HOW CAN WE GENERALISE LEARNING DISTRIBUTED REPRESENTATIONS OF GRAPHS?

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a general framework to construct unsupervised models capable of learning distributed representations of discrete structures such as graphs based on R-Convolution kernels and distributed semantics research. Our framework combines the insights and observations of Deep Graph Kernels and Graph2Vec towards a unified methodology for performing similarity learning on graphs of arbitrary size. This is exemplified by our own instance G2DR which extends Graph2Vec from labelled graphs towards unlabelled graphs and tackles issues of diagonal dominance through pruning of the subgraph vocabulary composing graphs. These changes produce new state of the art results in the downstream application of G2DR embeddings in graph classification tasks over datasets with small labelled graphs in binary classification to multi-class classification on large unlabelled graphs using an off-the-shelf support vector machine.

1 INTRODUCTION

A fundamental prerequisite for machine learning algorithms to *learn* about input data is the ability to discern one observation from another. Even more powerful is the ability to determine how *similar* or dissimilar such observations are from one another to make more detailed associations. For observations represented in Euclidean space with feature vectors the concept of similarity between observations is intuitive as it may be computed as *distance* using Euclidean distance or cosine similarity formulas. Unfortunately, for observations represented as graphs defining the notion of similarity, and even more so designing methods for computing similarity has been a long ongoing challenge in maths and computer science. This is partly because assessing comparability of graphs does not only consist of looking for similar elements (nodes) but also structural similarities between the substructures within. An obvious real world example on the importance of this distinction can be found in chemistry where molecules called isomers exhibit identical chemical formulas but different structural properties which induce different behaviours and traits (Petrucci et al., 2017). The difficulty of comparing graphs is highlighted by the graph isomorphism test (Garey & Johnson, 1990), which despite its complexity only gives a binary evaluation of the structural equivalence between two graphs which is insufficient for demands of fine grained machine learning tasks.

Consequently, the graph learning domain predominantly features kernel methods which approximate the comparability of graphs using invariants or substructures such as nodes, subgraphs and random walks within the graphs (Vishwanathan et al., 2010). Whilst they are powerful and intuitive, such kernels are dependent and often tied to certain methods such as support vector machines (SVM) to perform learning tasks (Yanardag & Vishwanathan, 2015). Hence, existing methods utilizing kernel methods are often unable to handle other downstream learning tasks such as regression or clustering without significant revision.

More recently, deep learning approaches for graph representation learning have gained significant research activity with the successful interpretation of graph convolutional methods for learning node representations (Kipf & Welling, 2017; Scarselli et al., 2009; Veličković et al., 2018). Representations at the graph level are then constructed through application of different pooling operations which aggregate node representations into a single representation for the entire graph (Ying et al., 2018; Goyal & Ferrara, 2018).

Our approach takes a different perspective through the distributive modelling of discrete higher order subgraph patterns across the graph dataset. This builds fixed size vector representations of graphs within a distributed vector space created through training a neural language model (Bengio et al., 2003). The positions of the distributed vector representations are contextualized by the subgraph patterns within graphs with respect to the patterns seen within other graphs across the dataset. This is analogous to the distributive modelling of arbitrary sized text documents in Le & Mikolov (2014). The construction and application of distributed representations of graphs can be summarized in a three step framework. The first step involves reducing graphs into higher order subgraph patterns to create *graph documents* summarizing the patterns seen in each graph. Together the graph documents constitute a *corpus* of graphs. The second stage trains a neural language model on the graph document corpus, thereby building a distributed embedding for each graph within the dataset. In the third and final stage the embeddings may be used with off-the-shelf learning systems for downstream tasks such as classification, regression, clustering, and so on. Alternatively one can even use the trained model for transfer learning purposes.

We implemented G2DR as an instance of this 3 stage framework and extension on Graph2Vec. For the first stage, G2DR decomposes graphs into subtree patterns using an algorithm inspired by the Weisfeiler-Lehman graph isomorphism test (Weisfeiler & Lehman, 1968; Shervashidze et al., 2011), extending the implementation in Narayanan et al. (2017) with an efficient iterative algorithm which also enables learning beyond labelled and undirected graphs to all combinations of labelled/unlabelled and directed/undirected graphs. For stage 2, we utilise a neural language model based on doc2vec (Le & Mikolov, 2014; Narayanan et al., 2017) with a minimum frequency threshold on the vocabulary to reduce the specificity of the subgraph pattern vocabulary and reduce the size of the neural network representing the neural language model. In the third stage we chose to evaluate G2DR on graph classification tasks with publicly available graph kernel datasets (Kersting et al., 2016). Datasets and tasks were selected to cover a range of graphs from small labelled graphs to large unlabelled graphs exhibiting complex substructures. To be comparatively fair within our assessment with kernel methods that inspire G2DR we utilised support vector machines (SVM) on the distributed representations and outperformed kernel methods as well as popular supervised graph neural methods. This helps validate the distributed perspective as a useful inductive bias (Battaglia et al., 2018) for constructing graph level representations alongside pooling efforts, and as a possible research avenue for more unsupervised techniques to graph-structured data.

In section 2 we provide a comprehensive background into graph learning and related methods such as graph kernels. Section 3 describes our 3 stage framework and G2DR in more detail. Section 4 describes the downstream application of the distributed representations in graph classification tasks with details about datasets and experimental setup. This is followed by reporting of results in section 5 and a discussion of the strengths and limitations of G2DR.

2 BACKGROUND AND RELATED WORK

Many real world phenomena such as chemical compounds (Petrucci et al., 2017), protein structures (Borgwardt et al., 2005), application process calls (Gascon et al., 2013), and social networks (Yanardag & Vishwanathan, 2015) can be naturally represented using graphs. For example, in chemistry the graph makes an intuitive model for a molecule where nodes represent atoms and edges the bonds between them. Here the graph is an appropriate representation as it captures not only the presence of the atoms in the molecule, but the edges also capture the specific bonding patterns between the atoms which is important for distinguishing isomers that a classical chemical formula cannot describe (Petrucci et al., 2017). In other words, the resulting graph topology created by the relationships between the nodes in a graph reveal a structural complexity that can be analysed as a source of information in pattern recognition problems.

Described more formally, a graph is an abstract structure which defines a set of entities which are related in some way. Graphs contain *nodes* representing said entities with related nodes being connected by an *edge* which records the relation. We define $\mathcal{G} = (V, E)$ as a graph where V is a set of nodes and $E \subseteq (V \times V)$ be a 2-tuple set of edges in the graph. Hence if u and v are nodes in $\mathcal{G}$, their relation is recorded with an edge as $(u, v) \in E$. The *neighbours* of a node v in graph $\mathcal{G} = (V, E)$, is the set of nodes which share an edge with v, denoted $\mathcal{N}(v) = \{u | (v, u) \in E\}$.

Graphs can be categorised depending on the attributes of the nodes and edges. A *labelled graph* is a graph whose nodes or edges have labels, which may or may not be unique. Nodes and/or edges can be labelled, with the graphs then being called node- or edge-labelled graphs respectively. Otherwise it is simply known as an *unlabelled graph*. Edges can either directed or undirected. Directed edges are uni-directional relations from a starting to node u to a target node v and recorded as $(u, v) \in E$ and $(u, v) \neq (v, u)$. Undirected edges describe bi-directional relationships between nodes u and v, hence $(u, v) = (v, u)$.

2.1 Graph learning and kernel methods

An operational assumption made in learning with graph-structured data is that similar phenomena represented by graphs will also exhibit similar topological properties. Hence the ability to quantify the similarity of graph topologies is central to graph learning algorithms. As topological patterns are not intuitively well represented using classic feature vectors, research has predominantly focused on using *kernel methods* for machine learning tasks involving graphs. Kernel methods are machine learning algorithms which rely on a *kernel* for the pattern recognition task. Kernels are functions which define a relation or more contextually, a similarity over pairs of data points using their raw representations. Subsequently one can use kernel methods which can operate on kernels such as support vector machines (SVMs) (Yanardag & Vishwanathan, 2015).

Ideally a kernel would be a similarity function $\text{sim}(\mathcal{G}, \mathcal{G}') = d, d \in \mathbb{R}^+$ where d or "distance" between graphs $\mathcal{G}$ and $\mathcal{G}'$ is small if they have similar structural properties, and a larger distance otherwise. The most intuitive measure of similarity is the binary indication of whether two graphs are topologically identical or *isomorphic*.

This is also known as the Graph Isomorphism (GI) test. Despite being a rudimentary measure of similarity, the complexity of the GI test is in NP and has neither been proven to be NP complete nor solved by a polynomial time algorithm (Garey & Johnson, 1990). Out of the twelve computational complexity problems listed in Garey and Johnson (Garey & Johnson, 1990), only the GI problem and integer factorisation remain unsolved. In addition to being computationally expensive, the binary measure of similarity provided by GI based measures requires graphs to be identical or contain large identical subgraphs in order to be considered similar. This is too restrictive to be used effectively by machine learning methods. As a result a number of more flexible kernels based on approximate and inexact matching of graphs were proposed to address this problem.

Examples of these approximate kernels include *graph edit distance* methods and *invariant based* methods. Graph edit distance methods as proposed by Bunke & Allermann (1983); Neuhaus & Bunke (2005) define a set of graph edit operations and associates a "cost" with each operation. The distance between the graphs can then be approximated by the minimum number and cost of edits needed to transform one graph into another. Slightly less intuitive, but powerful kernels exploit graph invariants. Kondor & Borgwardt (2008) introduced the skew spectrum where the invariant feature known as the graph skew is computed from the graph and extracted bispectral invariants can be compared into a kernel value. Less than a year later, Kondor et al. (2009) proposed the graphlet spectrum which computes a spectrum of matrices relative to a set of subgraphs. These features capture the number and position of the subgraphs which could then be compared between graphs.

Yanardag & Vishwanathan (2015) noted that kernel methods such as the graphlet spectrum are part a larger family of graph kernels which evaluate the similarity between graphs $\mathcal{G}$ and $\mathcal{G}'$ by decomposing them into atomic substructures such as random walks, shortest paths, graphlets, and subgraph patterns. The kernel value is then calculated by some function such as counting the number of common substructures over $\mathcal{G}$ and $\mathcal{G}'$. These kernel values would then be exploited by kernel methods performing the machine learning task. Such count based graph kernels can largely be grouped into three major families: those based on finite size subgraphs (Kondor et al., 2009; Horváth et al., 2004; Shervashidze et al., 2009), subtree patterns (Shervashidze et al., 2011; Shervashidze & Borgwardt, 2009; Ramon & Gärtner, 2003), and walks or paths (Borgwardt & Kriegel, 2005; Kashima et al., 2003; Vishwanathan et al., 2010).

Graph kernels are intuitive, efficient and perform well on smaller benchmark datasets however exhibit two limitations. Firstly, most kernels do not create explicit graph embeddings. This makes many out of the box machine learning algorithms that rely on vector embeddings such as Random Forests, Neural Networks, Naive Bayes, etc. unable to work with graph data. Secondly, the substructures which the graphs are decomposed to have to be determined manually with well defined functions that help extracting such substructures from graphs. When such substructures are used in very large datasets this can lead to building extremely high-dimensional, sparse, and non-smooth representations of graphs (Narayanan et al., 2017).

2.2 Deep learning approaches

More recently, success in the node classification tasks with graph convolutional networks (GCN) (Kipf & Welling, 2017) and successful translation of attention models (GAT) (Veličković et al., 2018) brought a flurry of attention to learning representations of substructures and whole graphs using deep learning. Whilst a majority of research in this regard has focused on learning representations of nodes (Goyal & Ferrara, 2018; Battaglia et al., 2018) methods have been proposed to pool or aggregate substructure representations into graph level representations (Goyal & Ferrara, 2018). A notable example is Ying et al. (2018) which hierarchically clusters and coarsens substructure representations towards a graph representation with stacks of graph neural networks. These have produced excellent empirical results on graph classification datasets overcoming some of the limitations of graph kernels on datasets with large graphs. However these methods face interesting challenges as they are sensitive to network initialisations and jumping knowledge structures, and simple structure-unaware MLPs and single layer GCN models have been shown to display comparable performance to the most powerful models despite their great algorithmic and memory complexity as shown in Luzhnica et al. (2019).

One deep learning approach that works slightly differently is Niepert et al. (2016) PATCHY-SAN. This method incorporates ideas from work in kernels and assigns labels to nodes using the labelling procedure from the Weisfeiler-Lehman kernel (Shervashidze et al., 2011), and sorts the node labels into a line. Subsequently, PATCHY-SAN defines a receptive field around each node by selecting a fixed number of nodes in d-hop neighbourhood where d is a natural number chosen by the practitioner. It then uses a standard convolutional neural network to learn a representation of the graph. This method features prominently as a state of the art supervised approach to graph classification.

2.3 Deep Graph Kernels and Graph2Vec

Our work is inspired by Deep Graph Kernels (Yanardag & Vishwanathan, 2015) and Graph2Vec (Narayanan et al., 2017). Yanardag & Vishwanathan (2015) recognised that the many graph kernel methods can be formulated as instances of the R-Convolutional kernel framework (Haussler, 1999) which decomposes discrete structures such as graphs into smaller substructure patterns to define kernels compatible with kernel methods like SVMs. Yanardag & Vishwanathan (2015) then utilised graph-edit distance methods or learning methods based on neural language models to compute a similarity matrix over the substructures to define kernel functions.

Graph2Vec (Narayanan et al., 2017) builds upon principles of Deep Graph Kernels. The authors defined a recursive subgraph decomposition algorithm based on the Weisfeiler-Lehman test (Weisfeiler & Lehman, 1968) to find subtree patterns for each node in labelled and undirected graphs, subtree patterns are recorded as strings in documents for each graph (we will delve into this algorithm further in the next section as we developed a general variant of this algorithm). A key deviation from Deep Graph Kernels is the subsequent input of the documents into a neural language model based on word2vec's skipgram architecture (Mikolov et al., 2013). This produced a scalable unsupervised approach to building distributed vector representations of graphs which captures generic structural properties of graphs. Yet the effectiveness of the embeddings, at least where data is publicly available, was only shown with classification datasets consisting of small labelled graphs. Nonetheless, the potential of the underlying methodology and potential motivates the general framework and G2DR we describe in the next section.

3 General methodology

We present a 3 stage framework for constructing models that combines the generality of Deep Graph Kernels and performance of Graph2Vec.

1. **Substructure pattern extraction**: The first stage consists of decomposing each graph into subgraph patterns using methods that respect the R-Convolution kernel framework, such as graphlets, subtree patterns, and walks. During this process a hash function is used to

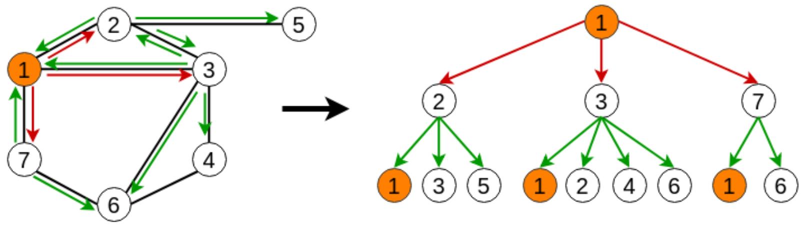

Figure 1: A subtree of degree $d = 2$ rooted at node 1. The red arrows are the trajectories of the first hop, whilst the green arrows are of the second hop. These hops generate a subtree pattern of height 2 with root node 1 seen on the right. Note that repeated visits of nodes is allowed.

record patterns to unique string labels within a dictionary which defines a *vocabulary* of the different subgraph patterns found across the dataset. The string labels of patterns found in a graph are recorded in an associated text document called a *graph document*. This process creates a collection of graph documents corresponding to each graph in the dataset forming a graph document *corpus* representative of the input graph dataset.

2. **Distributed representation learning**: This stage borrows concepts from statistical language modelling where the distributional hypothesis (Harris, 1954) is used to imbue *semantic meaning* to words and documents based on *context*. This distributional hypothesis suggests that words which are used and exist in the same context have similar meanings (Harris, 1954). We may adopt the same view about graphs by hypothesising that graphs which exhibit similar subgraph patterns entail similar properties and classifications. Therefore fixed distributed representations of graphs may be learned via neural language models such as word2vec (Mikolov et al., 2013), doc2vec (Le & Mikolov, 2014), or GloVe (Pennington et al., 2014) on the graph documents of stage 1.
3. **Downstream application:** The third and final stage involves the downstream application of the distributed representations learned in stage 2. As the learning models in stage 2 produce fixed-size vector representations of each graph, any off-the-shelf learning algorithm can readily be applied such as SVMs and multi-layer perceptrons for classification, or K-Means for clustering, or decision trees for regression. This opens up a host of use-cases for graph datasets that are enabled by the distributed vector representations.

3.1 G2DR

G2DR is an instance of this 3 stage methodology which specifically addresses some limitations of existing kernel methods and incorporates the distributive hypothesis to graph representations.

3.1.1 Stage 1: Subtree decomposition via Weisfeiler-Lehman reduction

In the first stage G2DR utilises a subtree decomposition algorithm based on the Weisfeiler-Lehman (WL) graph isomorphism test. Amongst the possible algorithms tackling graph isomorphism the Weisfeiler-Lehman (WL) test (Weisfeiler & Lehman, 1968) stands out as a particularly effective solution that differentiates graphs based on the composition of the subtree patterns extracted from the nodes, and works for most cases except those covered in Cai et al. (1992). In its essence, the WL test consists of iteratively assigning unique labels to the multisets formed by a nodes label with the labels of its neighbours. As this is performed on each node within the graph, the d^{th} iteration of node relabeling is equivalent to encoding a subtree pattern formed of a node and the neighbourhood within d-hops as shown in Figure 1. If the composition of assigned labels differs between graphs at any iteration, the WL test considers them non-isomorphic.

We wish to find subtree patterns of degree/depth d for every node within the graph of every graph in the dataset. This is achieved as a byproduct of the WL test's node relabeling (Shervashidze et al., 2011) and is fully described in Algorithm 1. G2DR utilises subtree patterns over other atomic substructures which could compose graphs for two reasons. Firstly, subtrees are higher order substructures over nodes which offer richer description of not only the nodes in a graph but the local neighbourhood structures which when sampled can give better embeddings of graphs. Secondly, compared to linear patterns such as walks and paths, non-linear subtree patterns capture the structure of node neighbourhoods more generally (Narayanan et al., 2017).

Algorithm 1: WL-Relabel $(\mathbb{G}, d)$

```
Input  : $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$ a set of $n$ graphs
         $d$ the desired degree of rooted subgraph found from each node
Output: $\mathbb{G}$ with each of the node relabellings made at each iteration saved in all nodes of all
         graphs
1 for $i = 0$ to $d$, by $1$ do
2     Multiset-Label determination
          • if $i = 0$, set $M_i(v) = l_0(v) = \lambda(v)$
          • if $i > 0$, assign a multiset-label $M_i(v)$ to each node $v$ in $\mathcal{G}$ for all $\mathcal{G} \in \mathbb{G}$ which
            consists of the multiset $\{l_{i-1}(u) | u \in N(v)\}$
3     Sorting each multiset
          • Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$
          • Concatenate $l_{i-1}(v)$ as a prefix to $s_i(v)$ and set the resulting string as $s_i(v)$
4     Label Compression
          • Sort all of the strings $s_i(v)$ for all $v$ form $\mathcal{G} \in \mathbb{G}$ in ascending order.
          • Map each string $s_i(v)$ to a new compressed label, using a hash function
            $f : \Sigma^* \to \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ iff $s_i(v) == s_i(w)$
5     Node Relabelling
          • Set $l_i(v) = f(s_i(v))$ for all nodes in $\mathcal{G} \in \mathbb{G}$
6 end
```

Algorithm 1, generates rooted subgraphs $t^{(d)}$ of depth d around every node v of graph $\mathcal{G} \in \mathbb{G}$. In this iterative algorithm starting from $i = 0$ to $i = d$, a hash function is initialised at the beginning of each iteration which maps the subgraph pattern to compressed multiset labels. When $i = 0$ no subgraph needs to be extracted and hence the original label of node v is returned (line 1). If the graph is unlabelled, the degree of a node is set as its label. Using degree as the initial node label can also apply to labelled graphs and will effectively construct a vocabulary of structural patterns (Shervashidze et al., 2011). When $i > 0$ the labels of the previous iteration are used to create new compressed labels. For each node v we find the previous iterations' labels of all v's neighbours as a multi set $\{l_{i-1}(u) | u \in \mathcal{N}(v)\}$ and sort the elements in ascending order. The sorted set is then turned into a string $s_i(v)$ with v's label in the previous iteration prepended to it. These strings are passed into the hash function to assign new compressed labels for the multiset string $s_i(v)$. Note that this makes sure that if the same subgraph pattern of v is found in another graph, that node is given the same compressed label due to the hashmap, simultaneously building a ”vocabulary” of the different subtree patterns in $\mathbb{G}$. Furthermore, at each iteration of this algorithm compressed labels $l_i(v)$ are produced which correspond to subtree patterns of depth i rooted at v.

At the end of algorithm 1 each graph $\mathcal{G}_j \in \mathbb{G}$ will have d relabellings for each of the nodes $v \in \mathcal{G}_j$ in string form $s_i(v)$ which we store as attributes inside of a NetworkX graph object (Schult, 2008). For each graph $G \in \mathbb{G}$, we generate a *graph document* by writing each of the compressed relabellings $s_i(v)$ for all $i \in [1, d]$ of every node $v \in G$ into a text file. All of the graph documents together form our corpus of graphs.

Algorithm 1 deviates from the algorithm presented in the Graph2Vec paper (Narayanan et al., 2017) where multisets representing subtree patterns are recursively constructed using the labels of neighbours of a target node for each node of all graphs in the dataset, and then assigned to that node. Instead the proposed WL-Relabel algorithm iteratively assigns compressed multiset labels based on a nodes previous multiset label, and in the case of an unlabelled graph, runs an initial relabelling based on the degree of the node. Our changes bring three advantages. One, it allows extraction of subgraph patterns for unlabelled graphs, allowing G2DR to learn distributed representations of unlabelled graphs and labelled graphs; and can apply directly to directed graphs. Two, during the iterative process, the subtree patterns of each degree prior to the user specified degree are observed and can be used or saved without finding them again. Three, the compressed labels are a memory efficient alternative to saving entire subgraph patterns for each node.

3.1.2 Tackling Zipf's Law in Vocabulary Size

The degree of the subtrees to be extracted from the graphs in the dataset directly affects the vocabulary size eventually used in the training of the neural language model. As the degree to be analysed increases, the number of unique subgraphs which enter the vocabulary also increases (Yanardag & Vishwanathan, 2015). The increase in specificity allows very similar graphs to be drawn even closer, and is useful in small datasets with small graphs and low numbers of distinct labels where finding patterns that set graphs apart may be useful. However, as the complexity and size of graphs and graph datasets increases, the number of substructure patterns explodes leading to extraction of patterns which occur infrequently in the dataset, which is a limitation of graph kernels in general. A pattern to frequency analysis will show that the 'language' of subgraph patterns follows Zipf's law (Powers, 1998). Zipf's empirical law describes how the frequency of a word in a given language is inversely proportional to its rank in a frequency table. The phenomenon brings the forth diagonal dominance seen in graph kernels (Yanardag & Vishwanathan, 2015). A naive yet natural fix which addresses the untuned network parameters corresponding to infrequent subgraphs is to modify the corpus and vocabulary so that a rooted subgraph has to occur a set minimum number of times in the dataset to be included in the vocabulary and hence the language model. We chose to set a minimum threshold of 2 occurrences of a subgraph pattern to be included in the vocabulary if the half of the tail of single occurrences was larger than the number of graphs in the dataset.

3.1.3 Stage 2: Modified PV-DBOW with negative sampling

Once the graph documents have been generated using algorithm 1 it is possible to learn a distributional vector space in a completely unsupervised manner using a neural language model. We follow Graph2Vec (Narayanan et al., 2017) and use a PV-DBOW model with negative sampling (Le & Mikolov, 2014; Mikolov et al., 2013) as outlined in Algorithm 2.

```
Algorithm 2: Train-Graph2Vec ($\mathbb{G}, D, \delta, \mathfrak{e}, \alpha$)
Input  : $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$ a set of $n$ graphs
         $D$ the desired degree of rooted subgraph found from each node
         $\delta$ the desired dimensionality of the vector embeddings
         $\mathfrak{e}$ the number of epochs to train the neural network
         $\alpha$ the learning rate
Output: $\Phi$ Matrix of distributed representations of the graphs $\Phi \in \mathbb{R}^{|\mathbb{G}| \times \delta}$
1  Initialisation of $\Phi$: randomly sample $\Phi$ from $\mathbb{R}^{|\mathbb{G}| \times \delta}$;
2  for $e \leftarrow 0$ to $\mathfrak{e}$, by $1$ do
3      $\mathfrak{G}$ = SHUFFLE($\mathbb{G}$);
4      foreach $G_i \in \mathfrak{G}$ do
5          foreach $v \in V_i$ do
6              for $d = 0$ to $D$, by $1$ do
7                  $t_v^{(d)} = GetSubgraph(G_i, v, d)$;
8                  $J(\Phi) = -\log P(t_v^{(d)} | \Phi(G_i))$;
9                  $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$;
10             end
11         end
12     end
13 end
```

Calculating the posterior probability in line 9 of the algorithm can be prohibitively expensive because $t_v^{(d)} \in \mathbb{T}_{\text{vocab}}$ and the size of the vocabulary, $|\mathbb{T}_{\text{vocab}}|$ can be very large. To mitigate this problem, the neural language model is trained with negative sampling as suggested by Mikolov et al. (2013). For every training epoch, given a graph $\mathcal{G}_i \in \mathbb{G}$ and a set of subtrees in its context, $\mathfrak{C}(\mathcal{G}_i) = \mathfrak{c} =$

$\{t_{i1}, t_{i2}, ...\}$, a fixed number of k subgraphs are randomly selected as negative samples forming $\mathfrak{c}' = \{t_{i1}, t_{i2}, ..., t_{ik}\}$ such that $\mathfrak{c}' \subset \mathbb{T}_{\text{vocab}}$ and $k << |\mathbb{T}_{\text{vocab}}|$ and $\mathfrak{c} \cap \mathfrak{c}' = \{\}$. In other words the negative samples $\mathfrak{c}'$ for graph $\mathcal{G}_i$ is a set of k samples from $\mathbb{T}_{\text{vocab}}$ that are not present in $\mathfrak{c}(\mathcal{G}_i)$. For training, the target-context pairs $(\mathcal{G}_i, \mathfrak{c})$ are trained and the embeddings of the corresponding subgraphs are updated. Subsequently only the embeddings of the negative samples $\mathfrak{c}'$ are updated, instead of the whole vocabulary. The number of negative samples selected is a hyperparameter, and G2DR uses a default of 10 negative samples.

The training process produces representations which achieves the original kernel design aims. Given a pair of graphs $\mathcal{G}_i$ and $\mathcal{G}_j$, Train-Graph2Vec makes their vector embeddings $\Phi(\mathcal{G}_i)$ and $\Phi(\mathcal{G}_j)$ closer if they share or have similar subgraphs in their contexts, i.e. $\mathfrak{c}(\mathcal{G}_i)$ is similar to $\mathfrak{c}(\mathcal{G}_j)$.

3.1.4 Stage 3: Downstream application

After training the graph neural language model, Φ contains the distributed representations of every graph $\mathcal{G}_i \in \mathbb{G}$, where the embedding of $\mathcal{G}_i$ is denoted $\Phi(\mathcal{G}_i)$. The task agnostic nature of the embeddings allows any downstream task such as classification, regression, clustering, etc. to be performed on the graphs of $\mathbb{G}$ simply by applying the appropriate methods which accept vector inputs.

4 Evaluation on downstream graph classification

We utilised the embeddings trained on graph classification datasets to assess the suitability of the approach, and as a benchmark against other graph learning methods. One of the key motivations of this work was to assess the suitability of the distributed representations beyond small labeled graphs towards larger (also unlabelled) graphs where the limitations of specification in graph kernels were expected to show. For ease of replicability a suite of public datasets were selected from Kersting et al. (2016) based on frequency in papers and size. The selection of benchmark datasets and tasks range from smaller, classic datasets with graphs averaging 17 nodes in size tasked on binary classification, to significantly larger graphs averaging over 500 nodes in size over 11929 examples in a multi-community prediction task. This list includes the MUTAG (Debnath et al., 1991) , PTC (Helma et al., 2001), PROTEINS (Borgwardt et al., 2005) , NCI1, NCI109 (Wale et al., 2008), REDDIT-BINARY, REDDIT-MULTI-5K, REDDIT-MULTI-12K (Yanardag & Vishwanathan, 2015) datasets. The datasets represent applications from a number of research domains including: chemoinformatics, bioinformatics, and social networks whose properties are summarised in Table 3 in the supplementary material.

We were particularly keen to investigate the case of unlabelled graphs and large datasets with multiple graph labels where the 'large' applies to both the number of data points as well as the size of the graphs therein. The large graphs make the representation learning inherently more complex as the number of expected subgraph patterns substantially increases (Yanardag & Vishwanathan, 2015) and questions the necessary specificity required to compare large graphs to each other as well as large graphs to small graphs.

4.1 Experimental setup

For fairer comparative analysis with graph kernel methods and experimental setups of Deep Graph Kernels Yanardag & Vishwanathan (2015) and Graph2Vec Narayanan et al. (2017), G2DR uses an off-the-shelf C-Support Vector Machine implemented in the SciKit-Learn package (Pedregosa et al., 2011) on the learned embeddings for each dataset. 10 fold cross validation was used to evaluate test classification accuracy. The C value for each fold was independently tuned using training data from that fold. In order to exclude random effects of the fold assignments, each experiment was repeated 10 times, and mean classification accuracies with their standard deviations were recorded.

Results are presented over two section, one dedicated to datasets whose graphs are labelled, and the other dedicated to datasets of graphs which are initially unlabelled and hence get labelled through Algorithm 1 using the degree of the nodes as in Shervashidze et al. (2011). For comparative purposes of existing methods we have selected prominent methods applicable to each of the graph types and recorded the results they have published.

Table 1: Mean classification accuracy and standard deviation on labelled graph classification

Dataset	MUTAG	PTC	PROTEINS	NCI1	NCI109
node2vec	72.62 ± 10.20	55.85 ± 8.00	57.49 ± 3.57	54.89 ± 1.61	52.68 ± 1.56
sub2vec	61.05 ± 15.79	59.99 ± 6.38	53.03 ± 5.55	52.84 ± 1.47	50.67 ± 1.50
Graphlet Kernel	81.66 ± 2.11	57.26 ± 1.41	71.67 ± 0.55	62.28 ± 0.29	62.60 ± 0.19
Shortest Path Kernel	85.22 ± 2.43	58.24 ± 2.44	75.07 ± 0.54	73.00 ± 0.24	73.00 ± 0.26
WL Kernel	80.72 ± 3.00	56.97 ± 2.01	72.92 ± 0.56	80.13 ± 0.50	80.22 ± 0.34
Deep GK	82.66 ± 1.45	57.32 ± 1.13	71.68 ± 0.50	62.48 ± 0.25	62.69 ± 0.23
Deep SP	87.44 ± 2.27	60.08 ± 2.55	75.68 ± 0.54	73.55 ± 0.51	73.26 ± 0.26
Deep WL	82.94 ± 2.68	59.17 ± 1.56	73.30 ± 0.82	80.31 ± 0.46	80.32 ± 0.33
Graph2Vec	83.15 ± 9.25	60.17 ± 6.86	73.30 ± 2.05	73.22 ± 1.81	74.26 ± 1.47
PATCHY-SAN (K=10)	88.95 ± 4.67	62.29 ± 5.68	75.89 ± 2.76	76.34 ± 1.68	76.37 ± 1.43
G2DR + SVM	**92.13 ± 0.06**	**67.14 ± 0.08**	**76.42 ± 0.03**	**84.91 ± 0.02**	**84.35 ± 0.02**

G2DR was experimented using embedding dimensions of 16, 32, 64, 128, 256, 512, 1024, 2048 across a consideration of subtree degrees of 1,2,3. For each configuration the neural network was trained over 1000 epochs with an initial learning rate of 0.5 set on an linear decay over the epochs towards 0.001. A negative sampling size of 10 was used throughout as in Graph2Vec (Narayanan et al., 2017). The best downstream results on the different embedding configurations is reported in the results of the next section.

4.2 Results: Labelled graphs

Table 3 shows that the labelled graph datasets have a range of different properties, including datasets with small graphs and a small number of distinct labels such as the MUTAG dataset to larger datasets with larger graphs and number of distinct labels. It was expected that lower-order substructure embedding methods such as node2vec (Grover & Leskovec, 2016) and sub2vec (Adhikari et al., 2017) would perform well in datasets of small graphs with few labels, but gradually perform worse as the graphs became more complex. As shown in Table 1, these methods initially perform well on smaller datasets, but suffer from the naive agglomeration schemes for larger datasets, barely performing better than random on NCI1 and NCI109. Sub2vec samples only one random walk of fixed length from the given graph and learns its representations using fixed length linear context skipgram models. This prevents sub2vec from learning structurally meaningful embeddings of the entire graph, and suffers considerably when applied to larger graphs as can be seen in the difference between MUTAG and NCI109.

The graph kernels perform better overall, as they have been purpose built for this task, with the WL-kernel performing particularly well in the larger datasets, which also initially prompted the decision to use subtrees in G2DR. The deep graph kernels (Yanardag & Vishwanathan, 2015) attempt to overcome the limitation of previous methods by addressing the sparsity problem of graph kernels. This raises the performance significantly over node2vec and sub2vec, however only marginally better than the traditional graph kernels upon which they are based. This is attributed to the assumption of linear context between substructures and the assumption that every substructure in any given graph has an identical number of subgraphs in its context. The latter assumption is too strict and violates the general neighbourhood structure of nodes and subgraph patterns in graphs.

The proposed embedding learning system combined with an out-of-the-box C-SVM returns highly competitive results. The system returns the best mean classification accuracy across the labelled graph datasets. The success of this method is attributed to the descriptive power of non-linear patterns extracted through subtrees and the usage-based representations learnt of the subgraphs within the graphs leading to similar graphs being closer to each other, and dissimilar graphs more distant, allowing the C-SVM to compute more effective hyper planes which maximise the separation of graphs of different classes. We also attribute the improvement over Graph2Vec on labelled graphs mainly on the consideration of vocabulary pruning which overcomes issues of diagonal dominance in graph comparison (Yanardag & Vishwanathan, 2015).

Table 2: Mean classification accuracy and standard deviation on unlabelled graph classification

Datasets	REDDIT-BINARY	REDDIT-5K	REDDIT-12K
Graphlet Kernel	77.34 ± 0.18	41.01 ± 0.17	31.82 ± 0.08
Deep GK	78.04 ± 0.39	41.27 ± 0.18	32.22 ± 0.10
PATCHY-SAN (K=10)	86.30 ± 1.58	49.10 ± 0.70	41.32 ± 0.42
2D-CNN	**89.12 ± 1.70**	52.11 ± 2.24	**48.13 ± 1.47**
G2DR + SVM	83.25 ± 0.01	**53.48 ± 0.02**	41.44 ± 0.02

4.3 Results: Unlabelled graphs

The social network datasets of Yanardag & Vishwanathan (2015), distinguish themselves from the previous set of graphs in 3 ways. Firstly, the graphs within the dataset are unlabelled hence the proposed system labels the nodes within the graphs by their degree and cannot be utilised in the unmodified Graph2Vec system. Secondly, the graphs are considerably larger than those seen in the classic benchmarks, having a direct effect on the complexity and number of substructures within, which in turn have a significant effect on the vocabulary size used in the neural language model. Finally, for REDDIT-5K and REDDIT-12K, is a multi-class classification task as opposed to the previous binary classification tasks. As a result, several of the previous methods are unsuitable for this task, and we have included another system, 2D-CNN (Tixier et al., 2019), which represents graphs as multi-channel image-like structures so that they can be fed into image convolutional neural networks. This methodology produces the best results amongst the considered approaches.

As seen in table 2, G2DR performs better than the GK kernel and the Deep Graph Kernel variant Deep GK as expected. However, it fails to achieve the performances of 2D-CNN, as the specificity and sheer number, of unique subgraph patterns makes limitations of Graph2Vec apparent. A quick investigation showed that the lower performance could partly be attributed to the extremely large vocabulary of subtrees that is produced from the combination of very large complex graphs and a high number of data points in the dataset. This manifests into two related problems in the system. Foremost, whilst Graph2Vec reduces the effect of diagonal dominance by exploiting the distributional hypothesis and dimensionality reduction, the sheer number of infrequently occuring subgraphs (relatively untuned from its initial random state in the corresponding weights of the neural language model) can inadvertently cause diagonal dominance. Secondly and on a related note, the size of the subtree vocabulary $\mathbb{T}_{\text{vocab}}$ sets the number of weights and biases in the neural language model, the large number of parameters makes minimisation of the loss function more difficult and computationally intensive. Hence, even models using subtree patterns of lower degree, which would otherwise be too unspecific to find meaningful similarities/dissimilarities fare better in contrast to the case in smaller datasets. This suggests that better methods will be needed to tackle the issue of sparsity and specificity of the vocabularies utilised to train the distributed vector space models of larger graphs, as well as classification tasks considering more than two classes.

5 Conclusion

We have presented a general framework for constructing models which learn distributed representations of graphs, that can be further generalised to other discrete structures under the R-Convolution kernel framework. This combines the theoretic frameworks of Deep Graph Kernels and advancements under Graph2Vec exemplified by our own instance, G2DR, which builds upon both previous works. G2DR is a scalable system which can learn task agnostic distributed representations of unlabelled/labelled and directed/undirected graphs of arbitrary size automatically. The suitability of this unsupervised approach is validated against graph kernels and supervised neural approaches in downstream graph classification tasks, where G2DR displays strong state of the art results despite its unsupervised nature and use of an off-the-shelf SVM. This enables the application of G2DR as a tool to numerous real world questions on graphs in different research domains. Furthermore, this suggests that the distributional perspective of graphs as compositions of discrete substructures is an useful inductive bias for building smooth vector representations of graphs along other research in hierarchical graph coarsening and substructure pooling.

References

Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B. Aditya Prakash. Distributed representations of subgraphs. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 111–117, 2017.

Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018. URL `https://arxiv.org/pdf/1806.01261.pdf`.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=944919.944966`.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pp. 74–81, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2278-5. doi: 10.1109/ICDM.2005.132.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1): 47–56, January 2005. ISSN 1367-4803.

H Bunke and G Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245 – 253, 1983. ISSN 0167-8655. doi: https://doi.org/10.1016/0167-8655(83)90033-8. URL `http://www.sciencedirect.com/science/article/pii/0167865583900338`.

Jin-Yi Cai, Martin Fürer, and Neil Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, Dec 1992. ISSN 1439-6912. doi: 10.1007/BF01305232. URL `https://doi.org/10.1007/BF01305232`.

Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, Feb 1991. ISSN 0022-2623. doi: 10.1021/jm00106a046.

Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455.

Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. Structural detection of android malware using embedded call graphs. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*, AISec '13, pp. 45–54, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2488-5. doi: 10.1145/2517312.2517315.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78 – 94, 2018. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2018.03.022. URL `http://www.sciencedirect.com/science/article/pii/S0950705118301540`.

Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 855–864, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939754. URL `http://doi.acm.org/10.1145/2939672.2939754`.

Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954. doi: 10.1080/00437956.1954.11659520.

David Haussler. Convolution kernels on discrete structures. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA, 1999.

C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000 2001. *Bioinformatics*, 17(1):107–108, 2001. doi: 10.1093/bioinformatics/17.1.107.

Tamás Horváth, Thomas Gärtner, and Stefan Wrobel. Cyclic pattern kernels for predictive graph mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pp. 158–167, New York, NY, USA, 2004. ACM. ISBN 1-58113-888-1. doi: 10.1145/1014052.1014072.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 321–328. AAAI Press, 2003. ISBN 1-57735-189-4. URL `http://dl.acm.org/citation.cfm?id=3041838.3041879`.

Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016. URL `http://graphkernels.cs.tu-dortmund.de`.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

Risi Kondor and Karsten M. Borgwardt. The skew spectrum of graphs. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pp. 496–503, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390219. URL `http://doi.acm.org/10.1145/1390156.1390219`.

Risi Kondor, Nino Shervashidze, and Karsten M. Borgwardt. The graphlet spectrum. In *ACM International Conference Proceeding Series*, volume 382, pp. 67, 01 2009.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. II–1188–II–1196. JMLR.org, 2014. URL `http://dl.acm.org/citation.cfm?id=3044805.3045025`.

Enxhell Luzhnica, Ben Day, and Pietro Liò. On graph classification networks, datasets and baselines. In *36th International Conference on Machine Learning*, ICLM'19, 2019.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL `http://arxiv.org/abs/1301.3781`.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *CoRR*, abs/1707.05005, 2017. URL `http://arxiv.org/abs/1707.05005`.

M. Neuhaus and H. Bunke. Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(3):503–514, June 2005. ISSN 1083-4419. doi: 10.1109/TSMCB.2005.846635.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. *CoRR*, abs/1605.05273, 2016. URL `http://arxiv.org/abs/1605.05273`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

R.H. Petrucci, F.G. Herring, J.D. Madura, and C. Bissonnette. *General Chemistry: Principles and Modern Applications*. Pearson Education, 2017. ISBN 9780133400588. URL `https://books.google.co.uk/books?id=E1CTDAAAQBAJ`.

David M. W. Powers. Applications and explanations of zipf's law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, NeMLaP3/CoNLL '98, pp. 151–160, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics. ISBN 0-7258-0634-6. URL `http://dl.acm.org/citation.cfm?id=1603899.1603924`.

Jan Ramon and Thomas Gärtner. Expressivity versus efficiency of graph kernels. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences*, pp. 65–74, 2003.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *Trans. Neur. Netw.*, 20(1):61–80, January 2009. ISSN 1045-9227. doi: 10.1109/TNN.2008.2005605. URL `http://dx.doi.org/10.1109/TNN.2008.2005605`.

Daniel A. Schult. Exploring network structure, dynamics, and function using networkx. In *In Proceedings of the 7th Python in Science Conference (SciPy)*, pp. 11–15, 2008.

Nino Shervashidze and Karsten M. Borgwardt. Fast subtree kernels on graphs. In *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, NIPS'09, pp. 1660–1668, USA, 2009. Curran Associates Inc. ISBN 978-1-61567-911-9. URL `http://dl.acm.org/citation.cfm?id=2984093.2984279`.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In David van Dyk and Max Welling (eds.), *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pp. 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *J. Mach. Learn. Res.*, 12:2539–2561, November 2011. ISSN 1532-4435.

Antoine J.-P. Tixier, Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Graph classification with 2d convolutional neural networks. In Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2019: Workshop and Special Sessions*, pp. 578–593, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30493-5.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *J. Mach. Learn. Res.*, 11:1201–1242, August 2010. ISSN 1532-4435.

Nikil Wale, Ian A. Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14(3):347–375, March 2008. ISSN 0219-1377. doi: 10.1007/s10115-007-0103-5.

B. Weisfeiler and A. A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 9(9):12–16, 1968.

Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1365–1374, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2783417. URL `http://doi.acm.org/10.1145/2783258.2783417`.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4805–4815, USA, 2018. Curran Associates Inc. URL `http://dl.acm.org/citation.cfm?id=3327345.3327389`.

A SUPPLEMENTARY MATERIAL: FIGURES AND TABLES

Datasets	Size	Classes	Avg. Nodes	Distinct Labels
MUTAG	188	2	17.93	7
PTC	344	2	25.5	19
PROTEINS	1113	2	39.06	3
NCI1	4110	2	29.87	37
NCI109	4127	2	29.68	38
REDDIT-BINARY	2000	2	429.61	unlabelled
REDDIT-MULTI-5K	4999	5	508.52	unlabelled
REDDIT-MULTI-12K	11929	11	391.41	unlabelled

Table 3: Properties of the benchmark graph classification datasets.