical applications include mining substructures for the comparison of chemical compounds, predicting compound bio-activity using classification, regression and ranking [14].

Web Data: Typical web data are in the form of hypertext documents, shopping histories, browsing histories and search histories, for example. A webpage, is represented as a vertex and the hyperlinks between the pages as edges. Web pages modelled as graph objects are for the purpose of capturing the linkage structure. Given vast amounts of internet data, one goal would be to develop models that leverage the network topology in order to extract relational knowledge [15].

Text Data: Unprecedented growth of text data on the web has given room to model such data in a variety of ways. One approach is to use probabilistic (or neural probabilistic) models. Another approach is to use graphs. For example, in one case words could be represented as vertices and relations between the words as edges, and in another, topics represented as vertices and edges as relations between those topics. Vertices can also denote features of text. Examples of some popular graphs using text data include the co-occurrence graph, the semantic graph and the hierarchical keyword graph [16]. Organisations that leverage machine learning models, to learn from text, stand to gain insight into areas such as user sentiment analysis, trend detection, for example in the case of Twitter.

Relational Data: Relationships tend to form between i.e individuals or organisations, for example, for reasons of common interest. Popular examples of social ties and social interest can be seen in networks such as Facebook, Instagram, Twitter and Flickr. Analysis of network data using the vertex and edge features, often carried out for descriptive or inferential purposes, has resulted in the study of relational network science as a growing interdisciplinary field and it has found applications in a wide range of areas such as sociology [18], [19], physics [20], biology [21], computer science[22].

SocialMedia Data: This type of data encompasses a variety of data streams connected through a network structure. It possesses many unique characteristics, given its size and the evolving dynamic content i.e video, images and text resulting in a heterogeneous mixture. For social network data, the key characteristics that are important are the connections, preferences, status, comments and tags. As the images, text, video and audio co-occur and each item (i.e image, post, video clip, audio clip) is defined by number of characteristics, learning features from such data is a

non-trivial task. Further, interaction between the users through links or feedback adds another dimension to feature learning. Learning joint-embeddings of multiple data streams has resulted in a new research avenue.

2 GRAPH THEORY

2.1 Concepts

Graph theory terminology is described to provide background for forthcoming discussions involving graph data. A graph $\mathcal{G}$ is an ordered pair $(\mathcal{V}, \mathcal{E})$. Set $\mathcal{V}$ is the *vertex set* with $n \equiv |\mathcal{V}|$ denoting the *order* of the graph. Set $\mathcal{E}(\mathcal{G})$ is the corresponding *edge set*, with e_{ij} as the edge between vertex i and j. We use the notation $\mathcal{V}(\mathcal{G})$ and $\mathcal{E}(\mathcal{G})$ to denote a vertex and an edge set of given graph $\mathcal{G}$.

Types of Graphs: *Simple* graphs are considered throughout this paper. Simple graphs have pair of vertices connected by one edge only. Other graphs discussed in this survey are undirected, directed and weighted. *Undirected* graphs have each edge as an unordered pair $\{v, w\}$. In a *directed* graph, the edges are ordered pairs. In a *weighted* graph, a weight function $\omega : f \rightarrow \mathcal{R}$ assigns a weight on each edge. A graph is *connected* if there exist paths between all pairs of vertices. If all the vertices of a graph have the *same degree* then we have a *regular* graph. A *complete* graph is one in which there is an edge between every pair.

Degree, Walk, Cycle, Path, Distance, Height and Depth: The *degree* of a vertex u, denoted $deg(u)$, is the number of edges incident on u. A *walk* is a sequence of adjacent vertices and the corresponding edges, with the *length* of the walk given by the number of edges included. We will sometimes denote the vertices in a walk of length k as the sequence $v_0, \ldots v_k$. If $v_0 = v_k$ (i.e the start vertex is equal to the end vertex) then the walk is a *cycle*. A walk is a sequence of alternating vertices and edges. The term *path* denotes walks where no vertex appears more than once. Moreover, the distance between two vertices, denoted $dist(u, v)$, is defined as the length of the shortest path between them. The *height* of a vertex is the number of edges on the longest path top-down between the respective vertex and a leaf vertex. The *depth* of a vertex is the number of edges from the vertex to the tree's root vertex.

Subgraph: A subgraph $\mathcal{G}_1$ of a graph $\mathcal{G}$ is a graph $\mathcal{G}_1$ whose vertex and edge sets are subset of those of $\mathcal{G}$. A *clique* is a complete subgraph of a graph. A *cycle* is also a connected subgraph where each vertex has exactly two neighbours and a graph that contains no cycles is a *forest*. A connected forest is a *tree*. A *subforest* is an acyclic subgraph; a *subtree* is a connected *subforest*. The set of neighbours for a given vertex v is called the *neighbourhood* of v and is denoted by $\mathcal{N}_v$.

Graph Isomorphism: Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be two graphs. $\mathcal{G}$ and $\mathcal{G}'$ are said to be *isomorphic* if there exists a bijective function $f : \mathcal{V} \rightarrow \mathcal{V}'$ such that, for any $u, v \in \mathcal{V}$, we have that $f(u)$ and $f(v)$ are adjacent in $\mathcal{G}$ iff they are adjacent in $\mathcal{G}'$. Solving *graph isomorphism* problems is relevant to machine learning as this provides a way of detecting similarities among data points. However, graph isomorphism is a challenging problem. There is no known polynomial time algorithm for graph isomorphism. Early approaches to solve the graph matching problem proposed the use of graph edit-distances [23] as well as topological descriptors [24]. Using graph edit-distances involves counting the number of key operations that would transform graph $\mathcal{G}_1$ into $\mathcal{G}_2$ offering flexibility to assign costs; however, this approach suffers from the need to choose some optimal cost function for different operations as well intermediate steps of subgraph isomorphism. The use of topological descriptors mapping each graph to a feature vector again suffers from the loss of topological information through the transformation step. A practical tractable alternative consists of using substructures formed from graphs that are computable in polynomial time is popularly known as the *bag-of-structures* approach and it is discussed in detail in section four.

2.2 MATRIX REPRESENTATION OF GRAPHS

2.2.1 Types of Matrices

We need to work with the input representations of the matrices to generate the features. These are as follows: the *Adjacency Matrix*, the *Degree Matrix* and the *Laplacian Matrix*. The adjacency matrix of a graph is denoted as $\mathcal{A}$ and encapsulates the whole topology of the graph in the $n \times n$ form as below.

$$\mathcal{A}_{ij} = \begin{cases} 1, & if(i,j) \in \mathcal{A} \\ 0, & (i = j) \end{cases} \tag{1}$$

The degree matrix D_{ii} is a *diagonal matrix* where d_i is the degree of vertex i.

$$\mathcal{D}_{ii} = \begin{cases} d_i & (i = j) \\ 0, & (i \neq j) \end{cases} \tag{2}$$

For an unweighted graph, the normalised Laplacian matrix $\mathcal{L}$ is of the form

$$\mathcal{L}_{ij} = \begin{cases} -1 & if\,(i,j) \in \mathcal{E} \\ d_i & if\,i = j \\ 0 & otherwise \end{cases} \tag{3}$$

The spectral decomposition of a normalised Laplacian $\mathcal{L}$ is as follows. $\mathcal{L}$ is a symmetric positive semi-definite matrix and can take the form $\mathcal{L} = \Phi\Lambda\Phi^T$, where $\lambda = diag(\lambda_1, \lambda_2, \lambda_3 \ldots \lambda_{|V|})$ is the diagonal matrix with ordered eigenvalues of $\mathcal{L}$ as elements and $\Phi = (\phi_1, \phi_2, \ldots \phi_{|V|})$ is the matrix with ordered column eigenvectors. The *spectrum of a graph* is the study of the eigenvalues of the adjacency matrix.

2.2.2 Relationship between the Matrices

The normalised form of the adjacency matrix is $A_{\mathcal{N}} = \mathcal{D}^{-\frac{1}{2}}\mathcal{A}\mathcal{D}^{-\frac{1}{2}}$. The graph Laplacian can also be computed using the degree matrix and the adjacency matrix as $\mathcal{L} = \mathcal{D} - \mathcal{A}$. The normalised Laplacian is written as $\mathcal{L} = \mathcal{D}^{-\frac{1}{2}}\mathcal{L}\,\mathcal{D}^{-\frac{1}{2}}$ and this follows on to $\mathcal{L} = \mathcal{D}^{-\frac{1}{2}}(\mathcal{D} - \mathcal{A})\mathcal{D}^{-\frac{1}{2}}$

3 USE CASES

Popular use cases of graph data: graph comparison, graph classification, graph clustering, link prediction, graph compression and graph visualisation are discussed.

3.1 GRAPH COMPARISON

The task of graph comparison is to determine the dissimilarity or similarity between two graphs through a mapping $s : \mathcal{G} \times \mathcal{G} \rightarrow \Re$. Traditional graph comparison algorithms have been classified into set based, subgraph based and kernel based [25]. In set based approach, the graph is represented using either a set of edges or a set of vertices whereas in subgraph based approach the subgraphs are extracted from graph and comparison is done using the number of matching subgraphs. Kernel method based approaches are discussed in section four.

3.2 GRAPH CLASSIFICATION

The graph classification problem is of two types: one entails a vertex classification problem and other entails a whole graph classification. In the whole graph classification, given a dataset $\mathcal{D}$ consisting of graphs, where each graph $\mathcal{G}_i$ has a vertex-edge set such as $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i)$, the goal of graph classification is to learn the model $f : \mathcal{D} \rightarrow \mathcal{Y}$ and classify the graphs into one or more classes. Each graph has a corresponding class label typically given as $\mathcal{Y} = \{1 \ldots y\}$. Data is divided into the *training set* and the *test set* and the trained model is evaluated on the test set. The accuracy is tested by comparing the predicted output $\hat{y}_i$ with the true class label y_i.

3.3 LINK PREDICTION

A priori, we do not know which links are missing or which future links are going to be formed. The high-level view of the task of link prediction is to predict how the structure of the network evolves as existing participants form new links, break links [26]. For example, in protein-protein interaction networks, link prediction can identify novel interactions between proteins [27]. Following the definition in [28], given a network graph $\mathcal{S} = (\mathcal{V}, \mathcal{E})$, the task of link prediction is defined as follows. Consider $\mathcal{U}$ to be a universal set containing $\frac{|\mathcal{V}|(|\mathcal{V}|-1)}{2}$ possible links where $|\mathcal{V}|$ denotes the number of elements in the set. Therefore, the task of link prediction is to find the links in the set $\mathcal{U} - \mathcal{E}$. The dataset is divided randomly into two parts $\mathcal{E}^T$- training set and the $\mathcal{E}^P$- test set. Globally, the *Network Growth Prediction* problem is described as an extension of the link prediction problem. In social network analysis, it is used to predict user preferences for forming new friendships resulting in the user's social network growth [29].

3.4 GRAPH CLUSTERING

In graph clustering, the edge structure plays an important role. The vertices of the graph are grouped into clusters in such a way that there are many edges *within* the formed cluster and comparatively *fewer* between the clusters [30].

There are two main approaches: *within-graph* clustering and *between-graph* clustering. As the name implies, within-graph clustering methods divide the vertices within a graph into clusters, whereas in the case of between-graphs, the clustering algorithm works on a set of graphs which are divided into different clusters. In biology, the applications of graph clustering are in gene regulatory networks, metabolic networks, neural networks and food webs. In social networks, clustering is an approach used for community detection, for example in [31] the authors attempt to identify a user's circles, each a subset of her friends and this problem of circle detection is formulated as a clustering problem. Clustering is also used to identify communication networks, organisational networks, online communities and this is finely discussed in [32].

3.5 FURTHER USE CASES

Large scale graphs, such as web or social media graphs typically contain more more than a billion edges and are growing quickly. Learning from large graphs is extremely challenging from a computational viewpoint. Two use cases have gained recent traction to address this challenge: Graph Compression and Graph Visualisation. A compressed representation of a graph encodes its topological structure [33]. Constructing a good graph representation is a space-saving approach and several compression schemes have been researched to propose various graph representations [34]. Graph visualisation explicitly shows us the connections between vertices, communities or subgraphs. The visual graphic of a graph can show some interesting properties to enable the reader to study the network from another angle. Some interesting visualisations can be seen in [35], [36] and [37]. Nevertheless, challenges of customisability, layout and generating dynamic visualisations remain an ongoing case to solve.

4 KERNEL METHODS

Kernel methods are a widely used class of algorithms that could be applied to any data structure. Kernel approaches are also used as building blocks in some representation learning methods described in the following section. A *kernel function* is the inner product of two vectors in feature space. It *isolates* the *learning algorithm* from the instances. This means that the learning algorithm relies exclusively on the kernel values between the instances without the need to explicitly use the original data representation.

Formally, let $\mathcal{X}$ be a non-empty set and let $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{R}$, where $\times$ denotes set product. Kernel k is symmetric if $k(x, y) = k(y, x)$ and k is positive definite given that $n \geq 1$ and $x_1 \dots x_n \in \mathcal{X}$ and matrix k is defined by $k_{ij} = k(x_i, x_j)$ is positive definite i.e we have $\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0 \; \forall \; c_1 \dots c_n \in \Re$. An alternative way of writing a kernel function is $k(x, x') =< \phi(x), \phi(x') >$, where $\phi(x)$ is a feature vector. In the scope of this survey, kernel methods for graph-structured data are discussed.

4.1 KERNELS METHODS FOR GRAPHS

Learning dictionaries of structured data is an approach that started in the late 1990s. The *Bag-of-Structures* approach is one in which each data point is a derived vector representation for a given type of graph substructure. Using the bag-of-structures approach, the feature representation for each kernel type is fixed and each dimension corresponds to a type of substructure. This results in very high dimensional kernel space. Formally, let $\mathcal{G}$ be a non-empty set of graphs, then kernel $k : \mathcal{G} \times \mathcal{G} \rightarrow \mathcal{R}$ is called a *graph kernel*, here $< \phi(\mathcal{G}), \phi(\mathcal{G}') >$ are the respective feature vectors.

$$k(\mathcal{G}, \mathcal{G}') =< \phi(\mathcal{G}), \phi(\mathcal{G}') > \tag{4}$$

Existing graphs kernels are an instance of $\mathcal{R}$-convolution kernels. The $\mathcal{R}$-convolution framework is formed on pairs of graphs following the decomposition of two structured objects. [38] proposed the idea of decomposing an object into atomic substructures. Each new decomposition relation $\mathcal{R}$ results in a new graph kernel.

$$k_{conv}(x, x^{'}) = \sum_{(x_d, x) \in \Re} \sum_{(x_d^{'}, x^{'}) \in \Re} \mathcal{K}_{parts}(x_d, x_d^{'}) \tag{5}$$

There are two fundamental learning approaches when working with graph kernels: learning *Kernels defined on Graphs* and learning *Kernels defined between Graphs*. Kondor and Lafferty [39] proposed the idea of Kernels on Graphs which are kernels formed between vertices of a single graph. This was further extended by Smola and Kondor [40]. The Kernel between graphs approach was proposed by Gartner [41]. We review some of the kernels using the bag-of-structures approach and these are categorised into three major families: *Kernels on Walks and Paths*, *Kernels on Subtrees* and *Kernels on Subgraphs*. However, there are other methods used to derive graph features that do not depend on the bag-of-structures approach and these are discussed towards the end of this section.

4.1.1 Kernels on Walks and Paths

Random Walk Kernels proposed by Gartner, are based on counting the number of walk based substructures formed from the sequences of vertices between graphs in dataset $\mathcal{D}$. To find common walks in two graphs, a product graph is used. A product graph is formed from identically labeled vertices and edges from $\mathcal{G}_1$ and $\mathcal{G}_2$. Here (p_1,p_2) are the starting probabilities and (q_1, q_2) the stopping probabilities of the random walk. $\mathcal{A}_1, \mathcal{A}_2$ are the adjacency matrices of $\mathcal{G}_1$ and $\mathcal{G}_2$. The number of length l common walks on the direct product graph $\mathcal{G}$ where $\otimes$ is the Kronecker product of two matrices is computed as follows.

$$(q_1 \otimes q_2)(\mathcal{A}_1^T \otimes \mathcal{A}_2^T)(p_1 \otimes p_2) \tag{6}$$

Formally, the random walk kernel between two graphs can be defined as follows.

$$k(\mathcal{G}_1, \mathcal{G}_2) = \sum_{l=0}^{\infty} (q_1 \otimes q_2)(\mathcal{A}_1^T \otimes \mathcal{A}_2^T)^l (p_1 \otimes p_2)(q_1 \otimes q_2) \\ (I - \lambda(\mathcal{A}_1^T \otimes \mathcal{A}_2^T))^{-1}(p_1 \otimes p_2) \tag{7}$$

Here, λ is the discounting factor applied to long walks, with all the common walks summed for all different lengths. In a neater form, the random walk kernel is defined as follows where $q = (q_1 \otimes q_2)$ and $p = (p_1 \otimes p_2)$.

$$k(G_1, \mathcal{G}_2) = q^T(1 - \lambda\mathcal{A})^{-1}p \tag{8}$$

For the random walk approach, the runtime is $\mathcal{O}(n^6)$. Artificial kernel values that are a consequence of repeated vertex-edge visits. This is called *tottering*, a phenomenon in which different walks include repetitions of vertex and edges resulting in a high structural similarity score as the same vertex or edge is repeatedly counted. Another phenomenon, *halting* is associated with random walk kernels where a weight factor λ down weights the longer walks. Random walk kernels have been extensively studied with suggestions for improvement such as fast computation of random walks, label enrichment and methods to prevent tottering [42], [43].

Shortest Path Kernel [44] is computed by computing all pairs of the shortest-path p, for given length n, for each graph in dataset $\mathcal{D}$. Given the shortest paths, p and p' for the graphs $\mathcal{G}$ and $\mathcal{G}'$, the kernel is formed as the sum over all pairs of edges $\mathcal{E}_p$ and $\mathcal{E}_{p'}$ from p and p' using a sensible choice of kernel on the edges.

$$\mathcal{K}(\mathcal{G}, \mathcal{G}') = \sum_{p \in \mathcal{E}_p} \sum_{p' \in \mathcal{E}_p} \mathcal{K}^n(\phi(\mathcal{G}_p), \phi(\mathcal{G}'_p)) \tag{9}$$

To overcome the problems associated with walk kernel methods that suffer from tottering and halting, the approach is to define kernels on paths. However, computing all paths is intractable, while computing all pairs-shortest-paths is $\mathcal{O}(n^3)$ and comparing all pairs shortest paths is $\mathcal{O}(n^4)$. Considering large graphs, this becomes computationally expensive.

Cyclic Pattern Kernel [45] is computed by counting the number of common cycles that appear in each of the graphs in $\mathcal{D}$. Formally, it is defined as follows where $\phi(\mathcal{G})$ of the feature of the graph.

$$\mathcal{K}(\mathcal{G}, \mathcal{G}') = \mathcal{K}(\phi(\mathcal{G}_{cp}), \phi(\mathcal{G}'_{cp})) \tag{10}$$

However, to compute the cyclic pattern kernels, the cyclic and tree patterns from each graph need to be computed and then a form of intersection is applied. Finding all the cycles in a graph in polynomial time is a challenge and since there is no known algorithm to compute this, researchers have resorted to sampling and time-bounded measures.

4.1.2 Kernels on Subtrees

Subtree Kernel, proposed by Ramon and Gartner [46], is computed by finding the common subtrees in each of the graphs in dataset $\mathcal{D}$ and comparing them. By definition, a *subtree* of graph $\mathcal{G}$ is a connected subset of distinct vertices of $\mathcal{G}$ with an underlying tree structure. Finding common tree-like neighbourhoods amongst graphs in $\mathcal{D}$ amounts to counting pairs of identical subtrees with given height h. The advantage is that we have a richer representation of the graph structure encapsulating its topology. Subtree kernels on graphs is the sum of subtree kernel on vertices.

$$\mathcal{K}(\mathcal{G}, \mathcal{G}') = \sum_{a \in \mathcal{V}(\mathcal{G})} \sum_{b \in \mathcal{V}(\mathcal{G}')} \mathcal{K}_h(a, b) \tag{11}$$

However, the runtime is affected by the recursion depth of the subtree like patterns and therefore subtree kernels also suffer from *tottering*. Computing a subtree pattern will have a signature represented by the sequence of labels of the

vertices in the sequence [47].

Weisfelier-Lehman Kernel [48] (WL) is a fast subtree kernel. It uses the Weisfelier-Lehman isomorphism test composed of the steps of iterative multi-set label determination, label compression and relabelling. Here, h is the depth, with l given to be a relabelling function, the WL kernel is defined as follows

$$k(\mathcal{G}, \mathcal{G}') = \sum_{i=0}^{h} k(l^i(\mathcal{G}), l^i(\mathcal{G}')) \tag{12}$$

The 1-d WL algorithm is a type of colouring scheme. It starts with iterating through each vertex label and its neighbouring set. Each vertex is relabelled with the compressed label that is used at the next iteration. The algorithm goes through k iterations using the compressed labels to construct a frequency vector for each graph. The recolouring of the vertices converges when the number of distinct colours stop increasing, which means that the vertices in the graph cannot be further partitioned. The output contains the frequency of compressed labels occurring in k iterations. However, the WL kernel does exact matching and therefore the labels *EFGH* and *EFGJ* return a zero similarity match and so it fails to take partial similarity into account. Nevertheless, it scales well to large, labeled graphs.

4.1.3 Kernels on Subgraphs

Subgraph kernel is computed using the idea that similar graphs tend to have similar subgraphs which could be used for graph comparison. Connected non-isomorphic subgraphs of size k are referred to as *graphlets*. A k-sized graphlet is defined as $\mathcal{G}_k = \{g_1, g_2, g_3 \dots g_{n_k}\}$ where n_k is the unique number of graphlets of size k. Following [25], let $\phi(\mathcal{G}_f)$ be the normalised vector of length n_k whose i^{th} element is the frequency of the graphlet g_i in $\mathcal{G}$ and let s_j denote the number of times g_k occurs as a subgraph of $\mathcal{G}$. That is,

$$\phi(\mathcal{G}_f) = \Big(\frac{s_1}{\sum_j^{n_k} S_1} \cdot \dots \cdot \frac{s_j}{\sum_j^{n_k} S_j}\Big) \tag{13}$$

Graphlet kernels compute the similarity between two graphs using the dot product of the count vectors of all possible connected subgraphs of some order k.

$$\mathcal{K}(\mathcal{G}, \mathcal{G}') = \big(\phi(\mathcal{G}_f), \phi(\mathcal{G}'_f)\big) \tag{14}$$

Weighted Decomposition Kernel [49], applied to the use case of protein sequence and molecule graph classification uses a subgraph s from graph $\mathcal{G}$ called *selector*, with associated kernel δ, weighted according to the match within a *context of occurrence* i.e a set of subgraphs, $z = (z_1 \dots . z_D)$ with associated kernel k. Substructures are matched according to an equality predicate measure and the kernel is computed as follows.

$$\mathcal{K}(\mathcal{G}, \mathcal{G}') = \sum_{(s,z) \in R^{-1}(G), (s', z') \in R^{-1}(G')} \delta(s, s^{'}) \sum_{d=1}^{D} k(z_d, z_d^{'}) \tag{15}$$

4.1.4 Challenges: Working with Bag-of-Structures approach

Diagonal Dominance: The bag-of-structures approach recursively decomposes structured objects into substructures but this results in various challenges. For example, one of the challenges is the *Diagonal Dominance* problem, where the kernel matrix becomes closer to the identity matrix. This happens when different substructures are regarded as different features and as these substructures grow in number, the feature set grows larger. Therefore, the probability that given two graphs will contain similar substructures diminishes. Hence, the highest similarity of a graph is to itself as compared to other graphs within the training set.

Substructure Sparsity & **Substructure Dependence**: Other practical issues include *Substructure Sparsity*, the problem of where only few of the substructures are common across the graph. *Substructure Dependence* is the problem where subgraphs that occur are not independent, as one subgraph could be found inside another or could be arrived at by making modifications to vertices and edges of other subgraphs. Therefore, features that are represented by these subgraphs turn out to be similar in nature. Finally, most graph kernels consider each substructure as separate feature and this not only increases the feature set but also results in similar features. Therefore, the substructures that occur frequently, those that often encompass the lower order substructures, tend to dominate the occurrence index.

4.1.5 Other Approaches

Intensive research on kernels for structured data with associated applications can be found in [50],[51], [52], [53], [54], [55], [56]. Nevertheless, there are other kernel approaches that work equally well on structured data and are used

to derive graph features. For example, using hash kernels for structured data [57] results in hashing that preserves information and also facilitates dimensionality reduction. Neighbourhood hash kernel [58] is one such example that uses hashing techniques for labeled graphs encoding the vertex neighbourhood and topology information using the bit arrays and logical operations. Each vertex label is transformed into a bit label with a mapping function. Using the XOR operation on bit labels for a given vertex label around the neighbourhood of that vertex results in a unique encoding for that vertex and its neighbourhood. Each of these unique encodings combined into a feature matrix, are used to learn the whole graph representation. The use of heat kernel for generating graph representations has has found numerous uses. For example, authors Xiao and Handcock, explore numerous ways to compute the heat kernel for graph clustering [59]. Many kernels designed for structured data have leveraged the use of probabilistic graphical models to discriminate features. Consider the Fisher Kernel [60] which compares two objects by fitting a generative model to the entire dataset and then using the fisher information matrix and the fisher score for each data point defining the kernel in this manner. Another example, the Probability Product Kernel [61] is based on the central idea to define kernels between distributions. Themes of such existing methods have been incorporated into recent approaches for learning graph representations.

5 LEARNING GRAPH REPRESENTATIONS

Five taxonomies are proposed by categorising them according to a set of baseline techniques used to construct the methods in each of the following approaches. These are: Kernel approaches, Convolution approaches, Graph neural network approaches, Graph embedding approaches and Probabilistic approaches. The term *graph representation* is defined as a *learned* graph feature which is obtained from a neural where each *learned* representation encodes the respective topological information about the graph.

5.1 KERNEL APPROACHES

Recent advances have highlighted the relation between neural networks and kernel methods. For example, Cho and Saul [62, 63] construct kernels that mimic neural networks whilst Mairal et al [64], show the connection between convolutional neural networks and kernels. Further references are [65],[66] and [67]. Kernel approaches are characterised by the use of kernel methods adapted to neural learning.

Deep Graph Kernels: [68] is one of the foremost approaches to combine graph kernels with deep learning techniques championed by Yanardag and Viswanathan. They tackle the challenge of capturing meaningful semantics between substructures. The bag-of-structures approach suffers from the issues of substructure dependence, substructure sparsity and diagonal dominance (section 4.1.4). The authors alleviate these by introducing the encoding matrix $\mathcal{M}$, a $|\mathcal{S}| \times |\mathcal{S}|$ positive semidefinite matrix that *encodes* the relationships between the substructures where $|\mathcal{S}|$ is the size of vocabulary of substructures extracted from the training data. This is achieved by designing $\mathcal{M}$ such that it respects the similarity of the substructure space. The kernel is then defined as

$$\mathcal{K}(\mathcal{G}, \mathcal{G}') = \phi(\mathcal{G})^T \mathcal{M} \phi(\mathcal{G}') \tag{16}$$

Approaches to calculating $\mathcal{M}$ are as follows, first by using the edit-distance relationship between the substructures and second by learning the latent representations of substructures using probabilistic neural language models (section 5.3.2). Data corpuses are generated such that a co-occurrence relationship is partially preserved. The neural language model is built using the continuous bag-of-words or Skip-Gram and trained using hierarchical softmax. Interesting performance results for deep graph kernels are shown on social network and bioinformatics dataset.

Kernel Neural Network: In [69], the authors leverage kernels defined over structured data such as sequences and graphs to derive neural operations. They design a new architecture using kernel inner product, embedding it into a recurrent neural network. Within the scope of this review, one such example is explained to illustrate the embedding of the graph kernel into the neural module. Given, the random walk kernel (6) concerning feature vectors f_x, the kernel and neural computations are linked as follows.

$$\begin{array}{lcl} c_1[v] & = & w^1 f_v \\ c_j[v] & = & \lambda \sum_{u \in N(v)} c_{j-1}[u] \odot W^j f_v, \quad 1 < j \leq n \end{array} \tag{17}$$

Here $\odot$ is the element wise product, $N_{(v)}$ represents the neighbourhood of the graph around vertex v, W is the weight matrix and $c_j[t]$ and $h[t]$ are the pre and post activation states. Equation (17) provides a model that embeds the random walk kernel into the neural framework. Here, $c_*[v]$ is the random walk count vector for a given vertex v and $h_{\mathcal{G}}$ is the latent representation of the graph aggregated from vertex vectors and this learned representation could be used for

either classification or regression.

$$h_{\mathcal{G}} = \sigma(\sum_{v} c_n[v]) \quad 1 < j \leq n \tag{18}$$

In this manner, illustrating with the random walk kernel example, the neural module embeds sequence similarity within the architecture. Kernels could be used for the single and multiple layer constructions in which case through stacking the output states $h^l[t]$ of the l^{th} layer are fed into the $(l+1)$ layer as the input sequence. The authors derive similar templates for a variety of graph kernels families supporting their theory with numerous experiments.

5.2 CONVOLUTION APPROACHES

Early researchers Fukushima [70], Atlas [71], LeCun [72] contributed to the development of convolutional neural networks (CNNs). Recent contributions by Mallet [73] and Wu [74] focus on the CNN theoretical framework. CNN architectures are able to extract representations from data that have an underlying spatiotemporal grid structure, making them suited for working with image, video [75] and speech data [76]. CNNs are designed to extract local features across the signal domain by extracting the local stationarity property of the input data.

Learning with CNNs involves two key operations *convolution* and *pooling*. Localised convolutional filters learned from data identify similar features. Convolutional filters are shift invariant and location independent hence recognising identical features independently. For example, with image data the CNN convolution operator takes an input and convolves kernel filter over it using stride s and appropriate parameters for tuning. Feature maps generated as a result of the convolution operator are then fed into pooling layer to provide a compressed output. Further a fundamental concept for designing convolutional neural network architectures uses a *receptive field* which in essence is the *local region* of the input. Selecting vertices of a graph for creating a convolution is similar to selecting receptive field in a classic neural network. Addressing the practical concern in the increase of the number of parameters when features in one layer are connected to the features in the layer beneath, Coates et al [77] propose connecting each feature extractor to a *local region* i.e the receptive field of inputs. Features are grouped together based on a similarity measure to limit the number of connections between the two layers.

Convolutions on graph. Many data have irregular structure in their underlying graph, due to the underlying irregular spatial geometry, such data is known as *non-euclidean* data. Regular, lattice type, underlying structure is found in time-series and image data whereas irregular underlying structure is found in text data, sensor data, mesh data, social network data and gene data, for example. To design convolutional networks for graph data, we need to use a similar convolutional operator but *one that works on graph data domain, one that works on an irregular domain*.

We define concepts that are used to formulate the graph convolution operator in the papers discussed below. A *graph signal*, considering undirected graphs, is a function mapping $x : \mathcal{V} \rightarrow \Re$ defined on the vertices of the graph and represented by vector $x \in \Re^N$, where the n^{th} component of the vector x represents signal value at the n^{th} vertex in $\mathcal{V}$. One can think of the data as tied to the vertex of the graph, for example a vertex could denote a single gene in a gene-gene interaction network.

Classical fourier transform of a function f, frequency w is the inner product of f with eigenfunction $exp(2\pi iwt)$.

$$\widehat{f(w)} = \int_{-\infty}^{\infty} f(t)\, exp(-2\pi iwt)\, dx \tag{19}$$

The graph fourier transform of a function $f : \mathcal{V} \rightarrow \mathcal{R}$ is the expansion of f in terms of the eigenfunctions of the graph Laplacian. $\mathcal{L}$ is positive semidefinite, has $\{u_l\}_{l=0}^{n-1} \in \Re^n$ orthonormal set of eigenvectors and the nonnegative eigenvalues $\{\lambda_l\}_{l=0}^{n-1} \in \Re^n$ and the eigendecomposition as $\mathcal{L} = \mathcal{U}\,\lambda\mathcal{U}^T$ where $\lambda = diag[\lambda_0.....\lambda_{n-1}]$, $\mathcal{U}$ the fourier basis. Fourier transform converts a signal from a time domain into the frequency domain. The graph fourier transform $\hat{x}$ of a spatial signal x is $\hat{x} = \mathcal{U}^T x \in \Re$ followed by the inverse $x = \mathcal{U}\hat{x}$ [78]. Formally, *convolution* is defined through an integral that expresses the amount of overlap of given function g as it is shifted over another function f. Mathematically, it is written as follows [79].

$$f * g = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{+\infty} g(\tau)f(t-\tau)d\tau \tag{20}$$

Convolution is a linear operator that diagonalises the fourier basis and as we can express a meaningful translation in the fourier domain instead of the vertex domain. The convolution operator on the graph is defined as follows, where $\odot$ is the element wise product.

$$x_{\mathcal{G}} * y = \mathcal{U}((\mathcal{U}^T x) \odot (\mathcal{U}^T y)) \tag{21}$$

Discretised convolutions used in CNN, commonly for image data are defined on regular grids both 2D and 3D and hence not applicable to the graph data domain. For irregular grids such as graphs, we need to define localised filters and these are known as *spectral graph convolutions*. Our interest is to obtain spectral convolutions on graphs. Spectral graph convolutions exploit the fact that convolutions are multiplications in the fourier domain. A spectral convolution of the signal x with a filter g_θ as follows.

$$y = g_\theta\left(\mathcal{L}\right) x = g_\theta\left(\mathcal{U}\,\lambda\,\mathcal{U}^T\right) x = \mathcal{U}\, g_\theta(\lambda)\,\mathcal{U}^T\, x \tag{22}$$

Here, $g_\theta(\lambda) = diag(\theta)$, where $\theta \in \Re$ is vector of fourier coefficients.

5.2.1 Spatial and Spectral Approaches

Two main methods based on convolutional approaches as proposed in literature for graph data. These are defined as the *spatial approach* and the *spectral approach*. Spatial approach is characterised using the notion of localised receptive fields formed by the neighbourhood of a vertex in the context of graph data for CNNs. The receptive fields are formed as a direct measure of distance in a graph, where given a vertex considered to be the center of the filter, we look around at vertices within a particular number of hops away. Spectral approach is characterised by using measures of distance based on decompositions of the graph Laplacian. Both approaches require careful consideration, creating spectral convolutions is dependent on graph structure. For spatial convolution, there is the need to create shift-invariant convolutions for graph data, as well as the problem specific need to determine vertex ranking and neighbourhood ordering. Another drawback observed with CNNs is that convolution operations are only applied to vertex features assuming that the graph domain is fixed but in many cases the graph can be noisy and some graphs are computed *a priori*, this is not necessarily reflects the relationships between the actors.

5.2.2 Spatial Approach

Spatial Convolutions. The use of spatial approach can be noted in PATCHY-SAN(PS)[80]. Here, it is used to learn the graph representations using a CNN in a supervised fashion creating receptive fields in the manner similar to how classical CNNs work on images. PS uses a number of steps to create graph derived receptive fields. During vertex sequence selection of a section, for a given graph $\mathcal{G}$, a sequence of vertices is identified and in the neighbourhood assembly step the neighbours are identified for creating receptive fields. Thus the receptive field for a given vertex results in a *neighbourhood* receptive field. Following the creation of a neighbourhood receptive field, a normalisation procedure is implemented which in essence is a form of vertex ordering to create a vector in vector space for deriving graph features used to learn graph representations. [81] is another example of spatial approach, using random walk (6) to select spatially close vertices. It forms the convolutions by associating the i^{th} parameter w_i with the i^{th} power of the transition matrix (P^i). Using the transition matrix results in $\mathcal{O}(N^2)$ complexity and [82] suggests a simple method for thresholding graphs resulting in optimum memory reducing complexity to $\mathcal{O}(N)$.

Generalised Convolutions: [83] take a generalised approach to CNNs, allowing for generating convolutions on graphs of different sizes. This proposition uses a spatial approach based on a random walk based transition matrix on the graph to select ranked k neighbours. The transition matrix P is then used to derive $\mathcal{Q}_{ij}^k$ which calculates the expected number of visits from a given vertex X_i to given vertex X_j in k steps. The convolution over the graph vertex X_i is represented as a tensor product of the form $(M, N, d) \rightarrow (M, N, p, d)$ where the $4D$ tensor includes the top p neighbours of each feature selected by Q^k, for M observations, N features at depth d. The authors test the approach on the Merck molecular activity Kaggle challenge.

Motif CNN. Motif are small subgraphs patterns demonstrating specific connections amongst vertices. [84] present a motif-based CNN where the motifs are defined to create a receptive field around a target vertex of interest. A motif-convolutional unit at vertex v_i is defined such that the features of all vertices that are locally connected through that motif are weighted according to their respective semantic roles. *motif-cnn* uses an attention mechanism to integrate features extracted from multiple motifs for semi-supervised learning. To the tackle the explicit assumption used by the laplacian for undirected graphs i.e symmetric laplacian matrix, MotifNet uses motif induced adjacencies by constructing a symmetric motif adjacency matrix. [85].

In [86], Bronstein et al, discuss non-euclidean data, coined under the umbrella term *Geometric deep learning* detailing the learning problems. [87] treat matrix completion problem, using spatial patterns extracted by the CNN architecture designed to work on multiple graphs. The column graph is thought of as a social network capturing relations between users and similarity of their tastes whereas row graph represents the similarities of the items items.

5.2.3 Spectral Approach

Spectral Networks are introduced by [88] and show a construction technique for connecting locally formed neighbourhood graphs. The idea behind using spectral networks is to generalise the convolutional network through the graph Fourier transform. For spectral construction, the spectrum of the graph Laplacian is exploited to generalise the convolution operator. Each of the constructions are tested on variations of the MNIST dataset. In [89], authors build on the above work using spectral graph convolution filters (22). The authors train a *graph convolution layer* performing forward and backward pass given a Fourier matrix U, an interpolation kernel $\mathcal{K}$ and weights w. During the forward and backward pass the tasks amounts to learning spectral filters on the graph. In the first variation, coordinates are extracted from subsampled MNIST data, forming convolutions via the Laplacian spectrum. In the second variation, MNIST data are projected onto 3D sphere.

Applications to molecular data. [90] shows a special case of an application for predicting properties of molecules described as graphs. Using a custom-made CNN, a local filter is applied to an atom and its neighbourhood. Molecular graphs, representing the properties of the original molecules are of particular interest in predicting properties of molecules. State of the art method, Quantitative Structure Activity Relationship (QSAR) [91] uses circular fingerprints with a fully connected neural network. The authors use existing methods to create circular fingerprints to construct a differentiable fingerprint changing each non-differentiable operation to a differentiable one. A differentiable fingerprint is created inputting a molecular graph resulting in a neural fingerprint. Differentiable fingerprints can be optimised to include only relevant features and using similarity between fragments, neural fingerprints become more meaningful. In the spirit of neural fingerprints, molecular graph convolutions [92] is another architecture proposed to learn from undirected small graphs of molecules. In [93], the authors show how to use original graph data consisting of diverse graph structures by constructing a customised graph Laplacian that uses a filter by combining the neighbourhood features according to the unique graph topology. [94] adapt the use of convolutions with auto-encoders to reconstruct/learn the latent/damaged fingerprint representations.

Generalising CNNs to Graphs:[95] provides a method to generalise CNNs to graph data using thorough spectral theoretical formulation. During the feature extraction phase, the authors perform graph signal filtering and graph coarsening. In the graph filtering phase, following a spectral formulation, strictly localised filters are defined within k radius ball. In the graph coarsening phase, Graclus [96] a fast graph clustering software. It computes the normalised cut and ratio association for a given undirected graph without any eigenvector computation [97]. As pooling compresses the output, meaningfully arranged graph neighbourhoods need to be defined. To do this optimally, the authors devise an efficient binary tree arrangement of the vertices resulting in a strategy similar to constructing a 1D signal.

Graph Convolutional Network:[98] introduces the Graph Convolutional Network (GCN) for vertex classification in a semi-supervised setting. A neural network model $f(X, A)$ is used to encode the graph structure using a layer-wise propagation rule where features X were derived using the popular Weisfeiler-Lehman algorithm (12) for adjacency matrix A. Given that evaluating the spectral convolution $g_\theta * x = U g_\theta U^T x$, where x is a signal for every vertex, would be computationally expensive, particularly for large graphs, following Hammond et al [99], g_θ could be approximated by Chebyshev polynomials $g_{\theta'}(\lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\widetilde{\lambda})$. [100] shows an application of GCN in population graphs for brain analysis combining imaging and non-imaging data. Modelling relational data stored in knowledge bases is useful in a number of tasks such as question answering, knowledge completion and information retrieval.[101] propose a relational graph convolution network (R-GCN), an extension of GCN, for the task of link prediction, predicting facts and entity classification. The network consists of an encoder a R-GCN that produces latent representations of the entities and a decoder which is a tensor factorisation model.

Recognition and Attention Mechanism (GAT). In [102], 2D feature maps are transformed into a graph structure where vertices define regions and edges capture the relationship between regions. Using the steps of *graph projection*, *graph convolution* and *graph re-projection*, context modelling and recognition is done with graph structure. [103] use attention based architecture to perform vertex classification of graph structured data. It computes the *importance* of each edge by processing only features of its incident vertices. This model is then applicable for inductive learning where it can be generalized to unseen graphs.

5.3 GRAPH NEURAL NETWORKS

We define a Graph based Neural Network (GNN) framework as one in which the connectivity among units follow the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ structure.

Graph Neural Network:[104] is one of the earliest approaches to propose a neural network architecture motivated by graph structure. A state vector x_v is attached to each vertex v given the information contained in its neighbourhood, where each vertex contains vertex-level label information l_v. Two main steps form the GNN. First, a parametric *transition function* $x_v = f_w(l_v, l_{N(v)})$ (which expresses the dependence between a vertex v, its label l_v and its neighbourhood $N(v)$) propagates the learned vertex representations. Second, the local output function $O_v = g_w(x_v, l_v)$ maps vertex representations to respective graph labels. The encoding network is built by storing the states of each vertex and updating the state when activated. GNN learns in a recursive manner, with a recurrent relation used to obtain the embedding x_v for each vertex v in the euclidean space. The model is differentiable end-to-end with learning parameter w tasked as a minimisation of a quadratic cost function.

Graph Gated Sequence Neural Network and Gated Graph Transformer Network: [105] is a modification of GNN functionality. The graph gated sequence neural network (GGSNN) allows for non-sequential outputs. A classic example of this is outputting logical formulas. Using Gated Recurrent Units (GRUs)[106], GGSNN unrolls a recurrence for a fixed number of steps. Backpropagation through time is used to compute the gradients. The propagation model is in the same spirit of GNN, with each vertex representation updated following a recurrence relation. The output model is defined per vertex and is a differentiable function mapping to an output. [107] proposes the use of Gated Graph Transformer Neural Network (GGT-NN) in order to solve reasoning problems. A number of graph transformations such as node addition, node state update, edge update, propagation and aggregation are combined to solve question answering and graph construction tasks.

5.4 GRAPH EMBEDDING APPROACHES

Embedding graphs into a low dimensional space encompass a set of techniques that deal with the transformation of an input graph into its respective vector representation mapping it to a point in space with a function mapping. A variety of graph embedding techniques find application in visualization, community detection, localization of wireless devices, power grid network routing, for example. Graph embedding techniques focus on preserving the proximity such that vertices within the same neighbourhood share nearby euclidean space. Historically, graph embedding methods have been been successfully used for obtaining low-level graph data representations. Consider the ISOMAP [108] in which a neighbourhood ball is used to convert the data points into a graph, using Djikstara's algorithm to compute the geodesic distance between the vertices. Another approach, the Multidimensional Scaling (MDS) [109] which when applied to the matrix of geodesic distances results in a reconstructed manifold and is often used to locate well-formed manifolds of complex datasets. Locally Linear Embedding[110], considered a variant of PCA, reduces the dimension of the data using a nearest neighbour approach. [111] provide a brilliant treatise on the use of auto-encoders to generate graph representations. Following their notation, we have a *pairwise* encoder-decoder framework to get a pair of embeddings (z_i, z_j) such that on reconstruction we have the following loss function $\mathcal{L}$.

$$\begin{aligned} DEC(ENC(v_i, v_j)) = DEC(z_i, z_j) \sim S_{\mathcal{G}}(v_i, v_j) \\ \mathcal{L} = \sum_{(v_i, v_j) \in \mathcal{D}} l(DEC(z_i, z_j), S_{\mathcal{G}}(v_i, v_j)) \end{aligned} \tag{23}$$

The general idea is that the encoder *ENC* maps the vertices to vector embeddings and the decoder *DEC* accepts a set of embeddings and decodes the user-specified graph statistics from these embeddings. The general setup adopted by a majority of authors is to find a similarity function defined on the graph, followed by a pairwise encoder-decoder that learns the embedding and $\mathcal{L}$ is a loss function which determines the performance. A number of methods combine neural learning techniques with natural language are discussed in the following sections.

The use of **Natural Language Techniques** as a tool [112],[113],[114] for learning vertex and edge representations of graphs and has dominated early research in learning graph representations. The intuition to encode word tokens into a vector in some $\mathcal{N}$ dimensional space where each dimension would represent some semantic meaning in speech for example, is adapted with a bold conjecture for the case of learning graph representations in a similar manner such that each vector would encode the topological information of the graph. In the following paragraph, we discuss some natural language processing theory basics.

Notion of context: Differentiating between the notion of *vertex context* and *word context*, we define *word context* as set of words surrounding the current word to be of length l_c in a given corpus of n words formed with a window of size k around word w. Sliding windows result in dynamic word contexts. We define a *vertex context* as a set of those vertices composed into a set through the process of some graph traversal algorithm. A vertex context will automatically encode the topological information around the vertex neighbourhood that is particularly determined by

the choice of some graph algorithm.

Probabilistic language models: A natural start is to use a probabilistic language model defined for using the word to be predicted, from a set of n words as $target\, t$, given $target\, history\, h$. The target t is defined as the probability of the next word given the target history h which is defined as a set of preceding words. The training set is typically a sequence of words $w_1 \dots w_T$. The model is as follows.

$$p(w_1 \dots w_T) = \prod_{t=1}^{T} p(w \mid \underbrace{w_{t-1}, \dots, w_{(t-n+1)}}_{h}) \tag{24}$$

Continuous bag-of-words (CBOW) model uses *context* c, a set of words, from which a word is to be predicted or generated.

$$p(w_t \,|c) = \frac{exp(v_{w_t}^T \cdot v'_{w_t})}{\sum_{w=1}^{v} exp(v_{w_t}^T \cdot v'_{w_t})} \tag{25}$$

Here, v_w corresponds to the input vector representation of word w and v'_{w_t} represents the output vector representation. Skip-Gram, proposed by Mikolov et al, maximises the probability of the surrounding words in context c given the current word w. Following notation from [115], the conditional probability using the softmax is as follows. Here, v_c and $v_w \in \Re^d$ are vectors representing c, w respectively, $C(w)$ is the set of contexts of word w and $\mathcal{D}$ is the set of all word and context pairs.

$$p(c|w;\theta) = \frac{exp(v_c \cdot v_w)}{\sum_{c' \in \mathcal{C}} exp(v_{c'} \cdot v_w)} \tag{26}$$

Using optimal parameters, θ, we maximise the following objective. The intuition here is that similar words have similar vecctors.

$$\begin{aligned} &\underset{\theta}{argmax} \prod_{w,c \in \mathcal{C}(w)} p(c|w;\theta) = \\ &\sum_{(w,c) \in D} \Big(log\, exp(v_c \cdot v_w) - log \sum_{c'} exp(v_{c'} \cdot v_w) \Big) \end{aligned} \tag{27}$$

The idea is to set the parameters in such a manner so that (27) is maximised. However, maximising (27) may turn out to be expensive and therefore an alternative *Hierarchical Softmax* seems a suitable option. Proposed by Morin and Bengio [116], given a vocabulary $\mathcal{V}$ of n words for a given word w in $\mathcal{V}$, computing the softmax probability of the word would require normalising over the probabilities of all words. Instead of using a flat layer, a hierarchical layer is used to decompose the probabilities of observing the next word in the sequence. The vocabulary of words is converted into a sequence tree structure which is balanced. Now, the probability for a given word is computed using path to the word from the root followed by the tree path. Modelling probabilities in this manner makes it a cost efficient way of defining a distribution. Negative sampling, an alternative form of Skip-Gram is often used as an efficient way of deriving embeddings but optimises a different objective.

5.4.1 Recent Advances

Walk based approaches: Deepwalk [35] one of the foremost methods to combine deep learning techniques with natural language models to learn representation of random walks on a graph. Deepwalk uses truncated random walk to transform the sampled linear sequence vertices into a co-occurrence matrix. Skip-Gram model is used to obtain low-dimensional representations for vertices. Using social network data, Deepwalk learns social representations that are latent features of the vertices to capture neighbourhood similarity and community membership. It separates the label space from the graph structure to build the feature space and takes an unsupervised approach to capture the network topology using the hierarchical softmax function. Deepwalk is suited to work on single large unweighted graphs instead of multiple graphs with a focus on learning similarities between vertices. Nevertheless, it has emerged as one of the most popular baseline models against which new approaches discussed below are measured and developed.

[117] is similar to Deepwalk in the manner that it uses random walk (8) to generate sequences and that it is scalable to large graphs. It differs in the use of p and q parameters which are preassigned and generate the walks in such a manner that they return to their parent vertex or not far from it. However, several models have to be generated and a subset of labelled vertices are sampled to find the best p, q values. Node2Vec operates in a semi-supervised setting with the graph based objective using stochastic gradient descent. Learning in the network is formulated as

a maximum likelihood optimisation problem with two standard assumptions about conditional independence and symmetry in feature space. In a similar spirit, [118] leverages random walks using two strategies: first to choose initial distributions $\mathcal{P}_o$ to produce invariant random walk features $r(\mathcal{P}_o)$, the second to generate random walk features localised to each vertex. [119] uses an approach similar to Deepwalk but overcomes the need to use its slow sampling process for generating sequences by using a random surfing model that directly constructs a probabilistic co-occurrence matrix from a weighted graph. Next, a high- dimensional positive pointwise mutual information matrix (PPMI) is calculated and used as an input to stacked denoising autoencoders to learn the low-dimensional vertex representations. [120] uses a k-step approach, with different k values to capture the relational information amongst vertices from the graph directly. Using global transition matrices defined over the graph overcomes the shortcomings of graph sampling processes that typically involves tuning parameters such as maximum length of linear sequences and sampling frequency for each vertex. The k-step transition probability is used to define a k-step loss function over the complete graph. Following Mikolov, noise-contrastive estimation is used to define the objective function. A matrix factorisation approach using singular value decomposition is used to optimise the proposed loss.

Subgraph based embedding approaches: In an attempt to overcome the shortcomings of subgraph assumptions (section 4.1.4) in [68], [121] extends the Weisfeiler-Lehman (section 4.1.2) relabelling strategy defining a radial context to alleviate the problem of selecting fixed-length sequences. The radial Skip-Gram captures sequences of varying lengths. The proposed algorithm consists of two main steps: first, the Weisfeiler-Lehman relabelling for building rooted subgraphs; second, the radial Skip-Gram for learning the embeddings of the subgraphs. The motivation for computing the subgraphs is to leverage the *local information* from the neighbourhood of the vertices in order to learn their latent representations. [122] leverages the Weisfeiler-Lehman as a neural machine, that takes the subgraph of a target link, encoding it into an adjacency matrix, for the use case of link prediction. [123] is another example of learning distributed representations of subgraphs with the goal to embed these into a low-dimensional continuous vector space. Using the local proximity definition that measures how many vertices, edges and paths are shared by two subgraphs, similarity between two subgraphs is learned with with an embedding function. In [124], the authors use a diffusion-like process to extract a subgraph, from which vertex sequences are used to extract features called *hitting frequencies* that are used to construct graph embeddings which are then learned using a neural network. Learned embeddings are used to cluster vertices for the purpose of community detection.

Multimodal data graphs: The authors in [125] leverage the increasing amount of social media data (section 1.2) to learn joint multimodal embeddings of text and images. A *scene graph*[126], defined by its objects, attributes and relationships is a directed graph constructed as $\mathcal{G} = (\mathcal{O}, \mathcal{E})$ where $o \in \mathcal{O}$ is an object in image $\mathcal{I}$ for a given $t \in \mathcal{E}$, a labeled directed edge. The proposed method learns joint representations of scene graph g and image x with respective embedding functions $f_i(x)$ and $f_g(g)$ that provide continuous representations in $\Re^D$ for input images and scene. Three embedding strategies: the bag-of-words, subpath representation and a graph neural network are used to learn the embeddings. In a similar spirit, for the use case of visual question answering, [127] build graphs over the scene objects and over the question words. A deep neural network model is used such that one leverages the inherent graph structure in these representations.

Inductive framework: In [128] the authors propose an *inductive framework*, the case where predictions are made on instances unobserved in the graph at the training time, here the embeddings are defined as a parameterised function of input feature vectors. The model is formulated using a feed-forward neural network, with input feature vector x, hidden layer defined as $h^k(x)$ = ReLU $(W^k h^{k-1}(x) + b^k)$. The loss function is defined as $\mathcal{L}_s + \lambda\mathcal{L}_u$. In the transductive formulation k layers are applied on the input feature vector to obtain $h^k(x)$ and l layers on embedding e to obtain $h^l(e)$ followed by a probability of $p(y\,|x, e)$ of predicting the label y and a transductive loss function. For the inductive case, the label y depends only on the feature x resulting in a respective loss function. Both models are trained using stochastic gradient descent. Another case of inductive learning, [129] proposes a function that generates embeddings by sampling vertex features from its context for unseen data.

5.5 PROBABILISTIC APPROACHES

Probabilistic approaches to learn representations of graph data encompass a variety of neural generative models, gradient based optimisation methods and neural inference techniques. Latent variable modelling involves modelling the relationship between a latent z and an observed variable x with associated parameters θ.

$$p_\theta(x, z) = p(z)p_\theta(x|z) \tag{28}$$

Here $p_\theta(x, z)$ is the joint distribution, $p(z)$ is the prior distribution and $p(x|z)$ is the likelihood. Inference in bayesian models is performed by conditioning on the data calculating the posterior $p(z|x)$. In many cases, calculating the posterior is not straightforward due to complex densities and hence the need for approximate inference tools. The

variational approach [130] encompasses a general family of methods for approximating complicated densities by a simple class of densities represented by a new approximate posterior distribution $q_\theta(z|x)$. A new family of variational methods called Variational Autoencoders [131] leverages the formulation of neural network using backpropagation techniques to form a new class of neural generative models. The action happens in mapping the data from the latent space to the observed space and that mapping is done using the neural network.

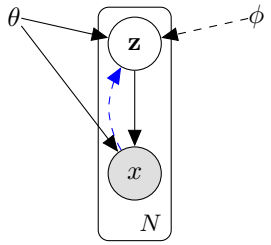

Figure 1: Graphical Model for the Encoder-Decoder Latent Variable Model

5.5.1 Recent Advances

[132] uses a variational auto-encoding approach to learn the graph representations. A GCN (section 5.3) is used as an encoder and *inner product* as the decoder. The proposed model inference is as follows.

$$q(Z|X,A) = \prod_{i=1}^{N} q(z_i|X,A) \tag{29}$$

X is the vertex features matrix derived using Weisfelier-Lehman (section 4.1.2) and A is the adjacency matrix with

$$q(z_i|X,A) = \mathcal{N}(z_i|\mu_i, diag(\sigma_i)) \tag{30}$$

Here, μ and σ are parameterised by the GCN(X,A). The generative model is given in the form of

$$p(A|Z) = \prod_{i=1}^{N}\prod_{j=1}^{N} p(A_{ij}|z_i,z_j) = \sigma(z_i^T z_j) \tag{31}$$

where $\sigma(\cdot)$ is the logistic function. Learning is performed by optimising the variational lower bound as follows.

$$\mathcal{L} = E_q(Z|X,A)[log\, p(A|Z)] - \mathcal{KL}_q[(Z|X,A)||p(Z)] \tag{32}$$

Application to Molecular Data: In [133], the focus is on generating discrete string representations of chemical molecules using a VAE. Discrete chemical molecules are represented using SMILES (Simplified Molecular Input Line Entry System) format . The encoder network takes each molecule and converts it into a vector into latent space (the space of all such vectors is termed the latent space). The decoder then reproduces a corresponding SMILES string. The network is trained using gradient based optimisation. However, the decoder could associate high probability to strings that are not valid SMILE strings. To address this problem [134] introduce a *Grammar Variational Auto-encoder* (GVAE) where a parse tree from context free grammar is used to describe a valid discrete object. The advantage of generating parse trees over text is to ensure that all outputs are valid and follow grammar rules, though their approach does not guarantee chemical validity. [135] propose a graph-based generative model for *de novo* molecular design. The authors compare the model's performance against the SMILES based approach by comparing the the rate of valid outputs.

The use of **Graphical models** for feature space design is championed in [136]. A Markov random field is postulated such that a latent variable h_i is associated with each observed variable x_i according to a Markovian structure motivated by a corresponding graph data point. Representations are motivated by the posterior distribution of each latent variable given observations, which are implemented by a set of equilibrium equations mapping features of the posterior of the neighbours of a vertex to its own posterior features, as inspired by the equations of loopy belief propagation and other variational methods. Such distributional features are tweaked by supervised learning within a given prediction problem, resulting in a representation of the graph data point.

Generating Graphs using a neural generative setting is an open problem. Real-world graphs, with edge connections between vertices are formed through arbitrary connections. Generating graphs involves discrete decisions

which may not be differentiable and [137] addresses this issue by using a decoder to return a fully-connected graph of a predefined size. A neural network is devised to translate vectors in a continuous space to graphs with the output matched accordingly with a graph matching algorithm. In [138] deep generative graph models, that make no structural assumptions, use a deep neural network to learn the distributions over any arbitrary graphs. A *graph net*, transforms the graph into a sequence of actions, with modules that provide the probabilities such as $(f_{addnode}, f_{addedge})$, adding nodes and edges respectively, to help build the structure of the graph offering the opportunity to use a number of different generative models. Further scope remains in terms of node ordering, reduction in the sequence of decisions, scalability and overcoming the challenges in training the neural network.

6 FUTURE DIRECTIONS

Some of the emerging research in the field looks at the problem of encoding graph data within the prior distribution, learning representations of weighted graphs, learning representations of temporal graphs, learning representations of temporal motifs, addressing non-euclidean graph domain specific challenges, addressing challenges of working with directed graphs and there remains further scope in developing novel probabilistic methods to learn representations of graph data. In this paper, five main approaches: Kernel approaches, Convolution approaches, Graph Neural Network approaches, Graph embedding approaches and Probabilistic approaches have been identified, grouped, surveyed and discussed. While representation learning tackles the leveraging of information from data as an automated process, there does not exist a universally preferred method for working with graph data. A practitioner could use this review as a road map to first gain knowledge about the recent advances and second to use it as a tool to guide further experimentation.

Acknowledgment

Mital Kinderkhedia is thankful to Engineering and Physical Sciences Research Council UK (grant number EP/G036306/1) for funding the doctoral research programme. Many thanks to Christopher Jefferson, Andrew Elliott, Matt Kushner and Jon Crowcroft for insightful discussions. Further, many thanks to anonymous reviewers whose suggestions and insights have considerably broadened my understanding and improved the quality of this manuscript.

References

[1] Srivastava, Jaideep, et al. Web usage mining: Discovery and applications of usage patterns from web data. ACM Sigkdd Explorations Newsletter 1.2, 2000: 12-23.

[2] Collins, Michael, and Nigel Duffy. Convolution kernels for natural language. NIPS. Vol. 14. 2001.

[3] Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks 2.5 (1989): 359-366.

[4] Dahl, G., Mohamed, A.R. and Hinton, G.E., 2010. Phone recognition with the mean-covariance restricted Boltzmann machine. In Advances in neural information processing systems (pp. 469-477).

[5] Dahl, G.E., Yu, D., Deng, L. and Acero, A., 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on Audio, Speech and Language Processing, 20(1), pp.30-42.

[6] Deng, L., Seltzer, M.L., Yu, D., Acero, A., Mohamed, A.R. and Hinton, G., 2010. Binary coding of speech spectrograms using a deep auto-encoder. In Eleventh Annual Conference of the International Speech Communication Association.

[7] Sermanet, P., Chintala, S. and LeCun, Y., 2012, November. Convolutional neural networks applied to house numbers digit classification. In Pattern Recognition (ICPR), 2012 21st International Conference on (pp. 3288-3291). IEEE.

[8] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[9] Bengio, Yoshua, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. IEEE transactions on pattern analysis and machine intelligence 35.8 (2013): 1798-1828.

[10] Lacroix, Vincent, Cristina G. Fernandes, and Marie-France Sagot. Motif search in graphs: application to metabolic networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 3.4 (2006): 360-368.

[11] Yook, S.H., Oltvai, Z.N. and Barabási, A.L., 2004. Functional and topological characterization of protein interaction networks. Proteomics, 4(4), pp.928-942.

[12] Wale, Nikil, Xia Ning, and George Karypis. Trends in chemical graph data mining. Managing and Mining Graph Data. Springer US, 2010. 581-606.

[13] S. J. Swamidass, J. Chen, J. Bruand, P. Phung, L. Ralaivola, and P. Baldi. Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. Bioinformatics, 21(1):359–368, 2005.

[14] A. R. Leach and V. J. Gillet. An Introduction to Chemoinformatics. Springer, 2003.

[15] Zanghi, Hugo, Christophe Ambroise, and Vincent Miele. Fast online graph clustering via Erdős–Rényi mixture. Pattern Recognition 41.12 (2008): 3592-3599.

[16] Sonawane, S. S., and P. A. Kulkarni. Graph based representation and analysis of text document: A survey of techniques. International Journal of Computer Applications 96.19 (2014).

[17] Zachary, W. W. (1977), 'An Information Flow Model for Conflict and Fission in Small Groups', Journal of Anthropological Research 33, Issue 4, Month Dec, 452-473.

[18] Wasserman, Stanley, and Joseph Galaskiewicz, eds. Advances in social network analysis: Research in the social and behavioral sciences. Vol. 171. SAGE Publications, 1994.

[19] Palla, Gergely, Albert-László Barabási, and Tamás Vicsek. "Quantifying social group evolution." Nature 446.7136 (2007): 664-667.

[20] Adamic, Lada A., et al. Search in power-law networks. Physical review E 64.4 (2001): 046135.

[21] Michailidis, George. Statistical challenges in biological networks. Journal of Computational and Graphical Statistics 21.4 (2012): 840-855.

[22] Faloutsos, Michalis, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. ACM SIGCOMM computer communication review. Vol. 29. No. 4. ACM, 1999.

[23] Gao, Xinbo, et al. A survey of graph edit distance. Pattern Analysis and applications 13.1 (2010): 113-129.

[24] Prado, Adriana, et al. Mining graph topological patterns: Finding covariations among vertex descriptors. IEEE Transactions on Knowledge and Data Engineering 25.9 (2013): 2090-2104.

[25] Shervashidze, Nino, et al. Efficient graphlet kernels for large graph comparison. International conference on artificial intelligence and statistics. 2009.

[26] Wang, Dashun, et al. Human mobility, social ties, and link prediction. Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011.

[27] Lei, C. and Ruan, J., 2012. A novel link prediction algorithm for reconstructing protein-protein interaction networks by topological similarity. Bioinformatics, Feb 2013, 29(3), pp.355-364.

[28] Lü, L. and Zhou, T., 2011. Link prediction in complex networks: A survey. Physica A: statistical mechanics and its applications, 390(6), pp.1150-1170.

[29] Al Hasan, M., Chaoji, V., Salem, S., & Zaki, M. (2006, April). Link prediction using supervised learning. In SDM06: workshop on link analysis, counter-terrorism and security.

[30] Schaeffer, Satu Elisa. Graph clustering. Computer science review 1.1 (2007): 27-64.

[31] Leskovec, J. and Mcauley, J.J., 2012. Learning to discover social circles in ego networks. In Advances in neural information processing systems (pp. 539-547).

[32] Fortunato, S., 2010. Community detection in graphs. Physics reports, 486(3), pp.75-174.

[33] Feder, T. and Motwani, R., 1991, January. Clique partitions, graph compression and speeding-up algorithms. In Proceedings of the twenty-third annual ACM symposium on Theory of computing (pp. 123-133). ACM.

[34] Maneth, S. and Peternek, F., 2015. A survey on methods and systems for graph compression. arXiv preprint arXiv:1504.00616.

[35] Perozzi, Bryan, et al. Deepwalk: Online learning of social representations. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014.

[36] Wang, D., Cui, P. and Zhu, W., 2016, August. Structural deep network embedding. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1225-1234). ACM.

[37] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. and Mei, Q., 2015, May. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web (pp. 1067-1077). International World Wide Web Conferences Steering Committee.

[38] Haussler, D. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, UC Santa Cruz, 1999.

[39] Lafferty, R. I., and J. Kondor. Diffusion kernels on graphs and other discrete structures. Machine Learning: Proceedings of the 19th International Conference. 2002.

[40] Smola, Alexander J., and Risi Kondor. Kernels and regularisation on graphs. COLT. Vol. 2777. 2003.

[41] Gartner, Thomas, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. Learning Theory and Kernel Machines. Springer Berlin Heidelberg, 2003. 129-143.

[42] N. Kriege, M. Neumann, K. Kersting, and M. Mutzel, Explicit versus implicit graph feature maps: a computational phase transition for walk kernels, in 2014 IEEE International Conference on Data Mining, 2014, pp. 881–886.

[43] U. Kang, H. Tong, and J. Sun, Fast random walk graph kernel, in Proceedings of the 2012 SIAM International Conference on Data Mining,2012, pp. 828–838

[44] Borgwardt, Karsten M., and Hans-Peter Kriegel. Shortest-path kernels on graphs. Data Mining, Fifth IEEE International Conference on. IEEE, 2005.

[45] Horváth, Tamás, Thomas Gartner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2004.

[46] Ramon, J. and Gärtner, T., 2003, January. Expressivity versus efficiency of graph kernels. In Proceedings of the first international workshop on mining graphs, trees and sequences (pp. 65-74).

[47] Shervashidze, N. and Borgwardt, K.M., 2009. Fast subtree kernels on graphs. In Advances in neural information processing systems (pp. 1660-1668).

[48] Shervashidze, Nino, et al. Weisfeiler-lehman graph kernels. The Journal of Machine Learning Research (2011): 2539-2561.

[49] Menchetti, S., Costa, F. and Frasconi, P., 2005, August. Weighted decomposition kernels. In Proceedings of the 22nd international conference on Machine learning (pp. 585-592). ACM.

[50] Watkins, C., 1999. Dynamic alignment kernels. Advances in neural information processing systems, pp.39-50.

[51] Jaakkola, T., Diekhans, M. and Haussler, D., 2000. A discriminative framework for detecting remote protein homologies. Journal of Computational Biology, 7(1-2), pp.95-114.

[52] Leslie, C.S., Eskin, E., Cohen, A., Weston, J. and Noble, W.S., 2004. Mismatch string kernels for discriminative protein classification. Bioinformatics, 20(4), pp.467-476.

[53] Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N. and Watkins, C., 2002. Text classification using string kernels. Journal of Machine Learning Research, 2(Feb), pp.419-444.

[54] Collins, M. and Duffy, N., 2002. Convolution kernels for natural language. In Advances in neural information processing systems (pp. 625-632).

[55] Kashima, H. and Koyanagi, T., 2002, July. Kernels for semi-structured data. In ICML (Vol. 2, pp. 291-298).

[56] Gartner, T., 2003. A survey of kernels for structured data. ACM SIGKDD Explorations Newsletter, 5(1), pp.49-58.

[57] Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A. and Vishwanathan, S.V.N., 2009. Hash kernels for structured data. Journal of Machine Learning Research, 10(Nov), pp.2615-2637.

[58] Hido, Shohei, and Hisashi Kashima. A linear-time graph kernel. Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on. IEEE, 2009.

[59] Xiao, Bai, and Edwin R. Hancock. Trace formula analysis of graphs. Lecture notes in computer science 4109 (2006): 306.

[60] Jaakkola, Tommi, and David Haussler. Exploiting generative models in discriminative classifiers. Advances in neural information processing systems. 1999.

[61] Jebara, Tony, Risi Kondor, and Andrew Howard. Probability product kernels. Journal of Machine Learning Research 5.Jul (2004): 819-844.

[62] Cho, Youngmin, and Lawrence K. Saul. Kernel methods for deep learning. Advances in neural information processing systems. 2009.

[63] Cho, Youngmin, and Lawrence K. Saul. Large-margin classification in infinite neural networks. Neural computation 22.10 (2010): 2678-2697.

[64] Mairal, Julien, et al. Convolutional kernel networks. Advances in Neural Information Processing Systems. 2014.

[65] Zhang, Yuchen, Jason D. Lee, and Michael I. Jordan. l1-regularized neural networks are improperly learnable in polynomial time. International Conference on Machine Learning. 2016.

[66] Hazan, Tamir, and Tommi Jaakkola. Steps toward deep kernel methods from infinite neural networks. arXiv preprint arXiv:1508.05133,2015.

[67] Mitrovic, Jovana, Dino Sejdinovic, and Yee Whye Teh. Deep Kernel Machines via the Kernel Reparametrization Trick. ICLR Workshop Track (2017).

[68] Yanardag, Pinar, & S. V. N. Vishwanathan. Deep graph kernels. Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

[69] Lei, Tao, et al. Deriving Neural Architectures from Sequence and Graph Kernels. arXiv preprint arXiv:1705.09037 2017

[70] Fukushima, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4):193–202, 1980

[71] Atlas, Les E., Homma, Toshiteru, and Marks, Robert J. II. An artificial neural network for spatio-temporal bipolar patterns: Application to phoneme classification. In Anderson, D.Z. (ed.), Neural Information Processing Systems, pp. 31–40. 1988.

[72] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. Neural Comput., 1(4):541–551, 1989.

[73] Mallat, Stéphane. Understanding deep convolutional networks. Phil. Trans. R. Soc. A 374.2065 (2016): 20150203.

[74] Jianxin Wu, An Introduction to Convolutional Neural Network, Tutorial, https://cs.nju.edu.cn/wujx/paper/CNN.pdf

[75] Taylor, G.W., Fergus, R., LeCun, Y. and Bregler, C., 2010, September. Convolutional learning of spatio-temporal features. In European conference on computer vision (pp. 140-153). Springer, Berlin, Heidelberg.

[76] Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N. and Kingsbury, B., 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, Nov, 29(6), pp.82-97.

[77] Coates, A. and Ng, A.Y., 2011. Selecting receptive fields in deep networks. In Advances in Neural Information Processing Systems (pp. 2528-2536).

[78] Shuman, D.I., Narang, S.K., Frossard, P., Ortega, A. and Vandergheynst, P., May 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Processing Magazine, 30(3), pp.83-98.

[79] Ronald Bracewell, The Fourier Transform & Its Applications, 1999.

[80] Niepert, Mathias, et al. Learning Convolutional Neural Networks for Graphs. Proceedings of the 33rd annual international conference on machine learning. ACM, 2016.

[81] Atwood, J. and Towsley, D., 2016. Diffusion-convolutional neural networks. In Advances in Neural Information Processing Systems (pp. 1993-2001).

[82] Atwood, J., Pal, S., Towsley, D. and Swami, A., 2017. Sparse Diffusion-Convolutional Neural Networks. arXiv preprint arXiv:1710.09813.

[83] Hechtlinger, Y., Chakravarti, P. and Qin, J., 2017. A generalization of convolutional neural networks to graph-structured data. arXiv preprint arXiv:1704.08165.

[84] Sankar, A., Zhang, X. and Chang, K.C.C., 2017. Motif-based convolutional neural network on graphs. arXiv preprint arXiv:1711.05697.

[85] Monti, F., Otness, K. and Bronstein, M.M., 2018, June. Motifnet: a motif-based graph convolutional network for directed graphs. In 2018 IEEE Data Science Workshop (DSW) (pp. 225-228). IEEE.

[86] Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A. and Vandergheynst, P., Jul 2017. Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine, 34(4), pp.18-42.

[87] Monti, F., Bronstein, M. and Bresson, X., 2017. Geometric matrix completion with recurrent multi-graph neural networks. In Advances in Neural Information Processing Systems (pp. 3697-3707).

[88] Bruna, J., Zaremba, W., Szlam, A. and LeCun, Y., 2013. Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203.

[89] Henaff, M., Bruna, J. and LeCun, Y., 2015. Deep convolutional networks on graph-structured data. arXiv preprint arXiv:1506.05163.

[90] Duvenaud, D.K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. and Adams, R.P., 2015. Convolutional networks on graphs for learning molecular fingerprints. In Advances in neural information processing systems (pp. 2224-2232).

[91] Lusci, A., Pollastri, G. and Baldi, P., Jul Aug 2013. Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. Journal of chemical information and modeling, 53(7), pp.1563-1575.

[92] Kearnes, S., McCloskey, K., Berndl, M., Pande, V. and Riley, P., 2016. Molecular graph convolutions: moving beyond fingerprints. Journal of computer-aided molecular design, 30(8), pp.595-608.

[93] Li, R., Wang, S., Zhu, F. and Huang, J., 2018. Adaptive Graph Convolutional Neural Networks. arXiv preprint arXiv:1801.03226.

[94] Svoboda, J., Monti, F. and Bronstein, M.M., 2017, October. Generative convolutional networks for latent fingerprint reconstruction. In 2017 IEEE International Joint Conference on Biometrics (IJCB) (pp. 429-436). IEEE.

[95] Defferrard, M., Bresson, X. and Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems (pp. 3844-3852).

[96] Kulis, B. and Guan, Y., 2010. Graclus-Efficient graph clustering software for normalized cut and ratio association on undirected graphs, 2008.

[97] Graclus Software
http://www.cs.utexas.edu/users/dml/Software/graclus.html

[98] Kipf, T.N. and Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[99] Hammond, D.K., Vandergheynst, P. and Gribonval, R., Mar 2011. Wavelets on graphs via spectral graph theory. Applied and Computational Harmonic Analysis, 30(2), pp.129-150.

[100] Parisot, S., Ktena, S.I., Ferrante, E., Lee, M., Moreno, R.G., Glocker, B. and Rueckert, D., 2017. Spectral Graph Convolutions on Population Graphs for Disease Prediction. arXiv preprint arXiv:1703.03020.

[101] Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I. and Welling, M., 2018, June. Modeling relational data with graph convolutional networks. In European Semantic Web Conference (pp. 593-607). Springer, Cham.

[102] Li, Y. and Gupta, A., 2018. Beyond Grids: Learning Graph Representations for Visual Recognition. In Advances in Neural Information Processing Systems (pp. 9245-9255).

[103] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. and Bengio, Y., 2017. Graph attention networks. arXiv preprint arXiv:1710.10903.

[104] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M. and Monfardini, G., Jan 2009. The graph neural network model. IEEE Transactions on Neural Networks, 20(1), pp.61-80.

[105] Li, Y., Tarlow, D., Brockschmidt, M. and Zemel, R., 2015. Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493.

[106] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

[107] Johnson, Daniel D, Learning graphical state transitions, ICLR, 2017.

[108] Tenenbaum, Joshua B., Vin De Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. science 290.5500 (2000): 2319-2323.

[109] Cox, Trevor F., and Michael AA Cox. Multidimensional scaling. CRC press, 2000.

[110] Roweis, Sam T., and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. science 290.5500 (2000): 2323-2326.

[111] Hamilton, W.L., Ying, R. and Leskovec, J., 2017. Representation learning on graphs: Methods and applications. arXiv preprint arXiv:1709

[112] Mikolov, Tomas, et al. Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems. 2013.

[113] Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. hlt-Naacl. Vol. 13. 2013.

[114] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. CoRR, abs/1301.3781, 2013

[115] Goldberg, Yoav, and Omer Levy. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 (2014).

[116] Morin, Frederic, and Yoshua Bengio. Hierarchical Probabilistic Neural Network Language Model. Aistats. Vol. 5. 2005.

[117] Grover, Aditya, & Leskovec, Jure. Node2vec: Scalable Feature Learning for Networks. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016

[118] Li, L., Campbell, W.M. and Caceres, R.S., 2017. Graph Model Selection via Random Walks. arXiv preprint arXiv:1704.05516.

[119] Cao, Shaosheng, Wei Lu, and Qiongkai Xu. "Deep neural networks for learning graph representations." Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI Press, 2016.

[120] Cao, Shaosheng, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, 2015.

[121] Narayanan, Annamalai, et al. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. arXiv preprint arXiv:1606.08928 (2016).

[122] Zhang, M. and Chen, Y., 2017, August. Weisfeiler-Lehman neural machine for link prediction. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 575-583). ACM.

[123] Adhikari, B., Zhang, Y., Ramakrishnan, N. and Prakash, B.A., 2017. Distributed Representation of Subgraphs. arXiv preprint arXiv:1702.06921.

[124] Rozemberczki, B. and Sarkar, R., 2018, March. Fast Sequence-Based Embedding with Diffusion Graphs. In International Workshop on Complex Networks (pp. 99-107). Springer, Cham.

[125] Belilovsky, E., Blaschko, M., Kiros, J.R., Urtasun, R. and Zemel, R., 2017. Joint Embeddings of Scene Graphs and Images.

[126] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.J., Shamma, D.A. and Bernstein, M.S., May 2017. Visual genome: Connecting language and vision using crowdsourced dense image annotations. International Journal of Computer Vision, 123(1), pp.32-73.

[127] Teney, D., Liu, L. and Hengel, A.V.D., 2016. Graph-structured representations for visual question answering. arXiv preprint arXiv:1609.05600.

[128] Yang, Z., Cohen, W.W. and Salakhutdinov, R., 2016, June. Revisiting semi-supervised learning with graph embeddings. In Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48 (pp. 40-48). JMLR.org.

[129] Hamilton, W.L., Ying, R. and Leskovec, J., 2017. Inductive Representation Learning on Large Graphs. arXiv preprint arXiv:1706.02216.

[130] Blei, D.M., Kucukelbir, A. and McAuliffe, J.D., 2017. Variational inference: A review for statisticians. Journal of the American Statistical Association, (just-accepted).

[131] Kingma, Diederik P and Welling, Max. Auto-encoding variational Bayes. In Proceedings of the International Conference on Learning Representations (ICLR), 2014.

[132] Kipf, T.N. and Welling, M., 2016. Variational Graph Auto-Encoders. arXiv preprint arXiv:1611.07308.

[133] Gómez-Bombarelli, R., Duvenaud, D., Hernández-Lobato, J.M., Hirzel, T.D., Aguilera-Iparraguirre, J., Adams, R.P. and Aspuru-Guzik, A., Automatic Chemical Design using Variational Autoencoders, 2016.

[134] Kusner, M.J., Paige, B. and Hernández-Lobato, J.M., 2017. Grammar Variational Autoencoder. arXiv preprint arXiv:1703.01925.

[135] Li, Y., Zhang, L. and Liu, Z., 2018. Multi-Objective De Novo Drug Design with Conditional Graph Generative Model. arXiv preprint arXiv:1801.07299.

[136] Dai, H., Dai, B. and Song, L., 2016, June. Discriminative embeddings of latent variable models for structured data. In International Conference on Machine Learning (pp. 2702-2711.

[137] Simonovsky, M. and Komodakis, N., 2018. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. arXiv preprint arXiv:1802.03480.

[138] Li, Y., Vinyals, O., Dyer, C., Pascanu, R. and Battaglia, P., 2018. Learning deep generative models of graphs. arXiv preprint arXiv:1803.03324.