Improving DDI Prediction Performance of Node2vec Embedding Using Graph Neural Network Based Models

Lihui Zhang[1], Sook S. Ha[2]*

1 Department of Computer Science, Virginia Polytechnic Institute and State University

2* The Bradley Department of Electrical and Computer Engineering, Virginia Polytechnic Institute and State University

*Correspondence: sook@vt.edu

ABSTRACT

A drug-drug interaction (DDI) is a reaction between two or more drugs that can reduce or increase the reaction of a medicine synergistically or cause adverse side effects. DDI detection, therefore, is an important objective in patient safety and pharmaceutical industry. Many researchers try to predict the DDI of unknown drugs by training the known DDI data in-silico approaches. In-silico approaches can be categorized into three groups: knowledge-based, similarity-based, and graph-based. Among them, graph-based approaches are known to have achieved great performance by casting DDI prediction as a link prediction problem on DDI graphs. In this paper, we explore how we can improve DDI prediction performance of the embedding learning method node2vec[6] using representation learning algorithms of graph neural networks (GNNs). We first created and trained node2vec model to obtain initial drug features; then we used three GNN based models to improve the learned node2vec drug embedding; finally, we used four different classifiers to implement link prediction, which is DDI prediction. Our experimental results showed that all four classifiers performance were improved using GNN learned embedding.

Keywords: classifiers, drug-drug interactions, drug-drug interaction prediction, embedding learning, graphical neural networks, node2vec

1. INTRODUCTION

Drug-Drug Interactions (DDIs) have been a major cause of preventable adverse drug reactions, which can cause a significant burden on the patients' health and the healthcare system. Therefore, many novel algorithms for DDIs prediction have been proposed. Vilar implemented DDI prediction through molecular structure similarity analysis[22] and large scale similarity-based modeling[23]. Raja applied machine learning workflow on the DDI prediction problem using four different algorithms(Bayesian network, Decision tree, Random forest, K-nearest neighbors) on DDI corpus data[15]. Jialiang Huang developed a systematic method to predict DDI through Protein-Protein-Interaction(PPI) Network[8]. Feixiong Cheng applied machine learning algorithms through integrating drug phenotypic, therapeutic, chemical, and genomic properties[1]. Among all methods, node embedding learning methods achieved remarkable success[26]. Node embedding is a collective term for techniques for mapping graph nodes to vectors of real numbers in a multidimensional space. When applying node embedding on DDI prediction task, drugs and interactions will be considered as nodes and edges in a graph. DDI diction problem can be considered as a link prediction task under this context. Some existing state-of-art node embed- ding methods like SVD[5], node2vec, LINE[19], SDNE[24] all achieved promising results. Zhang et al.[27] introduced the manifold regularization based on drug features and pro- posed a novel matrix factorization method named MRMF to predict potential DDIs Wen Zhang adopted SDNE to learn drugs features and using attention mechanism to enhance predicting performance and analyzing the impact of each factor by comparing attention weights[9]. Deepika deigned a meta-learning framework of representation learning(similar to node embedding)[16]. However, GNN based models and algorithms didn't receive enough attention on this problem. GNN is a relatively new type of machine learning model. Generally speaking, a GNN model learns one node's feature by aggregating it's neighbor's features. Different GNN models adopt different aggregating algorithms. In this paper, we studied whether we can use GNN models to im- prove embedding learning's performance on the DDI prediction problem. In practice, we propose using Graph Auto-Encoders(GAE)[20] architecture to implement DDIs prediction. The majority of

GNN-based matrix completion methods are based on GAE, which applies a GNN to the entire network to learn a representation for each node[17]. We use the performance of node2vec model as benchmark, and then use three different GNN models (GCN[12], GraphSage[7], GAT[21]) as encoders to learn aggregated embedding of drugs and use four types of decoders to predict DDIs.

The major contributions of this paper can be summarized as follows: (1) We utilized GAE models for implementing DDI predictions; (2) We improved the performance of node2vec embedding learning using GNN models.

The following sections explains the materials and the methods we used in our study, experiments we conducted, and analyzes our experimental results.

2. MATERIALS AND METHODS

2.1 Datasets

DrugBank[25] is the most used public database, which contains DDI dataset covering 2,682,157 DDIs. In this paper, we constructed our DDI dataset for our experiments using *DrugBank* database.

2.2 Overview of Methods

In this section, we will provide the overviews of the methods and algorithms we used in our experiments for the study.

2.2.1 Graph Auto-Encoders (GAE)

GAEs are end-to-end trainable neural network models for unsupervised learning, clustering and link prediction on graphs[20]. GAE models are known to be efficient in matrix completion problem[17]. If we view drugs as nodes and DDI as links in a graph, predicting a DDI can be considered as a task to complete a DDI adjacency matrix. GAE model consists of an encoder and a decoder. The encoder encodes drugs into scalars E and decoders use these scalars to rebuild the whole graph by predicting the existence of a link between a pair of nodes/drugs. The encoder can be viewed as representation learning methods and decoder can be viewed as classifiers. In this paper, we use four types of encoders, node2vec, GCN, GraphSAGE, and GAT; and four types of decoders, Inner-product(IP)[20], Logistic Regression(LR)[2], Deep Neural Network(DNN) and Soft Decision Tree(SDT)[4]. Figure 1 demonstrates the entire workflow of our DDI prediction analysis using those encoders and decoders. In Figure 1, e_{ni} denotes the embedding for the ith drug from node2vec drug embedding matrix; eg_i denotes the embedding for the ith drug in the GNN drug feature matrix; d_n denotes the dimension length of e_n; dg denotes the dimension length of e_g; the $\odot$ operator denotes the element-wise product.

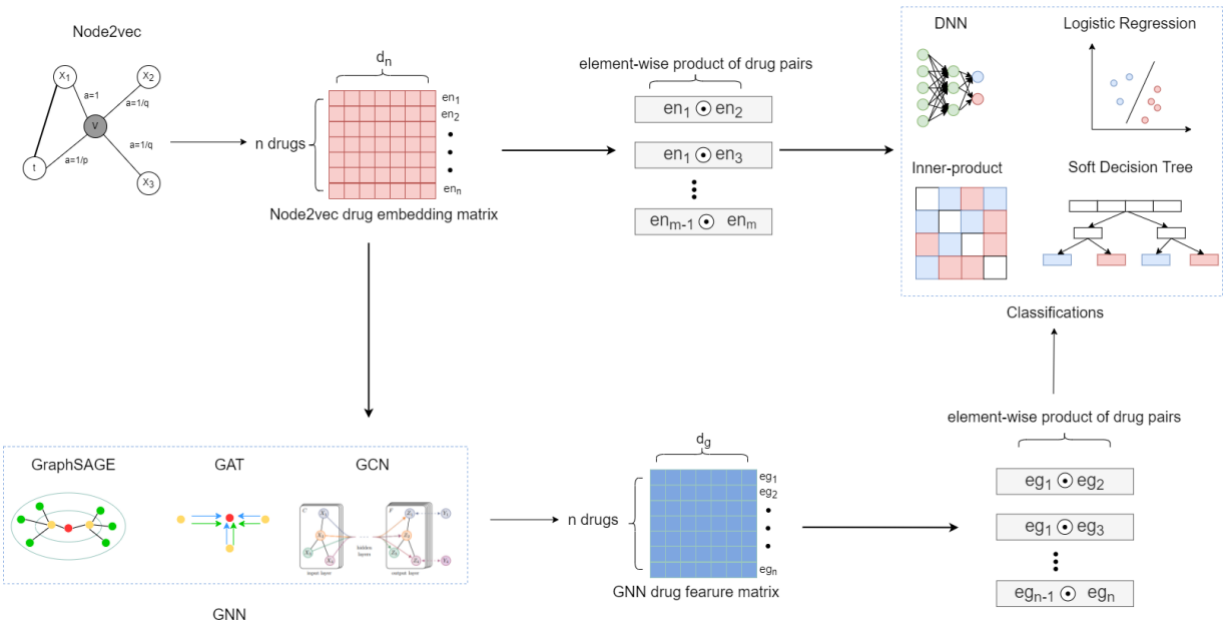


Figure 1: Overview of our DDI prediction analysis workflow. We first create the node2vec feature matrix (the pink feature matrix). Then we calculate the element-wise Hardmard product for each pair of drugs in the embedding matrix and feed the entire element-wise drug pair products into classification models. The classification models then classify whether a pair of drug has an interaction or not. We use these classification performances as benchmark and conduct experiments to find if GNN models can further improve the feature selection performance of node2vec. We feed the node2vec feature matrix into GNN models and get a new GNN feature matrix(the blue matrix). Then we perform Hardmard product on each pair of drugs and feed them into classifiers. Finally, we compare the classification performances of node2vec feature matrix and GNN feature matrix.

2.2.2 Node2vec

Node2vec is a DeepWalk[14] based algorithmic framework for learning on graphs. Compared to regular deep walk algorithm, it adopts a biased random walk algorithm using two parameters *p* and *q* to control the direction of random walks. Parameter p controls the probability of revisiting a node immediately and parameter *q* controls walking direction of inward or outward. Figure 2 demonstrates how the biased random walking algorithm choose next visiting node[6]. By tuning *p* and *q*, the learned node embedding should contain information about the neighbor structure and global role information simultaneously[6]:

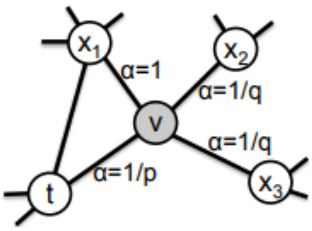


Figure 2. Random walk with parameters *p* and *q*

$$\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx} \qquad (1)$$

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

where π_{vx} is the unnormalized probability of visiting node x next at node v, t denotes the previous visited node, d_{tx} denotes the distance between node t and x, w_{vx} denote the weight of the edge between node v and x [6]. We set default edge weight of 1 for all edges. To be more specific, increasing p can decrease the probability of visiting the previous visited node, increasing q can decrease the probability of visiting a node that is not connected to the previous visited node. In other words, increasing the probability of walking outwardly. Visiting a node that is connected to the previous node is constant.

Node2vec first performs random walk starting from each node and repeat for several times (five by default) then taking nodes as words and paths as sentences and utilize skip-gram [10] architecture to optimize a mapping function $f : V \rightarrow E^n$, which maximizes the log-probability of observing a network neighborhood N(μ) for a node μ conditioned on its feature representation:

$$\max_f \sum_{\mu \in V} logPr(N(\mu)|f(\mu)), \qquad (3)$$

where *V* denotes drugs and E^n denote learned embedding. After training, we get drugs embedding $E^n \in R^{n \times d_n}$, where *n* denotes the number of drugs and d_n denotes the dimension of node2vec drug embedding scalars.

2.2.3 Graph Neural Networks (GNN)

GNN models seek to optimize an updating function: f : $E^n \rightarrow$! E^g, where E^n denotes node2vec drug embedding and E^g denotes GNN drug embedding. After training, we also should get drug representations $E^g \in \mathbb{R}^{n \times d_G}$. GCN builds on concepts from convolutional neural networks (CNNs) for images and extends them to arbitrary graphs by constructing locally connected neighborhoods from the input graphs. These neighborhoods are generated efficiently and serve as the receptive fields of a convolutional architecture, allowing the framework to learn effective graph representations[12]. In practice, the equations (4), (5), (6) shows GCN model's forward updating algorithm:

$$E^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}E^{(l)}W^{(l)}) \quad (4)$$

$$\widetilde{A} = A + I_N \quad (5)$$

$$\widetilde{D}_{ii} = \sum_j \widetilde{A_{ij}}, \quad (6)$$

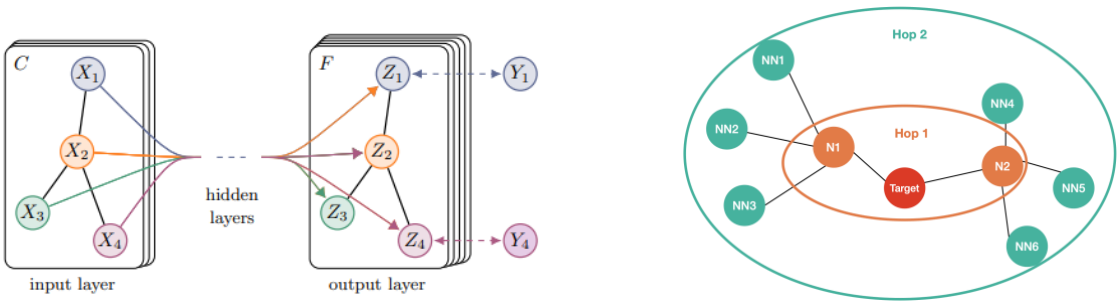
where $E^{(l)}$denotes current embedding of drugs, $\tilde{A}$ is the DDI graph adjacency matrix with self-loop, $\widetilde{D}$ is diagonal matrix, each element in its diagonal represents one node's degree, which is the number of total interactions for one drug, σ is activation function. In each forward message propagating process, GCN model will transfer adjacency matrix into a Laplacian matrix, which is basis of spectral decomposing. Then multiplying it with current features matrix and pass through a one-layer neural network. Figure 3 (a) demonstrates the structure of a GCN model.

GraphSAGE is an inductive graph embedding algorithm. It learns the embedding of nodes by sampling and aggregating neighborhood nodes features. Figure 3 (b) shows the overview of GraphSAGE model. For each target node, GraphSAGE model will first aggregate embedding within hop *k*, then concatenate it with target node's current embedding and pass through a one layer of neural network. In practice, we utilize the following equations:

$$E^k_{N(v)} = Aggregate_k(E^{k-1}, \forall \mu \in N(v)) \quad (7)$$

$$E^k_v = \sigma(W^k CONCAT(E^{k-1}_v, E^k_{N(v)})), \quad (8)$$

where *k* denotes depth, *N(v)* denotes neighbors of node *v*, μ is one of the neighbor nodes, the aggregate function can be averaging or multiplying. With deeper of *k*, GraphSAGE is able to aggregate more layers of neighbor information for each drug.



(a) (b)

Figure 3. Structure of Graph Convolutional Network model (a) ; structure of GraphSAGE model (b)

Graph Attention Networks (GATs) adopt the multi-head attention mechanism to learn drug embedding. Multi-head mechanism means utilizing attention mechanism for multiple times and concatenating calculated attention weights. Figure 4 demonstrates a GATs model with 3 attention heads. Each line indicates a "head" of attention weights.

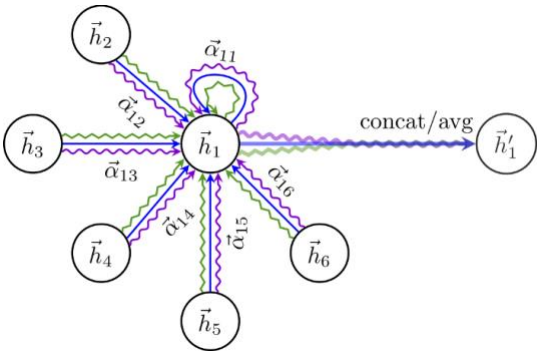


Figure 4: Structure of Graph Attention Networks model

Equations (9) and (10) show how the attention weights are calculated:

$$\alpha_{ij} = \frac{exp(\sigma(a^T[WE_i||WE_j]))}{\sum_{k\in N(i)} exp(\sigma(a^T[WE_i||WE_k]))} \tag{9}$$

$$E_i' = ||_{h=1}^{H}\sigma(\sum_{j\in N(i)} a_{ij}WE_j), \tag{10}$$

,

where T means transpose, || denotes concatenation, a denotes a one-layer attention network's parameters, W denotes parameter matrix, H denotes the number of attention heads. Each drug will aggregate attention weights of all interacting drugs.

2.2.4 Classifications

In this section, we will use y_{ij} to denote the ground truth of whether i^{th} and j^{th} drugs have interaction for all equations and use E to denote either type of drug embedding.

Inner Product(IP) is the simplest classifier compared two other trees. It just uses the dot product of two drug representations to make prediction:

$$h(E_i, E_j) = \begin{cases} 0 & \sigma(\sum_{k=0}^{d_g}(E_i \odot E_j)_k) < 0.5 \\ 1 & \text{otherwise,} \end{cases} \tag{11}$$

,

where E_i and E_j denote embedding of the ith and the jth drugs . We use the *Hardmard* product as the input for the four classifications, so we summarize the product to replace the dot product. The loss function L is defined as:

$$L = -\log\sum_i \sum_{j\in N(i)} E_i^T E_j - \log 1 - \sum_i \sum_{j\in NS(i)} E_i^T E_j, \tag{12}$$

,

where $N(i)$ and $NS(i)$ denote neighbor and negative sampling of the ith drug respectively.

Logistic regression(LR) is a statistical model that in its basic form uses a logistic function to model a binary dependent variable. The logistic regression model uses the following equation make predictions:

$$h(E_i, E_j) = \sigma(\sum_{k=1}^{d_g} \beta_k (E_i \odot E_j)_k + \beta_0), \qquad (13)$$

where β denotes regression weights, E_i and Ej are drug embedding. We also adopt cross entropy to calculate loss and use backward propagation to optimize weights. The loss function can be defined as:

$$L = -y_{ij} log(h(E_i, E_j)) - (1 - y_{ij}) log(h(E_i, E_j)), \qquad (14)$$

Soft decision tree (SDT) is a classification method improved from standard decision tree. There are 2 types of nodes in a SDT: inner node and leave node. Figure 6 demonstrates the structure of a SDT with a single inner node and two leaf nodes, where σ is activation function, w and b are weight matrix and bias. In practice, inner nodes are used to determine the path and leaf nodes are used to make predictions. When reaching a leaf node, the SDT will use the following equation to make prediction:

$$h(E_i, E_j) = \begin{cases} argmax(Q_l) & \sigma(xw + b) < 0.5 \\ argmax(Q_r) & \text{otherwise}, \end{cases} \qquad (15)$$

where Q_l and Q_r are positive and negative samples distribution at the node. SDT model can consist of multiple levels. With the increasing of the depth of the SDT, the training time will grow significantly. So we set max depth of 8 to limit the growth of the tree. In the original paper[4], the author offered 2 different ways to make predictions:

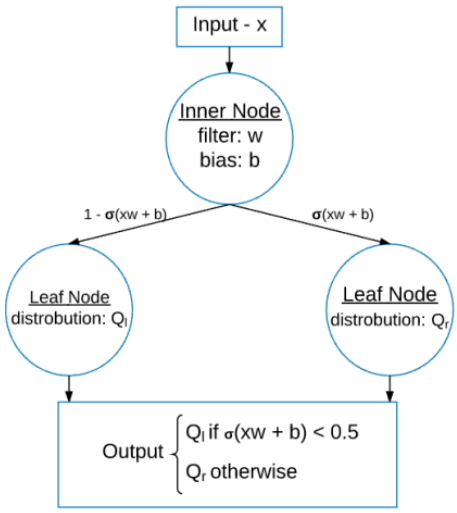


Figure 6. Structure of a soft decision tree

1. Following the path with greatest possibility and making prediction based on the distribution of the reached leaf node.
2. Weighted averaging all leaf nodes' distribution using respective path probabilities.

In this paper, we adopted the second method, and the loss function can be defined as:

$$L = -log(\sum_{l \in LeafNodes} P^l(x) \sum_k T_k log Q_k^l) \qquad (16)$$

$$Q_k^l = \frac{exp(\phi_k^l)}{\sum_{k'} exp(\phi_{k'}^l)}, \qquad (17)$$

where T is target distribution, $P^l(x)$ is the probability of arriving at l^{th} leaf node given the input x. Q^l_k denotes the probability distribution at the lth leaf, and each ∅ is a learned parameter at that leaf.

Deep Neural network (DNN) is known to be a better performing classifier. In this paper, we implemented a three-layer perceptron's model to make prediction. We adopted ReLU[11] activation function between hidden layers and Softmax activation function just before the output layer, so that the output of DNN can be considered as the probability of each class. DNN model uses the following equation to make predictions:

$$h(E_i, E_j) = \arg\max \sigma(f(E_i \odot E_j; \theta)), \qquad (18)$$

where f is the mapping function of the DNN classifier, θ denotes all parameters in the DNN model. We also adopt cross entropy to define loss. The loss function can be defined as:

$$P(E_i, E_j) = \max\left(\sigma(f(E_i \odot E_j; \theta))\right) \qquad (19)$$

$$L = y_{ij} \cdot \log P(E_i, E_j) + (1 - y_{ij}) \cdot \log 1 - P(E_i, E_j), \quad (20)$$

where function P denotes the probability of the prediction.

3. EXPERIMENT

3.1 General Setting

All models in this paper were developed using Pytorch[13] and Pytorch geometric[3] libraries.

3.1.1 Negative Sampling

One special problem in DDIs prediction is that we don't have a reliable negative dataset. Meta-learning is a solution under such circumstances[16] and achieved good results of DDI predictions. We can just randomly sample negative edges which do not exist in our dataset. But this approach may include potential positive edges into negative set, which can be a serious problem considering major DDI adverse effects which threat human lives. Hence, we performed negative sampling on older *DrugBank* dataset while assuring the edges in the negative set won't appear in the latest version of *DrugBank* dataset and conducted the case study using the latest DDI dataset.

3.1.2 Evaluation Metrics

When training the models, we adopted five evaluation metrics for measuring the performance of the models, such as Area Under Curve(AUC), Average Precision(AP), Accuracy, Recall rate, and F-1 score. We used AUC as the primary metric and preserved the model with the highest AUC on validation set for testing.

To verify the effectiveness and robustness of our models, 5-fold cross validation was performed. We randomly split the dataset into five subsets. For each fold, we further split 10% training set for validation and preserved the model with best validation performance for testing.

Due to the uncertainty of negative sampling and random walks process of node2vec, each set of experiment parameters were repeated for five times and the average test performances were compared, and the best performing model was identified. In practice, we trained each model using different parameters for 500 epochs and preserved the model with highest validating AUC. Then average testing AUCs were compared to decide which model achieved the best performance.

3.2 Parameter Setting

Node2vec and GNN models have their own unique parameters, and we will discuss how we set up these parameters to generate optimal output in our experiments.

3.2.1 Node2vec

For node2vec model, we optimized five parameters: $p \in \{0.5, 1\}$, $q \in \{0.5, 1\}$ which control the directions of random walks, embedding dimension $d_n \in \{8, 16, 32, 64, 128, 256\}$, walk length *w* and walks per node *n*, context size *s*. We didn't define the scope of *w*, *n*, and *s* because empirically longer walks, more walks per node, and larger context size would extract more information. Hence, we tried to optimize other three parameters first and then improved it by increasing the three parameters until model's testing performances were converged.

3.2.2 Graph Neural Networks

The different combinations of GNN models and classifiers may also create performance differences. Therefore, we enumerated all possible combinations and conducted experiments to determine their best performance. For each pair of GNN model and classifier, we considered following parameters: node2vec embedding dimension $d_n \in \{32, 64, 128\}$, GNN embedding dimension $d_g \in \{32, 64, 128\}$.

GNN models can take any dimension of input and output. Hence we first conducted experiments to determine which combination of d_n and d_g was the best for each GNN model. The three GNN models/encoders have their own unique parameters. For GCN, we considered the number of convolution layers $c \in \{1, 2, 3\}$. For GAT, we considered the number of attention heads $\alpha \in \{1, 2, 3\}$. For GraphSAGE, we considered the depth of sampling and aggregating $k \in \{1, 2, 3\}$. For all GNN encoders, we used ReLU[11] as activation function between hidden layers and used dropout layer to avoid overfitting[18]. For dropout layers, we considered dropout rate $E \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$. For all classfiers/decoders, we used Sigmoid activation function when generating output.

3.3 Experimental results

3.3.1 Performance of Node2vec

Empirically, d_n and the number of training epochs would lead to major performance differences. Hence, we first conducted experiments to determine the best d_n for each classifier by setting $p = 1$, $q = 1$, $w = 80$, $n = 10$, $s = 10$ as default parameters. Then we compared each classifier's best performances with their best *dn* after different training epochs. Finally, we tuned *p* and *q* parameters and increased *w*, *n*, *size* parameters until classification performance is converged. Table 1 demonstrates the classification performances of node2vec for the four different classifiers. After GAE models were tuned with thw best parameter values, we compared their performances.

Table 1: Node2vec model's performances for different classifiers

Classifier	AUC	AP	Accuracy	Recall	F-1
IP	0.8805	0.8832	0.7239	0.9257	0.7702
LR	0.9323	0.9241	0.8610	0.8370	0.8575
DNN	0.9473	0.9393	0.8845	0.8647	0.8822
SDT	0.8349	0.7674	0.6491	0.6248	0.6403

3.3.2 Performance of GNN

When optimizing GAE models, we adopted a consistent strategy. Empirically, the length of node2vec scalar, the length of GNN scalar and the structure of GNN models would make major performance difference. So, we first conducted our experiments to find the best combination of the node2vec scalar and the GNN scalar length for each GNN model with all classifiers. Then we selected the best structure of each different GNN model and tried to optimize other parameters. To compare the performances of node2vec and GNN embedding learning straightforwardly, we demonstrated the performance of each embedding learning for the same classifier applied.

Inner Product(IP). Figure 7 (a) demonstrates the experimental results of IP classifier. We

can see that all three GNN models' performances are better than node2vec performance for any metric. GCN achieved its best performance when the parameter values were $c = 2$, $d_n = 128$, $d_g = 128$, GAT when $\alpha = 3$, $d_n = 64$, $d_g = 128$, GraphSAGE when $k = 2$, $d_n = 64$, $d_g = 32$.

Logistic Regression(LR). Figure 7 (b) demonstrates the DDI prediction performance comparison of all models when LR classifier was applied. The three GNN models performed significantly better than node2vec. When LR classifier was applied, GCN performed best with the parameter values $c = 2$, $d_n = 128$, $d_g = 128$, GAT with $\alpha = 3$, $d_n = 128$, $d_g = 128$, $q = 0.5$, GraphSAGE with $k = 2$, $d_n = 128$, $d_g = 64$.

Soft Decision Tree(SDT). Figure 7 (c) demonstrates the result of SDT classifier for all models. All three GNN learned models outperformed original node2vec embedding.

Deep Neural network (DNN). Figure 7 (d) demonstrates the results of DNN classifier. Again, all three GNN learned models outperformed the original node2vec embedding. With DNN classifier, GCN performed best with parameter values $c = 1$, $d_n = 128$, $d_g = 128$, $p = 0.5$, GAT with $\alpha = 3$, $d_n = 128$, $d_g = 128$, $E = 0.2$, GraphSAGE with $k = 2$, $d_n = 128$, $d_g = 128$.

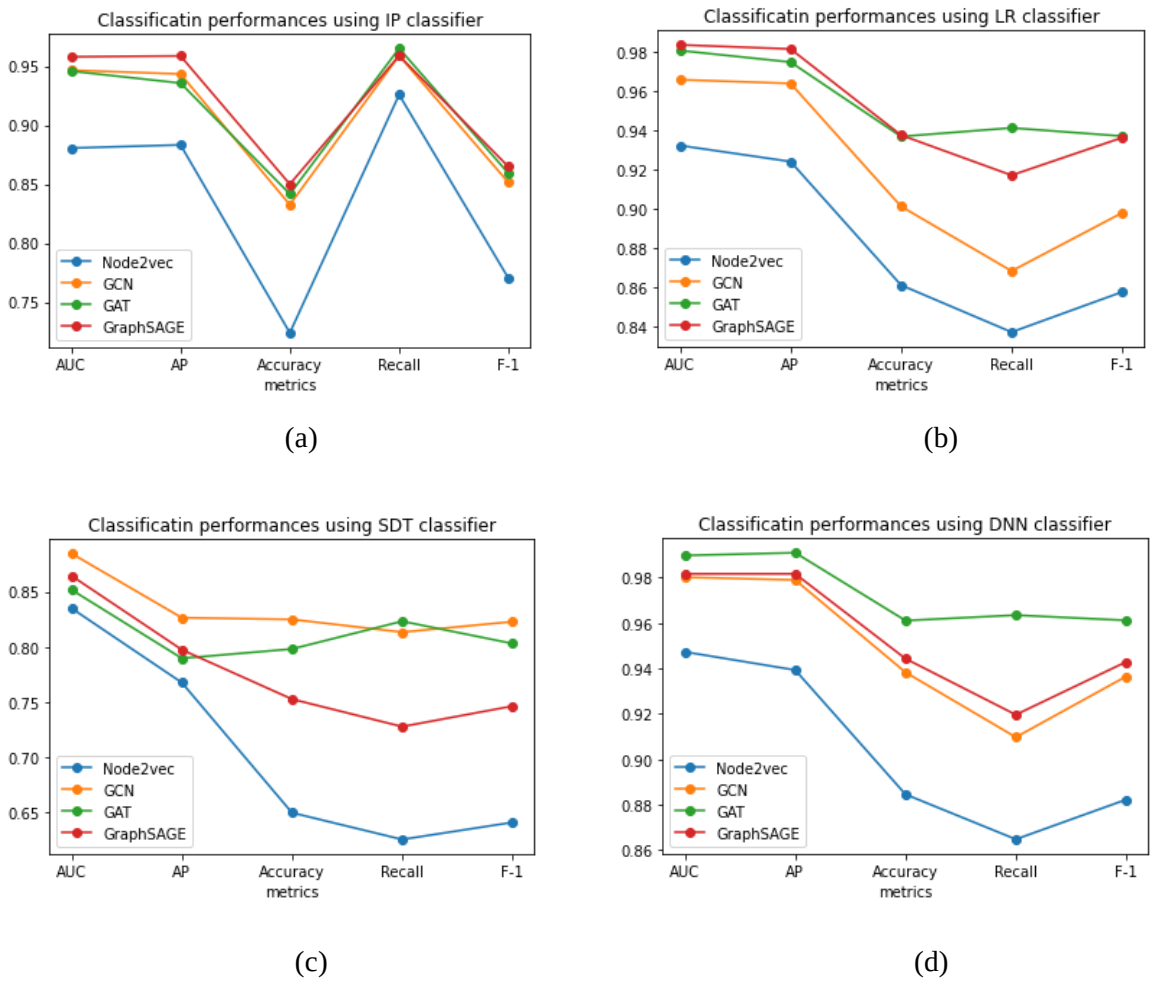


Figure 7: Classification results obtained using IP classifier (a); LR classifier (b); SDT classifier (c); DNN classifier (d). Each line in all figures indicates the performance of each algorithm for the specific classifiers.

Table 2 demonstrates the classification performance comparison of all embedding leaning algorithms for each different classifier. We can see that GNN models outperformed node2vec for any classifier and in any evaluation category.

Table 2. Performance comparison of GAE models and node2vec for all classifiers. All GAE models outperformed node2vec in all classification and in all evaluation categories

Model	AUC	AP	Accuracy	Recall	F-1
Node2vec+IP	0.8805	0.8832	0.7239	0.9257	0.7702
GCN+IP	0.9465	0.9433	0.8325	0.9588	0.8513
GAT+IP	0.9457	0.9356	0.8416	0.9650	0.8590
SAGE+IP	0.9579	0.9586	0.8499	0.9588	0.8647
Node2vec+LR	0.9323	0.9241	0.8610	0.8370	0.8575
GCN+LR	0.9659	0.9640	0.9012	0.8682	0.8979
GAT+LR	0.9808	0.9749	0.9369	0.9413	0.9371
SAGE+LR	0.9837	0.9816	0.9376	0.9171	0.9363
Node2vec+DNN	0.9473	0.9393	0.8845	0.8647	0.8822
GCN+DNN	0.9802	0.9790	0.9383	0.9097	0.9365
GAT+DNN	0.9898	0.9910	0.9611	0.9635	0.9612
SAGE+DNN	0.9817	0.9817	0.9443	0.9196	0.9429
Node2vec+SDT	0.8349	0.7674	0.6491	0.6248	0.6403
GCN+SDT	0.8847	0.8264	0.8249	0.8133	0.8228
GAT+SDT	0.8517	0.7895	0.7980	0.8232	0.8030
SAGE+SDT	0.8644	0.7973	0.7524	0.7274	0.7460

Table 3. The improvements made by the GAE models for all classifiers

model	AUC	AP	Accuracy	Recall	F-1
GCN + IP	+6%	+6%	+11%	+3%	+8%
GAT + IP	+6%	+5%	+12%	+4%	+8%
SAGE + IP	+7%	+7%	+13%	+3%	+9%
GCN + LR	+3%	+4%	+4%	+3%	+4%
GAT + LR	+5%	+5%	+7%	+11%	+7%
SAGE + LR	+5%	+6%	+7%	+8%	+8%
GCN + DNN	+4%	+4%	+5%	+4%	+5%
GAT + DNN	+4%	+6%	+8%	+10%	+8%
SAGE + DNN	+4%	+5%	+6%	+5%	+6%
GCN + SDT	+5%	+6%	+8%	+5%	+8%
GAT + SDT	+2%	+2%	+15%	+10%	+16%
SAGE + SDT	+3%	+3%	+11%	+10%	+10%

In summary, referring to the architecture of GAE model, we designed our DDI prediction workflow where learning models can be used as encoders and classifiers can be used as decoders. We first trained and optimized node2vec models; then using the node2vec embedding as input, we trained GNN models to further improve their predicting performance using the same classifiers (decoders) used on node2vec embedding. Setting up the best predicting performances of node2vec as benchmark, we built our GAE models. After GAE models were optimized, we compared their best performances against the node2vec benchmark performances, which are shown in Table 2. We observed that the predicting performances of the GNN learned models outperformed the benchmark performances of node2vec in all categories as shown. Table 3 demonstrates the improvement scales of GNN learned models in all evaluation categories. Accuracy and F-1 scores of the GNN models were significantly improved when IP classifier/decoder was applied. When LR classifier/decoder was applied, GAT and GraphSAGE performances were improved more

significantly than GCN performance, especially for the recall values.

4. CONCLUSION

In this paper, based on the great prediction performance of GNN based representation learning models, we examined if GNN models can improve the performance of representation learning algorithms. We examined the performance of GNN based models on DDI prediction. We evaluated different GNN models against one of the most classic representation learning algorithms, called node2vec. Our experimental results showed that all three GNN models outperformed all node2vec based models. This means GNN drug embedding contained more information than plain node2vec embedding. Among the three GNN models, GAT achieved the best performance in DDI prediction. From the final experimental results, GNN models could improve node2vec performance by 3% at least and upto10%. In the future, we consider examining whether GNN can be used to improve other representation learning algorithms such as SDNE and analyzing the performance difference in a wider perspective.

5. REFERENCES

[1] F. Cheng and Z. Zhao. Machine learning-based prediction of drug-drug interactions by integrating drug phenotypic, therapeutic, chemical, and genomic properties. 2014.

[2] S. W. P. M. S. J. S. S. G. K. L. S. F. V. Detrano R, Janosi A. International application of a new probability algorithm for the diagnosis of coronary artery disease. 1989.

[3] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[4] N. Frosst and G. E. Hinton. Distilling a neural network into a soft decision tree. *CoRR*, abs/1711.09784, 2017.

[5] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):205–224, 1965.

[6] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[7] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[8] J. Huang, C. Niu, C. D. Green, L. Yang, H. Mei, and J.-D. J. Han. Systematic prediction of pharmacodynamic drug-drug interactions through protein-protein-interaction network. *PLOS Computational Bi- ology*, 9(3):1–9, 03 2013.

[9] S. Liu, Y. Zhang, Y. Cui, Y. Qiu, Y. Deng, W. Zhang, and Z. Zhang. Enhancing drug-drug interaction pre- diction using deep attention neural networks. *bioRxiv*, 2021.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013.

[11] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[12] M. Niepert, M. Ahmed, and K. Kutzkov. Learn- ing convolutional neural networks for graphs. *CoRR*, abs/1605.05273, 2016.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Brad- bury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, L. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imper- ative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché- Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[14] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.

[15] K. Raja. Machine learning workflow to enhance pre- dictions of adverse drug reactions (adrs) through drug- gene interactions: application to drugs for cutaneous diseases. 2017.

[16] D. S S and G. Tv. A meta-learning framework using representation learning to predict drug-drug interaction. *Journal of Biomedical Informatics*, 84, 06 2018.

[17] W. Shen, C. Zhang, Y. Tian, L. Zeng, X. He, W. Dou, and X. Xu. Inductive matrix completion using graph autoencoder, 2021.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to pre- vent neural networks from overfitting. *Journal of Ma- chine Learning Research*, 15(56):1929–1958, 2014.

[19] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line. *Proceedings of the 24th International Conference on World Wide Web*, May 2015.

[20] M. W. Thomas N. Kipf. Variational graph auto- encoders. 2016.

[21] P. Veliockovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks, 2018.

[22] S. Vilar. Drug-drug interaction through molecular structure similarity analysis. 2012.

[23] S. Vilar. Similarity-based modeling in large-scale pre- diction of drug-drug interactions. 2014.

[24] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, page 1225–1234, New York, NY, USA, 2016. Association for Computing Machinery.

[25] G. A. S. S. H. M. S. P. C. Z. W. J. Wishart DS, Knox C. Drugbank: a comprehensive resource for in silico drug discovery and exploration. 2006.

[26] X. Yue, Z. Wang, J. Huang, S. Parthasarathy, S. Moosavinasab, Y. Huang, S. M. Lin, W. Zhang, P. Zhang, and H. Sun. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251, 10 2019.

[27] W. Zhang, Y. Chen, D. Li, and X. Yue. Manifold regularized matrix factorization for drug-drug interaction prediction. *Journal of Biomedical Informatics*, 88:90– 97, 2018.