Evaluating the Quality of Graph Embeddings via Topological Feature Reconstruction

Stephen Bonner [†], John Brennan [†], Ibad Kureshi [†], Georgios Theodoropoulos [§],
Andrew Stephen McGough [‡] and Boguslaw Obara [†]
[†] *Department of Computer Science, Durham University, Durham, UK*
{s.a.r.bonner, j.d.brennan, ibad.kureshi, boguslaw.obara}@durham.ac.uk
[‡] *School of Computing, Newcastle University, Newcastle, UK stephen.mcgough@newcastle.ac.uk*
[§] *School of Computer Science and Engineering, SUSTech, Shenzhen, China georgios@sustec.edu.cn*

***Abstract*—In this paper we study three state-of-the-art, but competing, approaches for generating graph embeddings using unsupervised neural networks. Graph embeddings aim to discover the 'best' representation for a graph automatically and have been applied to graphs from numerous domains, including social networks. We evaluate their effectiveness at capturing a good representation of a graph's topological structure by using the embeddings to predict a series of topological features at the vertex level. We hypothesise that an 'ideal' high quality graph embedding should be able to capture key parts of the graph's topology, thus we should be able to use it to predict common measures of the topology, for example vertex centrality. This could also be used to better understand which topological structures are truly being captured by the embeddings. We first review these three graph embedding techniques and then evaluate how close they are to being 'ideal'. We provide a framework, with extensive experimental evaluation on empirical and synthetic datasets, to assess the effectiveness of several approaches at creating graph embeddings which capture detailed topological structure.**

***Keywords*-graph embeddings; feature learning; deep learning**

I. Introduction

Graphs are a key and widely used construct for representing inherent relationships within datasets across social networks and many other different domains. The structure or topology of a graph can reveal important and unique insights into the data. Recently, analysing and making predictions about graph topology using modern machine learning techniques has shown significant advances over traditional approaches for a range of commonly performed tasks within graph mining, such as predicting new links within the graph – suggesting new friendship links – or classifying [1] – types of users in social network. However, graphs are inherently complex structures that do not fit well as input into existing machine learning methods, which often require a vector of real numbers as input.

We adopt here the commonly used notation for representing a graph or network[1] $G = (V, E)$ as an undirected graph which comprises a finite set of vertices (sometimes referred to as nodes) V and a finite set of edges E. The elements of E are an unordered tuple $\{u, v\}$ of vertices $u, v \in V$. Here G could be a graph-based representation of a social, citation or biological network [2]. The adjacency matrix $\mathbf{A} = (a_{i,j})$ for a graph is symmetric matrix of size $|V|$ by $|V|$, where $(a_{i,j})$ is 1 if an edge is present and 0 otherwise.

Graph embeddings are a set of techniques which learn latent representations of a graph, which can then be used as input to machine learning models for a certain downstream prediction task [3]. Thus, graph embeddings are becoming a key area of research as they act as a translation layer between a raw graph representation and some associated model. Due to their importance, a large number of competing techniques for learning a representation of the graph, each aiming to capture the most information from the graph's topology, have emerged in recent years [4].

The goal of all graph embedding techniques is the same; to transform a complex graph, with no inherent representation in vector space, into a low-dimension vector representation of the graph or its elements. Crucially, these low-dimensional representations can be utilised as input to other machine learning models and used for tasks such as classification [5]. The majority of graph embedding techniques have focused upon learning vertex embeddings [4] and reconstructing missing edges [3]. As such, the goal of a graph embedding technique is to learn some function $f : V \rightarrow \mathbb{R}^d$ which is a mapping from the set of vertices V to a set of embeddings for the vertices, where d is the required dimensionality of the resulting embedding. This results in f being a matrix of dimensions $|V|$ by d, i.e. an embedding of size d for each vertex in the graph. It should be noted that this mapping is intended to capture the latent structure from a graph, for example mapping similar topological features to similar positions within the vector space.

Currently, graph embedding approaches are designed primarily with a specific goal in mind, for example to improve classification of end users (vertices) within social networks [5], or to predict new edges (i.e. recommending new products) [3]. However, to date, little work has been

[1]To avoid confusion with neural networks we will use the term graph throughout the remainder of the paper without loss of generality.

performed on exactly which topological properties of a graph are preserved in the resulting embedding.

In this paper, we study three state-of-the-art, but competing, approaches for generating graph embeddings using unsupervised neural network techniques and aim to identify how much of the graphs topological structure is still identifiable after the embedding process. Here we hypothesise that a high quality graph embedding should be able to capture key parts of the graphs topology, thus, the resulting representation should be able to predict common measures of the original topology. We evaluate the approaches effectiveness at capturing a good representation of the graph's topology by predicting a series of topological features at the vertex level. This process allows us to explore which, if any, of these topological features is being captured in the embedding process. Topological features are a known way to accurately identify graphs and vertices [6] [7], however it is not currently known if graph embeddings are learning analogous features from the data, or if unique and novel features are being learned. We hypothesise that by predicting known topological features directly from the embeddings, we can go some way to identifying what, if any, existing structures the embeddings are using to represent vertices. We make the following contributions whilst exploring these issues:

- We propose a new framework for assessing the quality of graph embedding techniques, which directly measures their ability to capture a good representation of a graph's topology. We use this to explore which, if any, of the conventional topological properties are being learned in the embedding process.
- We directly compare the performance and runtime of three state-of-the-art graph embedding techniques, which to the best of our knowledge is the first time such a comparison has been performed.

In Section II we explore prior work, in Section III we discuss the different graph embedding approaches, in Section IV we detail our approach for evaluating the quality of graph embeddings, in Section V we present our results and in Section VI we conclude the paper along with suggesting further expansions of this work.

II. Previous Work

In this section we briefly explore the prior research regarding graph embedding techniques and any previous attempts to measure their quality at topological feature prediction.

A. Graph Embeddings

Being able to automatically discover some numerical based representation for a given graph is significant advantage and provides a timely solution to a common problem within the field of graph mining. Traditional approaches have relied upon extracting features – such as various measures of a vertices' centrality [8] – capturing the required information about a graph's topology, which could then be used in some down-stream prediction task [6] [7] [9]. However, such a feature extraction based approach relies solely upon the hand-crafted features being a good representation of the target graph. Often a user must use extensive domain knowledge to select the correct features for a given task, with a change in task often requiring the selection of new features [6].

Graph embeddings offer a way to learn a graph's topology in either a supervised or un-supervised manner, removing the need for a user to manually select representative features. For a recent review covering the complete history of graph embeddings, readers are referred to [10]. Our work focuses on *neural network* based approaches for graph embedding (as these have demonstrated superior performance compared with traditional approaches [4]), hence our review is limited to these.

The study of *Neural Networks (NNs)* is a field within machine learning inspired by the human brain [11]. NNs model problems via the use of connected layers of artificial neurons, where each network has an input layer, at least one hidden layer and an output layer. The activation of each neuron in a layer is given by a pre-specified function, with each neuron taking a weighted sum of all the outputs of those neurons to which it is connected. These weights are *learned* through training examples which are fed through the network, with modifications made to the weights via back-propagation to increase the probability of the NN producing the desired result [11].

1) Supervised Approaches: In the field of machine learning, supervised learning is perhaps the most studied and understood [11]. In supervised learning, the datasets contain labels which help guide the NN in the learning process. In the field of graph analysis, these labels are often present at the vertex level and contain, for example, the meta-data of a user in a social network.

Perhaps the most studied area of supervised graph embeddings is that of *Graph Convolutional Neural Networks (GCNs)* [12], both spectral [13] and spatial [14] approaches. Such approaches pass a sliding window filter over a graph, in a manner analogous with Convolutional Neural Networks from the computer vision field [11], but with the neighbourhood of a vertex replacing the sliding window. Current GCN approaches are supervised and thus require labels upon the vertices. This requirement has two significant disadvantages: Firstly, it limits the available graph data which can be used due to the requirement for labelled vertices. Secondly, it means that the resulting embeddings are specialised for one specific task and cannot be generalised for a different problem without costly retraining of the model for the new task.

2) Unsupervised Approaches: As these approaches form the main focus of our work, the methods for unsupervised graph embeddings are explored fully in Section III.

B. Evaluating Embedding Quality

To date, there has been little work exploring exactly how much of the rich topological structure from a given graph is captured by graph embedding models. However, recently Goyal and Ferrar [4] have presented a review paper on a selection of graph embedding techniques. The authors mostly focus on vertex classification (i.e. classifying a user in a social network) as a judgement of the embedding quality, which is not necessarily dependant upon capturing the best representation of the graph's topology. In addition, the authors do not consider embeddings taken from promising unsupervised techniques – such as the family of hyperbolic approaches, or the runtime performance nor; do they explore performance across imbalanced classes.

Lui *et al.* [15] explore the use of a graph's topological features as a way of validating the accuracy of a neural network based generative model they have developed. With their model, they aim to generate entirely new graph datasets which mimic the topological structure of a set of target graphs – a common task within the graph mining community [16]. To validate the quality of their model, they investigate if a new graph created from their generative procedure has a similar set of topological features to the original graph.

Hamilton *et al.* [17] speculate on the use of a graph's topological features as a way to improve the quality of vertex embeddings by incorporating them into a supervised GCN based model. They show how aggregating a vertex feature – even as simple as its degree – can improve the performance of their model. They present theoretical analysis to validate that their approach is able to learn the number of triangles a vertex is part of, arguing that this demonstrates the model is able to learn topological structure. We take inspiration from this work, but assess unsupervised approaches as well as exploring richer and more complicated topological features as a measure of embedding quality.

Many unsupervised graph embedding approaches have adapted models originally designed for language modelling [3] [5]. A recent work investigated how best to evaluate a variety of unsupervised approaches for embedding words into vectors [18]. They choose a variety of *Natural Language Processing (NLP)* tasks, which capture some aspects of the structure of language, and investigate how well the chosen embedding models perform. Interestingly, they conclude that no single word embedding model is "best" across all the tasks they devised. In this paper, we aim to explore how much of a graph's structure is captured by a selection of graph embedding approaches.

III. Unsupervised Graph Embeddings

There are numerous competing unsupervised graph embedding techniques available. In this section we will detail three of the current state-of-the-art approaches we will be evaluating in this paper.

A. Stochastic Embeddings

DeepWalk [5] and *Node2Vec* [3] are the two main approaches for random walk based embedding. Both of these approaches borrow key ideas from a technique entitled *Word2Vec* [19] designed to embed words, from a sentence, into vector space. The *Word2Vec* model is able to learn an embedding for a word by using surrounding words within a sentence as targets for a single hidden layer neural network model to predict. Due to the nature of this technique, words which co-occur together frequently in sentences will have positions which are close within the embedding space. The approach of using a target word to predict neighbouring words is entitled *Skip-Gram* and has been shown to be very effective for language modelling tasks [20].

1) DeepWalk and Node2Vec: These approaches are able to use this same technique for graphs instead of sentences, with vertices taking the place of words. The key insight of *DeepWalk* is to use random walks upon the graph, starting from each vertex, as the direct replacement for the sentences required by *Word2Vec*. A random walk can be defined as a traversal of the graph rooted at a vertex $v_t \in V$, where the next step in the walk is chosen uniformly at random from the vertices incident upon v_t [21], these walks are recorded as $w_0^t, ..., w_n^t$ (where t is the walk starting from v_t of length n, and $w_i^t \in V$), i.e. a sequence of the vertices visited along the random walk starting from $v_t = w_0^t$. *DeepWalk* is able to learn unsupervised representations of vertices by maximising the average log probability $\mathbf{P}$ over the set of vertices V:

$$\frac{1}{|V|} \sum\nolimits_{t=1}^{|V|} \ \sum\nolimits_{i=0}^{n} \sum\nolimits_{-c \le j \le c, j \ne 0} \log \mathbf{P}(w_{i+j}^t | w_i^t), \quad (1)$$

where c is the size of the training context[2] of vertex w_n^t. Note if $i + j < 0$ then we skip these from the sum.

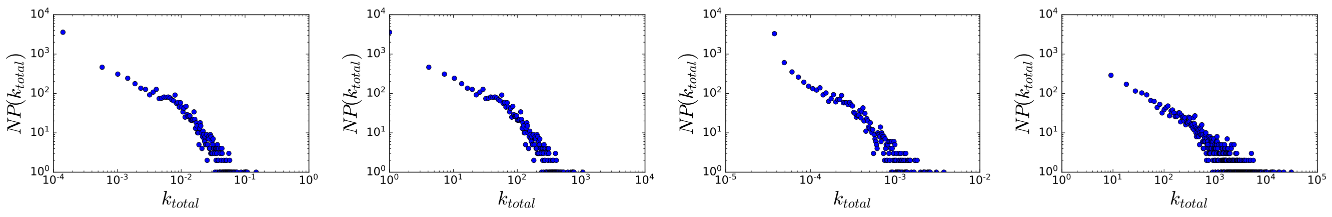
The basic form of *Skip-Gram* used by *DeepWalk* defines the conditional probability $\mathbf{P}(w_{i+j}^t | w_i^t)$ of observing a nearby vertex w_{i+j}^t from the vertex w_i^t from the random walk t can be defined via the softmax function over the dot-product between their features [5]:

$$\mathbf{P}(w_{i+j}^t | w_i^t) = \frac{\exp\left(\mathbf{W}_{w_i^t}^{\intercal} \mathbf{W'}_{w_{i+j}^t}\right)}{\sum_{t=1}^{|V|} \exp\left(\mathbf{W}_{w_i^t}^{\intercal} \mathbf{W'}_{v_t}\right)}, \quad (2)$$

where $\mathbf{W}_{w_i^t}$ and $\mathbf{W'}_{w_{i+j}^t}$ are the hidden layer and output layer weights of the *Skip-Gram* neural network respectively. Equations 1 and 2 show how vertices with similar contexts (similar surrounding vertices) will be embedded into a similar vector space.

2) Differences Between DeepWalk and Node2Vec: The key differences lie in how they generate the random walks of vertices to be used as input to the *Skip-Gram* model. Whilst *DeepWalk* uses a uniform random transition probability to

[2]Here the training context is defined as the vertices either side of the focus vertex w_n^t, set to 10 for the original *DeepWalk* paper.



(a) Degree Centrality (b) Degree (c) Page Rank (d) Triangle Count

Figure 1: Distribution of Topological Feature Values from the Wiki-Vote Dataset in Log Scale: (a) Degree Centrality Distribution, (b) Total Vertex Degree Distribution, (c) Page-Rank Distribution, (d) Distribution of Number Of Triangles for each Vertex.

move from a vertex to one of its neighbours, *Node2Vec* biases the random walks. This biasing introduces two user controllable parameters which dictate how far from, or close to, the source vertex the walk progresses. This is done to capture either the vertex's role in its local neighbourhood (homophily), or alternatively its role in the global graph structure (structural equivalence) [3]. Changing the random walk means that *Node2Vec* has a higher accuracy over *DeepWalk* for a selection of vertex classification problems [3].

B. *Hyperbolic Embeddings*

Recently, a new family of graph embedding approaches has been introduced which embed vertices into hyperbolic, rather than Euclidean space [22] [23]. Hyperbolic space has long been used to analyse graphs which exhibit high levels of hierarchical or community structure [24], but it also has properties which could make it an interesting space for embeddings [23]. Hyperbolic space can be considered "larger" than Euclidean with the same number of dimensions, as the space is *curved*, its total area grows exponentially with the radius [23]. For graph embeddings, this key property means that one effectively has a much larger range of possible points into which the vertices can be embedded. This property allows for closely correlated vertices to be embedded close together, whilst also maintaining more distance between disparate vertices, resulting in an embedding which has the potential to capture more of the latent community structure of a graph.

The hyperbolic approach we focus on was introduced by Chamberlain [23], and uses the *Poincaré Disk* model of 2D hyperbolic space [27]. In their model, the authors use polar coordinates $x = (r, \theta)$, where $r \in [0, 1]$ and $\theta \in [0, 2\pi])$ to describe a point in space for each vertex v in the Poincaré Disk, which allows for the technique to be significantly simplified [23]. Similar to *DeepWalk*, an inner-product is used to define the similarity between two points within the space. The inner-product of two vectors in a Poincaré Disk can be defined as follows [23]:

$$< x, y >= ||x||||y|| \cos(\theta_x - \theta_y), \tag{3}$$

$$= 4 \text{ arctanh } r_x \text{ arctanh } r_y \cos(\theta_x - \theta_y), \tag{4}$$

where $x = (r_x, \theta_x)$ and $y = (r_y, \theta_y)$ are the two input vectors representing two vertices and *arctanh* is the inverse hyperbolic tangent function [23].

To create their hyperbolic graph embedding, the authors use the softmax function of Equation 2, used by *DeepWalk* and others, but importantly replacing the Euclidean inner-products with the hyperbolic inner-products of Equation 3. Aside from this, hyperbolic approaches share many similarities with the stochastic approaches in regards to their input data and training procedure. For example, the hyperbolic approaches are still trained upon pairs of vertex IDs, taken from sequences of vertices generated via random walks on graphs.

C. *Auto-Encoder Based Approaches*

A different approach for graph embeddings which does not use random walks for input, is entitled *Structural Deep Network Embedding (SDNE)* [28]. Instead of a technique based upon capturing the meaning of language, *SDNE* is designed specifically for creating graph embeddings using Deep Learning [11] – specifically deep auto-encoders [29]. Auto-encoders are an un-supervised neural network, where the goal of the technique is to accurately reconstruct the input data through explicit encoder and decoder stages [30].

The authors of *SDNE* argue that a deep neural network, versus the shallow *Skip-Gram* model used by both *DeepWalk* and *Node2Vec*, is much more capable of capturing the complex structure of graphs. In addition the authors argue that for a successful embedding, it must capture both the first and second order proximity of vertices. Here the first order proximity measures the similarity of the vertices which are directly incident upon one another, whereas the second order proximity measures the similarly of vertices neighbourhoods. To capture both of these elements *SDNE* has a dual objective loss function for the model to optimise. The input data to *SDNE* is the adjacency matrix $\mathbf{A}$, where each row a represents the neighbourhood of a vertex.

The first part of the loss function captures the second order proximity of the vertices neighbourhood [28]:

$$\mathbf{L}_{second} = \sum_{i=1}^{|V|} ||(q_i' - q_i) \odot b_i||_2^2, \tag{5}$$

Table I: Topological Features for Measuring Embedding Quality.

Feature Name	Equation
PageRank Score (PR) - The PageRank centrality is commonly used to measure the local influence of a vertex within a graph [8] [25]. Where $\Gamma^-(v)$ is the set of incoming neighbours of v, N is the total number of vertices, $d^+(u)$ is the out-degree of u and d is a constant damping factor (0.85 for this work).	$PR(v) = \frac{1-d}{N} + d \sum_{u \in \Gamma^-(v)} \frac{PR(u)}{d^+(u)}$
Degree Centrality (DC) - This is the sum of both the in and out degree for the vertex v over the total number of vertices in the graph [7].	$DC(v) = \frac{1}{\|V\|}\Gamma^-(v) + d^+(v)$
Local Clustering Score (CLU) - Represents the probability of two neighbours of v also being neighbours of each other, where Φ is the number of pairs of $v's$ neighbours which are themselves connected [26].	$CLU(v) = \frac{2\Phi}{d^+(v)(d^+(v)-1)}$
Number Of Triangles (TR) - The number of triangles containing the vertex v [7].	$TR(v) = \Phi$

where q_i and q_i' are the input and reconstructed representation of the input, $\odot$ is the element wise Hadamard product and b_i is a scaling factor to penalise the technique if it predicts zero too frequently – a common problem with sparse matrices such as adjacency matrices [28].

The second part of the loss function captures the first order proximity of the vertices by iterating over the set of edges E:

$$\mathbf{L}_{first} = \sum_{u,v=1}^{|E|} a_{u,v} ||(w_u^{(k)} - w_v^{(k)})||_2^2, \tag{6}$$

where $w^{(k)}$ is the weights of the k^{th} layer in the auto-encoder technique [28].

Equation 5 and 6 can be combined to create the final loss function for the model to minimise: $\mathbf{L}_{final} = \mathbf{L}_{second} + \alpha \mathbf{L}_{first}$:

$$\mathbf{L}_{final} = \sum_{i=1}^{|V|} ||(q_i' - q_i) \odot b_i||_2^2 + \alpha \sum_{u,v=1}^{|E|} a_{u,v} ||(w_u^{(k)} - w_v^{(k)})||_2^2, \tag{7}$$

where α is a user-controllable parameter defining the importance of $\mathbf{L}_{first}$ in the final loss score [28].

To initialise the weights of the deep auto-encoder used for this approach, an additional neural network must be trained to find a good starting region for the parameters. This pre-training neural network is called a Deep Belief Network, and is widely used within the literature to form the initialisation step of deeper models [31]. However, this pre-training step is not required by either the stochastic or hyperbolic approaches as random initialisation is used for the weights, and adds significant complexity.

IV. Evaluating Embedding Quality

As we have explored above, the primary goal of an unsupervised graph embedding technique is to learn a representation of a given graph which captures the most information from the topology of that graph. Currently, each new graph embedding approach is commonly validated against a vertex labelling problem [3] [5] [28]. However, we feel that this does not necessarily mean that the embeddings are validated against a direct measure of its ability to truly capture a good representation of a graph's topology.

Inspired by recent work in validating the quality of word embeddings [18] and work on validating graph generative models [15], we propose to validate the quality of three graph embedding approaches (summarised in Table II) by predicting topological features from a graph. We feel that this approach will give us a direct measure of how successfully each approach has been in capturing the graph's topology in the resulting embeddings. We believe that optimising a graph embedding approach to predict topological features is a task independent way of evaluating the quality of the embedding. We hope that optimising models to better learn a graph's topology will result in better performance down stream in many machine learning tasks. Evaluating a graph embedding by its ability to recreate graph features also allows un-labelled graphs to have their embeddings validated; in addition these features can be generated for any graph irrespective of the presence of labels. Additionally, investigating which features the embeddings are best able to recreate, could give us some indication as to which kind of topological structures are being approximated by the embeddings for vertex representation.

A. Topological Feature Prediction

Numerous topological features have been identified, measuring various aspects of a graph's topology, at the vertex, edge and graph level [6]. As we are focusing our work upon vertex embedding approaches, we will focus on features which are measured at the *vertex level*. We have selected a range of vertex level features from the literature, detailed in Table I, which capture information about a vertex's local and global role within a graph [3], although it should be noted that any vertex level feature could be used. We then use these features as targets for the graph embeddings to predict as a down-stream task. Here we hypothesise that if an embedding truly has captured a good representation of the graph's topology, it should be able to make some accurate predictions about these features.

In order to transform the task from a regression into that of classification, and following a procedure similar to [32], we bin the real-valued features into a series of classes via the

Table II: Graph Embedding Approaches being Compared.

Approach	Year	Type	Published	Complexity
Node2Vec	2016	stochastic	KDD [3]	$O(\|V\|)$
SDNE	2016	auto-encoder	KDD [28]	$O(\|V\|\|E\|)$
Poincaré Disk	2017	hyperbolic	MLG [23]	$O(\|V\|)$

use of a histogram. One can consider each of these newly created classes as representing a range of possible values for a given feature. As an example, we could transform a vertex's continuous PageRank score [8] into a series of discrete classes via the use of a histogram with a bin size of 3, where each of the newly created classes represented a low, medium or high Page Rank score. In order to allow for a good distribution of feature values, we used a bin size of 100 for the histogram function, meaning that 100 discrete classes were created for each of the features in Table I.

B. Feature Distribution Imbalance

Many empirical graphs, especially those representing social, hyper-link and citation networks, have been shown to have an approximately power-law distribution of degree values [33]. This power-law distribution poses a challenge for a neural network model, as it means the features we are trying to predict are extremely unbalanced, with a heavy skew towards a low value if we are considering the distribution of degree value. Imbalanced class distribution creates difficulties for machine learning models, as there are fewer examples of the minority classes for the model to learn, which can often lead to poor predictive performance on these classes [11]. It has been shown that the distribution of other topological features can also follow a power-law distribution in many graphs [16]. Figure 1 shows the distribution of a range of topological feature values for the *wiki-vote* dataset. The Figure shows that indeed, the topological feature values all follow an approximately power-law distribution. This means that the embedding approaches must create a representation which is unique enough to identify the minority classes sufficiently well in the imbalanced dataset.

V. Results

In the following section we detail the setup of the experimental evaluation and present results.

A. Metrics

1) Presented Results: - All the reported results are the mean of five replicated experiment runs along with confidence intervals. For the runtime analysis, the presented results are the mean runtime for job completion, presented in minutes. For the classification results, all the accuracy scores presented are the mean accuracy after k-fold cross validation – considered the gold standard for model testing [34]. For k-fold cross validation, the original dataset is partitioned into k equally sized partitions. $k-1$ partitions are used to train the model, with the remaining partition being used for testing. The process is repeated k times using a unique partition for each repetition and a mean taken to produce the final result. To perform the actual classifications of the embeddings, we use a one-vs-rest Logistic Regression with L2 regularization from the Scikit-Learn Python package [35].

2) Precision Metrics: - For reporting the results of the vertex feature classification tasks, we report the $macrof1$ and $microf1$ scores with varying percentages of labelled data available at training time. This is a similar setup to previous works [3] [4].

The $macrof1$ score, when performing multi-label classification, is defined as the average $microf1$-score over the whole set of labels L:

$$macrof1 = \frac{1}{|L|}\sum_{l\in L} f1(l), \tag{8}$$

where $f1(l)$ is the $microf1$-score for the given label l.

The $microf1$-score calculates the $f1$-score for the dataset globally by counting the total number of true positives (TP), false positives (FP) and false negatives (FN) across a labelled dataset $|L|$. Using the notation from [4], $microf1$ is defined as:

$$microf1 = \frac{2\cdot P\cdot R}{P+R}, \tag{9}$$

where:

$$Precision(P) = \frac{\sum_{l=1}^{|L|} TP(l)}{\sum_{l=1}^{|L|} TP(l) + FP(l)},$$

$$Recall(R) = \frac{\sum_{l=1}^{|L|} TP(l)}{\sum_{l=1}^{|L|} TP(l) + FN(l)},$$

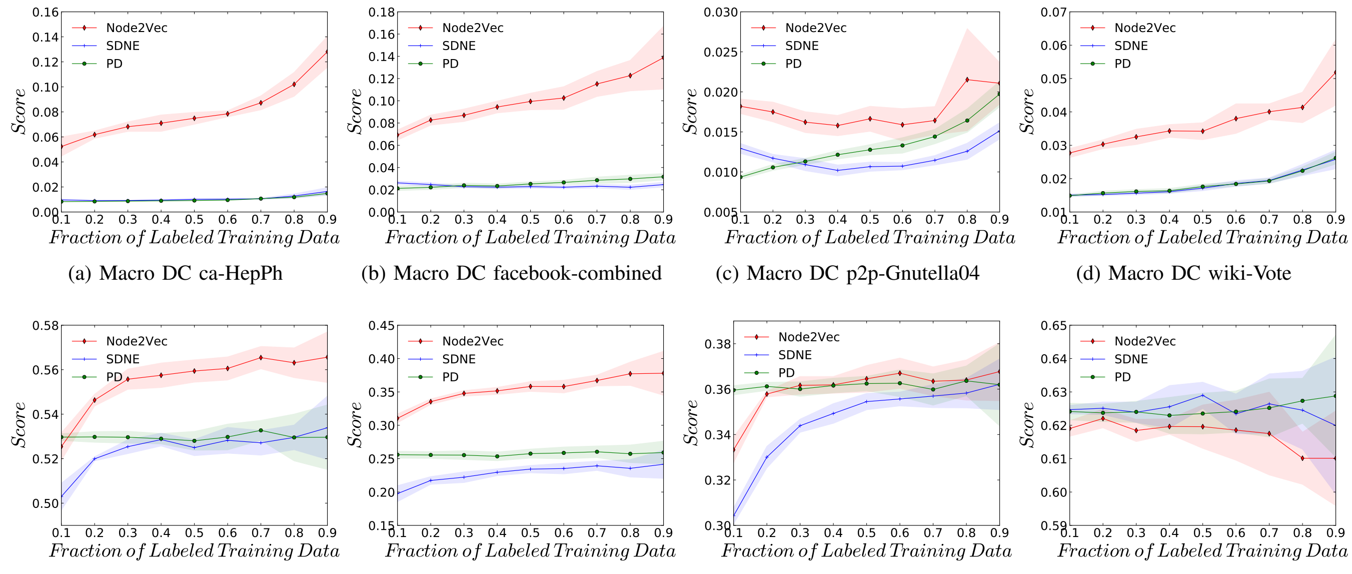
and $TP(l)$ denotes the number of true positives the model predicts for a given label l, $FP(l)$ denotes the number of false positives and $FN(l)$ the number of false negatives.

B. Experimental Setup

1) Implementation Details: - The three approaches were reimplemented in Tensorflow as the author-provided versions were not all available using the same framework. We also attempted to ensure the same Tensorflow-based optimisations were used across all the approaches [36]. NNs contain many hyper-parameters a user can control to improve the performance, both of the predictive accuracy and the runtime, of a given dataset. This process can be extremely time consuming and often requires users to perform a grid search over a range of possible hyper-parameter values to find a combination which performs best [11]. For setting the required hyper-parameters for the approaches, we took the default values provided by the authors in their respective papers [3] [23] [28] keeping them constant across all datasets. The key hyper-parameters used for each approach are detailed in Table III. We have open sourced

Table III: Key Hyper-Parameter Settings

Approach	**Learning Algorithm**	**Learning Rate**	**Specific Parameters**
SNDE	RMSProp	0.01	α=500, b=10, epochs=500
Node2Vec	SGD	1.0	p=0.5, q=2, epochs=5
Poincaré Disk	SGD	0.1	p=0.5, q=2, epochs=5



(a) Macro DC ca-HepPh (b) Macro DC facebook-combined (c) Macro DC p2p-Gnutella04 (d) Macro DC wiki-Vote

(e) Micro DC ca-HepPh (f) Micro DC facebook-combined (g) Micro DC p2p-Gnutella04 (h) Micro DC wiki-Vote

Figure 2: Micro and Macro F1 Scores for all Approaches when Predicting a Vertex's Degree Centrality (DC) Value.

Table IV: Graph Datasets Used.

Dataset	$\|V\|$	$\|E\|$	*domain*
Ca-HepPh	12,008	118,521	*Collaboration*
ego-Facebook	4,039	88,234	*Social*
p2p-Gnutella04	10,876	39,994	*Peer − to − peer*
wiki-Vote	7,115	103,689	*Wiki*

our implementations of these approaches and made them available online[3].

2) Experimental Environment: - Experimentation was performed on a compute system with 2 NVIDIA Tesla K40c's, 2.3GHz Intel Xeon E5-2650 v3, 64GB RAM and the following software stack: CentOS 7.2, GCC 4.8.5, CUDA 8.0, CuDNN v6, TensorFlow 1.3, scikit-learn 0.19.0, Boost 1.56, Python 2.7.5 and NetworkX 2.0.

3) Experimental Datasets: - All the empirical datasets used for evaluation were taken from the SNAP data repository [37] and are detailed in Table IV. The *domain* label provided is taken from the listings of the graphs domain provided by SNAP [37]. The synthetic *Barabási-Albert (BA)* [16] graphs used were generated using the NetworkX Python package [38], with a fixed average degree of 9 and a varying number of vertices.

C. Topological Feature Prediction

In this section, we present the experimental evaluation from the classification of topological features, presented in Table I, using the embeddings generated from the three approaches on the datasets detailed in Table IV. We present both the *macro* and *micro-f1* plotted against a varying amount of labelled data available during the training process.

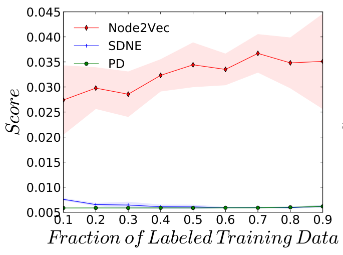
[3]https://github.com/sbonner0/unsupervised-graph-embedding/

Figure 2 shows the *macro* and *micro-f1* scores for all approaches, across all datasets, when classifying the Degree Centrality value for each vertex – here higher scores are better with the 'best' score of 1 meaning perfect classification accuracy. It can be seen that across the results presented in the figure, *Node2Vec* has the best performance across most datasets, particularly when considering the *macro-f1* scores. It should be noted that although we performed experiments for *DeepWalk*, there was no statistical difference between these results and the results for *Node2Vec*. Therefore, we only report here the *Node2Vec* results.

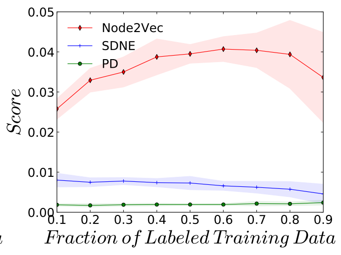
Figure 3 shows how the approaches perform at classifying Local Clustering Scores across all datasets. We can again see that *Node2Vec* is the best across some of the datasets, which is particularly evident in sub-figures a and b. However on other datasets, the approaches have much more similar levels of performance, with the *p2p-Gnutella04* showing very similar performance across both macro and micro scores.

Figure 4 shows how well the different approaches are able to correctly classify the Page Rank score of each vertex across the datasets. Interestingly, the figures show that the Hyperbolic approach performs comparatively well here across a range of datasets, beating the other approaches in terms of *macro* score across *facebook* and *p2p-Gnutella04*.

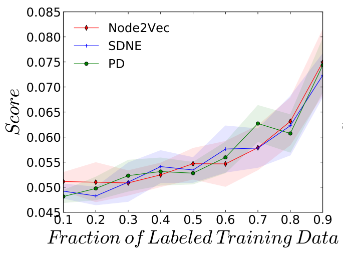
Finally, Figure 5 shows the performance of the approaches in predicting the number of triangles for each vertex across the datasets sizes. Here the approaches are again closely matched in performance, with *Node2Vec* having a clear lead in performance on the *Ca-HepPh* dataset.



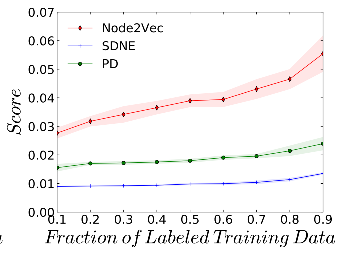
(a) Macro CLU ca-HepPh



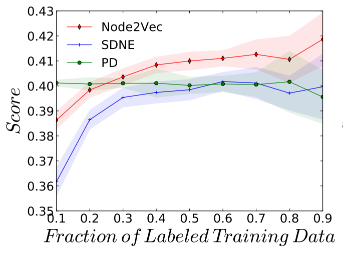
(b) Macro CLU facebook-combined



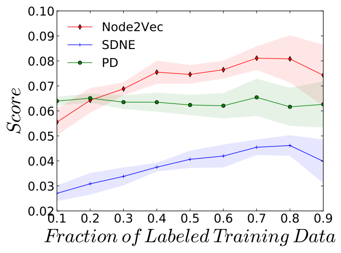
(c) Macro CLU p2p-Gnutella04



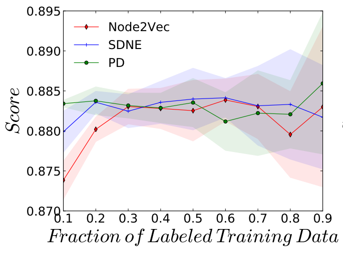
(d) Macro CLU wiki-Vote



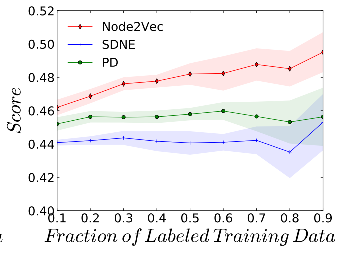
(e) Micro CLU ca-HepPh



(f) Micro CLU facebook-combined

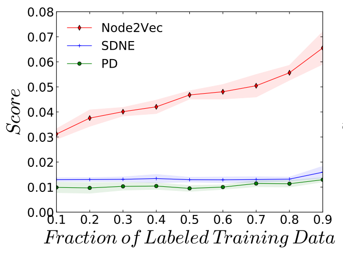


(g) Micro CLU p2p-Gnutella04

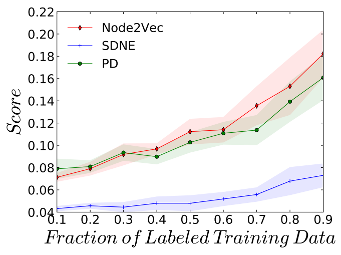


(h) Micro CLU wiki-Vote

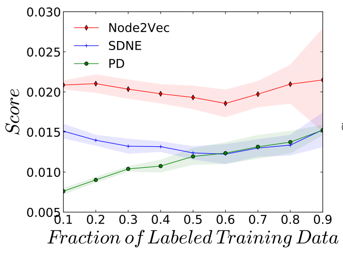
Figure 3: Micro and Macro F1 Scores for all Approaches when Predicting a Vertex's Local Clustering (CLU) Value.



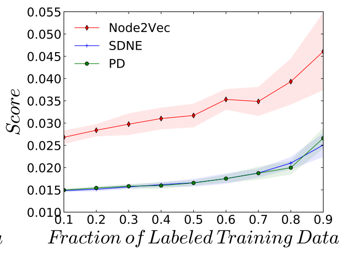
(a) Macro PR ca-HepPh



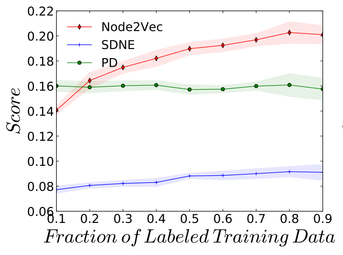
(b) Macro PR facebook-combined



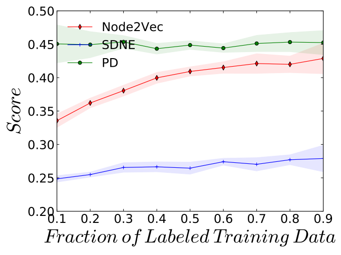
(c) Macro PR p2p-Gnutella04



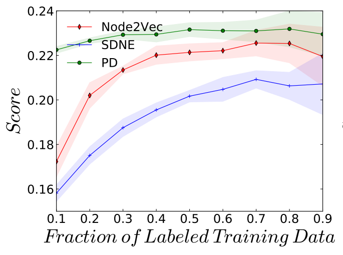
(d) Macro PR wiki-Vote



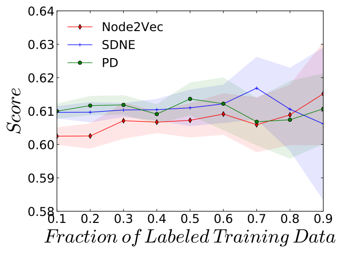
(e) Micro PR ca-HepPh



(f) Micro PR facebook-combined



(g) Micro PR p2p-Gnutella04

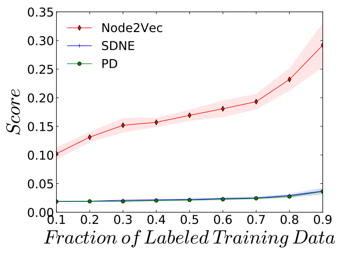


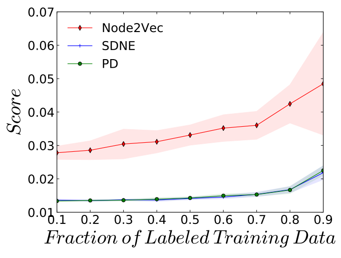
(h) Micro PR wiki-Vote

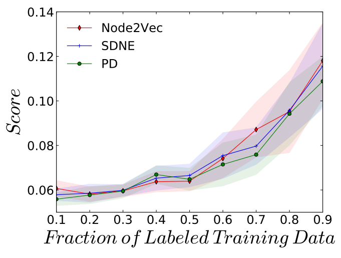
Figure 4: Micro and Macro F1 Scores for all Approaches when Predicting a Vertex's Page Rank (PR) Value.

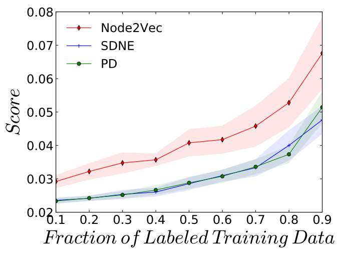
D. Run-Time Analysis

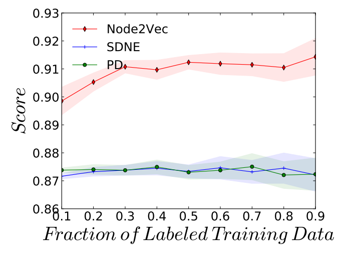
To assess the runtime of the approaches, we measured it across a series of synthetic *BA* graphs. Figure 6 displays the results of the runtime experiment when using the hyper-parameters from Table III, where the presented results are the mean of 5 repeated runs, with the error bars being one standard deviation. For both the *Node2Vec* and *PD* approaches, the presented runtimes include the walk generation process. Figure 6 shows the *SDNE* approach to have the lowest runtime across all dataset sizes, with the *Node2Vec* and *PD* approaches having almost identical runtime. However, the Figure also shows that the runtime for *Node2Vec* and *PD* increase close to linearly with dataset size, suggesting approaches based on random walks to be more scalable than other approaches. The discrepancy between the predicted computational complexity and the measured runtime for *SDNE* is interesting to note, although this could

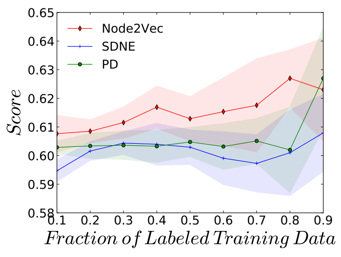

(a) Macro TR ca-HepPh

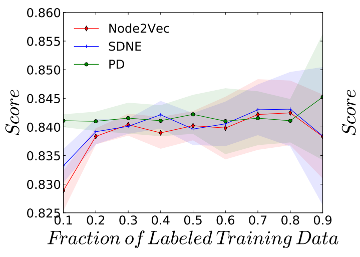

(b) Macro TR facebook-combined


(c) Macro TR p2p-Gnutella04


(d) Macro TR wiki-Vote


(e) Micro TR ca-HepPh


(f) Micro TR facebook-combined


(g) Micro TR p2p-Gnutella04

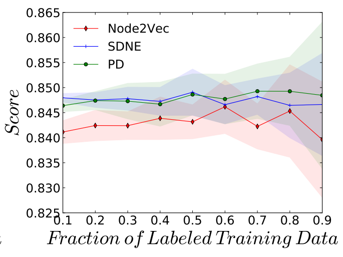

(h) Micro TR wiki-Vote

Figure 5: Micro and Macro F1 Scores for all Approaches when Predicting a Vertex's Triangle Count (TR) Value.

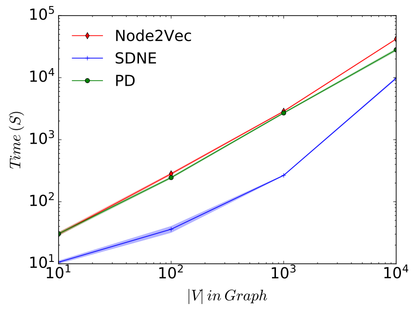


Figure 6: Runtime Of Approaches (Log Scale).

be due to the relativity modest graph sizes tested.

E. Discussion

Across all the datasets and features the three approaches have been tested against, it can be argued that no single approach has consistently better at classifying topological features. However, *Node2Vec* does demonstrate the better performance on the largest number of datasets. This is in contrast to other studies which have found *SDNE* to have the best performance in vertex labelling problems [4] [28]. This discrepancy could be explained by *SDNE* being a deeper model, it could be more sensitive to correct hyper-parameter selection than the other approaches, or it could be that *SDNE* learns embeddings which are less able to represent detailed topological features. It could also be that the biased random walk employed by *Node2Vec* means that it is more able to capture rich topological information. Finally, it is interesting to note the performance of Hyperbolic approach *PD*, as it has far fewer latent dimensions in which to capture topological information due to its limitation in modelling the space as a 2D disk. Empirically, *PD* shows largely similar performance to the other approaches on most datasets, providing strong evidence that the hyperbolic space is an appropriate space in which to embed vertices.

The general poor performance of the models at traditional topological feature prediction is interesting to note and could be taken as evidence to suggest that traditional topological features are *not* being learned during the embedding process. It seems likely that the neural networks used to create the embeddings are learning unique and non-traditional features from the topology in order to best represent the vertices.

VI. Conclusion

Graph embeddings have shown themselves to be a powerful neural network based technique for automatically learning a representation of a graph in low dimensional vector space. The goal of these approaches is to learn a representation which captures the largest quantity of information from a graph's topology. To investigate how much of this information is really captured in the embeddings, we propose a framework for assessing the quality of graph embeddings at topological feature reconstruction. We empirically demonstrate that none of the current state-of-the-art graph embedding approaches provide excellent feature identification, strongly suggesting that other representations are being learned during the embeddings process.

Further work needs to be conducted to investigate in greater detail which elements from the topology are being approximated by the embeddings, as it seems possible that traditional graph features are not being learned. We plan to investigate if optimising the embeddings to predict topological features more accurately results in an embedding which generalises more efficiently across all tasks.

Acknowledgment

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research. Additionally we thank the Engineering and Physical Sciences Research Council (EPSRC) for funding. We also thank the authors of papers [3], [23] and [28] for making implementations of their code publicly available.

References

[1] L. G. Moyano, "Learning network representations," *The European Physical Journal Special Topics*, 2017.

[2] B. Obara, V. Grau, and M. D. Fricker, "A bioimage informatics approach to automatically extract complex fungal networks," *Bioinformatics*, vol. 28, no. 18, pp. 2374–2381, 2012.

[3] A. Grover and J. Leskovec, "node2vec : Scalable Feature Learning for Networks," *Knowledge Discovery and Data Mining (KDD)*, 2016.

[4] P. Goyal and E. Ferrara, "Graph Embedding Techniques, Applications, and Performance: A Survey," *arXiv preprint arXiv:1705.02801*, 2017.

[5] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk: online learning of social representations," *Knowledge Discovery and Data Mining (KDD)*, 2014.

[6] G. Li, M. Semerci, B. Yener, and M. Zaki, "Effective graph classification based on topological and label attributes," *Statistical Analysis and Data Mining*, 2012.

[7] S. Bonner, J. Brennan, G. Theodoropoulos, I. Kureshi, and A. S. McGough, "Deep topology classification: A new approach for massive graph classification," in *IEEE International Conference on Big Data*, 2016.

[8] L. Page, S. Brin, R. Motwani, and T. Winograd, "The PageRank citation ranking:bringing order to the web." 1998.

[9] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "NetSimile: A scalable approach to size-independent network similarity," *arXiv preprint arXiv:1209.2684*, 2012.

[10] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation Learning on Graphs: Methods and Applications," *arXiv preprint arXiv:1709.05584*, 2017.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[12] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral Networks and Locally Connected Networks on Graphs," *International Conference on Learning Representations (ICLR)*, 2013.

[13] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering," *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[14] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," *ICML*, 2016.

[15] W. Liu, H. Cooper, M. H. Oh, S. Yeung, P.-y. Chen, T. Suzumura, and L. Chen, "Learning Graph Topological Features via GAN," *arXiv preprint arXiv:1709.03545*, 2017.

[16] R. Albert and A. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, 2002.

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

[18] T. Schnabel, I. Labutov, D. M. Mimno, and T. Joachims, "Evaluation methods for unsupervised word embeddings." in *EMNLP*, 2015, pp. 298–307.

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," *Conference on Neural Information Processing Systems (NIPS)*, 2013.

[20] ——, "Efficient Estimation of Word Representations in Vector Space," in *International Conference on Learning Representations (ICLR)*, 2013.

[21] L. Backstrom and J. Leskovec, "Supervised random walks: Predicting and Recommending Links in Social Networks," in *Web Search and Data Mining (WSDM)*, 2011.

[22] M. Nickel and D. Kiela, "Poincaré Embeddings for Learning Hierarchical Representations," *arXiv preprint arXiv:1705.08039*, 2017.

[23] B. Chamberlain, M. D. Clough, and James, "Neural Embeddings of Graphs in Hyperbolic Space," *KDD Workshop on Mining and Learning with Graphs (MLG)*, 2017.

[24] T. Munzner, "Exploring large graphs in 3d hyperbolic space," *IEEE Computer Graphics and Applications*, 1998.

[25] M. Han, K. Daudjee, K. Ammar, M. T. Ozsu, X. Wang, and T. Jin, "An Experimental Comparison of Pregel-like Graph Processing Systems," *Proceedings of the Very Large Databases (VLDB) Endowment*, vol. 7, no. 12, pp. 1047–1058, 2014.

[26] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks." *Nature*, 1998.

[27] D. B. Epstein, R. C. Penner *et al.*, "Euclidean decompositions of noncompact hyperbolic manifolds," *Journal of Differential Geometry*, 1988.

[28] D. Wang, P. Cui, and W. Zhu, "Structural Deep Network Embedding," *Knowledge Discovery and Data Mining (KDD)*, 2016.

[29] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*, 2011.

[30] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, 2009.

[31] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, 2010.

[32] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[33] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *ACM SIGCOMM Computer Communication Review*, 1999.

[34] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection *," *Statistics Surveys*, 2010.

[35] 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*, 2011.

[36] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," *arXiv preprint arXiv:1608.07249*, 2016.

[37] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, 2014.

[38] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Python in Science Conference (SciPy2008)*, 2008.