Improving Node Embedding by a Compact Neighborhood Representation

Ikenna Victor Oluigbo, Hamida Seba, Mohammed Haddad

▶ **To cite this version:**

Ikenna Victor Oluigbo, Hamida Seba, Mohammed Haddad. Improving Node Embedding by a Compact Neighborhood Representation. Neural Computing and Applications, 2022, 35 (9), pp.7035-7048. 10.1007/s00521-022-08076-6 . hal-03638206v2

HAL Id: hal-03638206
https://hal.science/hal-03638206v2

Submitted on 2 Dec 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Improving Node Embedding by a Compact Neighborhood Representation

Ikenna OLUIGBO, Hamida SEBA, Mohammed HADDAD
Univ Lyon, UCBL, CNRS, INSA Lyon, LIRIS,
UMR5205, F-69622 Villeurbanne
{Ikenna.oluigbo,hamida.seba,mohammed.haddad}@univ-lyon1.fr

Abstract

Graph Embedding, a learning paradigm that represents graph vertices, edges, and other semantic information about a graph into low dimensional vectors, has found wide applications in different machine learning tasks. In the past few years, we have had a plethora of methods centered on graph embedding using different techniques such as spectral classification, matrix factorization and learning. In this context, choosing the appropriate dimension of the obtained embedding remains a fundamental issue. In this paper, we propose a compact representation of a node's neighborhood, including attributes and structure, that can be used as an additional dimension to enrich node embedding, to ensure accuracy. This compact representation ensures that both semantic and structural properties of a node's neighboringhood are properly captured in a single dimension. Consequently, we improve the learned embedding from state-of-the-art models by introducing the neighborhood compact representation for each node as an additional layer of dimensionality. We leverage on this neighborhood encoding technique and compare with embedding from state-of-the-art models on two learning tasks: node classification and link prediction. The performance evaluation show that our approach gives a better prediction and classification accuracy in both tasks.

1 Acknowledgment

This preprint has not undergone any post-submission improvements or corrections. The Version of Record of this article is published in Neural Computing and Applications, and is available online at https://doi.org/10.1007/s00521-022-08076-6

2 Introduction

Graph embedding has gained so much popularity over the years, and it currently lies in the center of many graph analytic problems. Graph embedding is an algorithmic technique which basically transforms graph granularity (nodes, edges, substructures, or whole graph) into low dimensional continuous vector(s) of predefined number of dimensions. A typical graph (also known as network) has important properties and characteristic features such as topology, structural dependence, node-to-node correlations, neighborhood relationships, etc., which can be harnessed for a number of learning tasks and analytical problem solving. Therefore, the goal of a graph embedding technique is to ensure that these very valuable graph properties are preserved in the learned embedding. Since graph data can contain millions of vertices and billions of links between the vertices, providing an efficient way (both in time and space complexity) to analyze these massive data is very crucial.
Over the last few decades, several graph embedding models have been designed to learn latent low dimensional vector(s) from the high dimensionality of graph granular data. A popular way most models learn vector representations from a graph is to first compute an attribute graph (a graph which bears the pairwise similarity feature(s) between these granular data) from the given set of input high dimensional graph granular data. Each granular data having similar attributes are then mapped closely together on a new latent low dimensional embedding space, and the topological and structural characteristics of the granular data are encoded into its corresponding embedding vector [22]. To obtain low dimension vectors for graph data, several researches have embarked on different approaches ranging from deep neural architecture and spectral processing, to matrix factorization. Many deep architectural approaches make use of an autoencoder. First, an adjacency matrix is computed from the input network, this matrix is then fed into an autoencoder to derive a non-linear similarity vector representation which preserves the desired information in the embedding space [30]. Instead of computing an adjacency matrix to extract information from an input network (adjacency matrix becomes complex for massive networks), *DeepWalk* [24] and *Node2Vec* [10] use random walks to build a corpus of nodes (made up of a fixed length of paths, analogous to sentence generation of *Word2Vec*). The SkipGram model is thus applied on the corpus to learn continuous vector representations for nodes by optimizing a neighborhood preserving likelihood objective using Stochastic Gradient Descent.

In the matrix factorization approach, the graph structural properties are first represented as a matrix (node adjacency matrix and Laplacian matrix are popularly used), and then a linear or non-linear transformation manifold

learning procedure is used to factorize the computed matrix (factorization method depends on matrix property) so as to obtain a low dimensional vector embedding of the network [5], [25].

As much as embedding from popular state-of-the-art representation learning models have shown significant performance in machine learning tasks, it is important to highlight some weaknesses they still possess:

- *Complexity in Dimensionality Reduction:* Many embedding models, especially models needing to compute and factorize complex matrices can be very computationally expensive when transforming data from a high-dimensional space into a low-dimensional space. It becomes even more complex when learning embedding for massive networks while also maintaining high efficiency in preserving important features in the learned embedding. Deep architecture and Random walk approaches enjoy fast training time and low memory consumption but random walks do not consider other features like edge weights and distance, which means that whatever importance a node possesses cannot be preserved in the embedding. Matrix decomposition techniques on the other hand are computationally expensive when factorizing matrix powers.

- *Complexity in Information Gathering:* With the explosive growth of large networks, many existing models are faced with the challenge of capturing and preserving semantic aspects of the network. Some deep architectural models does not consider edge features, meaning each node's importance and frequency of occurrence cannot be preserved in the embedding, and such embedding is hard to generalize to new data that are not present when training the model.

- *Estimating Embedding Dimensionality:* In most existing graph embedding methods, the fundamentals involved in correctly appropriating the dimension for an embedding is not well defined. Many methods rely on Grid hyper-parameter estimation or domain experience which can result into issues such as high resource and time usage when training the model, as well as poor model performance [18].

- *Lack of Scalability:* Many existing representation learning techniques model the low dimension embedded output as depended wholly on the entire input graph which can be very expensive with massive graphs. As a result of the latency experienced with loading the entire graph into memory, processing can be cut short resulting in intermediate and inconclusive results. This has led to concepts such as embedding compressed subgraph of a massive graph [1], [3].

In this paper, we attempt to address scalability and dimensionality issues of node embedding. We propose to use a compact encoding of a vertex neighborhood that combines the structural and semantic features of the neighborhood of the vertex. This encoding process circumvents the technicalities involved with computing complex matrices and appropriating dimensions for learned embedding. Recent studies have shown that the dimensionality reduction techniques as well as selecting a predefined dimension length for an embedding have an impact on the efficacy of the representation learning model. Selecting a small number of dimensions for learned embedding limits the quality of information captured in the embedding; while too large dimension increases the possibility of noise and other unnecessary features captured in the embedding thereby limiting its accuracy in machine learning tasks, and the possible problem of overfitting also [18]. A number of literature have proposed techniques for estimating the number of dimension for embedding. In [11], the authors proposed a principled Pairwise Inner Product (PIP) loss technique (adapted from [33]) to choose a dimension value of a generic network embedding. This technique however is limited to capturing only information related to the structural characteristics of nodes in the network. The Pairwise Inner Product (PIP) metric, proposed in [33], uses a grid search algorithm to measure the dissimilarities between word embedding. The grid search technique becomes ineffective when the number of hyper-parameters in the model grows exponentially. In [12], the authors proposed a generative network model using an optimal inference algorithm to embed a graph. They achieved this by estimating the probability between pairs of nodes and their distance in the embedding space. Their model estimated three embedding dimensions to capture the learned embedding. In [16], the authors proposed a metric for k-dimensional structure information embedding of the graph, which is particularly concerned with the structure of the network ignoring its rich link structures. In [34], the authors proposed a robust graph dimensionality reduction algorithm to map high-dimension data into lower-dimensional intrinsic space by a transformation of matrix, which can get very computationally expensive for massive networks. Estimating an appropriate embedding dimension for an efficient and effective embedding in both small and large networks is quite a daunting task for existing embedding models.

The aim of the compact neighborhood encoding technique is to represent, with a bijective function, all the information surrounding a vertex into a simple numerical value. It can be computed without the need to load all the graph in memory eliminating the computational complexity encountered by many existing methods. It also contributes to resolve the challenges involved in estimating the dimension for an embedding. To achieve this objective we rely on specific kind of bijective functions called Cantor polynomials [26].

These polynomials have been used to index nodes' neighborhoods for subgraph isomorphism search [21] but to our knowledge there is no work using them to construct node embedding for deep leaning purposes. Our contributions are summarized as follows:

1. We use a compact neighborhood encoding to embed in one dimension both structural and semantic information of a node.

2. We design a link prediction and node classification model to evaluate the performance of our compact neighborhood representation against selected existing network embedding models on some learning tasks using real world datasets drawn from different domains. This comparison is done across varied dimensions.

3. We show from the results that the proposed compact neighborhood representation technique is efficient and effective for encoding a vertex with vital structural and semantic information around it. This is particularly important for tasks such as predicting new or missing links from/to a vertex.

3 Definitions and notation

In this study, we will use the terms *Network* and *Graph* interchangeably to mean the same thing. Also vertex and node are used interchangeably, as well as edge and link. The vertices are objects in the network, defined by some form of relationship known as links.

Definition 1 *We define an attributed Graph G as a 4-tuple $G = (|V|, |E|, \ell, \Sigma)$, where $|V|$ consists of the set of vertices in the graph, $|E|$ consists of the set of edges connecting the vertices in the graph, $\ell : V \cup E \longrightarrow \Sigma$ is a labeling function on the vertices and the edges where Σ is the set of attributes that can appear on the vertices and/or the edges. $\ell(u)$ represents the attribute of vertex u in a graph.*

Definition 2 *An undirected graph is a bidirectional graph such that edges (u, v) and (v, u) are equivalent. The neighbors of a vertex u denoted as $N(u)$ are all vertices adjacent to u. The degree of a vertex u, denoted as $deg(u)$, is the number of neighbors of u. The semantic information of a vertex u represents the features of the vertex denoted by a vector X_u.*

Definition 3 *An adjacency matrix A is an $n \times n$ matrix with non-negative weights, such that $A_{uv} = w_{i,j}$ for a weighted edge or $A_{uv} = 1$ for an unweighted graph if and only if an edge (u, v) exists; else $A_{uv} = 0$ if edge (u, v) does not exists in graph G.*

Definition 4 *Given the node features X_u, a link prediction model can output whether two nodes are connected by an edge. In a neural link prediction model, the features are aggregated to learn embedding for each node. The similarity score of two node embedding will decide whether they should be connected. Nodes are equally classified based on similarities of their features.*

Definition 5 *First-order proximity captures the local-pairwise relationship between directly connected vertices in terms of the proximity between vertices in a graph. For a weighted and directed graph having connected vertex pairs $(u, v) \in E$ in G, the first-order proximity defines the weight ω_{uv} between the vertex pair. The probability distribution of first-order proximity for two vertices u and v is defined as: $Pr(u, v) = \frac{1}{1+\exp(X_u^T X_v)}$ where X_u^T denotes the transpose of X_u.*

The empirical distribution for $Pr(u, v)$ is defined as $\frac{\omega_{uv}}{W}$ where W is the total weights of all edges in the network. With first-order proximity alone many edges in the network goes unobserved, therefore is not sufficient for preserving the entire structure of the graph.

Definition 6 *The second-order proximity defines the neighborhood similarity between a vertex pair (u, v) of a directed or undirected network. A second-order proximity exists if a vertex pair share a common 1-hop neighbor. By defining a common neighbor function, one can determine if there exists an edge between pair of vertices. The probability distribution over vertex v and vertex u is defined as: $Pr(v|u) = \frac{\exp(X_u^T X_v)}{\Sigma_{k=1}^{|v|} \exp(X_k^T X_v)}$*

Our study involves a two prone process; Given an attributed graph $G = (|V|, |E|, \ell, \Sigma)$, we first compute the compact neighborhood representation for every node in the graph. Our compact neighborhood representation ensures that each node is capable of retaining multiple information from surrounding nodes. Next, we learn embedding with varied dimensions for selected real life datasets using existing network embedding techniques. On the one hand, we experiment with the learned network embedding on two learning task (Link Prediction and Node Classification). On the other hand, we add the encoded compact neighborhood representation for each node as an additional feature in the embedding and we repeat the same set of experiments under similar conditions and parameters. After comparison, we focus on determining how

much improvement in accuracy and effectiveness we get with the inclusion of the new feature, i.e., the compact neighborhood representation, which has more information embedded in it. The results from the series of test is shown in subsequent sections.

4 Related Studies

In this section, we classify some of the existing network embedding models and briefly describe how they perform dimensionality reduction. These models have the same objective of learning embeddings for nodes in a graph, and these embeddings can be used in machine learning tasks.

***Random Walk Approach*:** Embedding models under this approach preserve the structural characteristics of each node or the similarity in node communities, or both [10]. Two main embedding models that leverages on the random walk approach are *Deepwalk* and *Node2Vec*, both of which use the *SkipGram Model* to learn continuous feature representations for words by optimizing a neighborhood preserving likelihood objective (dimensionality reduction). The SkipGram model is a language model (like every other *Word2Vec model*) that maximizes the co-occurrence probability among the words that appear within a window in a sentence [20]. To capture information of every node, *Deepwalk*, proposed in [24], takes a graph G and samples uniformly a random vertex u as the root of the random walk with a uniform length. A walk samples uniformly from the neighbors of the last vertex visited until the maximum length for the walk is reached. Upon reaching this maximum length, the walk uniformly samples another random vertex and performs another walk. This walk learns only the first-order proximity of a node, which is not sufficient enough to preserve the entire network structure. *Deepwalk* uses Hierarchical softmax to approximate the probability distribution $Pr(u, v)$ of first-order proximity and Stochastic Gradient Descent (SGD) to optimize the learning rate and other parameters for feature representation. In contrast, *Node2Vec*, proposed in [10], preserves both the first-order and second-order proximity of a node. For every source node u in the network, *Node2Vec* uses a random walk approach similar to Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms to sample the neighbors of u. In other words, to capture information around a node (Neighbors), *Node2Vec* uses a biased second-order random walk guided by two parameters p and q. Parameter p controls the likelihood of the walk immediately revisiting a node, while parameter q allows the walk differentiate between local view (immediate connected neighbors of a node - BFS) and exploratory view (nodes at farther distance to source node - DFS). The random walk continues

until all the nodes in the network are sampled into a large *"corpus"* of node sequences; the *"corpus"* is fed into the Skipgram model to learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes in a δ-dimensional feature space. As with most embedding models, the value of δ is estimated, which can have its impact on the performance of the model. In *BINE* model [9] for bipartite network, the authors capture information around each node into a *"corpus"* using a probabilistic random walk generator; they then define a joint optimization framework to learn vertex embedding while preserving both explicit and implicit relations simultaneously. In [2] and [28], the studies focus on attributed random walks to build sequences of nodes, and then define a conditional probability functions that map nodes with similar attributive relationship in a vector space.

***Spectral Processing Approach*:** There are different variants to this approach; while some architectures have leveraged on spectral heat wavelet diffusion, some other have achieved success through eigen matrix decomposition. For example, an unsupervised model *GRAPHWAVE* proposed in [7] learns a network neighborhood structure into a low-dimensional embedding by leveraging on the spectral wavelet (unit energy) property of a graph, and the wavelets probability distribution; such that the vertex neighborhood similarity property is preserved in the embedding space. The result from this study was evaluated in areas of node classification, and other evaluation metrics. A Laplacian eigenmaps spectral clustering techniques was proposed in [4] to represent each node into vector embedding by the graph Laplacian eigenvectors associated with its first k nontrivial eigenvalues. The Laplacian Eigenmaps model preserves first-order proximity of nodes in the network and thus, the model computes a larger penalty if two nodes with higher similarity are embedded far apart in the embedding space. Laplacian objective function is symmetric to each node pairs, therefore it cannot capture important edge features [19].

***Deep Architecture Approach*:** Most deep architecture models achieves the goal of embedding network properties by means of an encoder. A semi supervised model *SDNE* [31] was one of the erstwhile deep learning model to learn δ-dimensional embedding with non-linear functions while preserving the second-order proximity and first-order proximity structure for homogeneous networks. On the other hand, a variational graph autoencoder *VGAE* [14] framework captures node neighborhood information by computing an adjacency matrix A and a degree matrix d, which are entered into a graph convolutional network (GCN) encoder and a simple inner product decoder to learn the higher order dependencies between nodes for tasks such as link prediction. *LINE*, proposed in [13] is a representation learning algorithm that

learns an embedding model for real world information networks. To capture the structural features of a network, *LINE* uses both the first-order proximity to preserve the local pairwise proximity between the nodes and second-order proximity to preserve vertices sharing many connections to other vertices. To learn vector embedding for nodes in the network, the *LINE* model is optimized using negative sampling technique to preserve the first-order proximity and second-order proximity separately and then concatenate the embedding trained by the two methods for each vertex. A drawback for this model is that not only it considers the entire network as an input (large networks leads to higher computational complexity), but also it separately trains the objective functions of first-order and second-order proximity resulting into higher time and space complexity.

***Matrix Factorization Approach*:** Non linear matrix factorization maps vertices into low dimensional latent space by considering the eigenvectors of the k-smallest eigenvalues, through non linear transformation manifold learning methods such as Laplacian Eigenmaps (LE), Locally Linear Embedding (LLE), and Isomap. A linear matrix factorization such as Singular Value Decomposition (SVD) is a complex mathematical technique that factorizes an input adjacency matrix A into the product of three new matrices $U\Sigma V^T$, while ensuring that the three matrices possesses some characteristic properties of the original network. U and V are orthonormal $m \times m$ and $n \times n$ real or complex unitary matrix respectively, and Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal. These matrix factorization approaches usually has a high time computational complexity of $\Theta(|V|^2)$, and encounters scalability issues with massive networks.

5 A Compact Neighborhood Representation

As presented in the previous section, embedding graph granularities into low-dimensional continuous vectors can be a complex and computationally expensive task, with high memory utilization. The idea behind compact neighborhood representation (CNR) is to distill in a simple numerical value, i.e., an integer, the neighborhood information surrounding a vertex.

To compute such a representation for a vertex, we use a generalized Cantor Polynomial [26] $g_k : \mathbb{N}^k \longrightarrow \mathbb{N}$, defined as follows:
$\forall (x_1, x_2, x_3, \cdots, x_k) \in \mathbb{N}^k$ and $k > 0$

$$g_k(x_1, x_2, x_3, \cdots, x_k) = \sum_{j=1}^{k} \hbar(j, x_1 + ... + x_j) \tag{1}$$

where

$$\hbar(p,s) = \binom{s+p-1}{p} = \frac{(s+p-1)!}{p!(s-1)!} \tag{2}$$

This is a pairing function known to be a bijective function [17, 23, 29].

To use this bijection on a graph, it suffices to assign distinct integer values to distinct vertex attributes. This assignment can be simply achieved by numbering attributes parting from 1. So, the value of k is the number of distinct labels in the graph. To compute $\hbar(j, x_1 + ... + x_j)$, j corresponds to a vertex label, and x_j is the number of apparition of attribute j in the direct neighborhood of vertex u. With this definition, if two vertices in the graph are not isomorphic at one-hop, then they can never have the same CNR even if they have the same degree and attribute.

Example:

Figure 1 illustrates an example where we compute the CNR for the 5 vertices, u_1, u_2, u_3, u_4, and u_5, on a graph having 4 distinct attributes 1, 2, 3, and 4, i.e., $k = 4$:

$cnr(u_1) = g_4(0,2,0,0) = \hbar(1,0)+\hbar(2,0+2)+\hbar(3,0+2+0)+\hbar(4,0+2+0+0) = 0+3+4+5 = 12$. In fact, we can see that attributes 1, 3 and 4 do not appear in the neighborhood of of u_1 and attribute 2 appears 2 times. For the remaining vertices, we have similarly:

$cnr(u_2) = g_4(1,1,0,1) = \hbar(1,1)+\hbar(2,1+1)+\hbar(3,1+1+0)+\hbar(4,1+1+0+1) = 23$.

$cnr(u_3) = g_4(1,1,1,1) = \hbar(1,1)+\hbar(2,1+1)+\hbar(3,1+1+1)+\hbar(4,1+1+1+1) = 49$.

$cnr(u_4) = g_4(0,2,0,0) = \hbar(1,0)+\hbar(2,0+2)+\hbar(3,0+2+0)+\hbar(4,0+2+0+0) = 0+3+4+5 = 12$

$cnr(u_5) = g_4(0,1,0,0) = \hbar(1,0)+\hbar(2,0+1)+\hbar(3,0+1+0)+\hbar(4,0+1+0+0) = 3$.

We can see that nodes u_1 and u_4 which are isomorphic at 1-hop, i.e., they have a similar neighborhood, have the same CNR value.

It is worthy of note that two factors play a major role in calculating the CNR for nodes in the graph; the first is k which is bounded by the set of attributes in the graph, with k being the number of distinct attributes in Graph G. The second factor is the maximum degree in the input graph. The denser the graph, the larger the maximum degree of the graph; which also results into a big CNR integer values in relation to $\frac{\triangle^k}{k!}$ where $\triangle$ is the maximum degree of graph. To tackle this, we use a simple log function to scale the CNR values for each vertex. With this, we effectively handle the *"skewness"* among the CNR integer outputs without loss of inherent properties encoded within the integer. It is also interesting to see that the

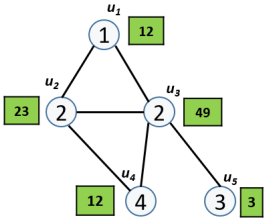


Figure 1: CNR computation on an example.

computation of CNR values does not require to load all the graph in memory. Only the direct neighbors of each vertex are needed.

Also, we use here the pairing function so that the computed CNR captures the one-hop neighborhood of a vertex but it is interesting to see that it possible to also capture the t-hops neighborhood, $t > 1$, by considering several values by vertex.

6 Experimental Evaluation

In this comparative study, we evaluate how the learned embedding from three state-of-the-art embedding models *DeepWalk* [24], *Node2vec* [10], and *LINE* [13] perform on *Link Prediction and Node Classification tasks*; first as stand-alone embedding, and then with the introduction of CNR encoding as an additional important feature. For each dataset used in this study, we learned embedding in 32, 64, and 128 dimensions. The aim is to find out how this additional feature affects the overall result of the tasks. Specifically, we focus on the following research question: To what extent does introducing an additional encoded feature (i.e., CNR) impacts on the overall performance in the link prediction and node classification tasks?

6.1 Datasets

In our experiments, we used seven real life datasets (see Table 1 for a summary) from different domains [27]:

- The popular Cora dataset consists of Machine Learning papers. These papers are classified into one of the following seven classes: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory; thus dividing the cora network into 7 labels. In the dataset corpus, every paper cites or is cited by at least one other paper.

Table 1: Datasets

Dataset	Nodes	Edges	Density	Maximum Degree	Num. of Classes
Cora	2708	5429	$1.4\ 10^{-3}$	169	7
FIRSTMM	56.6k	252k	$1.5\ 10^{-4}$	20	5
Proteins	43.5k	162.1k	$1.7\ 10^{-4}$	50	3
NCI-1	122.3k	265.5k	$3.5\ 10^{-5}$	8	25
Enzymes	19.5k	75k	$3.9\ 10^{-4}$	18	3
DHFR	32k	67k	$1.3\ 10^{-4}$	8	9
Political Retweets	18.5k	61.2k	$3.5\ 10^{-4}$	1k	2

- The FIRSTMM DB dataset contains a set of graphs corresponding to 3D point cloud data and categories of various household objects for semantic and graph-based object category prediction and has 33 training graphs and 4 graphs each for validation and testing. The dataset is from the Robotics Domain.
- The proteins dataset is a collection of proteins that are classified as enzymes or non-enzymes. Nodes represent the amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart.
- The NCI-1 dataset consists of small molecules with class labels representing toxicity or biological activity determined in drug discovery projects. Here, the nodes take the places of atoms and edges are the chemical bonds between each atom. Consequently, the labels encode atom and bond types, possibly with additional chemical attributes.
- The Enzymes dataset consists of 600 protein tertiary structures obtained from the BRENDA enzyme database. The enzymes dataset contains 6 enzymes, corresponding to its labels.
- The Dihydrofolate Reductase Inhibitors (DHFR) dataset contains values for 325 compounds. For each compound, 228 molecular descriptors have been calculated. Additionally, each sample is designated as "active" or "inactive".
- The Soc-political-retweet networks consists of the most retweeted user in a political debate. Nodes are the users of the twitter social network while the edges represents the retweets between them.

6.2 Methods and Settings

As already stated, we adopted the Link Prediction and Node Classification tasks to answer the questions posed in section 5. For the purpose of achiev-

ing the link prediction analysis, we designed a logistic regression model for labeled indices. We consciously opted for such model because (i) It is very efficient to train and fast at classifying unknown occurrences. (ii) Since our features are linearly separable with no multicollinearity, this model is ideal. (iii) It is effective at interpreting model coefficients based on important attributes in the features. (iv) It is less inclined to overfitting. The idea behind the link prediction task is to measure the accuracy at predicting missing links in the network. To achieve this, we consider a network with certain fractions of edges removed. We generated the training dataset by randomly removing varying percentage of edges from the network while also ensuring that the network obtained after removing the edges still remains connected. For the node classification task however, we designed a Graph Convolutional Network (GCN) model. The GCN hyperparameters includes 2 layers with hidden size 16, training was done using the Adam Optimizer [6] and softmax was used for output classification, with learning rate 0.01 and weight decay 5×10^{-4}.

To measure the models' performance, we use the following metrics:

- Area under the precision-recall curve: Recall measures what percentage of positives were returned. Precision measures what percentage of the results are true positives. These metrics are used to construct a Precision-Recall Curve which shows how the increase in recall affects precision.

- Area under the receiver operating characteristics curve: A combination of the True positive rate (recall) and the false positive rate which measures how many negatives were returned as false positives, are used to construct a Receiver Operating Characteristics (ROC).

- Precision@k: The Precision at k metric returns the percentage of true positives among only the top k ranked links. It is a metric for measuring link equality. Mathematically, precision is expressed as $\frac{True\ \ Positive(TP)}{True\ \ Positive(TP)+False\ \ Positive(FP)}$.

- F-score: The F-score (also known as F1-score) is a measure of a model accuracy on a dataset. The F-score is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model precision and recall. Mathematically, it is represented as $2 \times \left(\frac{precision \times recall}{precision + recall}\right)$.

6.3 Results for Link Prediction Experiments

For this task, we tried to evaluate how well our model predicted missing links using the information embedded in the learned representation. In order to get a baseline score for link prediction, we selected five heuristics metrics:

Table 2: Link Prediction Heuristic Scores for the seven selected networks

Heuristic Method	Cora	First MM	Protein	Enzymes	Political Retweets	DHFR	NCI 1
Common Neighbor	0.6631	0.6843	0.7119	0.7481	0.7012	0.7567	0.7579
Jaccard Neighbour	0.6580	0.6729	0.6932	0.6610	0.6521	0.6341	0.6737
Preferential Attachment	0.6873	0.6891	0.7141	0.6783	0.7012	0.6862	0.7119
Adamic/Adar Index	0.7063	0.7168	0.7585	0.7705	0.7164	0.7521	0.7781
Resource Allocation	0.6831	0.7045	0.6951	0.6859	0.7067	0.6872	0.7106

- *Common Neighbor (CN)* $= |\Gamma(x) \cap \Gamma(y)|$,
- *Jaccard's Coefficient* $= |\frac{\Gamma(x) \cap \Gamma(y)}{\Gamma(x) \cup \Gamma(y)}|$,
- *Preferential Attachment (PA)* $= |\Gamma(x)||\Gamma(y)|$,
- *Adamic/Adar Index (AI)* $= \Sigma_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{log|\Gamma(z)|}$, and
- *Resource Allocation Index* $= \Sigma_{z \in \Gamma(x) \cap \Gamma(y)} \frac{1}{|\Gamma(z)|}$.

where x and y denotes nodes, $\Gamma(x)$ and $\Gamma(y)$ denotes the neighbor set of these nodes, while z is the common neighbor of node x and node y [8]. Scores of heuristics for every dataset are given in Table 2.

To evaluate performance of learning models, all nodes and edges were taken into consideration; and we ensured that while splitting our data into training and test data, the network remained largely connected thus avoiding disconnected components. The dataset was split in the ratio 80:20 with 80% of the data used for training the model and the remaining 20% for evaluation. We first conducted a link prediction analysis using varied dimensions (32, 64, and 128) of learned embedding from the three selected state-of-the-art models. For the second part, we merge the originally learned embedding with the encoded CNR feature for each node and we perform the same series of link prediction analysis a second time using the newly merged features and under the same set of conditions and parameters. The results for link prediction analysis is discussed in this section.

We discuss the results of the link prediction analysis from the perspective of each of the embedding. The results in Table 3 pretty much shows the effect of the CNR encoding as an additional feature in the learned embedding.

While we got a decent prediction accuracy using learned embedding from state-of-the-art models, clearly the introduction of an extra encoded feature

for each vertex increased the accuracy of our link prediction model in predicting previously unseen connections considerably. This is possible due to a number of reasons which includes:

1. CNR encoding is sensitive to structural features of the network, since it encodes node based on other nodes in its neighborhood. And since the link prediction analysis involves predicting structural connection between nodes, it makes the introduction of CNR improve the predictive accuracy of the model.

2. For each vertex u, the information stored in the surrounding neighbors of u are encoded as a single integer for vertex u. Each neighbor of u also takes and encodes information about its respective neighbors. As a result of this, each node has the capacity of storing several metadata in a single Integer. This property gives graphs encoded with CNR technique the ability to preserve an entire graph property in a set of integers.

In addition, across the AUC results for all seven datasets, our model was able to correctly predict the true positive edges out of all the edges removed from the dataset for testing. This is reflected in the high Precision values, which grows even higher when we factored in the CNR encoding. Another interesting finding is that for every increment in the embedding dimensions, we record a lower link prediction accuracy which reflects for AUCs without CNR and with the introduction of CNR encoded integers. This could be as a result of adding several unnecessary features which reduces the performance of the embeddings in learning tasks.
Finally when we compare both results (link prediction analysis without CNR and link prediction with CNR) with standard heuristics score shown in Table 2, we get a much higher differentials for the link prediction analysis with CNR. From our findings, it seems clear that CNR encoding thrives well for massive networks which higher maximum degree since such networks have more surrounding nodes for each node in the network, and by extension more surrounding information to encode. This is evident in FirstMM, NCI-1 and proteins datasets, which are the biggest of all seven datasets used in this study, having much higher prediction accuracy of almost 90% with the introduction of the single encoded integer as an additional feature.

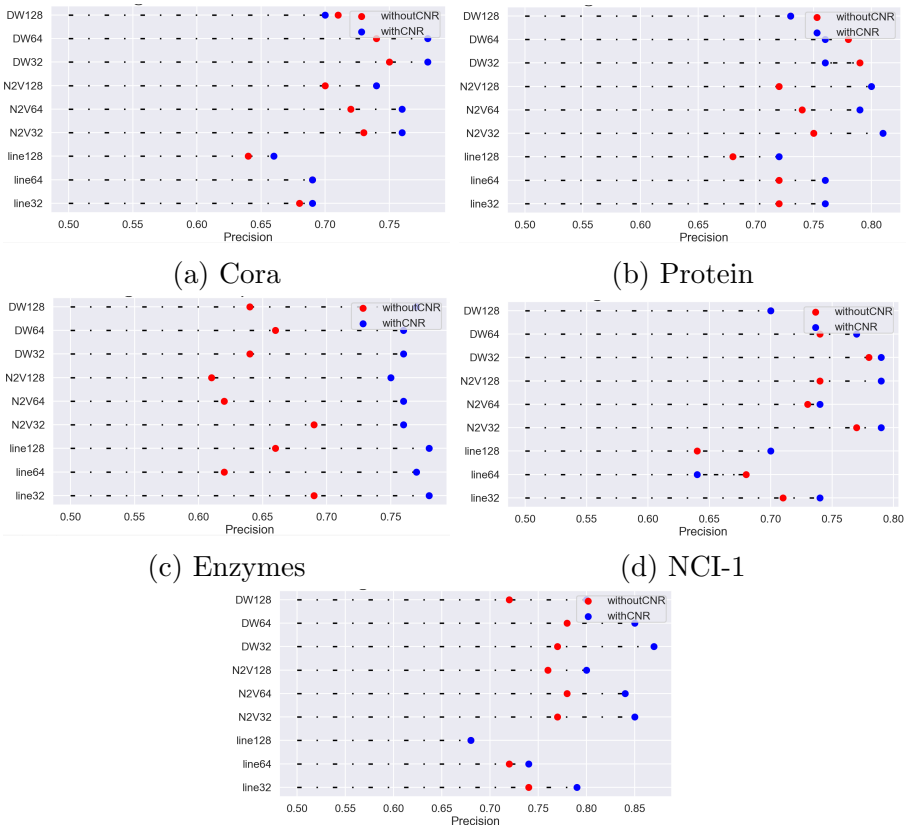
6.4 Results for Node Classification Experiments

In many real life networks, the nodes in the network can have some kind of relationships based on certain intrinsic characteristics of each node. For example, we might want to classify nodes based on the number of inward/outward

edges, or classify nodes on the same structural neighborhood, or sometimes find some unknown relationships in an unsupervised environment. In our study, we first learn vector embedding for nodes in an labeled graph (labels are assigned to nodes based on the class of interest they belong in), and then we designed a GCN model to classify the nodes using the learned embedding. The model was trained with 200 epochs, early stopping on validation loss with 10 epochs of patience, and a categorical cross-entropy loss function. On the other hand, we conduct the same experiment under the same environment and parameters but this time, with CNR encoding for each node added as an additional feature in the embedding. As a way of quality assurance, we repeat both sets of experiments with a 32, 64, and 128 dimension embedding. We evaluate the results from our experiments using the AUC precision metric and the F1-score to measure how precise the model accuracy was [32] [15].

The results in Figure 2 and F1-score in Table 4 shows significant degree of improvement in the classification accuracy with the additional CNR as an extra embedding layer for each node. The precision@k is used to measure the confidence level of the model classification result. in simple terms, Precision is the ratio between the True Positives (correctly predicted positive observations) and all the Positives (total predicted positive observations) in the dataset. Therefore a high precision means a low false positive rate, and vice versa. In the context of our study, we use precision to measure nodes that were correctly classified into their respective class out of all the nodes in each class. for example, if a class has 50 nodes, a precision of 0.75 means the model correctly classified about 38 nodes as belonging to that class out of the total 50 nodes. For each dataset, we run the classification model for 32, 64, and 128 dimension embedding for *Node2Vec*, *DeepWalk*, and *LINE* embedding models as shown in Figure 2. While we achieve a decent classification accuracy with Node2Vec and Deepwalk models, the addition of the compact neighborhood representation significantly improves the classification accuracy as seen in the precision output. With the addition of CNR as an extra layer of embedding the classification model accurately classified the nodes in Protein and Enzymes dataset by over 75% and by atleast 85% in the DHFR dataset. The F1-score takes both false positives and false negatives into account, with a good F1-score indicative of a good Precision accuracy. The result in Table 4 correlates with the precision accuracy in Figure 2, with the F1-score considerably higher with CNR encoding. As with the case of the Link prediction analysis, it is evident that as we vary the number of dimensions from 32 through 128, the classification accuracy reduces; the *curse of dimensionality theory* quickly springs to mind. This theory follows that as the number of dimensions increases, so does the addition of repetitive and

sometimes unnecessary features (perhaps noise) increases, which increases the model error and thus result in a loss in accuracy. It is therefore important to determine an optimal embedding length in a representation learning problem, so as to get a good embedding output for machine learning tasks.



(a) Cora

(b) Protein

(c) Enzymes

(d) NCI-1

(e) DHFR

Figure 2: Precision@k Results for Node Classification (DW and N2V denotes DeepWalk and Node2Vec respectively)

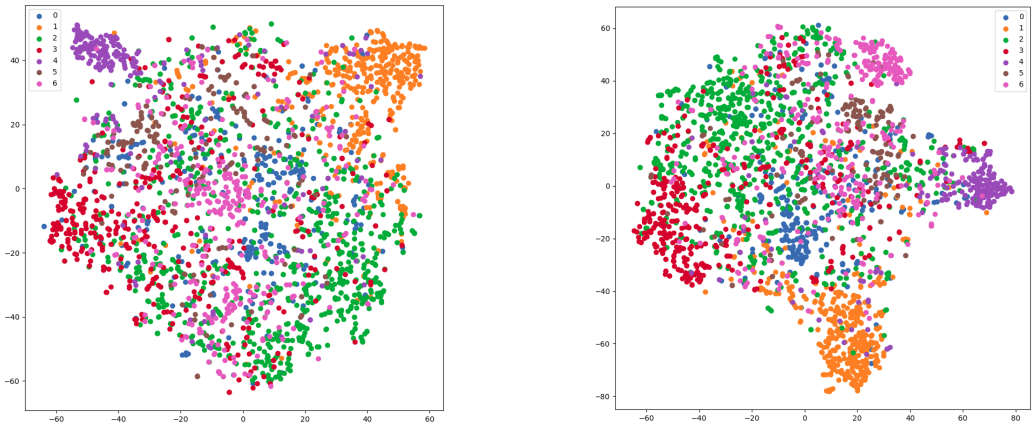
Table 3: Evaluation Results : **Accuracy Precision Recall**

Dataset	Node2Vec			Node2vec Plus CNR			DeepWalk			DeepWalk Plus CNR			LINE			LINE Plus CNR		
Dimension																		
Cora32	0.8703	0.8860	0.8703	0.8462	0.8642	0.8230	0.8526	0.8797	0.8526	0.8352	0.8567	0.8284	0.7512	0.7720	0.7512	0.7724	0.7969	0.7551
Cora64	0.7658	0.7905	0.7658	0.8364	0.8464	0.8275	0.7529	0.7941	0.7529	0.8475	0.8598	0.8308	0.5407	0.5750	0.5407	0.7630	0.7466	0.7630
Cora128	0.7892	0.7180	0.7892	0.8475	0.8692	0.8275	0.6525	0.6527	0.6525	0.8352	0.8602	0.8106	0.5395	0.5688	0.5395	0.6089	0.6091	0.6089
Proteins32	0.8891	0.9132	0.8979	0.8862	0.9195	0.8889	0.8923	0.9262	0.8923	0.8926	0.9334	0.8929	0.7921	0.7666	0.7921	0.8762	0.9099	0.8762
Proteins64	0.7378	0.7518	0.7378	0.8667	0.9169	0.8867	0.7736	0.7979	0.7736	0.8763	0.9160	0.8763	0.7103	0.7047	0.7103	0.8791	0.9298	0.8791
Proteins128	0.7364	0.7295	0.7364	0.8809	0.9241	0.8809	0.7501	0.7975	0.7501	0.8986	0.9335	0.8986	0.6708	0.6857	0.6708	0.8834	0.9321	0.8834
FirstMM32	0.9275	0.9353	0.9268	0.9773	0.9801	0.9773	0.8436	0.8582	0.8436	0.9836	0.9882	0.9836	0.7671	0.7993	0.7671	0.9874	0.9896	0.9874
FirstMM64	0.8419	0.8487	0.8341	0.9831	0.9843	0.9831	0.7464	0.7629	0.7464	0.9898	0.9912	0.9898	0.7533	0.7989	0.7533	0.9871	0.9893	0.9871
FirstMM128	0.8652	0.8678	0.8606	0.9869	0.9877	0.9869	0.7871	0.8360	0.7839	0.9779	0.9795	0.9779	0.7310	0.7732	0.7310	0.9474	0.9893	0.9474
Enzymes32	0.7904	0.7818	0.7815	0.9238	0.9579	0.9390	0.7676	0.7692	0.7687	0.9308	0.9362	0.9245	0.7844	0.7739	0.7527	0.8821	0.8754	0.8794
Enzymes64	0.7419	0.7233	0.7384	0.8648	0.8547	0.8543	0.7248	0.7176	0.7145	0.8740	0.8795	0.8634	0.7403	0.7601	0.7419	0.8101	0.8022	0.8029
Enzymes128	0.7257	0.7122	0.7241	0.8281	0.8123	0.8230	0.7104	0.6615	0.6973	0.8407	0.8377	0.8295	0.7485	0.7322	0.7477	0.8091	0.8076	0.8109
DHFR32	0.7631	0.7642	0.7681	0.9451	0.9518	0.9451	0.7569	0.7412	0.7510	0.8742	0.8806	0.8726	0.7422	0.7448	0.7369	0.7989	0.7815	0.7921
DHFR64	0.7730	0.7793	0.7750	0.9011	0.9128	0.9011	0.7487	0.7453	0.7490	0.8501	0.8494	0.8483	0.7124	0.7022	0.7124	0.7446	0.7501	0.7446
DHFR128	0.7311	0.7328	0.7239	0.8854	0.8902	0.8854	0.7178	0.7106	0.7131	0.8246	0.8297	0.8246	0.6889	0.6919	0.6910	0.7559	0.7505	0.7498
Politics32	0.8852	0.8743	0.8809	0.9717	0.9794	0.9717	0.8698	0.8703	0.8698	0.9683	0.9515	0.9683	0.7951	0.7903	0.7947	0.9643	0.9536	0.9643
Politics64	0.8365	0.8420	0.8365	0.9195	0.9255	0.9201	0.8665	0.8612	0.8665	0.9213	0.9228	0.9213	0.7538	0.7491	0.7557	0.7957	0.8009	0.7957
Politics128	0.7976	0.7933	0.7905	0.8823	0.8911	0.8823	0.7743	0.7718	0.7743	0.8841	0.8920	0.8841	0.7385	0.7485	0.7393	0.8164	0.8273	0.8164
NCI32	0.8852	0.8736	0.8835	0.9548	0.9630	0.9548	0.8733	0.8609	0.8739	0.9441	0.9501	0.9441	0.7752	0.7640	0.7721	0.8843	0.8861	0.8843
NCI64	0.8305	0.8365	0.8305	0.8894	0.8859	0.8890	0.8115	0.8296	0.8115	0.8355	0.8297	0.8355	0.7665	0.7513	0.7668	0.8388	0.8594	0.8412
NCI128	0.7622	0.7706	0.7622	0.8481	0.8329	0.8490	0.7733	0.7619	0.7733	0.8144	0.8189	0.8144	0.7361	0.7460	0.7361	0.8077	0.8125	0.8173

Table 4: F1-Score

Dataset Dimension	Node2Vec	Node2vec Plus CNR	DeepWalk	DeepWalk Plus CNR	LINE	LINE Plus CNR
Cora32	0.65	0.70	0.66	0.68	0.62	0.68
Cora64	0.64	0.62	0.60	0.60	0.52	0.55
Cora128	0.60	0.60	0.58	0.60	0.50	0.53
Proteins32	0.68	0.74	0.69	0.73	0.66	0.66
Proteins64	0.66	0.72	0.64	0.71	0.61	0.60
Proteins128	0.58	0.64	0.60	0.64	0.60	0.62
FirstMM32	0.67	0.72	0.65	0.70	0.63	0.68
FirstMM64	0.65	0.71	0.62	0.70	0.64	0.66
FirstMM128	0.68	0.67	0.62	0.68	0.62	0.62
Enzymes32	0.72	0.83	0.69	0.83	0.69	0.79
Enzymes64	0.70	0.82	0.70	0.80	0.68	0.77
Enzymes128	0.65	0.80	0.66	0.80	0.68	0.78
DHFR32	0.78	0.85	0.76	0.85	0.73	0.79
DHFR64	0.77	0.81	0.71	0.80	0.71	0.75
DHFR128	0.66	0.78	0.64	0.77	0.64	0.70
Politics32	0.79	0.83	0.77	0.83	0.73	0.80
Politics64	0.75	0.82	0.76	0.81	0.69	0.77
Politics128	0.78	0.80	0.74	0.81	0.70	0.78
NCI32	0.75	0.78	0.77	0.80	0.71	0.75
NCI64	0.73	0.73	0.73	0.75	0.70	0.72
NCI128	0.72	0.74	0.72	0.71	0.71	0.70

*Dim. denotes 32/64/128 Dimensions varied for each dataset.



(a) Without CNR (b) With CNR

Figure 3: Visual display for Cora Dataset with 7 Labels

Compact Neighborhood Representation can be an interesting solution to this problem since we are able to encode every information and characteristic feature surrounding a node in just a single integer.
In Figure 3, we try to visualize the hidden layer representations of our model in 2-dimensions. Each point on the plane represents each node in the network, while each color represents each class to which a node belongs in. Figure 3 shows the hidden layer activation for cora dataset which consists of machine learning papers grouped in the following seven label; Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory. Compared to the visualization in Figure 3a without the encoded neighborhood index integer, Figure 3b shows some degree of class distinction for the seven classes of papers in the dataset. We still see some papers in other clusters, this can be attributed to the fact that in the document corpus, some papers were cited more than once. However, we can still get a clear picture of the different classes of papers in Figure 3b compared to Figure 3a. Overall, the node classification analysis with CNR encoding out performs the classification without CNR by up to 15%, which is a significant improvement in accuracy. As described in Section 4, since each vertex in a network is capable of effectively holding higher-order information about other vertices (described in the node attributes), the neighborhood compact representation technique is quite effective for node classification.

7 Conclusion

This study involves a performance evaluation of state-of-the-art embedding models (Node2Vec [10], DeepWalk [24], and LINE [13]) against a compact neighborhood representation technique that captures in a single dimension both structural and semantic information around vertices. We conducted two machine learning tasks on seven real life networks, with learned vector representations from those models on one hand, and then we introduced the CNR encoding as an additional feature for an improved accuracy on the other hand. Results from the both experiments show that we were able to improve our link prediction and node classification accuracy significantly with the addition of the CNR feature as an additional layer of embedding. This confirms that this vertex representation carries important information about nodes. As part of ongoing work, we will like to explore how much role CNR encoding will play in massive temporal and streaming networks in areas of detecting anomalies and pattern matching. Another interesting area is to represent each distinct characteristic feature in a network with a CNR, thus being able to effectively distinguish a feature from another and perform certain tasks on

them separately. Is is also interesting to see that CNR can be easily extended to encode edge attributes and k-hop neighboring-hood. This may enhance the accuracy of the computed embedding in certain applications.

Acknowledgements

For the research leading to these results, Hamida Seba and Mohammed Haddad received funding from the Agence National de la Recherche under Grant Agreement No ANR-20-CE23-0002, Ikenna Oluigbo was supported by Petroleum Technology Development Fund, Nigeria with grant number PTDF/GFC/035 The source code of the algorithms is available at: `https://gitlab.liris.cnrs.fr/hseba/cnr`.

References

[1] B. Adhikari, Y. Zhang, N. Ramakrishnan, and B. A. Prakash. Sub2vec: Feature learning for subgraphs. In *PAKDD*, 2018.

[2] N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry. Learning role-based graph embedding. pages 1–8, 2018.

[3] O. Balalau and S. Goyal. Subrank: Subgraph embeddings via a subgraph proximity measure. *Advances in Knowledge Discovery and Data Mining*, 12084:487 – 498, 2020.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, 2002.

[5] S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management CIKM '15*, pages 891–900. ACM, 2015.

[6] K. Diederik and B. Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[7] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1320–1329. ACM, 2018.

[8] F. Gao, K. Musial, C. Cooper, and S. Tsoka. Link prediction methods and their accuracy for different social networks and network metrics. *Sci. Program.*, 2015:172879:1–172879:13, 2015.

[9] M. Gao, L. Chen, X. He, and A. Zhou. Bine: Bipartite network embedding. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval SIGIR '18*, pages 715–724. ACM, 2018.

[10] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[11] W. Gu, A. Tandon, Y.-Y. Ahn, and F. Radicchi. Principled approach to the selection of the embedding dimension of networks. *Nature Communications*, 12(3772), 2021.

[12] P. D. Hoff, A. E. Raftery, and M. S. Handcock. Latent space approaches to social network analysis. *Journal of the American Statistical Association*, 97:1090 – 1098, 2002.

[13] T. Jian, Q. Meng, W. Mingzhe, Z. Ming, Y. Jun, and M. Qiaozhu. Line: Large-scale information network embedding. *Proceedings of the 24th International Conference on World Wide Web*, pages 1067 – 1077, 2015.

[14] T. Kipf and M. Welling. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, ArXiv/1611.07308, 2016.

[15] L. Lan, P. Wang, X. Du, K. Song, J. Tao, and X. Guan. Node classification on graphs with few-shot novel labels via meta transformed network embedding. *ArXiv*, abs/2007.02914, 2020.

[16] A. Li and Y. Pan. Structural information and dynamical complexity of networks. *IEEE Transactions on Information Theory*, 62:3290–3339, 2016.

[17] M. Lisi. Some remarks on the cantor pairing function. *LE MATEMATICHE*, LXII-Fasc. I:55–65, 2007.

[18] G.-X. Luo, J. Li, H. Peng, C. Yang, L. Sun, P. S. Yu, and L. He. Graph entropy guided node embedding dimension selection for graph neural networks. In *IJCAI*, 2021.

[19] I. Makarov, D. Kiselev, N. Nikitinsky, and L. Subelj. Survey on graph embeddings and their applications to machine learning problems on graphs. *PeerJ Computer Science*, 7, 2021.

[20] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *Computing Research Repository (CoRR)*, abs/1301.3781, 2013.

[21] C. Nabti and H. Seba. Compact neighborhood index for subgraph queries in massive graphs. *arXiv: Databases*, 2017.

[22] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. P. Liu, and S. Jaiswal. graph2vec: Learning distributed representations of graphs. *ArXiv*, abs/1707.05005, 2017.

[23] M. Nathanson. Cantor polynomials and the fueter-pólya theorem. *The American Mathematical Monthly*, 123:1001 – 1012, 2015.

[24] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710. ACM, 2014.

[25] B. Perozzi, V. Kulkarni, H. Chen, and S. Skiena. Don't walk, skip!: Online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining ASONAM '17*, pages 258–265. ACM, 2017.

[26] G. P. Rudolf Fueter. Rationale abzählung der gitterpunkte, vierteljschr. *Naturforsch. Ges, Zürich*, 58:280–386, 1923.

[27] R. Ryan and A. Nesreen. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[28] N. Sheikh, Z. T. Kefato, and A. Montresor. gat2vec: representation learning for attributed graphs. *Journal of Computing*, 101(3):187–209, 2018.

[29] S. K. Stein. *Mathematics: The Man-Made Universe. New York: McGraw-Hill, 1999.* Dover Publications; 3rd Revised ed., March 21 2013.

[30] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu. Structural deep embedding for hyper-networks. *arXiv preprint arXiv:1711.10146*, 2017.

[31] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '16*, pages 1225–1234. ACM, 2016.

[32] R. Yacouby and D. Axman. Probabilistic extension of precision, recall, and f1 score for more thorough evaluation of classification models. In *EVAL4NLP*, 2020.

[33] Z. Yin and Y. Shen. On the dimensionality of word embedding. In *NeurIPS*, 2018.

[34] X. Zhu, C. Lei, H. Yu, Y. Li, J. Gan, and S. Zhang. Robust graph dimensionality reduction. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3257–3263. AAAI Press, 2018.