applied sciences

MDPI

Article

RLC-GNN: An Improved Deep Architecture for Spatial-Based Graph Neural Network with Application to Fraud Detection

Yufan Zeng [1] **and Jiashan Tang** [2,*]

[1] College of Telecommunications & Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing 210003, China; yufanzeng@outlook.com
[2] College of Science, Nanjing University of Posts and Telecommunications, Nanjing 210003, China
* Correspondence: tangjs@njupt.edu.cn

Abstract: Graph neural networks (GNNs) have been very successful at solving fraud detection tasks. The GNN-based detection algorithms learn node embeddings by aggregating neighboring information. Recently, CAmouflage-REsistant GNN (CARE-GNN) is proposed, and this algorithm achieves state-of-the-art results on fraud detection tasks by dealing with relation camouflages and feature camouflages. However, stacking multiple layers in a traditional way defined by hop leads to a rapid performance drop. As the single-layer CARE-GNN cannot extract more information to fix the potential mistakes, the performance heavily relies on the only one layer. In order to avoid the case of single-layer learning, in this paper, we consider a multi-layer architecture which can form a complementary relationship with residual structure. We propose an improved algorithm named Residual Layered CARE-GNN (RLC-GNN). The new algorithm learns layer by layer progressively and corrects mistakes continuously. We choose three metrics—recall, AUC, and F1-score—to evaluate proposed algorithm. Numerical experiments are conducted. We obtain up to 5.66%, 7.72%, and 9.09% improvements in recall, AUC, and F1-score, respectively, on Yelp dataset. Moreover, we also obtain up to 3.66%, 4.27%, and 3.25% improvements in the same three metrics on the Amazon dataset.

Keywords: graph neural network; fraud detection; deep architecture; residual structure; layered architecture

Citation: Zeng, Y.; Tang, J. RLC-GNN: An Improved Deep Architecture for Spatial-Based Graph Neural Network with Application to Fraud Detection. *Appl. Sci.* **2021**, *11*, 5656. https://doi.org/10.3390/app11125656

Academic Editor: Steven Walczak

Received: 16 May 2021
Accepted: 12 June 2021
Published: 18 June 2021

1. Introduction

In the current information age, things are moving towards cyberspace. The Internet has become a significant part of modern society. Moreover, fraud activities have subsequently expended from offline to online. Fraudsters post fake content anonymously, which disrupts the normalcy of cyberspace, and benign users may suffer losses as a result. It is inefficient and costly to manually censor tens of millions or even hundreds of millions of complicated online content to make sure whether the information is fraud or not. Performing fraud detection efficiently has important research significance.

Considering a social networking service or a shopping site, we abstract its users, reviews, or any information into nodes, and regard the relationship between nodes as edges. Then, we can build a graph, and GNNs are applied to detect the suspicious characteristics of nodes so as to find out the fraudsters. In fact, graph neural networks (GNNs) extend application field of classic deep learning to tasks with non-Euclidean data [1]. By dealing with relation camouflages and feature camouflages, CAmouflage-REsistant GNN (CARE-GNN) is proposed and achieves state-of-the-art results recently [2].

One of primary reasons for the success of neural network models in the area of computer vision is the ability to train deep network architectures (e.g., 19 layers for VGG [3], 22 layers for GoogLeNet [4], and 152 layers for ResNet [5]). However, GNNs suffer from a intrinsic limit which makes it difficult for GNNs to be trained with deep architectures. As a consequence, most state-of-the-art GNNs are restricted to have no more than four layers [6]. Experimental results of CARE-GNN show that it performs the best with single-layer

architecture. Moreover, its performance drops rapidly as the number of layers increases. In this case, the single-layer CARE-GNN aggregates neighboring information for every node only once. The model has no chance to fix its potential mistakes and to revise its inferences. We call the problem above as single-layer learning. Briefly, single-layer learning has two meanings: (1) intrinsic shallow limit of GNNs and (2) no chance to correct mistakes made by previous layer and to revise inferences.

Based on the CARE-GNN algorithm, inspired by the work in [7], we first use a different method to stack multiple layers. The classical architecture of multi-layer for GNNs is defined by hop. We take the calculation process of a node v as an example. If we first take v's neighbors as central nodes and aggregate information from the neighbors' neighbors for each one, and then aggregate the neighbors' information into node v, we define that the model has two hops, which also means that the model has two layers. Thus, as we increase the number of layers, more and further information can be aggregated. However, the most valuable information always exists in neighbors that are directly connected to central nodes. The further the distance (i.e., the hop) is, the less relevant the information will be. Therefore, we consider how to make the most effective use of the information of directly connected nodes.

Based on the above considerations, we propose a new improved algorithm named Residual Layered CARE-GNN (RLC-GNN). We use a different method to define the concept of multi-layer. We utilize a layered structure to collect the most valuable information from neighbors. The architecture and working mechanism of RLC-GNN is similar to the models for classic deep learning tasks (e.g., image classification). In an iteration, every layer processes a same batch of nodes. Moreover, we do not update original dataset with each layer's output features. Furthermore, we introduce the residual structure into our algorithm to compensate the losses of information during propagation. Thus, RLC-GNN has the ability to correct its own mistakes and "think" more deeply and comprehensively. The two structures can form a complementary relationship and make the most effective use of neighboring information. Experiments are implemented on Yelp dataset and Amazon dataset provided in the work [2]. Numerical results show that the new algorithm obtains a significant improvement in performance w.r.t the CARE-GNN algorithm. The contribution of this work includes the following: (1) Adapt the idea of the new definition of multi-layer in a spatial-based GNN. (2) Present an improved algorithm named RLC-GNN to successfully train deep spatial-based GNN. (3) Introduce the residual structure into our multi-layer case to form a complementary relationship and present empirical analysis of how the combination of the two structures, layered architecture and residual structure, improve the performance. (4) Some experiments on a Yelp dataset and an Amazon dataset are conducted, and the proposed RLC-GNN achieves significant improvements with the application to fraud detection.

2. Related Works

GNNs are originally designed to deal with the complicated non-Euclidean data [8], which the classic deep learning algorithms are unable to process systematically and reliably in general. There are two main types of GNNs [6]: spectral-based GNNs and spatial-based GNNs. The first proposed spectral-based GNN algorithm implements the convolutional operation on topological graphs using spectral graph theory [9] (i.e., using the eigenvalues and eigenvectors of the Laplacian matrix of the graph to study the properties of graphs). Since then, many improved algorithms based on graph convolutional network (GCN) have appeared one after another. The first spatial-based GNN algorithm [10] was actually proposed much earlier, which iteratively aggregates neighborhood information and updates the node embeddings by global shared local transition function and local output function. As spatial-based GNNs are in possession of more flexibility in algorithm designing, most proposed GNN variants belongs to this type. Moreover, so is our proposed RLC-GNN.

Fraud detection is essentially a semi-supervised node classification problem. There are two reasons which make the fraud detection task special: One reason is that sample

distribution of datasets is extremely unbalanced, and the other reason is that fraudsters will actively conceal their features to avoid to be detected [2,11]. These characteristics make it difficult for classic deep learning algorithms to learn implicit rules in data. In this case, even if a classification model is obtained, it always have serious bias, which results in poor generalization. Various methods have been presented recently to solve the problem. The Graph Embeddings for Malicious accounts algorithm [12] is the first heterogeneous graph neural network approach for detecting malicious accounts, establishing multiple homogeneous subgraphs based on the types of nodes, and aggregating neighborhood information under each subgraph. Moreover, the GEM algorithm uses an attention mechanism to learn the importance of each type. The GCN-based Anti-Spam algorithm [13] is the first GCN-based spam detector which applies on heterogeneous graph and directly aggregates information of three types of nodes at the same time. Then, research of the Adversary Situation Awareness algorithm [14] gave the solutions to the fact of the confrontational behaviors of fraudsters, e.g., adding special symbols into texts which makes detectors unable to recognize the original semantic information or avoiding detection by switching devices and networks.

Current GNN algorithms are limited to shallow depth in the conventional sense which is defined by k-hop. Some approaches have been proposed to help train deeper GNNs. In the traditional CNN area, the work of ResNet [5] presents a residual learning framework to ease the training of very deep neural networks (as many as 152 layers). Moreover, the works in [15,16] apply the residual structure to break the shallow limit of GCNs. The work in [17] proposes a Node Normalization technique to reduce feature correlation and increase the smoothness of models, successfully helping train deeper GCN. The research of H-GCN [18] proposes coarsening procedure and correspondingly makes GCN deeper to enlarge the receptive field for each node.

However, many algorithms and methods are designed for spectral-based GNNs of which the calculation theories are not similar to those of classic deep learning models (e.g., convolutional neural network (CNN)). They cannot utilize well-developed techniques to assist training and make good use of the flexibility of structure design of spatial-based GNNs.

3. Preliminaries

3.1. Graph Representation Learning

Graphs are a general language for describing and analyzing entities with interactions. Many types of real-world data are graphs and form complex systems, e.g., computer networks, social networks, economic networks, code graphs, and molecules. The relation structure of these data contains valuable information of which we can take advantages for better prediction. A graph is represented as $\mathcal{G} = \{V, X, E\}$, where V is the set of nodes. For each node v_i, it is represented by a d-dimension feature $x_i \in \mathbb{R}^d$ in X. Each edge $e_{i,j} \in E$ indicates that node v_i and node v_j are connected due to a certain relationship between them. For example, in Figure 1, we show the famous Zachary Karate Club Network [19] which uses nodes to represent individuals and uses edges to represent friendship between two individuals. During Zachary's research, the club was divided into two communities which were led by the instructor and the club's chairman, namely, node 1 and node 34, respectively. Moreover, Zachary correctly predicted which individuals would join each community based on this graph structure.

An image is composed of many pixels that are regularly placed. Therefore, a unified and regular operation can be designed to process images. We show a typical processing method of a single CNN layer on an image (Figure 2). However, real-world graph data have some inconvenient characteristics that have arbitrary size, a complex topological structure, and no fixed node ordering, which makes it hard to apply classic deep learning models to solve graph learning tasks (Figure 3).

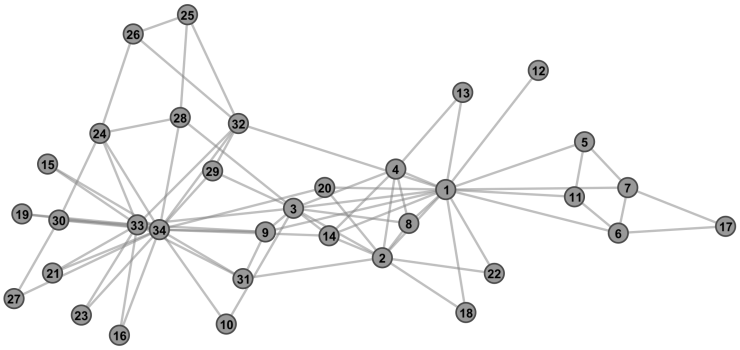


Figure 1. Zachary's social network of friendships between 34 members of a karate club at a US university in the 1970s. An edge connects two individuals if they socialized outside of the club.

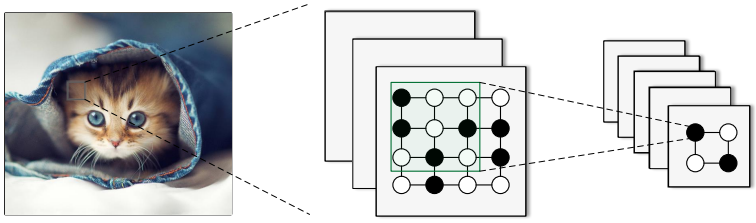


Figure 2. Single CNN layer with 3×3 filter. The filter slides over the feature maps of the image with a fixed step size. At each step, a 3×3 block of the feature map is taken, and the filter uses the block to calculate new features in a specific way (e.g., mean).

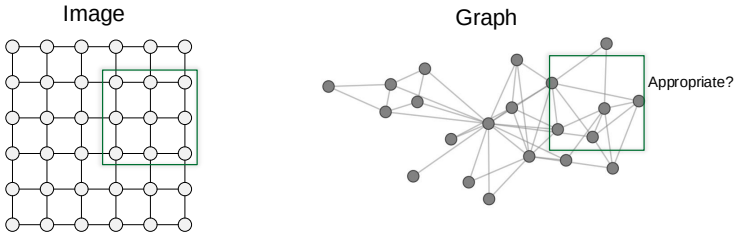


Figure 3. In the practice of classic deep learning, it is reasonable and effective to apply sliding window on images. However, there is no fixed notion of locality on graph, and we cannot apply sliding window on the graph data.

A disadvantage of traditional machine learning approaches on graphs is that feature engineering is an inescapable procedure, which makes these approaches of less flexibility. Node embedding, the graph representation learning method, alleviates the need to do feature engineering every single time. This method is an encoder–decoder framework [20], as is shown in Figure 4. There are two key components in encoder–decoder frameworks. Furthermore, the main problem in these networks is how to learn the mapping function (i.e., the encoder). For example, a graph encoder can use node features around each node to generate an embedding. The corresponding decoder extracts information from the embedding. Moreover, this might be information about the node's classification label. The generalized encoder architecture is often called GNN.

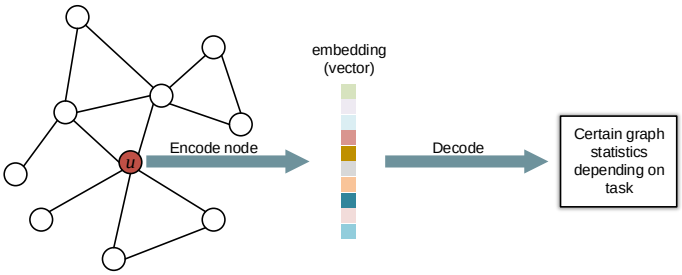


Figure 4. The encoder–decoder framework. The encoder is a function that maps nodes to low-dimension vectors (i.e., node embeddings). The decoder reconstructs certain graph statistics from the node embeddings that generated by encoder, e.g., predicting u's category on node classification task.

3.2. General GNN Framework

The key idea of GNNs is to transform information at the neighbors and combine it. In Figure 5, we show a typical architecture and the working mechanism of GNN. As can be seen, the aggregate function consists of two steps. The first step is message passing. And the second step is the formal aggregate operation (e.g., mean, pool, and long short-term memory). For a simplest GNN, the message passing function could be a constant and the aggregate operation could be a summary (i.e., we do nothing but simply adding neighboring information together). It is not hard to find that we can improve GNN's performance from at least three aspects: selecting strategy, aggregate function, and stacking multiple layers. For instance, if the importance of neighbors to central node is inconsistent, we can apply attention mechanism at the first step of aggregate function. Now, we can give the general formulation of GNNs:

$$h_{out}^{(k)} = Update(Aggregate(\{h_u^{(k-1)}, \forall u \in N(u)\}, W_{agg}^{(k)}), h_{in}^{(k)}, W_{update}^{(k)}) \quad (1)$$

where $h_{in}^{(k)}$ and $h_{out}^{(k)}$ denote input and updated node embedding at k-th layer, respectively. $h_u^{(k-1)}$ are neighbors' embeddings from previous layer. $W_{agg}^{(k)}$ and $W_{update}^{(k)}$ denote trainable matrices in aggregate function and update function at k-th layer, respectively. Moreover, $W_{update}^{(k)}$ usually denotes the k-th layer's trainable parameters of neural networks after the aggregate function. We show the general architecture of GNN in Figure 5.

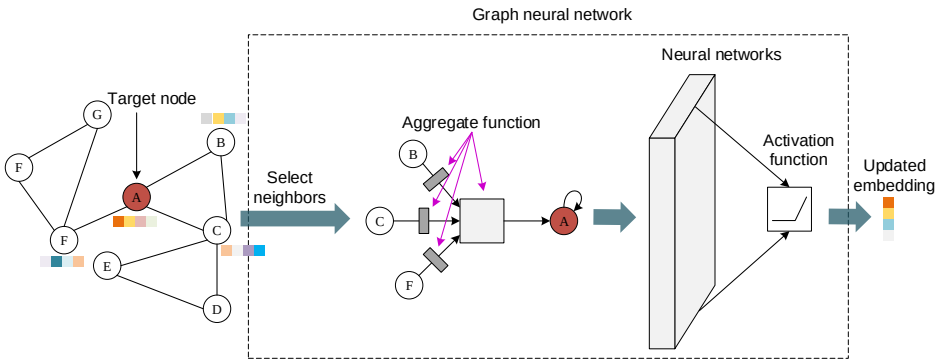


Figure 5. A typical and basic architecture and processing procedures of GNN. First, GNN selects neighbors with a certain strategy. Then, an aggregate function is applied to extract information around the central node. At last, the aggregated information passes through a neural network to be performed nonlinear transformation. The output is updated representation of central node.

The research of GraphSAGE [21] proposed a general framework for inductive node embedding:

$$h_v^{(k)} = \sigma([A_k \cdot AGG(\{h_u^{(k-1)}, \forall u \in N(u)\}), B_k h_v^{(k-1)}]) \quad (2)$$

where $h_v^{(k)}$ and $h_v^{(k-1)}$ denote the representation of node v at current layer k and previous layer $(k-1)$, respectively. Moreover, $h_u^{(k-1)}$ indicates the node embeddings of the neighbors from previous layer. A_k and B_k are the trainable weight matrices. σ is a nonlinear activation function. AGG is the aggregator. $h_v^{(k)}$ is generated by the following steps. First, each node aggregates features from its local neighborhood with the AGG aggregator into a single vector $h_{N(v)}^{(k)}$. After the aggregate of neighbors' features, GraphSAGE concatenates the node's previous representation with its neighborhood feature vector. Then, the concatenated vector is fed through a multi-layer perceptron (MLP) with ReLU activation function, and the output is the new representation of node v which will be used as the input at next layer.

Like classic deep learning, for supervised learning (i.e., we are given input $\boldsymbol{x}$, and the goal is to predict label $\boldsymbol{y}$), a graph learning task is also formulated as an optimization problem:

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x})) \quad (3)$$

where Θ is a set of parameters we optimize and $\mathcal{L}$ is a loss function. f denotes graph neural network function which can be very complex. The output of f is predictions of nodes. Our goal is to make the loss which measures the gap between predictions and actual labels as lower as possible.

3.3. CARE-GNN

The key idea of CARE-GNN is to convert heterogeneous graph into homogeneous graph (i.e., aggregating features separately under each relation) [2]. At each layer, the input node embeddings (i.e., a batch of dataset) are fed through a MLP. Then, the label-aware similarity measure will be performed to calculate the l_1-distance between each central node in the minibatch and its neighbors:

$$D^{(k)}(v,u) = \left\| \sigma(mlp^{(k)}(h_v^{(k-1)})) - \sigma(mlp^{(k)}(h_u^{(k-1)})) \right\|_1 \quad (4)$$

where $D^{(k)}(v,u)$ denotes the l_1-distance between central node v and one of its neighbors u at k-th layer. The similarity between two nodes is defined as

$$S^{(k)}(v,u) = 1 - D^{(k)}(v,u) \quad (5)$$

where each layer has its own similarity measure module. A reinforcement learning (RL) module is designed to dynamically learn threshold, $p_r^{(k)} \in [0,1]$, which is used by similarity-aware neighbor selector to perform top-p sampling of neighbors for feature aggregate (see in [2] for more details on the threshold and top-p sampling). The ability of similarity measure module, discriminating whether the categories of neighbors are the same as that of central node, is in a dynamic state during the training. If the algorithm choose same nodes in every epoch, it will be difficult to effectively adapt to the state of network. If the average distance between a central node and its neighbors is decreasing, it means that similarity between the central node and its neighbors is increasing and there may be more homogeneous nodes. In this case, more neighbors can be selected to extract richer information to help model convince better of the identity of the central node (i.e., we increase the corresponding $p_r^{(k)}$). Conversely, it means that there are too many different nodes that have been selected to perform aggregate. Moreover, the aggregate operation will cover up the true characteristics of the central node. Therefore, we need to decrease the corresponding $p_r^{(k)}$. This process is done automatically by the RL module.

As CARE-GNN applies multi-relation aggregate, the node embedding at each layer is composed of two steps: First, the intra-relation aggregate is performed under each relation:

$$h_{v,r}^{(k)} = \sigma(AGG_r^{(k)}(\{h_u^{(k-1)}, u \in N(v), (u_i, v) \in E_r\})) \quad (6)$$

where $h_{v,r}^{(k)}$ denotes the embedding of node v after intra-relation aggregate under relation r and nonlinear transformation at k-th layer. We take mean aggregator as the intra-relation aggregator. When the first step is completed, we obtain the embeddings under all the relation $\{h_{v,1}^{(k)}, h_{v,2}^{(k)}, \cdots, h_{v,3}^{(k)}\}$. Then, we use the threshold $p_r^{(k)}$ as the weight of embedding under relation r to perform inter-relation aggregate:

$$h_v^{(k)} = \sigma(h_v^{(k-1)} + AGG^{(k)}(\{p_r^{(k)} \cdot h_{v,r}^{(k)}, r = 1, 2, \cdots, R\})) \quad (7)$$

where $h_v^{(k-1)}$ denotes the embedding of node v at previous layer.

4. Methodology

In the field of graph learning, designing the deep structure is always difficult. As is shown in Figure 6, CARE-GNN with single-layer architecture performs the best and suffers from the performance degradation as the number of layers increases. Based on CARE-GNN, we employ the layered graph neural network [7] architecture and residual structure [5] to make the following improvements.

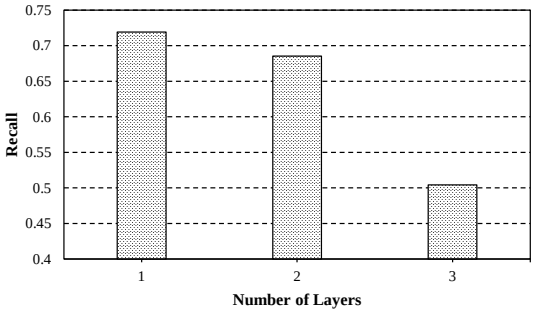


Figure 6. Recall of CARE-GNN with different number of layers.

First, we utilize layered architecture in Figure 7 to expend the original model to deep structure. The input to the model is a batch of central nodes and the original dataset. Each layer is an independent GNN, of which the input includes the original dataset and the node embeddings calculated by previous layer. The model is trained layer by layer, and each layer will correct mistakes made by previous layers. Intuitively, it indicates that each layer of GNN can focus on solving a simpler sub-problem. Moreover, the sub-problem is caused by the invalid or incorrect feature extracted by previous layers. Therefore, the layer-by-layer progressive learning process means the model is less reliant on a certain layer than the single-layer model, which allows every layer to make mistakes to some extent. Moreover, the mistakes will soon be continuously corrected by subsequent layers. Moreover, the fact that each layer inherits the results calculated by previous layer and its input includes the original graph enables each layer to make better choices on sampling neighbors. In other words, each layer is able to select a different and better set of neighbors based on the results calculated by previous layers. As a result, more nodes can be taken into account, and the neighborhood can be expended. The nodes' information will be richer and more accurate with the layer-by-layer training. As the depth increases, the model is able to "think" for more times, and each time the model can "think" more deeply and comprehensively based on the achievements of previous layer, thereby making better inferences.

We also take advantage of residual structure [5] to help the training process. The research points out that when adding multiple nonlinear layers to a shallow model, if the added layers can learn an identity mapping, which means the output equals to the input, then the performance of the model will at least not deteriorate. However, the fact is that it is

difficult for the complicated neural networks to fit a potential identity mapping. To reduce the learning difficulty for the networks, we let the model optimize the residual mapping instead of the original mapping. The added networks will learn a function $\mathcal{F}(G, \{W_i\})$ to fit a target mapping $\mathcal{H}(G)$, where G and $\{W_i\}$ are the input graph and a set of learnable parameters, respectively. As it is difficult for the networks to fit the identity mapping $\mathcal{H}(G) = G$, we turn it to fit its residual function $\mathcal{H}(G) - G$. Now, the function to be learned is $\mathcal{F}(G, \{W_i\}) = \mathcal{H}(G) - G$, and the target mapping is $\mathcal{F}(G, \{W_i\}) + G$. Therefore, the added nonlinear layers with residual structure still fits the desired underlying mapping. According to the form of the target mapping, we lead out a shortcut connection from input directly to the output. The work in [16] shows that adding shortcut connection for every GNN layer reaches the best result (see Figure 8).

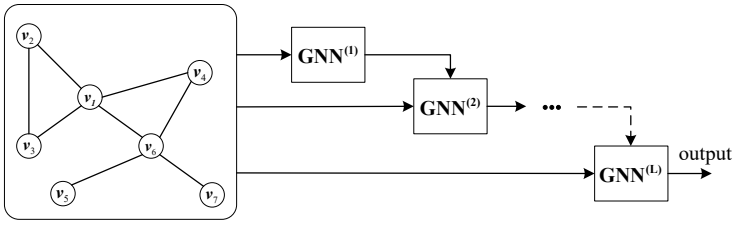


Figure 7. Layered architecture.

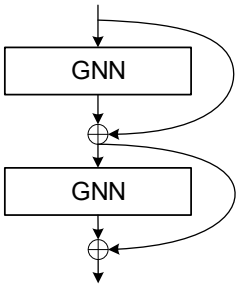


Figure 8. The residual structure for GNN.

Combining the layered structure and residual structure, we show details of a single layer of RLC-GNN in Figure 9. For each layer, the input of similarity measure module consists of two parts. One is the original dataset, and another one is updated node embeddings from previous layer except the first layer. More precisely, at the first layer, the similarity measure module will directly use a batch of dataset and its neighbors' node embeddings from dataset to do the measurement. After the first layer, the input is the node embeddings calculated by previous layer (i.e., output of previous layer).

For a specific node v, based on the discussion above, we now perform the similarity measure to a neighbor node u as follows:

$$D^{(k)}(v,u) = \left\| \sigma(mlp^{(k)}(o_v^{(k-1)})) - \sigma(mlp^{(k)}(h_u^{(G)})) \right\|_1 \tag{8}$$

where $o_v^{(k-1)}$ is the updated embedding of node v at previous layer $(k-1)$, $h_u^{(G)}$ is a neighbor node embedding that is directly from the dataset. Moreover, due to the existence of mpl, the dimension of output embeddings may change. The new node embedding at k-th (for $k > 1$) layer is

$$\begin{aligned} o_v^{(k)} &= \sigma(o_v^{(k-1)} + AGG^{(k)}(\{p_r^{(k)} \cdot h_{v,r}, r = 1, 2, \cdots, R\})) + o_v^{(k-1)} \\ &= GNN^{(k)}(o_v^{(k-1)}, G) + o_v^{(k-1)} \end{aligned} \tag{9}$$

where $h_{v,r}$ is the node embedding after intra-aggregate under relation r. In particular, we do not adopt shortcut connection at the first layer. Moreover, the way to calculate node embedding at the first layer is

$$\begin{aligned} {o_v}^{(1)} &= \sigma(h_v^{(G)} + AGG^{(1)}(\{p_r^{(1)} \cdot h_{v,r}, r = 1, 2, \cdots, R\})) \\ &= GNN^{(1)}(h_v^{(G)}, G) \end{aligned} \tag{10}$$

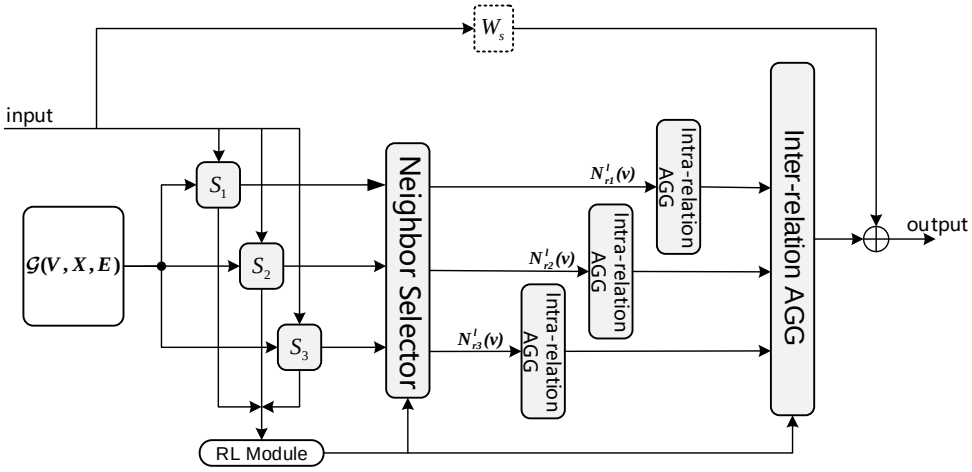


Figure 9. Details of a layer of RLC-GNN. W_s is a trainable weight matrix to match the embedding's dimensions between the input nodes and output nodes of the current layer. S_i denotes the similarity measure module for the i-th relation.

Now we give the expression of output at k-th layer:

$$\boldsymbol{o}^{(k)} = GNN^{(k)}(\boldsymbol{o}^{(k-1)}, G) + \boldsymbol{o}^{(k-1)} \tag{11}$$

where $\boldsymbol{o}^{(k)}$ is the new node embeddings of the batch $(o_0^{(k)}, o_1^{(k)}, \cdots, o_{n-1}^{(k)})$. We have mentioned that the dimension of input must equal the dimension of output. If it is not the case, we perform a linear projection with a trainable weight matrix W_s [5]:

$$\boldsymbol{o}^{(k)} = GNN^{(k)}(\boldsymbol{o}^{(k-1)}, G) + W_s\boldsymbol{o}^{(k-1)} \tag{12}$$

The overall structure of RLC-GNN is shown in Figure 10. At each iteration, the input of model is a batch of nodes with features and the entire graph. The input of each layer includes the output from the previous layer and the entire graph. The output of each layer is summary of the new features extracted by the current layer from the neighbors in original graph and the input from shortcut connection. In the training process, final output loss is given by

$$\mathcal{L}_\Sigma = \sum_{k=1}^{L} \mathcal{L}_{Simi}^{(k)} + \mathcal{L}_{GNN} \tag{13}$$

where $\mathcal{L}_\Sigma$ is final output loss which is obtained by adding up two parts. $\mathcal{L}_{Simi}^{(k)}$ is loss of scores which is used by the similarity measure module at k-th layers to do the label-aware similarity measure. Moreover, $\mathcal{L}_{GNN}$ is loss of the classifier that predicts labels of nodes. Both losses are calculated by using cross-entropy:

$$\mathcal{L} = -\sum_{v \in V_{batch}} y_v \cdot \log(\sigma(mpl(h_v))) \tag{14}$$

where y_v is actual node label of node v.

We show the pseudocode of the proposed RLC-GNN in Algorithm 1. Given a heterogeneous graph, at each iteration, we initialize thresholds of neighbor selector with manually specified values and randomly select a batch of nodes with features as input for the first layer. For all subsequent layers, the input of each layer is the updated embeddings generated by previous layer, which is the result of layered structure. We first do the similarity measure, top-p sampling, and intra-relation aggregate under each relation as is shown in Equations (5) and (6). This step determines the output dimension of current layer. Then, we apply interrelation aggregate (Equation (7)) and add up the result and input (Equation (11) or Equation (12), depending on whether input and output dimensions match). Here, we get updated node embeddings. Moreover, then we calculate losses of similarity measure modules and RLC-GNN (Equations (13) and (14)), and do backpropagation to update trainable parameters. At last, we use the reinforcement learning modules to update selector thresholds.

Algorithm 1: RLC-GNN

Input: Graph with features: $\mathcal{G} = \{V, X, E\}$; Number of layers, epochs, batches and relationships: L, E, B, R; Similarity measures: $\{S^{(k)}\}|_{k=1}^{L}$; Selector thresholds: $\{p_1^{(k)}, p_2^{(k)}, ..., p_R^{(k)}\}|_{k=1}^{L}$; Inter-relation AGGs: $\{AGG^k\}|_{k=1}^{L}$; Intra-relation AGGs: $\{AGG_1^k, AGG_2^k, ..., AGG_R^k\}|_{k=1}^{L}$

Output: Nodes embeddings after last layer: $\boldsymbol{o}_v, v \in V$

initialization;
for $e = 1$ *to* E **do**
  for $b = 1$ *to* B **do**
    for $k = 1$ *to* L **do**
      if $k = 1$ **then** `/* Layered structure */`
        $(h_v^{(1)}, h_u) \leftarrow (x_v, x_u), x \in X, v \in V_b, u \in V$;
      else
        $(h_v^{(k)}, h_u) \leftarrow (o_v^{(k-1)}, x_u), v \in V_b, u \in V$;
      end
      for $r = 1$ *to* R **do**
        $D^{(k)} \leftarrow$ Equation (8), $v \in V_b$;
        $S^{(k)} \leftarrow$ Equation (5), $v \in V_b$; `/* Similarity measure */`
        $X_{v,r}^{(k)} \leftarrow$ top-p sampling ; `/* Select neighbors */`
        $h_{v,r}^{(k)} \leftarrow$ Equation (6), $v \in V_b$; `/* Intra-relation aggregate */`
      end
      $h_v^{(k)} \leftarrow$ Equation (7), $v \in V_b$; `/* Inter-relation aggregate */`
      if $d_i = d_o$ **then** `/* Residual structure */`
        $o^{(k)} \leftarrow$ Equation (11), $v \in V_b$; `/* Dimensions match */`
      else
        $o^{(k)} \leftarrow$ Equation (12), $v \in V_b$; `/* Dimensions not match */`
      end
      $\mathcal{L}_{Simi}^{(k)} \leftarrow$ Equation (14), $v \in V_b$;
    end
    $\mathcal{L}_{GNN} \leftarrow$ Equation (14), $v \in V_b$;
    $\mathcal{L}_{\Sigma} \leftarrow$ Equation (13), $v \in V_b$;
    for $k = 1$ *to* L **do**
      for $r = 1$ *to* R **do**
        update $p_r^{(k)}$; `/* See [2] Section 3.3 for more details */`
      end
    end
  end
end

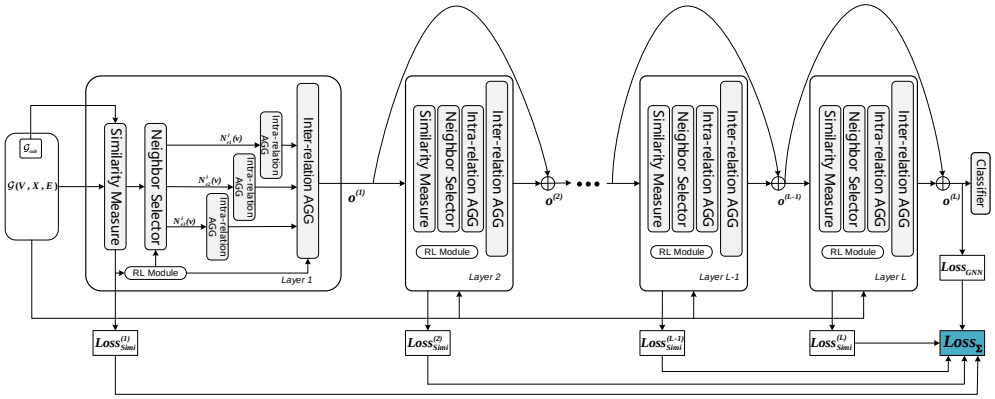


Figure 10. Proposed RLC-GNN architecture.

When the node embeddings are input to a layer, there may be the case that the current layer has not been properly trained and the benign samples cannot be effectively filtered out during the similarity measure, which results in the features of fraudsters being covered up by benign samples' features. After the introduction of residual structure to the model, if the current layer causes an adverse effect on the classification, the input embeddings passing through the shortcut connection will reduce the losses during the propagation. Therefore, RLC-GNN has the ability to skip the layers which have not been trained well and perform the rollback of embeddings (i.e., we avoid "bad" layers blocking normal training of subsequent layers). The model will have opportunities to further learning the characteristics of fraudsters based on the knowledge that has been acquired by previous layers. Thus, as the learning process progresses layer by layer, more information will be taken into account and better selection will be made.

In Figure 11, we show our implementation for RLC-GNN with various number of layers. If dimensions of input and output not match, we adopt a linear projection W_s to match the dimensions. Moreover, we show details of the architecture for RLC-GNN with various number of layers in Table 1.

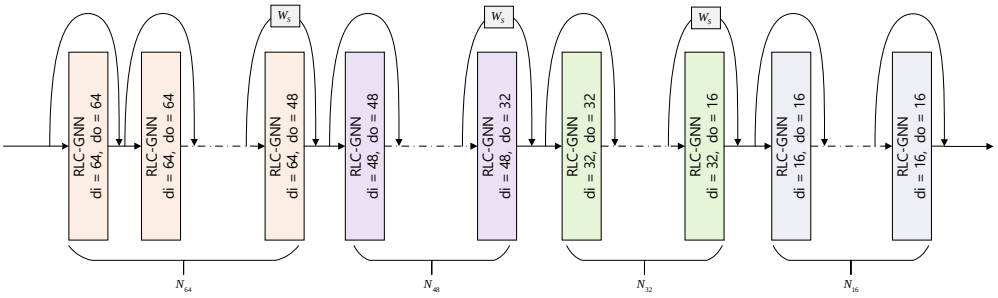


Figure 11. Our implementation for RLC-GNN architecture in experiments. d_i and d_o denote dimensions of input and output, respectively. See Table 1 for details of architectures.

Table 1. Implementations for RLC-GNN with various number of layers. N_d denotes the number of layers whose dimension of input node embeddings is d.

Parameters	4-Layers	6-Layers	11-Layers	19-Layers	27-Layers
N_{64}	1	2	3	3	5
N_{48}	0	0	3	5	7
N_{32}	1	2	3	6	10
N_{16}	2	2	2	5	5

5. Experiments

5.1. Datasets

In order to ensure the comparability of the experimental results, we conduct experiments on the Yelp Dataset and Amazon Dataset provided in research [2]. The Yelp Dataset is the public internal dataset of Yelp, the largest review site in USA, and covers business, reviews, user information, and so on. In our experiment, we use the reviews to build the graph which includes 45,954 nodes (14.5% are fraud reviews) and 3,846,979 edges. Amazon Dataset is an open source dataset created by Amazon platform and it includes more than 140 million reviews and product metadata under 24 product categories. We use the reviews under musical instruments and take the users as nodes of the graph and the graph includes 11,944 nodes (9.5% are fraudsters) and 4,398,392 edges.

There are three types of relationships between the nodes of each dataset. Yelp Dataset: (1) R-U-R: two reviews are posted by the same user; (2) R-S-R: two reviews are under the same product with the same star rating; (3) R-T-R: two reviews under the same product are posted in the same month. Amazon Dataset: (1) U-P-U: It connects two users who have reviewed at least one same product; (2) U-S-U: It connects two users who have rated the same star within a week; (3) U-V-U: It connects two users of whom 5% reviews are similar.

5.2. Implementation

In the experiment, we use Pytorch 1.7.0 to implement RLC-GNN and use cross-entropy as the loss function. We choose an Adam optimization algorithm and set the learning rate to 0.01. Each dataset is divided into two parts: 40% as the training set and 60% as the test set. We utilize the mini-batching training skills to improve the training efficiency [22]. We verify the performance of RLC-GNN with 4, 6, 11, 19, and 27 layers under both datasets. The final dimension of node embeddings is 16. All experiments are running on Python 3.7.6, Windows 10 OS, AMD Ryzen 7 4800H CPU, 16GB RAM, Nvidia RTX 2060 GPU.

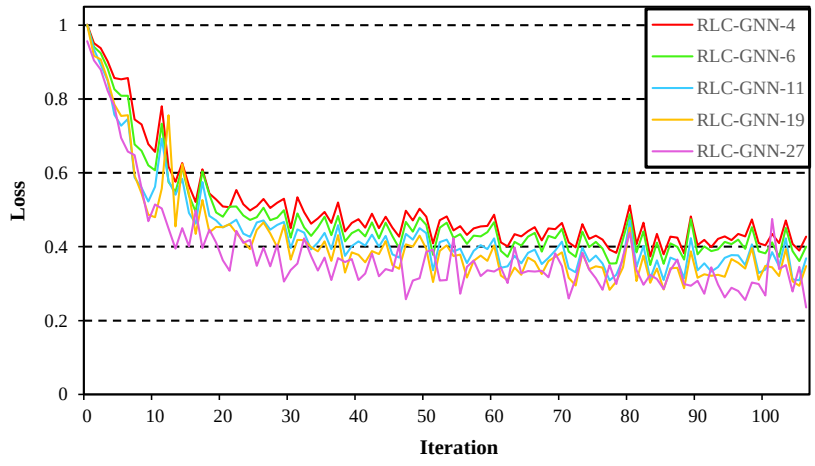
5.3. Evaluation Metrics

For the fraud detection task, we concern about the model's capability to correctly identify the fraud samples. Therefore, we use recall as one of our metrics. However, if model is not learning effectively and simply predicts all samples as frauds, we also get high recall. To avoid the confusing results, meanwhile, we consider the ratio of correct predictions in all predictions as frauds. We use F1-score as one of metrics. Furthermore, due to the extremely imbalance of sample distribution (ratio of fraud samples to benign samples is about 1 to 9), we use AUC (insensitive to distribution of samples) as the third metric to evaluate our model more fairly [23].

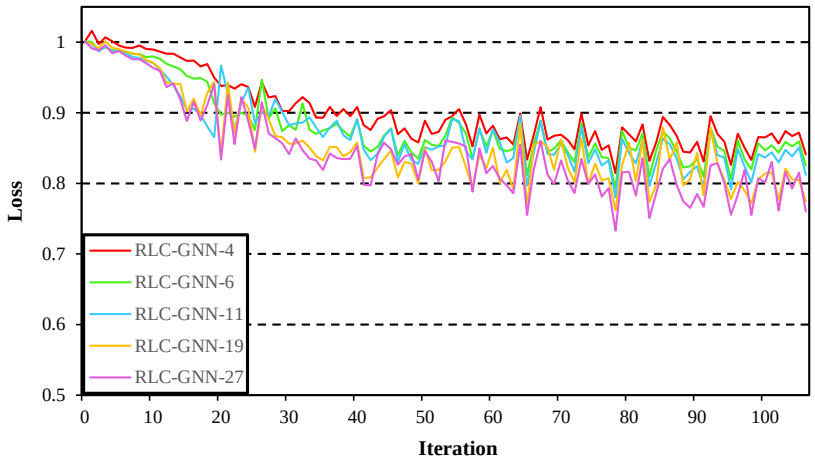
5.4. Results

First, we show the normalized training loss for RLC-GNN with 4, 6, 11, 19, and 27 layers in Figure 12. As the number of layers increases, the overall normalized training loss goes down. As we have discussed above, the final loss, $Loss_{\Sigma}$, consists of the loss of similarity measure modules from every layer. In other words, $Loss_{\Sigma}$ contains more items as the number of layer increases. However, we note that more layers leads to greater loss reduction ratio on the contrary. Tendency of training loss of each layer in Figure 13 shows that latter layer can make better inferences based on inherited knowledge of previous layers, which shows the effectiveness of proposed method on dealing with the single-layer learning problem. Moreover, we notice that the second layer makes the greatest improvement. Although the latter layers do not obtain as much improvement as the second layer, they indeed perform better layer by layer. Here we give the empirical analysis. As the number of layers increases, the problems to be solved for each layer become simpler. For instance, the single-layer model has to solve the entire problem alone. There is no other layers to share its learning pressure and to correct its mistakes. When we add more layers, each layer only needs to deal with partial problems (i.e., problems become easier for every layer). The remaining problems and the mistakes made by previous layers will be solved

and corrected by subsequent layers. More precisely, because of the input of each layer consisting of original dataset, aggregated information and progress already made, each layer has sufficient information to judge the correctness of inferences (i.e., giving higher confidence to correct inferences, and correcting the wrongs). Furthermore, if a layer has not yet been trained well, input features can skip the layer by passing through shortcut connection. The training will not be interrupted, and the problem will be directly handed over to the next layer for processing. With such cooperation mechanism, the whole problem can be solved more smoothly.



(**a**)



(**b**)

Figure 12. The normalized training loss for RLC-GNN with varying depth on Amazon dataset (**a**) and Yelp dataset (**b**). As it can be seen, an obvious characteristic on both dataset is that the overall training loss is lower with the increasing of layers.

We show the performance of RLC-GNN and various GNNs on the fraud detection tasks on Yelp and Amazon datasets in Table 2. GCN, GAT, GraphSAGE, and GeniePath are designed to run on homogeneous graphs. Multiple relations are merged into a single relation (i.e., heterogeneous to homogeneous) in the experiments of these GNNs [2]. Compared with the single-relation GNNs, we can see that multi-relation GNNs have great advantages on tasks based on heterogeneous graphs. Furthermore, it might point out a direction for future development of GNNs. Moreover, based on this superiority, proposed RLC-GNN introduces the mechanism of progressive-learning and self-correcting, which makes the use of neighboring information as more effective as possible and once again achieves significant improvements.

Table 3 shows the experiment results of RLC-GNN with various depth. According to the experimental results, RLC-GNN outperforms CARE-GNN significantly, especially on the more complex Yelp dataset. On the Yelp dataset, the model with 27 layers achieves the best performance in our experiments, and recall, AUC, and F1-score increase by 5.66%, 7.72%, and 8.90%, respectively. On the Amazon dataset, when RLC-GNN has 11 layers, overall results outperform other settings, which Recall, AUC and F1 increase by 3.22%, 4.05%, and 3.25%, respectively.

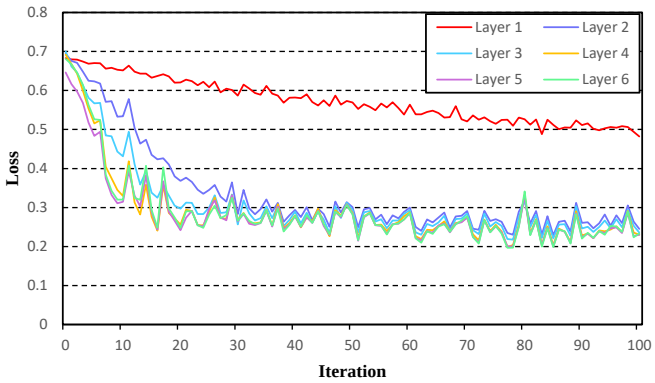


Figure 13. Training loss of each layer of RLC-GNN-6 on Amazon dataset. We note that training loss of later layer is lower, which means that the later layers can make inferences with higher confidence.

Table 2. Performance of RLC-GNN-27 and various GNNs on Yelp and Amazon datasets.

Model	Yelp		Amazon	
	AUC	Recall	AUC	Recall
GCN	54.47%	50.81%	74.34%	67.45%
GAT	56.24%	54.52%	75.16%	65.61%
GraphSAGE	54.00%	52.86%	75.27%	70.16%
GeniePath	55.91%	50.94%	72.65%	65.41%
GraphConsis	62.07%	62.08%	85.46%	85.53%
CARE-GNN	77.72%	71.02%	93.21%	88.17%
RLC-GNN-27	85.44%	76.68%	97.48%	91.83%

The original feature dimension size of Yelp dataset and Amazon dataset are 32 and 25, respectively. Generally, the more complex the features are, the more complex model we will need to fit data. Under current hyperparameter settings, the experiment results show that recall and AUC nearly grow all the time on both datasets, which means the model can identify more fraudsters with the number of layer increasing. We also notice different degrees of decline in F1-score. Usually, large networks have better ability of generalization than small networks' [24]. However, when a model is too complicated relative to the dataset, it will suffer from the overfitting, which reduces the generalization of the model [25]. In this case, the model needs more data to be trained sufficiently. Moreover, we argue that our deep models are facing this problem. As a consequence, there is slightly decrement in the performance of F1-score when the depth increases to certain extent.

We notice that our focus is the problem faced by classic deep learning models instead of the GNN-specific over-smooth problem that leads to shallow limit. In other words, we successfully deal with the shallow structure limit to GNNs with the application of fraud detection. Furthermore, note that the same hyperparameters (e.g., learning rate and weight decay for optimizer) are used to train all the models and no optimization is made for models with various depth on different dataset. If we carefully adjust the hyperparameters, we may reach better results.

Table 3. Results of RLC-GNN with various number of layers and the baseline. We report results after running 100 epochs.

Dataset	Model	Recall	AUC	F1-Score
Yelp	CARE-GNN	71.02%	77.72%	61.13%
	RLC-GNN-4	74.20%(+3.18%)	81.39%(+3.67%)	66.09%(+4.96%)
	RLC-GNN-6	74.66%(+3.64%)	83.29%(+5.57%)	68.45%(+7.32%)
	RLC-GNN-11	74.72%(+3.70%)	83.90%(+6.18%)	70.36%(+9.23%)
	RLC-GNN-19	76.03%(+5.01%)	85.13%(+7.41%)	70.22%(+9.09%)
	RLC-GNN-27	76.68%(+5.66%)	85.44%(+7.72%)	70.03%(+8.90%)
Amazon	CARE-GNN	88.17%	93.21%	87.81%
	RLC-GNN-4	89.43%(+1.26%)	95.53%(+2.32%)	89.03%(+1.22%)
	RLC-GNN-6	89.83%(+1.66%)	96.77%(+3.56%)	90.08%(+2.27%)
	RLC-GNN-11	91.39%(+3.22%)	97.26%(+4.05%)	91.06%(+3.25%)
	RLC-GNN-19	91.16%(+2.99%)	97.33%(+4.12%)	90.46%(+2.65%)
	RLC-GNN-27	91.83%(+3.66%)	97.48%(+4.27%)	89.18%(+1.37%)

6. Conclusions

This work proposes RLC-GNN algorithm, an improved spatial-based GNN algorithm that could be trained with deep architecture. We utilized a layered structure to deal with the single-layer learning problem and introduced the concept of residual network to complement the layered structure to assist training, which forms a type of cooperation and enables the model to be much deeper. Therefore, we can enjoy the benefits of depth without being trapped by the intrinsic shallow limit of graph neural networks. The experiments on fraud detection of Yelp dataset and Amazon dataset show that the proposed RLC-GNN algorithm obtains significant improvements under three metrics, recall, AUC, and F1 score, which could be further improved within some extend as the number of layers increases.

We verified the effectiveness of the proposed algorithm with the application of fraud detection. In future research, we will extend experiments to more application domains of graph neural network, and we will further explore widely applied techniques to deal with overfitting problem faced by deep RLC-GNN. Moreover, we have indicated some intuitive reasons for the combined effect of the layered structure and residual structure. The theoretical analysis will be explored in the future research.

Author Contributions: Conceptualization, Y.Z. and J.T.; investigation, Y.Z.; validation, J.T.; methodology, Y.Z.; software, Y.Z.; formal analysis, Y.Z.; supervision, J.T.; writing—original draft preparation, Y.Z.; writing—review and editing, J.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bronstein, M.M.; Bruna, J.; LeCun, Y.; Szlam, A.; Vandergheynst, P. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Process. Mag.* **2017**, *34*, 18–42. [CrossRef] [CrossRef]
2. Dou, Y.; Liu, Z.; Sun, L.; Deng, Y.; Peng, H.; Yu, P.S. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In Proceedings of the 29th ACM International Conference on Informationand Knowledge Management (CIKM), Online, 19–23 October 2020; pp. 315–324. [CrossRef]
3. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. In Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, 7–9 May 2015. Available online: https://arxiv.org/pdf/1409.1556 (accessed on 27 September 2020).
4. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9. Available online: https://arxiv.org/pdf/1409.4842 (accessed on 9 October 2020).

5. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778. [CrossRef]
6. Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; Sun, M. Graph Neural Networks: A Review of Methods and Applications. *AI Open* **2020**, *1*, 57–81. [CrossRef]
7. Bandinelli, N.; Bianchini, M.; Scarselli, F. Learning long-term dependencies using layered graph neural networks. In Proceedings of the 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, Spain, 18–23 July 2010; pp. 1–8. [CrossRef]
8. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**. [CrossRef] [CrossRef] [PubMed]
9. Bruna, J.; Zaremba, W.; Szlam, A.; LeCun, Y. Spectral networks and locally connected networks on graphs. In Proceedings of the 2nd International Conference on Learning Representations (ICLR 2014), Banff, AB, Canada, 14–16 April 2014; Bengio, Y., LeCun, Y., Eds.; Conference Track Proceedings. 2014. Available online: https://arxiv.org/pdf/1312.6203 (accessed on 16 October 2020).
10. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [CrossRef] [CrossRef] [PubMed]
11. Liu, Z.; Dou, Y.; Yu, P.S.; Deng, Y.; Peng, H. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), China, 25–30 July 2020; pp. 1569–1572. [CrossRef]
12. Liu, Z.; Chen, C.; Yang, X.; Zhou, J.; Li, X.; Song, L. Heterogeneous graph neural networks for malicious account detection. In Proceedings of the 27th ACM International Conference on Informationand Knowledge Management (CIKM 2018), Torino, Italy, 22–26 October 2018; pp. 2077–2085. [CrossRef]
13. Li, A.; Qin, Z.; Liu, R.; Yang, Y.; Li, D. Spam review detection with graph convolutional networks. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM), Beijing, China, 3–7 November 2019; pp. 2703–2711. Available online: https://arxiv.org/pdf/1908.10679 (accessed on 14 October 2020).
14. Wen, R.; Wang, J.; Wu, C.; Xiong, J. ASA: Adversary situation awareness via heterogeneous graph convolutional networks. In *Companion Proceedings of the Web Conference*; ACM: New York, NY, USA, 2020; pp. 674–678. [CrossRef]
15. Li, G.; Muller, M.; Thabet, A.; Ghanem, B. DeepGCNs: Can GCNs go as deep as CNNs? In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea, 27 October–2 November 2019. Available online: https://arxiv.org/pdf/1904.03751 (accessed on 19 October 2020).
16. Li, G.; Müller, M.; Qian, G.; Delgadillo, I.C.; Abualshour, A.; Thabet, A.K.; Ghanem, B. DeepGCNs: Making GCNs Go as Deep as CNNs. *arXiv* **2019**, arXiv:1910.06849.
17. Zhou, K.; Dong, Y.; Wang, K.; Lee, W.S.; Hooi, B.; Xu, H.; Feng, J. Understanding and Resolving Performance Degradation in Graph Convolutional Networks. *arXiv* **2020**, arXiv:2006.07107.
18. Hu, F.; Zhu, Y.; Wu, S.; Wang, L.; Tan, T. Hierarchical graph convolutional networks for semi-supervised node classification. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, 10–16 August 2019; pp. 4532–4539. Available online: https://arxiv.org/pdf/1902.06667 (accessed on 4 November 2020).
19. Zachary, W.W. An information flow model for conflict and fission in small groups. *J. Anthropol. Res.* **1977**, *33*, 452–473. [CrossRef]
20. Hamilton, W.L.; Ying, R.; Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv* **2017**, arXiv:1709.05584.
21. Hamilton, W.L.; Ying, R.; Leskovec, J. Inductive Representation learning on large graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, USA, 4–9 December 2017; pp. 1025–1035. Available online: https://arxiv.org/pdf/1706.02216 (accessed on 9 November 2020).
22. Goyal, P.; Dollár, P.; Girshick, R.B.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv* **2017**, arXiv:1706.02677.
23. Powers, D.M.W. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *arXiv* **2020**, arXiv:2010.16061.
24. Caruana, R.; Lawrence, S.; Giles, L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In Proceedings of the 13th International Conference on Neural Information Processing Systems (NIPS), 27 November–2 December 2000; pp. 381–387. Available online: http://papers.nips.cc/paper/1895-overfitting-in-neural-nets-backpropagation-conjugate-gradient-and-early-stopping.pdf (accessed on 30 October 2020).
25. Ying, X. An Overview of Overfitting and its Solutions. *J. Phys. Conf. Ser.* **2019**, *1168*, 022022. [CrossRef] [CrossRef]