Neural Processing Letters (2024) 56:226
https://doi.org/10.1007/s11063-024-11683-0

Finding Efficient Graph Embeddings and Processing them by a CNN-based Tool

Attila Tiba[1] · Andras Hajdu[1] · Tamas Giraszi[1]

Accepted: 16 August 2024
© The Author(s) 2024

Abstract

We introduce new tools to support finding efficient graph embedding techniques for graph databases and to process their outputs using deep learning for classification scenarios. Accordingly, we investigate the possibility of creating an ensemble of different graph embedding methods to raise accuracy and present an interconnected neural network-based ensemble to increase the efficiency of the member classification algorithms. We also introduce a new convolutional neural network-based architecture that can be generally proposed to process vectorized graph data provided by various graph embedding methods and compare it with other architectures in the literature to show the competitiveness of our approach. We also exhibit a statistical-based inhomogeneity level estimation procedure to select the optimal embedding for a given graph database efficiently. The efficiency of our framework is exhaustively tested using several publicly available graph datasets and numerous state-of-the-art graph embedding techniques. Our experimental results for classification tasks have proved the competitiveness of our approach by outperforming the state-of-the-art frameworks.

Keywords Graph embedding · Ensemble of embeddings · Graph datasets · Convolutional neural networks · Graph classification

1 Introduction

Data preparation in machine learning revolves around three key concepts: data collection, data preprocessing, and data transformation. Generally speaking, the more quality data fed to a machine learning algorithm, the better because present-day problems require finding a recognizable pattern in the dataset. These patterns form the foundations of machine learning models; therefore, using well-preprocessed data is critical. The first problem that algorithms tackle is reading data so that it can be stored and subsequently used for learning. A standard solution is to embed the data in a finite-dimensional real vector space $\mathbb{R}^d$. This vector-

✉ Attila Tiba
tiba.attila@inf.unideb.hu

Andras Hajdu
hajdu.andras@inf.unideb.hu

Tamas Giraszi
giraszi.tamas@inf.unideb.hu

[1] Faculty of Informatics, University of Debrecen, Kassai str. 26, Debrecen, Hajdu-Bihar 4028, Hungary

ization step is sometimes quite simple and obvious. For example, the spatial domain can be quite easily handled by convolutional neural networks (CNNs) since digital images are already represented as matrices of pixels. On the other hand, data generated by integrating objects is hard to embed since we have a non-vector graph representation in such cases. Graph-based approaches are very much required in many application fields, including medical research, chemistry, social networks, and search engines since this representation can generally describe objects and their relations. According to the above discussion, for these scenarios, we have to efficiently transform the graphs to $\mathbb{R}^d$ before machine learning occurs; the present work will focus on related issues.

By definition, graphs are mathematical structures consisting of objects and their relations between them. More formally, a graph $G = (V, E)$ is composed of a set of vertices $V = \{v_1, \ldots, v_n\}$ and a set of edges $E \subseteq V \times V = \{(v_i, v_j) \mid i, j = 1, \ldots, n,\ i \neq j\}$. We will denote the number of vertices by $|V|$. If the graph is directed, the pairs of vertices that form edges are ordered. A weighted graph is denoted by $G = (V, E, W)$ where $W : E \rightarrow \mathbb{R}$ assigns a real value to each edge. The degree of a vertex is the number of edges that are incident to it. The transformations of the graphs into vectors are commonly referred to as graph embeddings [1], and graph convolutions [2]. Thus, graph embedding is a mapping from a collection of vertices and edges to $\mathbb{R}^d$. We will primarily consider graph embeddings as a function $f : G \rightarrow \mathbb{R}^d$ with $d \leq |V|$.

This paper analyzes and compares the most popular graph embedding algorithms using different performance metrics in classification tasks. The evaluation of these algorithms is based on several publicly available graph databases. We also introduce a complex general neural network that can build on any graph embedding approach to extract essential information from graph-based objects. Our proposed CNN architecture is fully optimized for the graph embedding process.

In addition, we also investigate the best-fit graph embedding for each of our databases to make recommendations for selecting the appropriate algorithms by describing the advantages and disadvantages of using them. These evaluations are based on various performance metrics, and we also introduce an inhomogeneity measure for a graph database to support finding the most efficient embedding. To further improve graph embedding, we suggest ensemble-based systems to highlight situations where significant performance improvements can be achieved using the fusion of multiple embedding algorithms. To test the effectiveness of our proposed techniques, we also benchmarked them against one of the most widely used high-quality deep learning tools, DeepChem [3].

The rest of the work is organized as follows. In Sect. 2, we present the publicly available graph datasets included in our experimental analysis. These databases also reflect the wide occurrence of graph representations, including health (DNA, AIDS, diet), molecule, scientific citations, and fingerprint-related ones. A detailed description of the investigated graph embedding approaches is enclosed in Sect. 3. As for novel methodology, here we also introduce

- our proposed CNN with a unique architecture, which provides a unified and efficient processing unit for different graph embeddings,
- our ensemble creation methods to join different graph embedding techniques, exploring the possibility of interconnected ensembles in addition to majority voting-based ones, and demonstrate the effectiveness of ensemble-based approaches also for graph embedding,
- a statistical method for estimating the level of inhomogeneity of a graph dataset to select the appropriate embedding.

Section 4 describes the standard classification performance metrics used for evaluation and our experimental results for each dataset regarding all the embedding approaches. The dominance of our CNN model over other state-of-the-art methods is also demonstrated here. Finally, some discussions are made, and conclusions are drawn in Sect. 5.

2 Graph Databases

This section presents the publicly available graph datasets that will be involved in our analysis.

2.1 MUTAG and Mutagenicity

The datasets MUTAG and Mutagenicity [4, 5] are similar since they both store mutagen data in graph representation. Mutagenicity can substantially affect the DNA, genes, chromosomes, or even the whole genome, which increases the prevalence of genetic mutations. The mutagenic effect is also classified as carcinogenic since it develops from cancerous mutations. Molecules with mutagenic or non-mutagenic effects are transformed into graph structures, incorporating atoms as nodes and chemical bonds as edges. This dataset can be used for binary classification to determine whether mutagenic effects are either carcinogenic or non-carcinogenic. The MUTAG database contains 188 heterocyclic aromatic compounds, which consist of 18 nodes on average. In contrast, the Mutagenicity database contains 4773 graphs, which consist of 30 nodes on average. Long-term health effects of exposure to these compounds were tested by professionals using the salmonella bacteria. Initially, the purpose of both databases was to help a research team to evaluate machine learning methods. The classification error rate ranged between 15 and 18% [6].

2.2 AIDS

The dataset AIDS [4, 5] consists of molecules, represented as graphs, that affect the AIDS disease caused by HIV. We used two of the same type of databases, the overall graph count being the only difference. The first database contains 1110 graphs with 16 nodes on average, whereas the second one includes 2000 graphs with equal node count. The database proposes a binary classification problem in which the two classes refer to whether a molecule has an impeding effect on spreading the virus.

2.3 COX2

The dataset COX2 [4, 5] contains nonsteroidal anti-inflammatory drugs that dissuade hemorrhages, gastrointestinal diseases, and unwanted kidney problems. These compounds are great alternatives to hormonal drugs, considering they solve the issues with minimal long-lasting effects. Our database contains 467 distinct compounds with similar results on COX enzymes, having 41 nodes on average. The database proposes a binary classification problem in which the two classes refer to whether a molecule has an impeding effect on a COX enzyme.

2.4 Tox21_ARE

The dataset Tox21_ARE [4, 5] contains various antioxidants that help with oxidative stress. The database initially consists of training, validation, and test parts. We used cross-validation; therefore, we had to unify the data before feeding it into our model. The Tox21_ARE database contains 7953 graphs with 18 nodes on average. The database proposes a binary classification problem in which the two classes refer to whether an antioxidant has an impeding effect on a free radical.

2.5 DBLP_v1

The dataset DBLP_v1 [4, 5] is part of a worldwide project aiming to help scientific papers' availability. The DBLP_v1 database contains 19 456 graphs with ten nodes on average where the edges connecting the nodes are also labeled. We consider each publication a graph where every node is a word, and the edges connecting those nodes represent the context. The graph data in the database contains 20 edges on average. The database proposes a binary classification problem where the positive class means that the publication falls into the data mining category; conversely, the opposite conveys that the publication falls into the machine learning category.

2.6 Fingerprint

The dataset Fingerprint [4, 5] consists of 15 classes, each representing a well-known fingerprint pattern. Graphs in the database consist of nodes representative of a given fingerprint. The database contains 2800 graph objects with 5 nodes on average. We removed graphs with less than 5 nodes to help suppress noise in the data. The database proposes a multi-class classification problem where each class represents a generalized structure of friction ridges in the fingerprints.

3 Methods

This section presents all the analyzed graph embeddings and the DeepChem [3] framework. We also give our approach for creating ensembles of graph embeddings, composing a general CNN model to process vectorized graph data and a noise level estimation technique to help select the most appropriate embedding for a given graph dataset.

3.1 Graph Embedding Algorithms

Graph objects do not have a vector form, so they cannot be used directly as inputs to neural networks. Representing graphs in $\mathbb{R}^d$ requires data transformation algorithms. These methods aim to transform the data to make it suitable for neural network training while minimizing data loss. These algorithms are generally based on well-known graph embedding models such as RandomWalk, Word2vec, and Skip-gram [7, 8]. These three approaches were the building blocks of the algorithms tested.

A random walk in a graph is a stochastic process that begins at some node $v_i \in V$ in a graph G, and at each step moves to another neighboring node v_j with probabilities $p_1, \ldots, p_m$,

where m is the number of neighboring vertices of v_i. The probabilities can and should be changed depending on the algorithm used. Finally, we feed the obtained walks to a Skip-gram model responsible for the embedding.

The Skip-gram model [7] is a well-defined neural network architecture in which the output layer consists of the same number of neurons as the input one. To perform learning on such a model, we need a dataset that contains word pairs (node pairs in our case). These word pairs are usually a product of processing lengthy sentences (random walks in our case). In the processing procedure, we take a window size and start giving context to every word in the sentence. We can formulate word pairs and feed them to the neural network using these contexts. This process is called the Word2vec process.

Next, we describe several embedding algorithms based on these different approaches we have included in our investigations.

3.1.1 DeepWalk

The DeepWalk [9] algorithm provides a vectorial representation of nodes in a graph. Namely, the embedded graph structure is a collection of walks transformed into vectors, creating a 2D feature space. The walks in the graph can be parameterized differently, which means we can customize every learning process to our needs. One of the crucial parameters is walk length, which defines the number of nodes to be traveled during the random walk. The more nodes in a graph, the more possibilities we have for optimization. Using the DeepWalk approach, the Skip-gram model updates itself at every iteration. Each update will reconstruct a paragraph vector containing all the nodes visited before.

3.1.2 node2vec

node2vec [10] is a modified random walk and Skip-gram-based algorithm, one of the most efficient approaches we have tested. It performs the best on semisynthetic structures but also works well on heterophile graph structures and structurally identical data. Its performance on diverse structures is due to the modified random walk algorithm. node2vec walks have an extra parameter allowing them to visit nodes that have already been visited. Balancing the use of breadth-first and depth-first searches creates a structurally almost identical representation of the initial graph. The probability of revisiting a node is given by a parameter p, while the parameter $q = 1 - p$ is the probability of visiting an unknown node.

3.1.3 graph2vec

The graph2vec [11] algorithm uses a so-called doc2vec architecture [12] for its basis, in which documents replace the usage of word vectors. In this interpretation, we can consider the graphs documents because they carry essential information about an object. Using this assumption, we can create and use a modified doc2vec architecture as the basis for our word2vec algorithm, which uses negative sampling and a modified Skip-gram model.

3.1.4 Large-Scale Information Embedding (LINE)

The Large-Scale Information Embedding (LINE) [13] algorithm is a modern graph embedding algorithm used on large information networks. Social networks can contain millions

of nodes, which must be transformed into low-dimensional vector spaces. This transformation is a resource-demanding task; therefore, choosing the Skip-gram model is impractical. Using LINE, we can easily preserve a graph structure's first- and second-order neighborhoods without compromising data loss. This method is ideal for low-level data visualization as well as link prediction. Our databases do not contain graph objects with millions of nodes on average; however, we could not find any studies on how LINE would perform on smaller datasets compared to a Skip-gram-based approach. The main idea of LINE is to encode the local and global network structure by preserving first- and second-order neighborhoods as much as possible.

3.1.5 Structural Deep Network Embedding (SDNE)

The Structural Deep Network Embedding (SDNE) [14] algorithm shares ties to LINE. It aims to transform a large amount of information and social networks into low-dimensional vector space while retaining structural complexity. Information networks share a non-linear structure, which makes data loss minimization a difficult task. The SDNE algorithm introduces a secondary semi-supervised learning model which implements many non-linear layers. Like LINE, it aims to maintain graphs' local and global structures. However, while LINE manages first- and second-order neighborhoods separately, SDNE controls them as a whole.

3.1.6 Patchy-San

The Patchy-San [15] algorithm is based on the concept of convolution used, e.g., in CNNs. It traverses the nodes of graphs in a pre-defined order, which walks become perception kernels later on. Nodes corresponding to each perception kernel are then stored as vectors.

3.1.7 Graph Convolutional Network (GCN)

Another well-documented approach represents nodes of a graph as small signals. This way, a signal can be written as a vector in $\mathbb{R}^d$, having values as graph nodes in a scalar representation. Usually, graph convolutional layers are considered to use classical convolution

$$g_\theta \circledast x = U g_\theta U^T x, \tag{1}$$

as the base of their operations [16], where $g_\theta = diag(\theta)$ is the kernel, $\theta \in \mathbb{R}^d$ is its parameter, U is the matrix from the eigenvectors of the symmetric normalized Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, where I_N, A, and D are the identity, adjacency, and degree matrices of the graph. The cost of using graph convolutional layers depends on the number of nodes. If the node count happens to be higher than optimal, using Chebyshev polynomials is a great way to approximate the final matrix. The GCN model uses two graph convolutional layers as input, followed by mean pooling and fully connected ones.

3.1.8 DeepGraphCNN

The DeepGraphCNN [17] algorithm introduces an architecture to process variable-sized graphs. The architecture has to overcome the following two challenges:

- How can we extract meaningful and readable information from abstract objects such as graphs?

- In what order should nodes be processed to produce a meaningful result?

DeepGraphCNN addresses the second issue by creating a so-called SortPooling layer. Its sole purpose is to determine a procedure for ordering nodes based on their features in a graph. The first challenge has already been addressed in Sect. 3.1.7, as using graph convolutional layers results in the same outcome. Ordering the nodes correctly yields a fixed-sized graph structure with nodes capable of being used as inputs for training neural networks. Consequently, we can consider SortPooling layers as a bridge connecting graph convolutional layers with the fully connected ones. This method is similar to the Weisfeiler–Lehman (WL) subgraph kernel [18] model, determining whether two graphs share isomorphic properties.

3.2 A General CNN Model to Process Embedded Vectors

Since it can be very challenging to determine a dedicated architecture for each embedding to perform classification, we were motivated to find a neural network model that can be generally suggested for embeddings. Graphs are interpreted as spatial objects, so our primary approach was to build a neural network that uses convolutional layers and can perform relatively well on all of our databases. Moreover, using the same classification model made it possible to compare and evaluate the efficiency of the different embedding algorithms.

We have experimented with many architectural styles, but one always came out on top. We called it the Drop Model since it visually represents a tiny water droplet turned sideways in Fig. 1. The model combines simple convolutional layers connected to an output one by two dense layers. Instead of lowering the size of convolutional layers as the general approach, we used 1×1 convolutions. The use of 1×1 convolution on embedded vectors became feasible because of the nature of the data and the theoretical context behind graph embedding. It gave the functionality of cross-channel pooling [19] and worked similarly to network-in-network [20]. Thus, we could reduce the number of features and extract the hidden information in the data. Due to the nature of our datasets, our CNN is prone to taking a long time to be trained. Incorporating pooling layers doubled the training time, making the training process infeasible. Moreover, we have experienced further deterioration of the data after embedding reduced accuracy. Therefore, we decided to leave these layers out.

To improve accuracy further, we optimized the Drop Model for each graph embedding using architecture optimizers with a tree-structured parzen estimator. As we will show in the results section, the Drop Model is efficient for the various embeddings and datasets.

Using the Drop Model after each embedding lets us compare the embedding algorithms' efficiency. However, to evaluate also the Drop Model's performance itself, we have compared it with neural networks suggested specifically for embeddings incorporating graph convolution and Fourier-transformation-based approaches. For the remaining embeddings node2vec, DeepWalk, LINE, SDNE, and graph2vec, we used the Drop Model. Furthermore, we wanted to test the Patchy-San algorithm equipped with the Drop Model to compare it with its originally used neural architecture. Accordingly, when the Drop Model is applied, we will refer to it as the modified Pachy-San algorithm (m-Pachy-San).

3.3 Creating Ensembles of Embeddings

The concept of ensemble learning has been around for years for a reason. It is a practical way to improve model accuracy when certain conditions are adequately fulfilled. The most

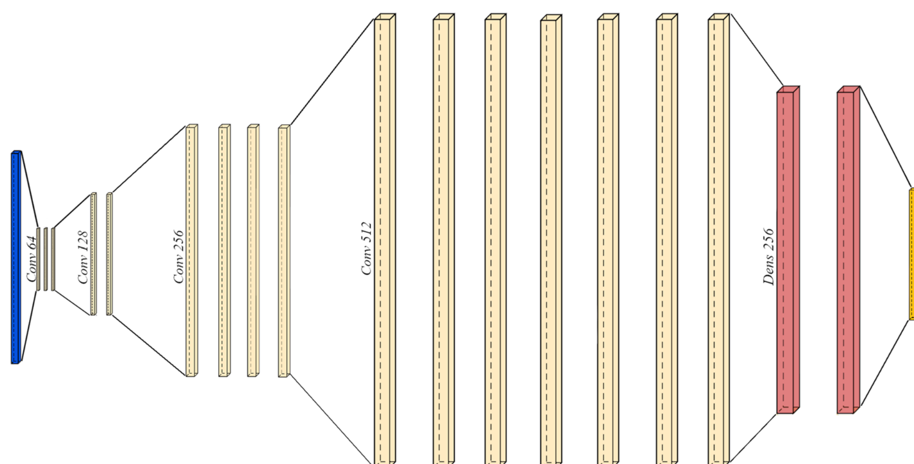


Fig. 1 The structure of our proposed Drop Model consisting of convolutional (yellow) and dense (red) layers (Colour figure online)

important aspect when creating an ensemble system is choosing diverse and sufficiently accurate members to complement each other and to improve overall performance.

Ensemble-based approaches are also used for graph embeddings. In [21], the authors proposed five different graph embedding methods (i.e., Laplacian Eigenmaps [22], HOPE [23], GraRep [24], DeepWalk [9], and GAE [25]) to accelerate the identification of lncRNA-miRNA interactions. They investigated their efficiencies both individually and as an ensemble-based system. The ensemble methods were applied to capture structural information of a heterogeneous network where some of the five embeddings were also optimized for similar problems. For example, the HOPE embedding method is optimized for directed graphs to preserve asymmetric transitivity. Two ensemble-based strategies were examined: prediction combination (similar to classical majority voting) and a deep attention neural network using a feature fusion-based architecture. In both cases, the ensemble approach yielded only slight improvement over the individual accuracies. Our present work, opposite to the architectural solutions of [21] optimized only for predicting lncRNA-miRNA interactions, introduces a general framework for different kinds of problems.

In [26], Knowledge Graphs (KGs) and Knowledge Graph Embedding Models (KGEMs) are investigated with a typical biomedical setup in the context of drug discovery for drug-disease prediction, where KGEMs are trained to predict triple links between drugs and diseases. The authors also used ensemble-based methods to improve accuracy using classical and weighted majority voting on graph-based datasets. They reported that the simple sum aggregation rule yielded better results than the weighted (averaged) one. In our current work, weighted majority voting has proved to be more efficient because we use a neural network to adjust the weights, which is able to find the optimal weighting scheme for a given ensemble. Furthermore, we notice that in [26] the graph embedding methods were selected based on the authors' former application in drug research, so again, this is not a general framework but a problem-specific one.

The work [27] investigated three graph embeddings and their multiclass ensemble-based system for link prediction, which aims to predict whether a new link exists between a pair of nodes or to discover hidden connections in the graph. The authors generated five classifiers and fed their outputs into a deep neural network for each pair of nodes. By aggregating the

results using this approach, they achieve more accurate results in link prediction than the individual classifiers. We further develop this approach here by proposing an interconnected neural network architecture to combine the above two steps. The interconnected system quickly and optimally determines each classifier's weights and parameters, eliminating the potential drawbacks of the two-step method of [27].

This work considers classification problems to discuss ensemble creation regarding such tasks. We provide two methods, namely, based on majority voting and interconnected networks. For the first approach Fig. 2a gives an example of an ensemble system with three members. The system gets a graph as an input, and the different embedding algorithms equipped with a classifier architecture classify it individually. Finally, majority voting occurs, where the final classification is based on individual decisions. Thus, the main components of an ensemble system, in our case, are:

- the members are graph embedding algorithms equipped with some classifiers,
- class labels as outputs of the members,
- a voting schema to determine the final decision result.

However, as we will see later, the embedding algorithms in the ensemble do not perform equally well. Therefore, dedicated weights should be assigned to each algorithm to improve classification performance. Thus, in this work, instead of simple majority voting, we use weighted majority voting (WMV) as an aggregation rule, which evaluates the individual performance of each classifier in the ensemble and adjusts their contribution to the class decision.

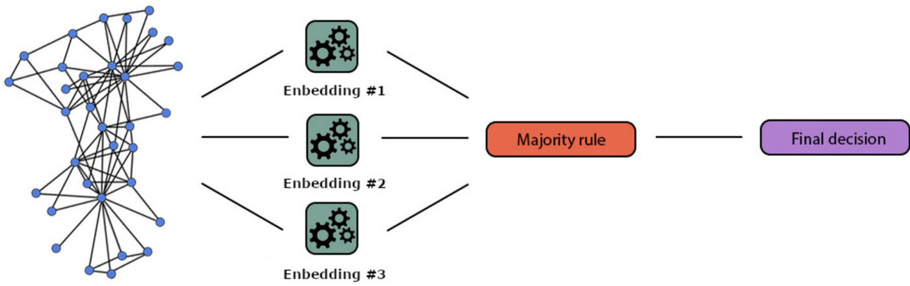
In addition to applying majority voting to aggregate the members' opinions, we have also examined the efficiency of an interconnected ensemble system shown in Fig. 2b. This involves organizing the individual algorithms into a supernetwork using a fusion-based approach, thus enabling the information flow among the members within the network. The combination of the members is achieved by linking them in a standard, fully interconnected layer. The advantage of this method is that this large architecture can be trained as a single neural network, allowing the embeddings that generate diverse networks to exchange information with each other. For implementation, we used a network similar to the Drop Model, where the number of filters in the CNN was doubled. It took approximately 60 epochs to train this network, i.e., half of the number of iterations compared to the original; all the other hyperparameters remained the same. Based on our experiments, we have combined only the best two embeddings for interconnected ensembles; adding others did not cause a significant rise in accuracy generally but did increase the training time.

In Sect. 4, we will see that ensemble systems can be used in various scenarios we study and can help improve the accuracy of decisions.

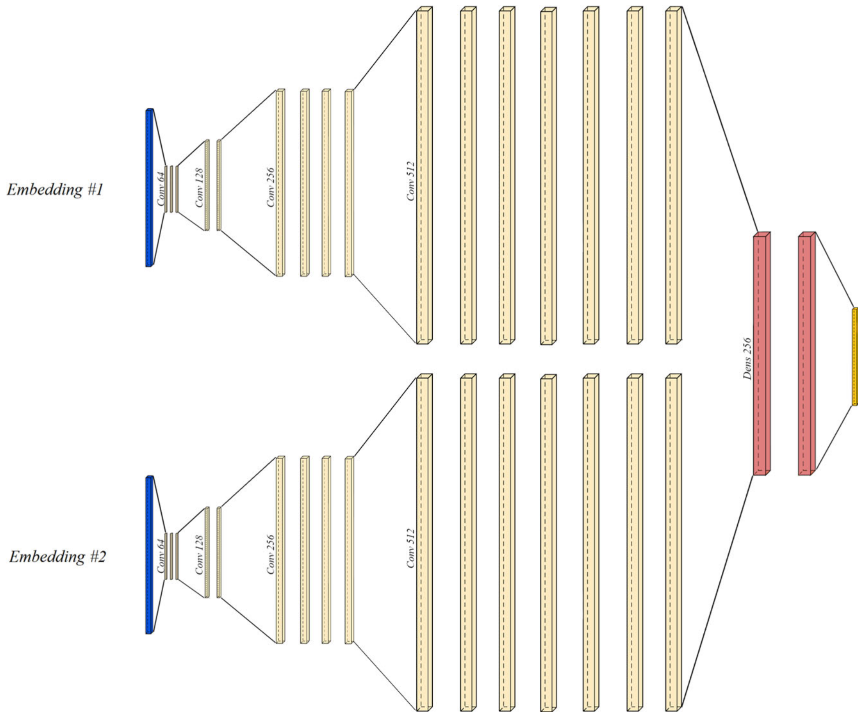
3.4 Estimating the Inhomogeneity of a Dataset to Select the Best Embedding

Training neural networks with graph embeddings can be time-consuming and resource-intensive. Each embedding works with different efficiency on different databases. We can save time by estimating which embedding might be the best choice. To address this issue, we propose a way to measure the inhomogeneity of a graph database using a statistical method. The resulting metric can be used to select the appropriate graph embedding.

Our approach is based on an algorithm that can measure Gaussian noise in digital images [28]; neural networks are particularly sensitive to this type of noise. The original method splits an image into patches, and a noise level σ^2 is determined using the eigenvalues of the



(a)



(b)

Fig. 2 Ensembles of graph embeddings for classification; **a** aggregation with majority voting, **b** creating an interconnected network

covariance matrix of the patches. The algorithm is non-parametric and runs in polynomial time.

As an interpretation for graph datasets, if the input is very inhomogeneous (has a high noise level), then the embedding process can amplify the effect of this "noise," and the network cannot be trained reliably. As a negative consequence, it will prevent the neural network from being trained successfully, which leads to worse results. More precisely, more considerable inhomogeneity is present if the graph has different types of vertices, resulting in

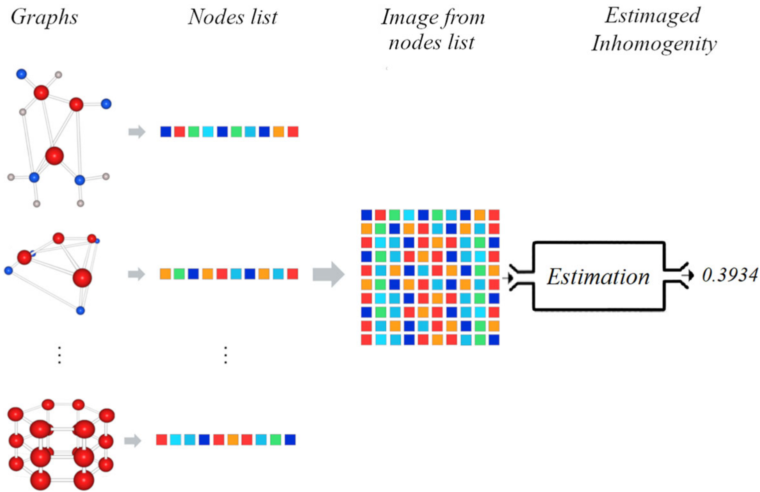


Fig. 3 Creating an image from a graph database to calculate an inhomogeneity measure

quite inhomogeneous embedded vectors. Our experiments showed that different embedding algorithms could handle high inhomogeneity with contrasting efficiency. The inhomogeneity of the vertices indicates that there may be different relationships between them, resulting in a complex pattern of graphs in the dataset that will be much more difficult to detect by a neural network.

To measure the input data's inhomogeneity, the graphs in the dataset are transformed into an image before embedding the dataset. To do so, each graph is lined up to form a row of the image (see Fig. 3) by applying zero-paddings to have the same length of rows. The image composed in this way is passed to the estimator algorithm [28] to determine the noise level. The resulting inhomogeneity descriptor helps select the best embedding for a neural network. In Sect. 4, we will see that this measure can be efficiently used to select an appropriate embedding algorithm for a given dataset.

It should also be noted that the noise level estimation described above does not result in any loss of information, as the conversion to images was for estimation purposes only. The actual processing and training used the original graph data as input.

4 Results and Evaluation

We show our experimental results for all the datasets and graph embeddings for the corresponding classification tasks. Moreover, we demonstrate how our novel ensemble creation techniques and inhomogeneity estimation can be exploited in selecting the most optimal embedding for the datasets. Throughout the section, we have applied the Drop Model except for m-Pachy-San after the graph embedding steps for classification as discussed in Sect. 3.2.

We have applied cross-validation and considered standard measures to evaluate the classification performance of the embeddings. Namely, stratified k-fold cross-validation was used with $k = 10$ during the training/testing processes to handle possible issues on unbalanced data. To follow the 60% train, 30% test, and 10% validation recommendation [29–31], we took out 30% of each database beforehand and used the remaining 70% for k-fold splits.

The standard measures for evaluation are built from the confusion matrix, namely from the following four key elements calculating the number of cases when the model:

- *TP*: correctly predicts the positive class,
- *TN*: correctly predicts the negative class,
- *FP*: incorrectly predicts the positive class,
- *FN*: incorrectly predicts the negative class.

From these values, we calculated several standard measures. For multi-class classification, we consider the micro/macro-averaging techniques, that is, when all the *TP*, *FP*, *TN*, and *FN* values are summed up for all the classes, and the measures are derived in the same way afterward:

- Accuracy $= \frac{TN + TP}{TN + TP + FP + FN}$;
- Precision $= \frac{TP}{TP + FP}$;
- Sensitivity/True Positive Rate (TPR) $= \frac{TP}{TP + FN}$;
- Specificity $= \frac{TN}{TN + FP}$;
- False Positive Rate (FPR) $= \frac{FP}{TN + FP}$;
- F1-score $= \frac{2TP}{2TP + FP + FN}$;
- Micro-Precision/-Sensitivity/-Specificity: The extension of precision/sensitivity/specificity for multi-class problems by adding the TN, TP, FP, and FN values for all the classes;
- Micro-F1: The harmonic mean of micro-precision and micro-sensitivity for multi-class classification;
- Macro-F1: The mean of F1-scores calculated for all the classes for multi-class classification;
- AUC score: Area under the Receiver Operating Characteristics (ROC) curve, which measures the performance in classification problems at various thresholds; it shows TPR against FPR.

4.1 Quantitative Results on Classification

All the graph embedding techniques listed in Sect. 3.1 were tested on all the datasets described in Sect. 2. We used an RTX 3080 video card for our neural networks and CUDA 10.2 in conjunction with cuDNN 8.1.0. The TensorFlow version we used for testing was 2.4. We show the classification results for each graph embedding technique on each dataset to evaluate our results.

4.1.1 MUTAG and Mutagenicity

For the dataset MUTAG, Table 1 clearly shows that DeepWalk, and node2vec outperformed the rest of the embedding algorithms. DeepWalk performed very well, probably because it intuitively uses random walks. Thus, relatively small graphs are mapped more efficiently. Node2vec also performed well, using a similar approach as DeepWalk with the possibility of revisiting known nodes. LINE and SDNE built for large social networks showed

Table 1 Classification results for graph embeddings on the MUTAG dataset

Embedding	Accuracy (train)	Accuracy (test)	Sensitivity	Specificity	F1-score	AUC score
DeepWalk	**96.11%**	**85.20%**	71.88%	35.71%	78.75%	0.878
node2vec	95.39%	82.48%	71.16%	42.42%	77.89%	**0.901**
graph2vec	68.63%	66.49%	**100.0%**	0.00%	**79.86%**	0.802
LINE	77.53%	62.19%	87.18%	32.39%	76.40%	0.832
SDNE	91.43%	60.11%	84.07%	40.00%	75.10%	0.826
Patchy-San	96.04%	82.45%	70.96%	**45.45%**	77.73%	0.825
m-Patchy-San	92.49%	73.91%	74.10%	44.89%	76.57%	0.853
GCN	70.48%	66.49%	**100.0%**	0.00%	**79.86%**	0.739
DeepGraphCNN	85.19%	84.51%	70.70%	45.16%	77.89%	0.869

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

Table 2 Classification results for graph embeddings on the Mutagenicity dataset

Embedding	Accuracy (train)	Accuracy (test)	Sensitivity	Specificity	F1-score	AUC score
DeepWalk	90.49%	55.08%	40.61%	49.69%	44.68%	0.761
node2vec	89.43%	54.55%	34.74%	**56.51%**	40.63%	0.741
graph2vec	91.43%	**72.95%**	**42.79%**	49.61%	**53.01%**	**0.831**
LINE	62.13%	59.01%	33.11%	48.51%	45.41%	0.749
SDNE	61.05%	58.36%	36.49%	52.15%	42.11%	0.765
Patchy-San	**93.06%**	67.46%	41.86%	50.38%	50.49%	0.820
m-Patchy-San	89.92%	66.68%	39.71%	53.26%	47.73%	0.804
GCN	58.95%	55.36%	39.41%	54.41%	40.10%	0.694
DeepGraphCNN	58.43%	56.42%	42.11%	52.87%	41.67%	0.713

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

medium performance, while graph2vec had the highest sensitivity with 0 specificity classifying all objects as positive. Patchy-San had a solid performance having a good balance between accuracy, sensitivity, and specificity.

In terms of algorithms using graph convolution, DeepGraphCNN consistently performed well across different metrics, showing a solid balance between sensitivity, specificity, and F1-score and having the highest AUC after DeepWalk and nod2vec. Having SortPooling layers act as a bridge between graph and regular convolutional layers might help to achieve a good result. On the other hand, GCN labeled each case positive, which is obviously not effective.

In conclusion, for domains providing datasets similar to MUTAG, sensitivity can be considered a descriptive metric in addition to accuracy for the choice of embedding. Sensitivity acts as a well-rounded performance measure since it quantifies the number of positive instances classified as positive by the classifier. The positive class represents the category we are interested in and want to investigate. When looking at compounds with non-carcinogenic and carcinogenic properties, the more crucial task is to recognize the latter. Our primary recommendation is Deepwalk since it has the best accuracy and high sensitivity.

The Mutagenicity dataset is similar to the MUTAG one since both contain mutagenic compounds. Mutagenicity encloses more and much larger graphs. However, their structures are

Table 3 Classification results for graph embeddings on the AIDS(v1) dataset

Embedding	Accuracy (train)	Accuracy (test)	Sensitivity	Specificity	F1-score	AUC score
DeepWalk	99.70%	98.82%	72.56%	30.77%	83.69%	0.984
node2vec	99.56%	98.37%	72.43%	**50.00%**	83.60%	0.952
graph2vec	94.88%	91.71%	76.22%	26.08%	83.34%	0.789
LINE	75.75%	70.90%	97.46%	10.22%	83.18%	0.713
SDNE	93.88%	74.14%	82.14%	43.21%	81.34%	0.693
Patchy-San	99.71%	98.91%	72.67%	16.66%	83.72%	0.988
m-Patchy-San	95.45%	79.63%	77.71%	**50.00%**	81.58%	0.733
GCN	71.05%	70.45%	**98.20%**	9.75%	83.20%	0.721
DeepGraphCNN	**99.88%**	**99.81%**	72.20%	22.35%	**83.76%**	**0.994**

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

less complicated. Thus, we achieved different outcomes as well. The corresponding results in Table 2 show that graph2vec, Patchy-San, and m-Patchy-San were the most effective embeddings here. m-Patchy-San, in this case, is a modified version of the original Patchy-San algorithm described in Sect. 3.1.6, which uses our Drop Model instead of its built-in neural architecture. Performing tests with both models can help determine our Drop Model's competitiveness. For this dataset, m-Patchy-San performed marginally worse since the Mutagenicity database has less complex compounds, but it produced much faster training times than the original model.

In the Mutagenicity dataset, half of the compounds carry mutagenic effects with large vocabulary sizes; thus, using negative sampling to update only a small amount of weights at a time shows its efficiency. Consequently, graph2vec was very effective since it encapsulates this technique. During embedding, graph2vec can effortlessly filter out nodes that are mutagenic compounds. This clear distinction can also be seen in its sensitivity and specificity scores. The database's simplicity and patternless nature caused LINE, SDNE, and random walk-based embeddings to perform sub-optimal.

In conclusion, considering the application domain of the Mutagenicity database, we propose relying on sensitivity when choosing an embedding to use on similar datasets. Processing nodes as signals when the graph structure is more practical meant that embeddings utilizing graph convolution performed poorly. Our recommended algorithm for similar datasets is graph2vec since it performed very well even when the positive and negative classes were balanced.

4.1.2 AIDS

For the AIDS database, we used two versions: one with 1110 graphs (v1) and one with 2000 graphs (v2). We decided to use both since we could also check whether the performance could be improved by increasing the dataset size. The results enclosed in Tables 3 and 4 show that the extension of the dataset led to higher sensitivity and specificity in general. Random walk-based algorithms performed well, probably because of the regular structures of the datasets. Namely, in the AIDS datasets, graphs can be split into equally sized sub-graphs, achieving consistent node density. Patchy-San could also find hidden patterns in the structure relatively faster while maintaining high accuracy. LINE and SDNE work well mainly on large datasets, so their performances were relatively low here.

Table 4 Classification results for graph embeddings on the AIDS(v2) dataset

Embedding	Accuracy (train)	Accuracy (test)	Sensitivity	Specificity	F1-score	AUC score
DeepWalk	**99.87%**	98.89%	80.33%	50.00%	88.81%	0.959
node2vec	99.42%	98.84%	80.52%	34.78%	88.83%	0.961
graph2vec	97.87%	94.75%	81.26%	**57.14%**	88.50%	0.886
LINE	86.11%	80.00%	96.67%	12.14%	87.54%	0.830
SDNE	95.91%	80.19%	89.15%	42.93%	87.72%	0.808
Patchy-San	99.76%	98.79%	80.61%	29.16%	88.83%	**0.996**
m-Patchy-San	97.38%	88.80%	84.45%	44.64%	88.22%	0.838
GCN	85.87%	80.00%	**98.34%**	10.04%	88.22%	0.784
DeepGraphCNN	99.84%	**99.84%**	80.12%	21.76%	**88.88%**	**0.996**

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

Table 5 Classification results for graph embeddings on the COX2 dataset

Algorithm	Accuracy (train)	Accuracy (test)	Sensitivity	Specificity	F1-score	AUC score
DeepWalk	**95.43%**	68.51%	5.63%	57.14%	8.98%	0.793
node2vec	95.09%	68.77%	4.98%	58.90%	7.94%	0.791
graph2vec	85.20%	71.91%	**11.30%**	48.85%	**17.22%**	0.759
LINE	79.77%	76.88%	5.47%	**94.44%**	9.68%	**0.814**
SDNE	94.81%	67.46%	6.35%	53.95%	9.87%	0.783
Patchy-San	94.36%	73.21%	8.77%	57.60%	14.11%	0.755
m-Patchy-San	76.66%	66.60%	6.75%	51.92%	10.31%	0.742
GCN	73.84%	74.54%	2.58%	78.15%	4.68%	0.703
DeepGraphCNN	80.15%	**78.15%**	5.45%	86.48%	9.31%	0.804

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

For both AIDS datasets, the best-performing algorithm was DeepGraphCNN. It remained stable throughout all the tests with the highest accuracy and F1-score. By amplifying node features and extending them to other vertices, it emphasized attributes that correlate with the effectiveness of the compounds on the virus.

4.1.3 COX2

The COX2 database contains three times more nodes on average than the others. The results in Table 5 show that algorithms performing well on smaller datasets now serve worse. It is visible that DeepGraphCNN, GCN, and LINE outperformed the rest. The GCN algorithm could filter out essential information as the database has a more straightforward structure but more nodes. Consequently, GCN had more nodes to extract recognizable patterns.

Utilizing more information throughout the learning process, LINE and SDNE had the opportunity to show their capabilities. Social networks are formed instinctively, while large information networks are built manually. The artificiality of databases creates formality and order, while social networks are more disordered. SDNE was designed to handle artificial data. For this reason, its performance here is worse than that of LINE. On the other hand, LINE aims to find hidden relations in the data. Compounds responsible for the oxidative

Table 6 Classification results for graph embeddings on the DBLP_v1 dataset

Embedding	Accuracy (train)	Accuracy (test)	Sensitivity	Specificity	F1-score	AUC score
DeepWalk	76.40%	61.80%	62.42%	32.58%	61.16%	0.803
node2vec	71.28%	66.56%	56.42%	40.37%	60.45%	0.772
graph2vec	83.12%	77.60%	52.84%	44.39%	62.88%	0.785
LINE	**87.29%**	60.43%	57.75%	45.60%	56.52%	0.698
SDNE	84.59%	57.97%	54.69%	45.96%	55.88%	0.704
Patchy-San	75.75%	76.04%	50.74%	**52.72%**	60.95%	0.835
m-Patchy-San	81.11%	**78.26%**	56.41%	37.22%	**63.29%**	**0.844**
GCN	56.12%	47.87%	**92.15%**	35.90%	53.67%	0.574
DeepGraphCNN	56.98%	48.76%	87.64%	42.46%	54.32%	0.596

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

effect are considered artificial; however, they were produced to eliminate the reactive agent of a naturally found enzyme; hence, we can consider the database as instinctive.

Both convolution-based embeddings performed relatively well. Though DeepGraphCNN can be expected to dominate for such data due to its excellent ability to emphasize crucial parts, GCN also performed fairly well. We believe that classifying compounds that do not carry impending abilities is much easier; therefore, we recommend using the LINE algorithm.

4.1.4 DBLP_v1

The DBLP_v1 database is the largest one we have included; therefore, it is a good baseline for finding embeddings that perform well on a large dataset. Table 6 shows that Patchy-San, mainly when it used the modified architecture by applying our Drop Model, had an excellent performance. The tree-like structure of the DBLP_v1 database helps the Patchy-San embedding to effectively use its coloring functionality to map all nodes of a graph.

Using random walk-based algorithms on DBLP_v1 containing graphs of 10 nodes on average proved ineffective since increasing the walk length beyond this number raises inconsistencies. Using negative sampling on such a database is also negligible; therefore, applying graph2vec to DBLP_v1 was ineffective. SDNE and LINE also performed relatively poorly since the graphs in the database were too small. The structures of the graphs in the DBLP_v1 database are artificial, which means lots of noise and redundancy are generated during the training process, which is a drawback for the graph convolutional algorithms. Consequently, GCN and DeepGraphCNN reached lower scores.

4.1.5 Fingerprint

The Fingerprint dataset consists of 14 classes. Consequently, for performance measurement, we consider here Micro-F1, Macro-F1, Micro-Sensitivity, and Micro-Specificity introduced in Sect. 3. From Tables 7 and 8, we can see that the m-Patchy-San algorithm outperformed all the other embeddings by a large margin. The dominance of the modified architecture is probably caused by the same feature mentioned for the DBLP_v1 database. GCN and DeepGraphCNN had low performances for reasons similar to those of DBLP_v1. Furthermore, since the Fingerprint dataset also has an artificial internal graph structure, these graphs are too small for the SDNE and LINE embeddings.

Table 7 Classification results for graph embeddings on the Fingerprint dataset

Embedding	Accuracy (train)	Accuracy (test)	Micro-sensitivity	Micro-specificity	Macro-F1	Micro-F1
DeepWalk	88.57%	53.09%	67.42%	67.42%	56.11%	67.42%
node2vec	74.19%	53.25%	80.42%	92.07%	85.85%	74.94%
graph2vec	48.34%	49.00%	49.00%	91.21%	63.75%	52.65%
LINE	49.56%	28.14%	32.01%	79.26%	38.24%	26.45%
SDNE	44.49%	24.33%	27.21%	89.04%	35.63%	24.14%
Patchy-San	56.92%	53.73%	67.64%	67.64%	57.38%	67.64%
m-Patchy-San	**94.92%**	**82.46%**	**82.46%**	**98.91%**	**89.94%**	**80.11%**
GCN	23.28%	25.09%	23.27%	86.37%	36.67%	26.57%
DeepGraphCNN	20.46%	19.65%	19.65%	85.65%	31.96%	22.14%

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

4.1.6 Tox21_ARE

The Tox21_ARE database is very similar to COX2 because it contains compounds that block the reactive agents in enzymes. This database has the particular behavior that it mainly includes graphs having either a small or a large number of nodes. Thus, considering the average node count purely can be misleading here. In Table 9, we can observe the increased amount of large graphs makes LINE and SDNE perform well. Moreover, the two Patchy-San embeddings have quite the same performance because of the variability of the data. The original Patchy-San algorithm finds a pattern in larger graphs better, while its modified variant has opposite characteristics. For this reason, we can assume that the numbers of large and small graph structures in the database are pretty close. Embeddings based on graph convolution, especially DeepGraphCNN, had the best performance supported by the natural internal structure of the database.

4.2 Efficiency of the Drop Model

The Drop model was used to test the node2vec, DeepWalk, LINE, SDNE, and graph2vec embeddings. Furthermore, we wanted to test the Patchy-San algorithm with the Drop Model (m-Pachy-San) to compare it with the original neural network architecture.

The Drop Model performed exceptionally well on the Mutag and Mutagenicity datasets using the DeepWalk and grap2vec embeddings (see Tables 1, 2). On the Mutagenicity database, the Drop model outperformed the graph convolution-based (GCN) architecture by nearly 15%; it surpassed the original Patchy-San architecture by over 5%. The Drop Model gave excellent results on the DBLP_v1 and Fingerprint databases as m-Patchy-San performed the most outstandingly. For example, Fingerprint (see Table 7), had almost 30% higher accuracy than the original Patchy-San and more than 50% than GCN. In the case of the AIDS, COX2, and Tox21_ARE databases, the graph convolution architecture and the original Patchy-San architecture performed better. However, notice that the Drop model is only a few percent behind in these cases (see Tables 3, 5, 9). Overall, the Drop model showed impressive performance as a universally applicable model after graph embeddings.

Table 8 AUC scores on the $C_1, \ldots, C_{14}$ classes of the Fingerprint dataset

Embedding	Train	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}	C_{14}
DeepWalk	0.712	0.791	0.843	0.815	0.902	0.711	0.635	0.422	0.801	0.729	0.722	0.791	0.467	0.626	0.698
node2vec	0.691	0.832	0.795	0.806	0.931	0.699	0.678	0.691	0.801	0.698	0.758	0.821	0.712	0.691	0.738
graph2vec	0.696	0.719	0.702	0.713	0.684	0.697	0.703	0.673	0.601	0.517	0.554	0.532	0.545	0.658	0.521
LINE	0.598	0.568	0.599	0.549	0.579	0.536	0.632	0.614	0.641	0.593	0.576	0.533	0.524	0.522	0.57
SDNE	0.643	0.612	0.59	0.514	0.597	0.534	0.544	0.555	0.548	0.588	0.562	0.554	0.584	0.534	0.596
Patchy-San	0.674	0.667	0.888	0.5	0.815	0.865	0.59	0.631	0.698	0.62	0.647	0.683	0.602	0.597	0.624
m-Patch-San	0.841	0.884	0.843	0.815	0.902	0.856	0.832	0.813	0.849	0.829	0.822	0.891	0.867	0.826	0.813
GCN	0.548	0.569	0.537	0.542	0.601	0.631	0.503	0.567	0.541	0.549	0.599	0.573	0.545	0.587	0.594
DeepGraphCNN	0.612	0.645	0.634	0.731	0.697	0.699	0.701	0.647	0.692	0.664	0.712	0.696	0.545	0.597	0.577

Table 9 Classification results for graph embeddings on the Tox21_ARE

Embedding	Accuracy (train)	Accuracy (test)	Sensitivity	Specificity	F1-score	AUC score
DeepWalk	93.00%	76.55%	3.78%	53.99%	6.41%	0.808
node2vec	92.58%	78.91%	2.56%	64.16%	4.56%	0.795
graph2vec	**94.24%**	81.39%	6.62%	54.59%	11.31%	0.798
LINE	88.21%	84.44%	5.41%	87.29%	9.87%	0.867
SDNE	87.45%	83.21%	6.20%	**89.91%**	8.79%	0.849
Patchy-San	91.50%	83.18%	3.44%	75.46%	6.34%	0.842
m-Patchy-San	85.63%	81.18%	2.47%	74.56%	4.56%	0.801
GCN	86.89%	84.21%	7.94%	87.51%	10.35%	0.804
DeepGraphCNN	89.44%	**85.44%**	**8.49%**	89.24%	**12.46%**	**0.891**

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

4.3 Performances of Ensembles of Embeddings

Based on our discussion on the individual performances of the embedding algorithms for each dataset, it seems a natural step ahead to combine the best-performing ones into ensembles using the methodology described in Sect. 3.3. We can expect a remarkable improvement if the best-performing embeddings are based on different principles since diversity is a precious feature in ensemble-based approaches. However, there are cases where the algorithms are not sufficiently diverse, or there is one algorithm with outstanding performance that an ensemble system cannot improve by adding other members. Considering this, we decided not to apply ensemble learning to three databases: AIDS(v1), AIDS(v2), and Fingerprint. Namely, we experienced the following for them.

AIDS: The individual embedding accuracies for the AIDS database are already close to 100%, so using an ensemble system results in negligible performance gains. Namely, the accuracy of DeepGraphCNN is 99.84%, which could not be improved significantly with an ensemble.

Fingerprint: Regarding the Fingerprint database, we can see that the m-Patchy-San algorithm has an excellent result (above 80%) on the test set, while the other algorithms perform around 50%. Adding these algorithms performing worse with that margin is useless, which was also validated by our experiments.

However, ensemble creation was justified for the remaining databases and improved classification accuracy. For each of these databases, algorithms with sufficiently high accuracy and diversity were selected, and their combined system was used to make the final classification decision. As for our first approach, we used weighted majority voting (WMV), which gave better results on the datasets under study than simple majority voting. The search for optimal weights was integrated into the learning process so the best combination of weights was found by a neural network. In our tests, we performed 10 runs on the remaining databases. The resulting performance metrics are the average values calculated from each run. Now, we go through all the datasets and present the corresponding results also in Table 10.

MUTAG: For the MUTAG database we found the DeepWalk, node2vec, and DeepGraphCNN embeddings to perform the best. The accuracy of their ensemble became 89.61% outperforming that of its best member (DeepWalk, 85.20%). The F1-score has also increased to 81.77%. These results suggest an ensemble-based system might be worth considering for similar datasets.

Table 10 Comparing the best-performing ensembles with the best individual embeddings and DeepChem

Database	Embedding	Accuracy (train)	Accuracy (test)	Train F1-score	Test F1-score
MUTAG	DeepChem	97.62%	87.90%	94.49%	77.95%
	DeepWalk	96.11%	85.20%	95.14%	78.75%
	Ensemble (WMV)	**98.81%**	89.61%	**96.94%**	81.77%
	Ensemble (Interconnected)	94.12%	**92.23%**	94.74%	**88.16%**
Mutagenicity	DeepChem	**97.40%**	61.79%	**87.38%**	51.68%
	graph2vec	91.43%	72.95%	60.23%	53.01%
	Patchy-San	93.06%	67.46%	62.41%	50.49%
	m-Patchy-San	89.92%	66.68%	59.76%	47.73%
	Ensemble (WMV)	96.99%	75.08%	67.49%	64.17%
	Ensemble (Interconnected)	85.15%	**77.03%**	74.08%	**79.11%**
COX2	DeepChem	**94.87%**	72.25%	51.14%	11.39%
	LINE	80.15%	78.15%	55.12%	9.68%
	Ensemble (WMV)	85.39%	82.25%	**66.81%**	**20.25%**
	Ensemble (Interconnected)	87.99%	**86.18%**	65.34%	19.18%
DBLP_v1	Patchy-San	75.75%	76.04%	77.11%	60.95%
	m-Patchy-San	81.11%	78.26%	66.94%	63.29%
	Ensemble (WMV)	83.43%	83.37%	77.49%	74.17%
	Ensemble (Interconnected)	**88.01%**	**85.30%**	**80.23%**	**76.61%**
Tox21_ARE	DeepChem	**98.17%**	70.09%	94.22%	15.81%
	DeepGraphCNN	89.44%	85.44%	92.41%	12.46%
	Ensemble (WMV)	92.23%	86.78%	**95.17%**	15.68%
	Ensemble (Interconnected)	89.16%	**88.47%**	94.32%	**16.02%**

The bold parts in the tables represent the best accuracy/result obtained for the given metric (column)

Mutagenicity: For this dataset, we also considered the three best-performing algorithms, and the combined system outperformed the better-performing algorithm graph2vec by 2.13%, and the F1-score improved by more than 10% on the test dataset.

COX2: For the COX2 database, we got similar results as for MUTAG. By combining the three top-performing embeddings (DeepGraphCNN, GCN, LINE), we have managed a 4.10% increase in accuracy and 8.86% in F1-score.

DBLP_v1: Selecting the best-performing embeddings for the ensemble involved Patchy-San and m-Patchy-San, which were based on similar principles. Thus, to improve the ensemble, both the node2vec and DeepWalk algorithms were added to it. In this way, the ensemble outperformed the best individual algorithm by 5.11% in accuracy and more than 10% in F1-score.

Tox21_ARE: For the Tox21_ARE dataset, we followed the same approach as for COX2, as their structure is similar. We selected the most efficient embeddings (DeepGraphCNN, GCN, LINE), and the overall accuracy increased by 1.34%.

As for our second ensemble creation approach, the embedding members of the interconnected ensembles for each of the investigated datasets have been selected as DeepWalk + DeepGraphCNN (MUTAG), grap2vec + Pachy-San (Mutagenicity), LINE + DeepGraphCNN (COX2), grap2vec + m-Pachy-San (DBLP_v1), LINE + DeepGraphCNN (Tox21_ARE). Compared to the majority voting-based ensembles, the results of the inter-

Table 11 The inhomogeneity estimations of the databases and the corresponding proposed embeddings

Database	Inhomogeneity	Average node number	Proposed embedding
MUTAG	0.5774	18	DeepGraphCNN (DeepWalk)
Mutagenicity	0.3934	30	graph2vec
AIDS	0.6085	16	DeepGraphCNN
AIDS2	0.644	16	DeepGraphCNN
COX2	1.6436	41	DeepGraphCNN
Tox21_ARE	0.4864	18	DeepGraphCNN
DBLP_v1	1.4797	10	m-Patchy-San
Fingerprint	1.391	6	m-Patchy-San

connected networks showed a further reasonable rise in accuracy and F1-score, as can be observed in Table 10. Moreover, the training times also dropped since only two embeddings were connected in these ensembles.

4.4 Selecting an Embedding Based on Inhomogeneity

In Table 11, we present how the inhomogeneity level estimation introduced in Sect. 3.4 can be used for selecting an adequate embedding for a specific dataset. We can see that DeepGraphCNN can be a universal approach for embedding, as it handles inhomogeneity in embedded vectors quite well. We also considered the average number of nodes in the database graphs when applying graph embedding according to the inhomogeneity metric. Large graphs can result in complex patterns that are difficult to embed.

Below the inhomogeneity level of 0.4, graph2vec proved the most efficient, as discussed in Sect. 4.1.1 for the Mutagenicity database. This dataset has a lower level of inhomogeneity, although the average number of nodes is 30. It suggests that it has a few different types of nodes composing global patterns that can be embedded successfully.

For an inhomogeneity above 0.4, depending on the number of nodes, we recommend DeepGraphCNN or m-Patchy-San. If there are more than 15 nodes per graph on average in the database, DeepGraphCNN is proven to be more efficient. If we have less than 15 nodes per graph, we recommend m-Patchy-San, which may be the best choice due to its convolutional architecture when there is no natural internal pattern in the graph, as in the case of DBLP_v1. According to the MUTAG dataset results, the DeepWalk algorithm performed slightly better, and DeepGraphCNN is also a good choice in such cases because of its stability.

4.5 Comparison with the DeepChem Framework

The DeepChem [3] project aims to build easy-to-use, high-quality tools for artificial intelligence engineers. The origin of DeepChem mainly focused on problems dedicated to chemistry and molecular biology, but the project has evolved further due to broad public interest. Nowadays, it can be used to predict the solubility of small drug-like molecules or analyze protein structures and extract meaningful descriptors. Since DeepChem also contains graph databases from the ones listed in Sect. 2, it is an ideal choice for benchmarking in this study. Namely, we could compare according to the MUTAG, Mutagenicity, COX2, and TOX21_ARE databases. We chose the original DeepChem implementation, considering a fully connected neural net-

work, for the classification task to provide a fair comparison. The whole test environment was the same as described in this work. We used the perhaps two most impactful metrics for evaluation: accuracy and F1-score. Our aim with this comparison was to compare the performances of our Drop Model and ensemble-based approaches with a state-of-the-art popular framework.

Table 10 suggests that DeepChem is primarily optimized for multi-class classification problems and smaller databases since it provides worse results for binary classification of large datasets. Namely, only the random walk-based embedding fell below 80% on the Tox21_ARE and 82% on the COX2 databases, while DeepChem achieved barely over 70% and 72%, respectively. The most substantial setback we found for DeepChem was its insufficient ability for binary classification problems, which generally resulted in more significant variance in the test results.

On the other hand, Table 10 also shows that DeepChem is efficient when used on smaller databases such as MUTAG. Regarding accuracy and runtime, it performed slightly above the best-performing graph embedding, DeepWalk, but worse than the ensemble-based approaches. However, even DeepWalk can be a good choice since it has a better F1-score. The Mutagenicity dataset is also more extensive, and the corresponding results further support the hypothesis that DeepChem performs better at small-scale classification problems.

5 Discussion and Conclusions

Diversity in databases and data types is far from negligible. Sub-optimal results can frequently occur when choosing the same embedding for all databases. We conclude that there is no best algorithm; it always depends on the context and problem we are trying to solve. However, the following descriptors might help to select an appropriate embedding: the number of graph objects in the dataset, the size of each graph, the number of nodes and classes, and finally, the number of attributes each node bears. Making the right decision can reward us with a 20–30% performance gain over a randomly chosen one. To this end, we proposed a procedure to help select appropriate embeddings based on the characteristics of the graphs in the database.

By examining the results, we found that using either the LINE or SDNE algorithm for larger graphs is a good practice. These embeddings were explicitly built for social and information networks. In addition, the results suggest that graph convolutional methods should be used on data with well-defined structures following a natural pattern. These signal-based algorithms are more efficient at preserving systematically constructed structures presented e.g., in MUTAG, COX2, and AIDS. We recommend using DeepGraphCNN over GCN since implementing SortPooling layers to connect convolutional and graph convolutional layers performed better in all situations. However, we do not advise using them on databases where graph objects are closer to being chaotic than systematic, like in Fingerprint and DBLP_v1.

In databases where finding a recognizable pattern is an issue, a good practice can be using a Patchy-San algorithm variant. It is based on convolution and can recognize less common structures. Stability was also vital to this embedding since it provided the most consistent specificity and sensitivity throughout our tests. Based on the results, our modified Patchy-San algorithm can be used well on more complex data with fewer nodes on average than usual (e.g., DBLP_v1 and Fingerprint).

Finally, when the elements of a database fall into the middle-sized category, it might be best to use Random Walk-based embeddings. Reasonable-sized walks contain plenty of

crucial information; therefore, knowing when to use them is a great asset. The MUTAG and AIDS databases are perfect fits since they contain easily recognizable patterns.

In addition to examining the individual embedding algorithms, the efficiency of their combined systems was also investigated. As a simple approach, we used the weighted majority voting method to make the final decision. Furthermore, we developed an interconnected network joining two embeddings, resulting in a system that can overcome the shortcomings of the majority voting-based ensemble systems. In conclusion, the results show that ensemble-based approaches are really efficient for graph embeddings. By using them, we can create a robust system that can significantly improve the efficiency of the underlying algorithms by compensating for their specific weaknesses.

In our experiments, the DeepGraphCNN and GCN model did not yield significantly better results when combined with our proposed architecture. While we recognize that the advantages of GNNs and GCNs include isomorphism detection and permutation invariance of nodes [32, 33], our aim here was to develop a more general approach that can be applied to a wider range of embedding methods. Future research can involve a detailed investigation of these algorithms to refine and optimize our approach further.

Another promising future direction may be to consider some stacking methods to exploit ensemble models more. We have already tested this approach on the Mutag and AIDS databases. However, we did not observe significant improvements in our experiments for the baseline architecture. Since we should generally expect an improvement using stacking, it suggests that a remarkable architectural and training optimization should take place to utilize this technique.

Acknowledgements This work is partly funded by the projects OTKA K143540, and TKP2021-NKTA-34, implemented with the support provided by the National Research, Development, and Innovation Fund of Hungary under the TKP2021-NKTA funding scheme.

Author Contributions Attila Tiba: Conceptualization, Methodology, Software, Validation, Resources, Writing—Original Draft, Writing-Reviewing and Editing, Supervision. Tamas Giraszi: Conceptualization, Methodology, Visualization, Investigation, Software, Validation. Andras Hajdu: Conceptualization, Methodology, Data curation, Resources, Writing—Original Draft, Writing- Reviewing and Editing, Resources

Funding Open access funding provided by University of Debrecen. The research was supported by the following funds: This work was supported by the project TKP2021- NKTA-34, implemented with the support provided by the National Research, Development, and Innovation Fund of Hungary under the TKP2021-NKTA funding scheme.

Availability of Data and Materials Not applicable.

Declarations

Ethical Approval Not applicable.

Conflict of interest Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Xu M (2021) Understanding graph embedding methods and their applications. SIAM Rev 63:825–853. https://doi.org/10.1137/20M1386062
2. Manessi F, Rozza A, Manzo M (2020) Dynamic graph convolutional networks. Pattern Recogn 97:107000
3. Ramsundar B et al (2019) Deep learning for the life sciences. O'Reilly Media, Sebastopol
4. TUDataset: a collection of benchmark datasets for learning with graphs. https://chrsmrrs.github.io/datasets/docs/datasets/. Accessed 03 Jan 2022
5. Morris C et al (2020) TUDataset: a collection of benchmark datasets for learning with graphs. www.graphlearning.io
6. Kazius J, McGuire R, Bursi R (2005) Derivation and validation of toxicophores for mutagenicity prediction. J Med Chem 48:312–320. https://doi.org/10.1021/jm040835a
7. Mikolov T, Chen K, Corrado G, Dean J, Bengio Y, LeCun Y (eds) (2013) Efficient estimation of word representations in vector space. In: Bengio Y, LeCun Y (eds) 1st international conference on learning representations, ICLR 2013, Scottsdale, Arizona, USA, May 2–4, 2013, workshop track proceedings
8. Jatnika D, Bijaksana MA, Suryani AA (2019) Word2vec model analysis for semantic similarities in English words. Procedia Computer Science 157:160–167. **The 4th International Conference on Computer Science and Computational Intelligence (ICCSCI 2019): Enabling Collaboration to Escalate Impact of Research Results for Society**
9. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. https://doi.org/10.1145/2623330.2623732
10. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. https://arxiv.org/abs/1607.00653
11. Narayanan A et al (2017) graph2vec: learning distributed representations of graphs. *CoRR* arXiv:1707.05005
12. Le Q, Mikolov T, Xing EP, Jebara T (eds) (2014) Distributed representations of sentences and documents. In: Xing EP, Jebara T (eds) Proceedings of the 31st international conference on machine learning, vol. 32 of proceedings of machine learning research. PMLR, Bejing, pp 1188–1196. https://proceedings.mlr.press/v32/le14.html
13. Tang J et al (2015) Line: large-scale information network embedding. https://doi.org/10.1145/2736277.2741093
14. Wang D, Cui P, Zhu W (2016) Structural deep network embedding. https://doi.org/10.1145/2939672.2939753
15. Niepert M, Ahmed M, Kutzkov K (2016) Learning convolutional neural networks for graphs. In: Balcan, MF and Weinberger, KQ (ed)Proceedings of The 33rd International Conference on Machine Learning. PMLR, New York, USA. https://proceedings.mlr.press/v48/niepert16.html
16. Kipf TN, Welling M (2017) Semi-supervised classification with graph graphembeddingdeeplearningutional networks. https://openreview.net/forum?id=SJU4ayYgl
17. Zhang M, Cui Z, Neumann M, Chen Y (2018) An end-to-end deep learning architecture for graph 735 classification. Proc AAAI Conf Artif Intell 32. https://ojs.aaai.org/index.php/AAAI/article/view/11782
18. Togninalli M, Ghisu E, Llinares-López F, Rieck B, Borgwardt K (2019) Wasserstein Weisfeiler–Lehman graph kernels. Curran Associates Inc., Red Hook
19. Liu L, Shen C, Hengel AVD (2017) Cross-graphembeddingdeeplearningutional-layer pooling for image recognition. IEEE Trans Pattern Anal Mach Intell 39:2305–2313
20. Lin M, Chen Q, Yan S (2013) Network in network. Computing Research Repository (CoRR) arXiv:1312.4400
21. Zhou S et al (2019) LncRNA-miRNA interaction prediction from the heterogeneous network through graph embedding ensemble learning. In: Yoo I, Bi J, Hu X (eds) 2019 IEEE international conference on bioinformatics and biomedicine, BIBM 2019, San Diego, CA, USA, November 18–21, 2019. IEEE, pp 622–627. https://doi.org/10.1109/BIBM47256.2019.8983044
22. Belkin M, Niyogi P, Dietterich T, Becker S, Ghahramani Z (2001) Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Dietterich T, Becker S, Ghahramani Z (eds) Advances in neural information processing systems, vol 14. MIT Press, Cambridge
23. Ou M, Cui P, Pei J, Zhang Z, Zhu W (2016) Asymmetric transitivity preserving graph embedding. https://doi.org/10.1145/2939672.2939751
24. Cao S, Lu W, Xu Q (2015) Grarep: learning graph representations with global structural information. https://doi.org/10.1145/2806416.2806512
25. Kipf TN, Welling M (2016) Variational graph auto-encoders. Online; Accessed 19 July 2022. arXiv:1611.07308

26. Rivas-Barragan D, Domingo-Fernández D, Gadiya Y, Healey D (2022) Ensembles of knowledge graph embedding models improve predictions for drug discovery. Brief. Bioinform 23:Bbac481. https://doi.org/10.1093/bib/bbac481
27. Chen Y-L, Hsiao C-H, Wu C-C (2022) An ensemble model for link prediction based on graph embedding. Decis. Support Syst. 157:113753
28. Chen G, Zhu F, Heng PA (2015) An efficient statistical method for image noise level estimation. 2015 IEEE Int Conf Comput Vis (ICCV) 477–485. https://doi.org/10.1109/ICCV.2015.62
29. Chollet F (2017) Deep learning with python. Manning Publications, New York
30. Raschka S, Mirjalili V (2017) Python machine learning: machine learning and deep learning with python, scikit-learn, and TensorFlow, vol 2, 2nd edn. Packt Publishing, Birmingham
31. Russel SJ, Norvig P, Davis E (2010) Artificial intelligence: a modern approach, 3rd edn. Prentice Hall, Upper Saddle River, pp 525–806
32. Scarselli F, Gori M, Tsoi A, Hagenbuchner M, Monfardini G (2009) Computational capabilities of graph neural networks. IEEE Trans Neural Netw 20:81–102
33. Xu K, Hu W, Leskovec J, Jegelka S (2019) How powerful are graph neural networks? arXiv:1810.00826

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.