node2vec Based Graph Convolutional Networks

Social Network Analysis for Computer Scientists — Course paper

Chenyu Shi
s3500063@umail.leidenuniv.nl
LIACS, Leiden University
Leiden, Netherlands

Shupei Li
s3430863@umail.leidenuniv.nl
LIACS, Leiden University
Leiden, Netherlands

ABSTRACT

In this project, we propose a novel method to combine two graph representation learning algorithms, node2vec and GCN, to improve the quality of embeddings and accelerate the training process. We verify the effectiveness of our proposed method on node classification task and link prediction task with five real-world datasets. Experimental results demonstrate the overall performance of our proposed model is better than that of the node2vec alone or the GCN alone. Besides, the visualization of training loss in node classification supports the proposed method can speed up the convergence.

KEYWORDS

node2vec, GCN, graph embeddings, social network analysis, network science

ACM Reference Format:
Chenyu Shi and Shupei Li. 2022. node2vec Based Graph Convolutional Networks: Social Network Analysis for Computer Scientists — Course paper. In *Proceedings of Social Network Analysis for Computer Scientists Course 2022 (SNACS '22).* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

Graphs are mathematical objects that can model complex relationships on non-Eucildean space. They are widely used in multiple domains such as molecular structure modelling, social network analysis, recommender systems, etc. To leverage the information contained in graphs, it is essential to develop efficient techniques for representing graph-structured data numerically.

Traditional statistical and machine learning methods are designed for extracting features from structured data on Eucildean space. For example, principal component analysis (PCA), uniform manifold approximation and projection (UMAP), and t-distributed stochastic neighbor embedding (T-SNE) are common techniques to reduce dimensions and capture features of data. Although these methods have achieved satisfactory performance on structured-data, they are hard to be generalized to graph-structured data, because they highly depend on properties of Eucildean space.

This challenge led to the development of techniques specifically for graph-based representation. There are two main types of these techniques: shallow embedding method and deep embedding method [9]. Shallow embedding methods use shallow encoder functions to map the original graph structure onto an Euclidean space and obtain the embedding matrix. If data is labelled, we can apply supervised learning algorithms, e.g. label propagation, to extract embeddings used in a subsequent supervised task. However, labels are not available or only partly available in most cases, where we need unsupervised learning or semi-supervised learning to distill information about graph structure. These methods can be divided into distance-based method and outer product method further [9]. Generally, distance-based methods select a metric function that indicates distances between any pairs of nodes and optimize the function to generate embeddings. Representative distance-based methods include multi-dimensional scaling and laplacian eigenmaps. Outer product-based methods use matrix operations to evaluate the similarity between nodes. Most of early studies in graph embedding field adopt matrix factorization to reduce dimensionality of data while preserve the structure information [2]. Another mainstream outer product-based method is inspired by the development in natural language processing. Existing research generalizes the skip-gram word embedding architecture to capture the graph embeddings, which has been proved to be efficient on many graph-related tasks [10][11]. node2vec [4], one of the algorithms addressed in this paper, is also a variation of skip-gram-based method.

node2vec is a semi-supervised algorithm whose goal is to learn features from networks [4]. It transforms the graph embedding learning problem into a maximum likelihood optimization problem in a similar way as skip-gram architecture of word embedding learning. Likelihood calculation requires a clear definition of the neighborhood. Textual data has the intrinsic semantic order that can be naturally employed as word neighborhoods. However, graph-structured data has no explicit neighborhood. node2vec introduces the idea of the second-order random walk into graph neighborhood sampling strategy. The emphasis of node2vec model is easy to switch between breadth-first sampling (BFS) and depth-first sampling (DFS) by adjusting hyperparameters. Moreover, its computational complexity is less than classical BFS and DFS strategies. Because of its efficiency and great performance on graph embedding learning task, node2vec is an ideal choice among shallow embedding methods.

In recent years, a lot of studies have focused more on deep embedding method rather than the shallow one. Deep embedding method usually refers to algorithms that learn graph features via graph neural networks (GNN). GNN is a class of artificial neural networks constructed for graph-structured data. Inspired by the success of convolutional neural networks (CNN) on grid data, many GNN architectures have been proposed to generalize the convolution operation on graphs. In this paper, we mainly focus on a method

SNACS '22, Master CS, Fall 2022, Leiden, the Netherlands

called graph convolutional networks (GCN) [6]. GCNs defines the graph convolution based on the graph Laplacian spectrum. It has achieved state-of-the-art performance on common graph related tasks, such as node classification, link prediction, etc.

We propose a novel method to extract embeddings from graphs in this paper. Our method is based on node2vec and GCNs. Motivated by the concept of meta learning, we regard the embeddings returned by node2vec as the meta information for GCNs. This prior knowledge helps to improve the quality of final graph embeddings, and therefore enhances the model performance in various tasks. Besides, we expect the prior knowledge will accelerate the training process of GCN.

The rest of the paper is organized as follows. Section 2 reviews related works on graph embedding learning methods. We illustrate basic task notations and preliminaries in Section 3. And then we describe approaches in detail in Section 4. Section 5 is an introduction to five open source datasets we use in the project. After that, we present our experimental set-up and results in Section 6. The paper ends with a conclusion section.

2 RELATED WORK

Our project draws inspiration both from the skip-gram-based shallow embedding method and the deep embedding method. The following will briefly review existing works related to these methods.

Skip-gram-based methods optimize graph embeddings to predict nodes in the defined context. Actually, this kind of methods leverages the matrix factorization technique implicitly [9]. Compared to early works that explicitly use matrix factorization, skip-gram-based methods are usually more computationally efficient. Deepwalk [10], an algorithm proposed in 2014, is one of pioneering studies that generalizes the idea of skip-gram model in language processing to graph embedding learning field. Deepwalk models the context of nodes by truncated random walks, which is analogous to sentences in textual data. It utilizes the local information obtained from random walks and learns a latent space that corresponds to features of vertices. Following the Deepwalk, LINE algorithm [11] is proposed to address the preservation of network properties and applicability on large-scale networks. The neighborhood sampling strategy of LINE is firstly simulating a BFS-style search for half of the feature dimensions and then a DFS-style search for remaining dimensions. Both Deepwalk and LINE are restricted to a specific sampling strategy of neighborhoods. node2vec [4] algorithm provides a more flexible option. It introduces a return parameter and an in-out parameter during second-order random walks that enables users to adjust the style of sampling. Its workflow is similar to Deepwalk and can be considered as a generalization.

GNNs have been extensively researched these years. Learning graph embeddings is one of the most important applications of GNNs. According to [9], GNN models used to devise graph embeddings are called deep embedding methods. A main challenge in designing GNNs is finding an efficient and easy-to-train filter to process graph signals. One solution is to apply graph Fourier transform to define the convolution operation on graphs. However, using a filter directly based on graph Laplacian matrix is computationally expensive. Defferrard et al. [3] suggest to approximate the filter by the k-order Chebyshev polynomial, which has a practical computational complexity. Kipf and Welling [6] propose GCN model that simplifies the approximation further. GCNs limit the order Chebyshev polynomial to one and assume the coefficients in polynomial are equal. Besides, GCNs consider the self-loops in the graph and introduce the renormalization trick, which improves the quality of learned graph embeddings. So far, GCNs have achieved the great performance on different graph related tasks.

Based on these works, we make the following contributions in the project.

(1) We propose a novel method to learn graph embeddings, inspired by meta learning concept. Our method fully utilizes the information from node2vec and the power of GCN architecture. We test our model on node classification and link prediction task.
(2) We evaluate node2vec, GCN, and our proposed method on five real-word datasets. In node classification task, we use additional metrics to assess the model performance, i.e. accuracy, recall, and macro F1 score, while the original paper of node2vec only reports macro F1 score.

3 PRELIMINARIES

3.1 Feature Learning Framework

Feature learning in networks is regarded as a maximum likelihood optimization problem [4]. We represent the given network as $G = (V, E)$, where V and E are nodes set and edges set respectively. Feature Learning task aims to find a projection $f : V \rightarrow R^d$, such that f optimizes the following objective function:

$$\max_f \sum_{u \in V} \log \ Pr(N_s(u)|f(u))$$

Projection f allocates each node an embedding vector with length d. In other words, projection f can be formulated as a matrix of size $|V| \times d$. $N_s(u) \subset V$ defines *network neighborhood* for a node u with neighborhood sampling strategy S. Please be aware, this *network neighborhood* is not equivalent to the commonly used concept *local neighborhood* which is only determined by the graph structure. On the contrary, *network neighborhood* is generated based on both network structure and sampling strategy. Therefore, the above formula aims to maximize the log-probability of observing a network neighborhood $N_s(u)$ for a node u conditioned on its feature embedding representation, given by projection f.

To simplify the above optimization formula and task, two standard assumptions are made. One is conditional independence, which means observing a neighborhood node is independent of observing any other neighborhood node. Then according to the property of independent event, the formula in the objective function above can be written in the form of multiplication:

$$Pr(N_s(u)|f(u)) = \prod_{n_i \in N_s(u)}^{N} Pr(n_i|f(u))$$

The other is symmetry in feature space, which means each source node and its neighborhood node have a symmetric effect over each other in feature space. In the reference [4], the conditional likelihood of each *(source node, neighbor node)* pair in the network

neighborhood is formulated as a softmax function:

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$$

Finally, the objective function can be rewritten using the above two assumptions in the form of:

$$\max_f \sum_{u \in V} \left[-\log \sum_{u \in V} \exp(f(u) \cdot f(v)) + \sum_{n_i \in N_s(u)} f(n_i) \cdot f(u) \right]$$

The first half of this formula $-\log \sum_{u \in V} \exp(f(u) \cdot f(v))$ is very expensive in computing. Therefore, a negative sampling method is applied to calculate it approximately [8]. For this objective function, stochastic gradient decent is applied to optimize it and obtain the projection f.

With a good designed network neighborhood sampling strategy S, the above optimization process could help to find projection f which makes the nodes embedding equipped with homophily and structural equivalence information [5], which could be taken advantage of in the downstream task.

The sampling strategy S is the most important part of feature learning framework. In this paper, second order random walk, the key point of node2vec, is applied as the sampling strategy. Before we dive into the details of node2vec and second order random walk in section 4.1, let us have a glance to two common downstream tasks, nodes classification and link prediction, which can be used to measure the performance of the nodes embedding generated by node2vec.

3.2 Downstream Task

Node embeddings are widely used in many common tasks related to networks, such as nodes classification and link prediction. These tasks, which leverage information from node embeddings, are called downstream tasks. Therefore, the best way to examine whether the node embeddings generated by Feature Learning Framework, like node2vec, is to check the performance of node embeddings on such downstream tasks.

In this paper, we consider two downstream tasks.

(1) Nodes Classification. Given a graph $G = (V, E)$ with part of node category labels, nodes classification task aims to classify the rest of nodes without labels into their category correctly.
(2) Link prediction. Given a subgraph $G' = (V', E)$ of the entire graph $G = (V, E)$, where $V' \subset V$ and the rest of edges are missing. Link prediction task aims to predict these missing edges correctly.

Node classification and link prediction are two basic but very important tasks in network analysis area. Therefore, it is of much importance to examine node2vec whether works well in these two tasks.

The downstream tasks also need a model, such as a classifier like Logistic Regression Classifier or Graph Neural Network. Determining which downstream model is better is not our focus in this paper. In this paper, we mainly care about whether the node embeddings generated from node2vec can make downstream model work better. Therefore, in Section 6, we will experimentally show the performance improvement gained by applying GCN with node embeddings from node2vec, compared to GCN without node embeddings.

4 APPROACH

4.1 node2vec

In Section 3.1, we discuss feature learning framework. And node2vec is such a feature learning model with special designed sampling strategy S. There are two classic search algorithms, BFS and DFS, can be naively used as network neighborhood sampling strategy S. However, according to the reference [4], directly applying BFS as network neighborhood sampling strategy leads the node embeddings to fully learn structural equivalence of networks. On the contrary, directly applying DFS as network neighborhood sampling strategy leads the node embeddings to fully learn homophily of networks. Since structural equivalence and homophily are both important similarities for nodes [8], it is definitely not a good way to throw away any one of them. Hence, we should use a method to combine BFS and DFS, and enable the sampling strategy to capture two kinds of similarity concurrently.

The reference [4] proposed to apply random walk algorithm, which combines the features from both BFS and DFS. Random walk is a method to take l steps to visit several nodes, and then consider the nodes which are visited in this walk as the network neighborhood of the source node u. Random walk can be defined as a Markov chain. Given a source node u and the i-th node c_i in the walk, the next node c_i to be visited is generated by the following distribution:

$$Pr(c_i = x|c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

where π_{vx} is the unnormalized transition probability between node v and x, and Z is the normalizing constant. In a weighted graph, π_{vx} is usually set to the edge weight between node v and x, namely w_{vx}. In an unweighted graph, $w_{vx} = 1$, that is to say, the next node to be visited is generated with equal possibility in the local neighborhood. We can notice that random walk process allow us not only to sample adjacent nodes that are close to the source node, but also to visit distant nodes which are far from the source node. Thus, this basic random walk has the capcility to capture both structural equivalence and homophily similarities simultaneously.

Nevertheless, it is hard for us to control and guide the basic random walk procedure due to its randomness. Hence, a second order bias random walk method is introduced by the reference [4] to solve this problem. Consider a basic random walk process that just traversed edge (t, v) and now resides at node v. And we want to decide the node to be visited in the next step of random walk. To make a decision, we will calculate the unnormalized transition probability $\pi_{v,x}$ on edge (v, x). Particularly, the unnormalized transition probability is defined as $\pi_{v,x} = \alpha_{pq}(t, x) \cdot w_{vx}$, where

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0, \\ 1 & \text{if } d_{tx} = 1, \\ \frac{1}{q} & \text{if } d_{tx} = 2. \end{cases}$$

p and q are two key hyperparameters introduced in this method, and d_{tx} represents the smallest distance between node t and node x. Notice that for each step, d_{tx} can only be three possible values: 0, 1, and 2. When $d_{tx} = 0$, it means the possible next step node x will lead the process to walk back to node t again. When $d_{tx} = 1$, it means the possible next step node x will lead the process to walk to the nodes which are the other direct local neighbor of node t. When $d_{tx} = 2$, it means the possible next step node x will lead the process to walk to the new node and get rid of node t's ego network. Hyperparameters p and q are called return parameter and in-out parameter respectively, which are the most significant hyperparameters in this walking process. By the adjustment of p and q, we are able to control and guide the walking process at the same time. To be more specific, when setting p to a higher value, the walking process is less likely to sample an already visited node. On the contrary, with a low value of p, the walking process is highly motivated to step back, then walk locally near the source node u. As for parameter q, when it is set to a high value, the walk will be biased towards contiguous nodes, and act more similarly to BFS.

Up to now, the second order bias random walk has been formulated. Then, we should apply it in the features learning framework as the network neighborhood sampling strategy for node2vec. In each walk from source node u with length L, we will obtain L network neighborhood nodes of u, which are used in the features learning framework as neighborhood network of each node to do obtain the optimized projection f. Finally, an embedding vector will be allocated to each node by node2vec.

Then the node embedding vectors generated by node2vec can be used in the downstream tasks. When dealing with node-related tasks, such as nodes classification, node embeddings can be directly applied in the downstream model. For example, in reference [4], authors use one-vs-rest logistic regression classifier, regarding the node embedding vectors as inputs.

However, in edge-related tasks like link prediction, we need to use edges' features rather than nodes' features. Thus, since each edge is composed by two nodes, we should apply a binary operation to transform the node embedding vectors obtained by node2vec into edges' features. Four commonly used operators g are shown in Table 1. The operator g outputs an embedding vector with the same size as the input vector u and v. And $g_i(\cdot)$ represents the i-th entry of the output vector.

Table 1: Commonly-used Binary Operators

Operator	Definition
Average	$g_i(u,v) = \frac{f(u_i)+f(v_i)}{2}$
Hadamard	$g_i(u,v) = f(u_i) * f(v_i)$
Weighted-L1	$g_i(u,v) = \lvert f(u_i) - f(v_i)\rvert$
Weighted-L2	$g_i(u,v) = \lvert f(u_i) - f(v_i)\rvert^2$

4.2 GCN

The Graph Convolution Network (GCN) is a specific method belonging to Graph Neural Network (GNN). In this section, we will briefly introduce the basic methodology of GCN, and then discuss our proposal — how to improve GCN by applying node2vec algorithm.

GNN is a deep learning framework for graph data. General graph neural network is formed of several layers that are composed of an aggregate process and an update process, which can be formulated in the following iteration formula.

$$h_u^{(k)} = \sigma\left(W_{self}^{(k)} h_u^{(k-1)} + W_n^{(k)} \sum_{v \in N(u)} h_v^{(k+1)} + b^{(k)}\right)$$

where $h_u^{(k)}$ denotes the embedding vector of node u in the output of k-th layer. $W^{(k)}$ and $b^{(k)}$ are k-th layer's trainable weights and bias respectively. Figure 1 illustrates the idea of iteration process. When conducting backpropagation, these parameters will be updated based on the gradients of the loss function. σ is a non-linear activation fuction, for example, ReLU or Sigmoid. Following this iteration process, each GNN layer takes the last layer's output embedding as its input, and applys aggregation and update process to output a new embedding using neighborhood information. Therefore, the graph structure information will be implicitly learned in the final output. And then this output will be used to calculate loss values together with ground truth labels, and update GNN's parameters just as what the common deep neural networks do.

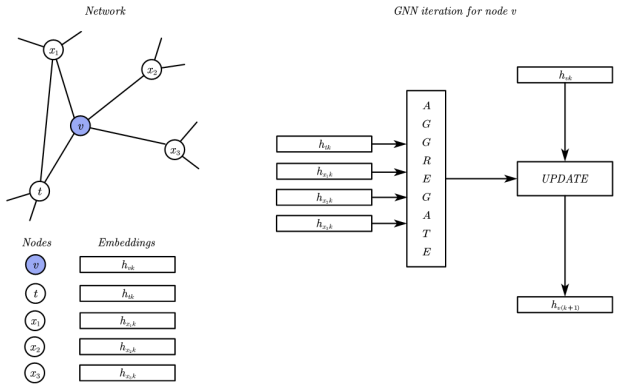


Figure 1: Iteration Process of GNN

GCN is one of the most popular GNN instances. Its aggregation and update process can be expressed as the following formula.

$$h_u^{(k)} = \sigma\left(W^{(k)} \sum_{v \in N(u) \cup u} \frac{h_v^{(k-1)}}{\sqrt{|N(u)||N(v)|}}\right)$$

Compared to the basic GNN, GCN applies a normalization factor to each input embedding vector based on its neighborhood information.

GCN has shown its great ability to capture graph information and deal with graph-related tasks. Nonetheless, like other GNN, GCN is as well an iteration process of embedding vector. So the problem is where the intial embedding vector comes from. There are three methods to generate the intial embedding vector:

(1) One-hot vector representation. One-hot vector is widely used in deep learning area. Hence, we can allocate each node a one-hot vector for representation. However, the drawback is obvious. The total number of nodes in a gragh is usually

very large, which means applying one-hot vector for each node will cause the input tensor too sparse. It is a waste of memory space and also will cause negative impact on training process.

(2) Transfer pretrained embedding. Another idea is transferring pretrained embedding vectors from other similar tasks. It has been commonly utilized in Natural Language Processing area. For instance, Google's well-designed word2vec model [7] has generated credible embedding vectors for a large number of words. When we apply GNN to a NLP-related task, we can transfer these pretrained embedding vectors as GNN's initial embedding vectors. This method is often quite effective. But the problem is that, we cannot always have such luck to find a similar task to our graph which have been conducted before. When we apply GNN to a brand-new task, this method loses its efficacy.

(3) Trainable embbeding layer. Most deep learning frameworks provide an embedding layer. We can allocate each node a trainable stochastic embedding layer by applying the embedding layer. However, the disadvantage is that randomly initialized embedding vectors could have a negative influence on training process.

Therefore, we propose a new method to combine GCN and node2vec to solve this problem. The idea is simple. We use node2vec to generate embedding vectors for each node, and then apply these vectors as intial embedding vectors in GCN model. Figure 2 shows the architecture of our proposed model. Due to the fact that the

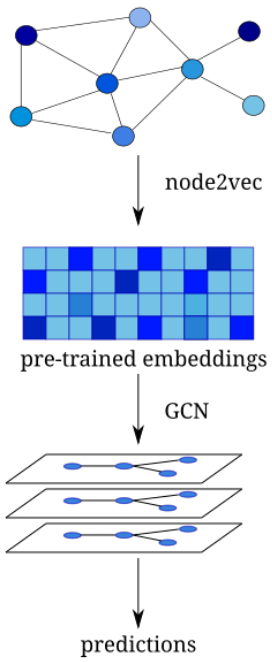


Figure 2: Architecture of Proposed Model

embedding vectors generated by node2vec are full of graph information, we can expect GCN can perform better with these vectors as initial embeddings. Besides, we also expect our proposed model will converge faster than the original GCN. In the next section, we test the effectiveness of our proposed method empirically via experiments.

5 DATA

We evaluate algorithms on five open source datasets. Three datasets, i.e. AIFB, MUTAG, and PubMed, are used in node classification task. Meanwhile, the goal of FB15K-237 and WordNet18RR data is link prediction. We apply built-in preprocessing APIs in Pytorch Geometric to convert the raw data into the desired format. The following is an overview of datasets.

AIFB. AIFB is a directed network modelling relationships among staff, research groups, and publications in AIFB research institute. The data set is firstly published for mining knowledge about instances with machine learning [1]. AFIB network has 8,285 nodes, 58,086 edges, and 4 node classes. The original data can be downloaded from https://data.dgl.ai/dataset/aifb.tgz.

MUTAG. MUTAG is a benchmark dataset for node classification task. It contains a collection of chemical compounds. We use the version provided by Deep Graph Library. It is a directed network with 23,644 nodes, 148,454 edges, and 2 node classes. The download link is https://data.dgl.ai/dataset/mutag.tgz.

PubMed. PubMed is a directed citation network in biomedical field provided by paper [12]. In the network, nodes represent documents, while edges denote citation links. It contains 19,717 nodes, 88,648 edges, and 3 node classes. The original data is on https://github.com/kimiyoung/planetoid/tree/master/data.

FB15K-237. FB15K-237 contains knowledge relationships sourced from the Freebase database. It is a modified version of FB15K dataset. FB15K-237 network is directed, including 14,541 nodes and 544,230 edges. The link of the original data is https://github.com/MichSchli/RelationPrediction/tree/master/data/FB-Toutanova.

WordNet18RR. WordNet18RR is an improved version of WordNet18 dataset, which is a large lexical database of English. The network is directed. It contains 40,943 nodes and 93,003 edges. The original data can be found on https://github.com/villmow/datasets_knowledge_embedding/tree/master/WN18RR/original.

Table 2 summarizes the datasets used in our experiments.

Table 2: Summary of Datasets

Data	Nodes	Edges	Classes
AIFB	8,285	58,086	4
MUTAG	23,644	148,454	2
PubMed	19,717	88,648	3
FB15K-237	14,541	544,230	-
WordNet18RR	40,943	93,003	-

6 EXPERIMENTS

6.1 Experimental Setup

We select accuracy, recall, and macro F1 score as metrics to comprehensively assess the model performance in node classification task, while the original node2vec paper [4] only reports the macro F1 scores. In link prediction task, we use area under the ROC curve (AUC) as metrics, which is consistent with the node2vec paper. We compare our proposed method with two benchmarks. One is the combination of node2vec and logistic regression in [4]. To be more specific, logistic regression model is fitted on the output features of node2vec and predicts the label — the class in node classification and the existence of the edge in link prediction. The other is GCN model [6]. All models are implemented with scikit-learn, PyTorch, and PyTorch Geometric packages in Python. Besides, hyperparameters in models are optimized by grid search technique. We automate and parallelize the hyperparameter tuning process with Ray Tune package. Table 3 summaraizes the software environment of our experiments.

Table 3: Summary of Software Environment

Item	Version	Function
Python	3.9	Programming language.
scikit-learn	1.1.2	Logistic regression.
PyTorch	1.12.0	Deep learning framework.
PyTorch Geometric	2.1.0	node2vec, GCN.
Ray Tune	2.1.0	Hyperparameter tuning.

As mentioned in Section 5, we use default pipelines to preprocess datasets. However, there is no negative sample in data for link prediction task. To avoid the adverse impact of data imbalance, we begin with recording all nonexistent edges in a list and then randomly sample edges from the list. We set the ratio of positive samples to negative samples as 1 to 1 in experiments. The random seed in all operations related to randomness is a fixed number to ensure reproducibility. We deploy our experiments on two servers. Table 4 summarizes our hardware environments.

Table 4: Summary of Hardware Environment

Item	CPU	GPU	RAM
Server 1	Intel Xeon Platinum 8358P	RTX A5000	80 GB
Server 2	Intel Xeon E5-2680	TITAN Xp	16 GB

6.2 Results: Node Classification

Table 5 reports the results of our experiments on node classification task. And Table 6 lists the main hyperparameter settings. Our proposed model directly uses the best node2vec setiings to obtain the input embeddings for GCN.

According to Table 5, the overall performance of our proposed model is better than the node2vec alone or the GCN alone. On AIFB dataset, our proposed method achieves higher accuracy and recall than benchmarks but performs slightly worse than GCN on macro F1 score. We conjecture two main factors contribute to the gap between GCN and our proposed model on macro F1 score. The first is the relative small size of AIFB data, and the second is the randomness in model training. The result of MUTAG is a little bit interesting. Our proposed method significantly improves the performance on recall and macro F1 score, compared to the original GCN. However, the accuracy of node2vec is about 10% higher than the proposed model. The high accuracy with the low recall of node2vec may indicate that the imbalance in data affects the training and evaluation processes. By contrast, predictions of the proposed model don't have a significant bias towards a specific class. On PubMed data, our proposed model outperforms other two baselines on all metrics, which supports the effectiveness of the proposed method.

Table 5: Summary of Results in Node Classification Task

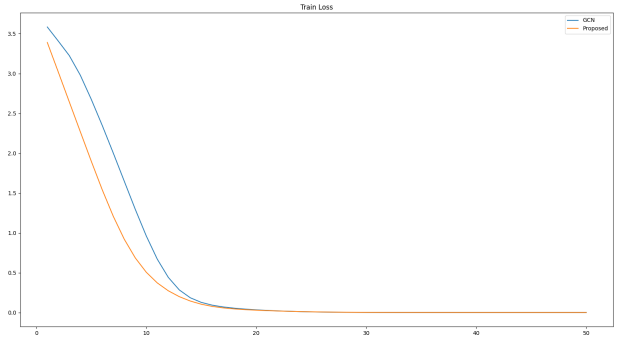
Dataset	Algorithm	Accuracy	Recall	Macro F1
AIFB	node2vec	0.9444	0.8750	0.9116
	GCN	0.9444	0.9375	**0.9508**
	Proposed	**0.9722**	**0.9583**	0.9416
MUTAG	node2vec	**0.7941**	0.5652	0.6500
	GCN	0.7353	0.6193	0.6151
	Proposed	0.6912	**0.7242**	**0.6857**
PubMed	node2vec	0.6650	0.6434	0.6438
	GCN	0.6770	0.6443	0.6482
	Proposed	**0.6920**	**0.6508**	**0.6616**

Table 6: Hyperparameter Settings of Node Classification

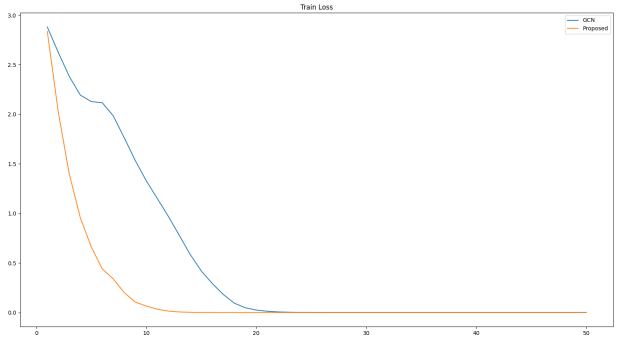
Algorithm	Parameter	Value		
		AIFB	MUTAG	PubMed
node2vec	p	0.25	2	4
	q	0.25	4	4
	walk length	30	30	20
	learning rate	0.1	0.01	0.01
	embedding dims	128	128	128
GCN	#layers	2	3	3
	learning rate	0.01	0.01	0.05
	epochs	50	50	50
Proposed	#layers	2	3	3
	learning rate	0.01	0.01	0.1
	epochs	50	50	50

Hyperparameter settings vary greatly from dataset to dataset. During the hyperparameter tuning process, we find that the selection of p and q is critical to node2vec, while the number of layers and the learning rate are important parameters in GCN for node classification task.

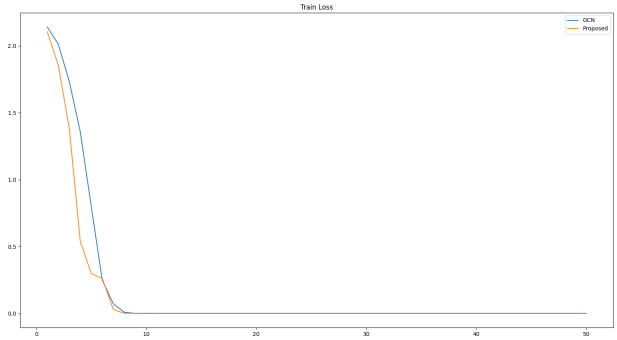
We also visualize the loss values of both GCN and our proposed method during the training. Figure 3 shows the visualization results on three datasets. The integration of pretrained embeddings accelerates the convergence rate of GCN, which is consistent with our assumption.



(a) AIFB: Training Loss



(b) MUTAG: Training Loss



(c) PubMed: Training Loss

Figure 3: Visualization of Training Loss. (yellow line: proposed method, blue line: GCN)

6.3 Results: Link Prediction

We perform all four types of binary operations mentioned in Table 1 on FB15K-237 and WordNet18RR data. Table 7 reports the results of link prediction. And Table 8, 9 summarize the hyperparameter settings used in two datasets respectively. It is worth noting that GCN link embeddings are generated by dot product operation according to the definition. Therefore, only the node2vec model and our proposed model can apply four binary operations, for they are concerned with utilizing node2vec link embeddings. We use the best node2vec settings to generate the pretrained link embeddings for the proposed model.

Models with the use of GCN perform much better than the node2vec alone in link prediction task. And our proposed method achieves higher AUC scores than the GCN model on both datasets. According to Table 7, different binary operations to generate node2vec link embeddings have a significant impact on model performance. Both the node2vec model and our proposed model have the best performance with hadamard operation on FB15K-237 dataset, while their highest AUC scores on WordNet18RR are achieved by applying weighted-L2 operation. Recall that the original node2vec paper [4] reports the hadamard operation has the best AUC sorces in link prediction task. Our conclusion is that the suitable operation depends on the properties of data.

Table 7: Summary of AUC in Link Prediction Task

Algorithm	Operation	Dataset	
		FB15K-237	WordNet18RR
node2vec	Average	0.7875	0.8567
	Hadamard	0.8626	0.8327
	Weighted-L1	0.8304	0.8454
	Weighted-L2	0.8276	0.8641
GCN	-	0.9816	0.8844
Proposed	Average	0.9810	0.8929
	Hadamard	**0.9819**	0.8897
	Weighted-L1	0.9817	0.8911
	Weighted-L2	0.9808	**0.8971**

Table 8: FB15K-237: Hyperparameter Settings

Algorithm	Parameter	Operation	Value
node2vec	p	Average	2
		Hadamard	2
		Weighted-L1	2
		Weighted-L2	2
	q	Average	0.5
		Hadamard	0.5
		Weighted-L1	0.5
		Weighted-L2	0.5
	walk length	Average	30
		Hadamard	20
		Weighted-L1	10
		Weighted-L2	10
	learning rate	Average	0.01
		Hadamard	0.01
		Weighted-L1	0.1
		Weighted-L2	0.1
	embedding dims	Average	128
		Hadamard	64
		Weighted-L1	128
		Weighted-L2	128
GCN	#layers	-	1
	learning rate	-	0.05
	epochs	-	50
Proposed	#layers	-	1
	learning rate	-	0.05
	epochs	-	50

Table 9: WordNet18RR: Hyperparameter Settings

Algorithm	Parameter	Operation	Value
node2vec	p	Average	2
		Hadamard	4
		Weighted-L1	0.25
		Weighted-L2	2
	q	Average	0.25
		Hadamard	2
		Weighted-L1	0.5
		Weighted-L2	1
	walk length	Average	20
		Hadamard	30
		Weighted-L1	10
		Weighted-L2	10
	learning rate	Average	0.1
		Hadamard	0.01
		Weighted-L1	0.1
		Weighted-L2	0.1
	embedding dims	Average	128
		Hadamard	64
		Weighted-L1	128
		Weighted-L2	0.1
GCN	#layers	-	1
	learning rate	-	0.005
	epochs	-	50
Proposed	#layers	-	1
	learning rate	-	0.005
	epochs	-	50

Hyperparameter settings are not only different from dataset to dataset but also from operation to operation. Even on the same dataset, we need to optimize hyperparameters when adopting another binary operation to obtain node2vec embeddings, which prolongs the tuning process. We use the parallelization strategy to speed up hyperparameter tuning in practice.

7 CONCLUSION

In this paper, we propose a method that applies node2vec to generate intial embedding vectors in order to improve the performance of GCN model. From experimental view, our proposed model outperforms individual node2vec and GCN model in most tasks.

We conduct several experiments mainly foucus on node classification and link prediction tasks. Although individual node2vec and GCN model already have a competitive performance, our model can enhance the final result in most cases. Especially for the link prediction task, our model can stably outperform individual node2vec and GCN model on both datasets. Experiments support that applying the embedding vectors contained graph information that are generated by node2vec can improve GCN's performance compared to the popular random initialization strategy. Notice that our model is composed of node2vec and GCN, which neither have special requirements of the graph data nor downstream task modality. Thus, our model is a general method to enhance GCN model in graph related tasks.

There are other kinds of GNN models rather than GCN. Because our model doesn't impose any specific requirement on GNN structure, it can be generalized to improve other GNN models as well. The extension to other GNN models is very straightforward and easy to accomplish. Thus, the future work includes conducting more experiments with the use of more kinds of GNN models to generalize our proposed method.

Considering our model is a combination of node2vec and GNN, the number of hyperparameters increase significantly in our model, which makes the tuning process more complicated. In our experiments, we only empirically search the best hyperparameter settings for our model, which may not be the global optimal. Therefore, the development of more sophisticated hyperparameter tuning strategy is also a possible direction for future works.

ACKNOWLEDGMENTS

We would like to thank Professor Frank Takes and Course Assistants for the help during the project.

REFERENCES

[1] Stephan Bloehdorn and York Sure. 2007. Kernel Methods for Mining Instance Data in Ontologies. In *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference (ISWC'07/ASWC'07).*

[2] HongYun Cai, Vincent W. Zheng, and Kevin Chen-Chuan Chang. 2018. A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637. https://doi.org/10.1109/TKDE.2018.2807452

[3] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16).*

[4] Aditya Grover and Jure Leskovec. 2016. Node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16).*

[5] Peter D Hoff, Adrian E Raftery, and Mark S Handcock. 2002. Latent space approaches to social network analysis. *Journal of the american Statistical association* 97, 460, 1090–1098.

[6] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations.*

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[8] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems* 26.

[9] Kevin P. Murphy. 2022. *Probabilistic Machine Learning: An Introduction.* MIT Press.

[10] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14).*

[11] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-Scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15).*

[12] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning (ICML'16).*