Article

GR-GNN: Gated Recursion-Based Graph Neural Network Algorithm

Kao Ge 1, Jian-Qiang Zhao 2 and Yan-Yong Zhao 3,*

1 Nanjing Institute of Software Technology, Institute of Software, Chinese Academy of Sciences, Nanjing 211135, China; gekao@nj.iscas.ac.cn
2 School of Mathematics and Statistics, Xuzhou University of Technology, Xuzhou 221018, China; zjqghh@xzit.edu.cn
3 Department of Statistics, Nanjing Audit University, Nanjing 211815, China
* Correspondence: yyzhao@nau.edu.cn; Tel.: +86-025-5831-8699

Abstract: Under an internet background involving artificial intelligence and big data—unstructured, materialized, network graph-structured data, such as social networks, knowledge graphs, and compound molecules, have gradually entered into various specific business scenarios. One problem that urgently needs to be solved in the industry involves how to perform feature extractions, transformations, and operations in graph-structured data to solve downstream tasks, such as node classifications and graph classifications in actual business scenarios. Therefore, this paper proposes a gated recursion-based graph neural network (GR-GNN) algorithm to solve tasks such as node depth-dependent feature extractions and node classifications for graph-structured data. The GRU neural network unit was used to complete the node classification task and, thereby, construct the GR-GNN model. In order to verify the accuracy, effectiveness, and superiority of the algorithm on the open datasets Cora, CiteseerX, and PubMed, the algorithm was used to compare the operation results with the classical graph neural network baseline algorithms GCN, GAT, and GraphSAGE, respectively. The experimental results show that, on the validation set, the accuracy and target loss of the GR-GNN algorithm are better than or equal to other baseline algorithms; in terms of algorithm convergence speed, the performance of the GR-GNN algorithm is comparable to that of the GCN algorithm, which is higher than other algorithms. The research results show that the GR-GNN algorithm proposed in this paper has high accuracy and computational efficiency, and very wide application significance.

Keywords: GR-GNN; graph neural network; bias random walks; GRU

MSC: 03D32; 68T05; 92B20

1. Introduction

Deep learning, under an internet background of artificial intelligence and big data, has achieved remarkable achievements in many fields, including computer vision, natural language processing, and speech recognition. The deep learning algorithm automatically extracts high-dimensional deep features from the input image information, text information, voice information, and other data by establishing a neural network. These deep features are used for downstream classification and prediction tasks via encoding and decoding layers, convolution layers, pooling layers, and activation function layers. Deep learning, compared with traditional machine learning algorithms, has significantly improved in automatic feature extraction, identification of information dimensions, and information quality, and it can generate high-dimensional abstract feature representations with stronger representation learning capabilities. However, researchers gradually discovered that deep learning cannot solve and adapt to the data in all scenarios. This is because, in the fields of computer vision, natural language processing, speech recognition, etc., images, text, voices,
and videos all have the same data characteristics and dimensions in data format; these kinds of data are called the Euclidean structure. In actual scenarios and businesses, there are graph data with non-European structures, such as social networks, knowledge graphs, proteins, compound molecules, the internet, etc. In these fields, deep learning algorithm performances are not very ideal, mainly because, in graph-structured data, each node has its own attribute characteristics, and the dependencies between nodes are complex and diverse. For example, graphs can be divided into directed graphs and undirected graphs according to whether the edges between graph nodes have directions; according to whether there is weight between graph nodes, they can be divided into weighted graphs and unweighted graphs, and so on. In the mathematical expressions of these graph-structured data, the dimensions of the feature matrix of each region are not uniform, as shown in Figure 1. As a result, neural networks such as CNN can no longer directly perform operations, such as convolution and pooling, on their graph-structured data, and models constructed from graph-structured data can no longer have local connections, weight sharing, and feature abstraction [1].

**Figure 1.** Feature extraction of graph structure data.

Therefore, how to apply deep learning algorithms to analyze graph-structured data has become challenging.

### 1.1. Research Status of GNN

In recent years, Gori et al. [2] first proposed the concept of a graph neural network (GNN), which uses RNN units to compress node and label information in graph-structured data; Bruna et al. [3] proposed a graph convolutional network (GCN), which formally uses CNN to process graph-structured data. The GCN network integrates the structural information of the source node and neighbor nodes through the Laplace matrix, and gives the fixed-dimensional feature vector of each node in the graph-structured data. Then the feature vector is input into the CNN neural network, which is applied to downstream tasks, such as node classification and link prediction. The GCN network algorithm provides annotations for analyzing graph-structured data using deep learning algorithms, after which various GNN variants are constructed one after another. The main variants of graph neural networks and related presentations are shown in Table 1.
Table 1. Main variants and introduction of graph neural networks.

| Graph Neural Network Classification | Improved Variant Algorithm | Research Content |
|-----------------------------------|----------------------------|------------------|
| Network embedding [4]             | DeepWalk [5], LINE [6], Node2Vec [7]. | These algorithms mainly learn fixed-length feature representations for each node by integrating the graph structure information of source nodes and neighbor nodes. |
|                                   | Spectral decomposition graph convolution: Chebyshev [8], CayleyNet [9], AGCN [10]. | The algorithm mainly performs feature extraction by convolving the eigenvectors of the Laplacian matrix of the graph. |
| Graph convolution network         | Spatial graph convolution: PATCHY-SAN [11], GraphSAGE [12], FastGCN [13]. | The algorithm mainly performs feature extraction using convolution operations on the structural information of the source node and neighbor nodes. |
| Graph auto-encoder (GAE)          | MGAE [14], AutoGCN [15]. | The algorithm mainly designs a stack denoising autoencoder to extract features from the structural information generated by the random walk model. |
| Graph generative network (GGN) [16] | Graph-GAN [17], NetGAN [18]. | The algorithm mainly uses the Bayesian network to learn the dependency structure between random node variables and uses the expectation propagation algorithm to train the generator and discriminator. |
| Graph recurrent network (GRN) [19] | GGT-NN [20], GraphRNN [21]. | The GRN algorithm is one of the earliest GNN models. It generally uses bidirectional RNN (Bi-RNN) and the long short-term memory network (LSTM) as the network structure and feature extraction. |
| Graph attention network (GAT)     | GATE [22], RGAT [23], GAM [24]. | The algorithm builds a multi-head attention mechanism on the basis of spatial graph convolution, and uses it to generate a feature aggregation function of a stacked spatial graph convolution for feature extraction. |

By researching the above papers, we found that: whether it is a graph neural network algorithm that solves tasks such as node classification and link prediction of graph-structured data, or a graph neural network algorithm that solves tasks such as text reasoning, shortest paths, and Euler cycles, the focus is on the feature fusion and extraction of the source node and its first-order and second-order neighbor nodes, but none of the deep chain-dependent features of the source node are extracted. For example, regarding a knowledge graph, such as an open source software supply chain, the quality of the underlying software open source framework and algorithm directly or indirectly affect the security and reliability of integrated software services, such as software product components and software product platforms. The impact is cumulative as the degree of software integration increases.
1.2. Related Work

This paper proposes a gated recursive graph neural network (GR-GNN) algorithm to solve the problems of extracting deep-dependent features between nodes and node classification. The algorithm mainly uses the biased random walk algorithm (bias random walk) in NodeVec [13], the degree matrix, and the adjacency matrix of the graph structure data to construct a deep chain adjacency matrix (deep chain adjacency matrix, DCAM). Through this deep chain adjacency matrix (DCAM), the deep chain-dependent features of the source node and its neighbor nodes are fused, and the feature representation of the source node is regenerated on the spatial structure. This chain-based dependency feature extraction is suitable for processing graph-structured data with strong entity node dependencies, such as open-source software supply chain knowledge graphs. In addition, the GRU neural network unit similar to the literature [19] is used in feature calculations.

This paper is structured as follows. The Section 1 presents the research background, research status, and research route of the paper. The Section 2 explains the concept definitions and technical theories involved in the algorithm proposed in this paper. The Section 3 demonstrates the theoretical derivation process of the GR-GNN algorithm. The Section 4 involves testing the GR-GNN algorithm proposed in this paper to verify its accuracy and superiority. This experiment mainly uses graph neural network algorithms, such as GR-GNN, GCN, GraphSAGE, and GAT on public datasets, such as Cora, PubMed, and CiteseerX to perform performance comparison tests on node classification tasks. The Section 5 presents the conclusion and future research directions.

2. Overview

This section provides definitions of key concepts and the related technical theory.

2.1. Symbols and Definitions

**Definition 1.** The graph is composed of nodes (vertex) and edges (edge) connected between nodes, denoted as \( G = (V, E, A, X, D, M) \), where \( V \) represents the set of nodes in the graph network structure, \( E \) represents the set of edges in the graph network structure, \( A^{N \times N} \) represents the adjacency matrix in the graph network structure, \( X^{N \times F} \) represents the feature matrix of nodes in the graph network structure, \( D^{N \times N} \) represents the degree matrix of the graph network structure, \( M^{N \times N} \) represents the random adjacency matrix generated by the nodes in the graph network structure according to the bias random walk strategy, \( N \) and \( F \) represent the number of nodes and the feature dimension of the nodes, respectively.

**Definition 2.** Deep chain adjacency matrix (DCAM), which is composed of the degree matrix, adjacency matrix, random adjacency matrix, and parameters \( \beta \):

\[
DCAM = D - \left[ A + \beta \ast (M - A) \right]
\]  

(1)

The matrix has non-zero values at the center node, first-order adjacency nodes, and depth random walk nodes, and the rest are zero.

**Definition 3.** Normalized depth chained adjacency matrix (N_DCAM):

\[
DCAM_{ij}' = \frac{DCAM_{ij}}{\max(DCAM_{ij}) - \min(DCAM_{ij})}
\]

(2)

where \( \max(\cdot) \) and \( \min(\cdot) \) represent the maximum and minimum values of the column.

2.2. Node2Vec Algorithm

The Node2Vec algorithm [7] was proposed by Grover A in 2016. It is a representation learning algorithm that maps each node in the network graph structure to a low-dimensional space through unsupervised learning. This algorithm improves the random walk algorithm in Deepwalk [5], introduces a weighted random walk strategy, forms a
series of linear sequences, and inputs these linear sequences into the Skip-gram model [25] for training word vector, which can better take into account the homogeneity and structural similarity of nodes.

According to Definition 1, the goal of the Node2Vec algorithm is to learn a map \( f : V \to \mathbb{R}^d \) (where \( F \ll N \)) that maps the structural features of each node to a feature representation \( \vec{w}_v \) in a low-dimensional space. The feature representation of this low-dimensional space can be used for downstream classification or prediction tasks. This paper does not directly use the final feature expression \( \vec{w}_v \) of the Node2Vec algorithm, but adopts the biased random walk sampling strategy in the algorithm to optimize the spatial structure characteristics of nodes, to improve the depth dependence characteristics between nodes.

For each node \( v \), its graph-structured neighbor is \( N_S(v) \subset V \) collected with a biased random sampling strategy \( S \). Similar to the Skip-gram algorithm, the optimization function of Node2Vec is: given a node \( v \), the maximum log-likelihood function of its neighbor node \( N_S(v) \) in the low-dimensional space, namely:

\[
\text{max}_f \sum_{v \in V} \log(P(N_S(v)|\vec{w}_v))
\]

where \( \vec{w}_v = f(v) \in \mathbb{R}^d \).

To solve the optimization problem of this objective, the Node2Vec algorithm makes two key assumptions:

1. **Conditional independence assumption**: given a node \( v \), any two nodes \( v_1 \) and \( v_2 \) in its neighbor node \( N_S \), \( v_1 \) and \( v \) becoming neighbors have nothing to do with \( v_2 \) and \( v \) becoming neighbors; that is:

\[
P(N_S(v)|\vec{w}_v) = \prod_{v_i \in N_S(v)} P(\vec{w}_{v_i}|\vec{w}_v)
\]

2. **Spatial symmetry**: given a node \( v \), it shares the same representation vector as a source node and as a neighbor node, so the conditional probability is constructed by the Softmax function:

\[
P(\vec{w}_{v_i}|\vec{w}_v) = \frac{\exp(\vec{\omega}_{v_i} \cdot \vec{w}_v)}{\sum_{\vec{v}' \in V} \exp(\vec{\omega}_{v'} \cdot \vec{w}_v)}
\]

Therefore, the objective function of Node2Vec is:

\[
J = \sum_{v \in V} [-\log Z_v + \sum_{v_i \in N_S(v)} \vec{\omega}_{v_i} \cdot \vec{w}_v]
\]

where \( Z_v = \sum_{v' \in V} \exp(\vec{\omega}_{v'} \cdot \vec{w}_v), i \in [0, \text{len}(N_S(v))] \).

For the sampling strategy \( S \), since the text is linear, the Skip-gram algorithm obtains the neighbors \( N_S(v) \) of the word \( v \) through the upper and lower continuous sliding windows of the text. However, the graph network structure is not linear, so it is difficult to directly define its neighbors. The Node2Vec algorithm proposes a weighted random walk algorithm. In this algorithm, for source node \( v \), a neighbor sequence of fixed length \( L \) is generated using the following conditional distribution:

\[
P(c_i = x|c_{i-1} = v) = \begin{cases} 
\frac{\alpha_{pq}}{Z}, & \text{if } (v, x) \in E \\
0, & \text{otherwise}
\end{cases}
\]

where \( c_i \) represents the \( i-th \) node in the random walk, \( \pi_{i}(vx) \) represents the transition probability from node \( v \) to node \( x \), \( Z \) is the normalization factor. In the random walk probability \( \pi_{i}(vx) = \alpha_{pq}(t, x) \cdot w_{vx} \), \( w_{vx} \) is the weight of edge \((v, x)\) , \( \alpha_{pq}(t, x) \) is given by the
following formula, where \( t \) is the previous node, \( v \) is the current node, and \( x \) is the next possible node.

\[
\alpha_{pq}(t, x) = \begin{cases} 
\frac{1}{p}, & d(tx) = 0 \\
1, & d(tx) = 1 \\
\frac{1}{q}, & d(tx) = 2 
\end{cases}
\] (8)

Among them, \( d(tx) \) represents the shortest path between node \( t \) and node \( x \), and by adjusting factors \( p \) and \( q \), a balance can be achieved between BFS and DFS sampling strategies.

In Figure 2, when \( q > 1 \), the probability of node traversal is more inclined to the neighbor node \( x_1 \) that is closer to node \( t \), which belongs to the BFS search strategy; when \( p > 1 \), the probability of node traversal is more inclined to nodes \( x_2 \) and \( x_3 \) those are farther from node \( t \), which belongs to the DFS search strategy. After obtaining the neighbor node set \( N_{S}(v) \), use the Skip-gram model to train the nodes to obtain the vector representation of the nodes.

![Figure 2. Schematic diagram of the neighbor node search route.](image)

Note: BFS is a breadth-first search strategy, which is conducive to considering the structural similarity of nodes; DFS is a depth-first search strategy, which is conducive to considering the homogeneity of nodes. The DFS strategy plays an important role in optimizing the gated recurrent graph neural network algorithm.

2.3. GRU Model

The recurrent neural network algorithm (RNN) is a very important algorithm in the field of deep learning, which is widely used in machine translation, speech recognition, and other fields with time series. However, RNN also has its own shortcomings; that is, it cannot handle the problem of long-term distance dependence. The long-term distance dependence will produce the problem of gradient disappearance and gradient explosion. Therefore, many variants of RNN have been proposed, for example, the more classic ones are the long short-term memory network (LSTM) and gated recurrent unit neural network(s) (GRU), as shown in Figure 3. The GRU network model simplifies the LSTM network model, and combines the forget gate and the input gate in the LSTM algorithm into an update gate, which greatly simplifies the algorithm structure of the LSTM network model, and improves the operation time and accuracy.
In Figure 3, $x_t$ represents the model input data at time $t$, $h_{t-1}$ represents the output result after the input data are calculated by the reset gate and the update gate at the previous moment, $r_t$ represents the information reset gate, $z_t$ represents the information update gate, $h_t$ represents the state to be activated, represents the output of the model at the current time $t$. The formula is as follows:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \quad (9)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \quad (10)$$

$$\hat{h}_t = \tanh(W \cdot [r_t \ast h_{t-1}, x_t]) \quad (11)$$

$$h_t = (1 - z_t) \ast h_{t-1} + z_t \hat{h}_t \quad (12)$$

Among them, $\sigma$ is the sigmoid function, $W$ is the weight matrix, $[A, B]$ represent the splicing of the feature matrix $A$ and the feature matrix $B$.

It can be seen from the above formula that the output at the current time $t$ is not only related to the input data of the current time, but also related to the input data of the previous time. Moreover, the parameters of the GRU unit are less than those of the traditional RNN network and LSTM network, so the amount of the calculation is relatively small, and the training time of the model is shorter.

3. Gated Recursion-Based Graph Neural Network (GR-GNN) Algorithm

This section will focus on the gated recursion-graph neural network (GR-GNN) proposed in this paper. The algorithm fully combines the advantages of the biased random walk algorithm and the GRU neural network, and the accuracy and effectiveness of the algorithm have been verified in the node classification task.

3.1. Feature Pre-Training

First, according to the biased random search strategy in Node2Vec, the graph depth walk sequence (denoted as $L_{N \times \text{len}_\text{seq}}$) is constructed for each node in the graph $G$, and we record the index of each node in the sequence (denoted as $L_{\text{Index}_{N \times \text{len}_\text{seq}}}$). Then a random adjacency matrix $M_{N \times N}$ is constructed from the sequence of depth walks ($L_{N \times \text{len}_\text{seq}}$); we construct a first-order adjacency matrix $A_{N \times N}$ and degree matrix $D_{N \times N}$ according to the network structure of graph $G$. Finally, a normalized depth chain adjacency matrix ($N_{\text{DCAM}}$) is generated according to Equation (2).
3.2. Model Construction

According to the generated depth chained adjacency matrix (N_DCAM) and depth walk sequence (L_{N\times \text{len}_\text{seq}}) in Section 3.1, we need to sort and concatenate the sparse features of the source nodes. The specific process is shown in Figure 4.

![Figure 4](image)

**Figure 4.** Feature ordering and splicing of source nodes.

It can be seen from Figure 4 that the red nodes represent the first-order neighbor nodes of source node 1, and the blue nodes represent the biased random walk nodes of source node 1 (DFS depth search strategy). The purpose of feature sorting is to construct the time series features of the source node, which is convenient for the GRU unit to perform dependent feature extraction. The feature concatenation length of the source node is \( \text{seq} \_\text{concat} = \text{len}_\text{seq} + \max(\text{nodes} \_\text{degree}) \). The GRU message transmission model is as follows:

\[
\begin{align*}
    h_v^{(1)} &= [x_v^T, x_{\text{sort} & \text{seq} \_\text{concat}}^T, 0]^T & \quad (13) \\
    a_v^{(t)} &= N \_\text{DCAM}_v^T \times \text{ReLU}(W_h [h_v^{(t-1)}; \ldots; h_N^{(t-1)}]^T) + b & \quad (14) \\
    z_v^t &= \sigma(W_z a_v^{(t)} + U z_{h_v}^{(t-1)}) & \quad (15) \\
    r_v^t &= \sigma(W_r a_v^{(t)} + U r_{h_v}^{(t-1)}) & \quad (16) \\
    \hat{h}_v^{(t)} &= \tanh(W_a a_v^{(t)} + U (r_v^t \ast \hat{h}_v^{(t-1)})) & \quad (17) \\
    h_v^{(t)} &= (1 - z_v^t) \ast h_v^{(t-1)} + z_v^t \ast \hat{h}_v^{(t)} & \quad (18) \\
    \text{output} &= W_{\text{final}} [h_1^{\text{final}}, h_2^{\text{final}}, \ldots, h_N^{\text{final}}]^T & \quad (19)
\end{align*}
\]

Among them, \( h_v^{(t)} \in R^{1 \times F \times \text{len}_{\text{concat}}} \) is used to determine how to transmit messages between nodes \( v \) at different times; \( N \_\text{DCAM}_v^T \) represents the normalized depth-chained adjacency matrix of node \( v \); \( \sigma(x) = 1 / (1 + e^{-x}) \) is the sigmoid function; \( z_v^t \) and \( r_v^t \) represent the update gate and reset gate of the network model; \( \times \) represents matrix multiplication; \( * \) means element-wise multiplication; \( \text{output} \in R^{N \times \text{classes} \_\text{num}} \).

The steps of the gated recurrent graph neural network algorithm are as follows:

Step 1: first, let the parameter \( p = 1.2 \) and \( q = 0.78 \) (the optimal parameter after random training) in Equation (8), the biased random walk strategy is more inclined to the DFS strategy. Then, according to the spatial network structure of graph \( G \) and the biased random walk strategy, a depth walk sequence \( (L_{N \times \text{len}_\text{seq}}) \) and its index sequence \( (L \_\text{Index}_{N \times \text{len}_\text{seq}}) \), a random adjacency matrix \( M^{N \times N} \), a first-order adjacency matrix \( A^{N \times N} \), and a degree matrix \( D^{N \times N} \) are generated.

Step 2: generate a normalized depth chain adjacency matrix \( N \_\text{DCAM} \) according to Equations (1) and (2) (where \( \beta = 0.5 \), the value obtained after random training and testing).

Step 3: according to Formula (1) and index sequence \( (L \_\text{Index}_{N \times \text{len}_\text{seq}}) \), generate the feature sorting and splicing vector \( X_{\text{sort} \_\text{concat}} \) of each node.

Step 4: the aggregated vector representation of graph nodes is abstracted and transformed by the GRU operator in Equations (13)–(19), and the cross-entropy loss function is
used to calculate the loss and the Adam algorithm to iteratively update the weight of the neural network;

Step 5: at the end, the optimal GR-GNN model is obtained.
The specific process is shown in Figure 5.

![GR-GNN algorithm process](image)

**Figure 5.** GR-GNN algorithm process.

### 4. Experimental Evaluation

This section mainly presents experiments conducted on the algorithm proposed in this paper on public datasets, and verifies the accuracy and effectiveness of the algorithm through the experimental results.

#### 4.1. Experiment Preparation

This subsection describes the experimental environment, experimental data, and evaluation criteria in detail.

##### 4.1.1. Experimental Environment

This lab was conducted on the deep learning framework PyTorch. PyTorch is the python version of Torch. It is a neural network framework open-sourced by Facebook and is especially designed for GPU-accelerated deep neural network (DNN) programming. Torch is a classic tensor library used for manipulating multidimensional data, with a wide range of applications in machine learning and other math-intensive fields. Table 2 shows the experimental environment parameters.

| Name                  | Value                      |
|-----------------------|----------------------------|
| Operating system      | Ubuntu 18.04               |
| Programming language  | Python                     |
| Development framework | PyTorch                    |
| CPU                   | Intel(R) Core(TM) i9-10900K|
| RAM                   | 64G                        |
| GPU                   | RTX 3090                   |

##### 4.1.2. Evaluation Index

This paper uses the Micro-F1 score to evaluate the model on the multi-classification task, calculating the precision and recall for all classification training and testing. The Micro-F1 method is mainly used to evaluate the indicators of multi-classification tasks. It balances precision and recall, and is also defined as the harmonic mean of precision and recall. The calculation formula of Micro-F1 is:

$$Micro - F1 = \frac{TP + FP}{TP + TN + FP + FN}$$ (20)
Among them, $TP$ represents true positives, $FP$ represents false positives, $TN$ represents true negatives, and $FN$ represents false negatives; the higher the value of Micro-F1, the better the model classification effect and the better the model performance.

4.1.3. Data Preparation

In this paper, a large number of comparative experiments were conducted on the proposed gated recurrent graph neural network (GR-GNN) model. We used public datasets Cora, PubMed, and CiteseerX as training and testing objects and compared them with the very classic graph neural network baseline models GCN, GAT, GraphSAGE, etc., in terms of algorithm accuracy and computational efficiency. The overview of the experimental dataset is shown in Table 3.

Table 3. Overview of the experimental dataset.

|          | Cora | CiteseerX | PubMed |
|----------|------|-----------|--------|
| Nodes    | 2708 | 3327      | 19,717 |
| Edges    | 5429 | 4732      | 44,338 |
| Features/node | 1433 | 3703      | 500    |
| Classes  | 7    | 6         | 3      |
| Training nodes | 140  | 120       | 60     |
| Validation nodes | 500  | 500       | 500    |
| Test nodes | 1000 | 1000      | 1000   |

4.2. Experimental Results and Analysis

Through training and testing on public datasets, such as Cora, CiteseerX, and PubMed, the calculation results of Micro-F1 of GCN, GAT, GraphSAGE, and GR-GNN are shown in Table 4.

Table 4. Comparison of Micro-F1 values in the test set of GCN, GAT, GraphSAGE and GR-GNN algorithms in the public dataset.

| Methods   | Cora | CiteseerX | PubMed |
|-----------|------|-----------|--------|
| GCN       | 81.5%| 70.3%     | 79.0%  |
| GAT       | 83.0%| 72.5%     | 79.0%  |
| GraphSAGE | 82.3%| 71.7%     | 78.3%  |
| GR-GNN    | 83 ± 0.5%| 73 ± 0.7%| 79.3 ± 0.3%|

As can be seen from Table 4, after repeating the training process for about 10 times, within the error range, the algorithm proposed in this paper is better than or equal to the classical graph neural network (GCN, GAT, GraphSAGE) in the average Micro-F1 value of the validation set. This shows that the GR-GNN algorithm proposed in this paper is suitable for various environments and has high accuracy.

In addition, in order to reflect the training effect of the model during training and validation, the training set loss and validation set accuracy were collected during the training process, respectively. Using the Cora dataset as an example, the comparison results of the training set loss and validation set accuracy of GCN, GAT, GraphSAGE, and GR-GNN are shown in Figure 6. The ROC curve can well represent the performance and accuracy of the model. The ROC curve of the validation set is shown in Figure 7.
Figure 6. Comparison of training error and validation accuracy of GCN, GAT, GraphSAGE, and GR-GNN on the Cora dataset.

Figure 7. Comparison of training error and validation accuracy of GCN, GAT, GraphSAGE, and GR-GNN on the Cora dataset.

The accuracy rate of GR-GNN in the validation set is the highest, and the loss of GR-GNN in the training set is the lowest. Moreover, when iterating 1000 times, $T_{\text{GAT}} = 148.0842$ s, $T_{\text{GCN}} = 2.66$ s, $T_{\text{GraphSAGE}} = 50.77$ s, $T_{\text{GR-GNN}} = 6.75$ s. It can be seen that the computational efficiency of the GR-GNN model is second only to the GCN model. There are two main reasons; one is that the GR-GNN model uses a more complex GRU operator, and the other is that when the node features are aggregated—the features of the first-order neighbor nodes as well as of the deep chain-dependent nodes are aggregated.

Because the Cora dataset is multi-class, the traditional true positive rate and false positive rate can no longer meet the ROC curve drawing of multi-class scenarios. Therefore, this paper averages the calculated true positive rate and false positive rate by converting the multi-class problem into a binary problem, and the ROC curve of Figure 6 is obtained. As can be seen from Figure 7, the computational performances of these four methods on the Cora dataset are not very different, but the GR-GNN algorithm proposed in this paper clearly outperforms the other classical algorithms.

5. Conclusions

We conducted extensive research on graph neural networks. This paper improves the biased random walk strategy in the Node2Vec algorithm and deeply mines the chain-dependent features of graph neural networks, reconstructing the iterative logic of the GRU unit operator, to extract, abstract, and transform the depth-dependent features of the nodes, and complete the downstream node classification task. Finally, in this paper, we propose a graph neural network algorithm based on gated recursion (GR-GNN). The algorithm, after training and testing on public datasets, was shown to have high accuracy.
and convergence rates. Experiments show that it has great advantages in node classification tasks, particularly in solving knowledge graph node depth dependence and node attribute feature extraction.

In future work, the authors will further verify the robustness and universality of the algorithm in practical application scenarios (knowledge graph, etc.), and further verify the effectiveness of the algorithm on tasks such as link prediction, shortest path, and the Euler cycle.

**Author Contributions:** K.G.: conceptualization, supervision, methodology, formal analysis, and writing—original draft preparation. J.-Q.Z.: implementation, formal analysis, and writing—original draft preparation. Y.-Y.Z.: methodology, and writing—review and editing. All authors have read and agreed to the published version of the manuscript.

**Funding:** Jian-Qiang Zhao’s research is supported by the National Social Science Foundation of China under grant no. 21BTJ070, and the third training object of the sixth “333 project” in Jiangsu Province. Yan-Yong Zhao’s research is supported by the Social Science Foundation of Jiangsu Province under grant no.20EYC008, the National Statistical Research Project of China under grant no.2020LJZ35, the Open Project of Jiangsu Key Laboratory of Financial Engineering under grant no.NSK2021-12, and the third level training object of the sixth “333 project” in Jiangsu Province.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflicts of interest.

**References**

1. Zhou, F.; Jin, L.; Dong, J. A Survey of Convolutional Neural Network Research. *Chin. J. Comput.* 2017, 40, 1229–1251.
2. Gori, M.; Monfardini, G.; Scarselli, F. A new model for learning in graph domains. In Proceedings of the International Joint Conference on Neural Networks, Montreal, QC, Canada, 31 July–4 August 2005; pp. 729–734.
3. Bruna, J.; Zaremba, W.; Szlam, A.; LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv* 2013, arXiv:1312.6203.
4. Qi, J.; Liang, X.; Li, Z. Representation Learning for Large-Scale Complex Information Networks: Concepts, Methods and Challenges. *Chin. J. Comput.* 2018, 41, 222–248.
5. Perozzi, B.; Al-Rfou, R.; Skiena, S. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 701–710.
6. Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; Mei, Q. Line: Large-scale information network embedding. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 1067–1077.
7. Grover, A.; Leskovec, J. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 855–864.
8. Defferrard, M.; Bresson, X.; Erdogan, P. Convolutional neural networks on graphs with fast localized spectral filtering. *Adv. Neural Inf. Process. Syst.* 2016, 29, 258–276.
9. Levie, R.; Monti, F.; Bresson, X.; Bronstein, M.M. CayleyNets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Process.* 2019, 67, 97–109. [CrossRef]
10. Spinelli, I.; Scardapane, S.; Uncini, A. Adaptive propagation graph convolutional network. *IEEE Trans. Neural Networks Learn. Syst.* 2020, 32, 4755–4760. [CrossRef]
11. Duvenaud, D.K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; Adams, R.P. Convolutional networks on graphs for learning molecular fingerprints. *Adv. Neural Inf. Process. Syst.* 2015, 28, 1594–1603.
12. Zhao, Z.; Zhou, H.; Qi, L.; Chang, L.; Zhou, M. Inductive representation learning via cnn for partially-unseen attributed networks. *IEEE Trans. Netw. Sci. Eng.* 2021, 8, 695–706. [CrossRef]
13. Liang, M.; Zhang, F.; Jin, G.; Zhu, J. FastGCN: A GPU accelerated tool for fast gene co-expression networks. *PLoS ONE* 2015, 10, e0116776. [CrossRef] [PubMed]
14. Kou, S.; Xia, W.; Zhang, X.; Gao, Q.; Gao, X. Self-supervised graph convolutional clustering by preserving latent distribution. *Neurocomputing* 2021, 437, 218–226. [CrossRef]
15. Luo, J.X.; Du, Y.J. Detecting community structure and structural hole spanner simultaneously by using graph convolutional network based Auto-Encoder. *Neurocomputing* 2020, 410, 138–150. [CrossRef]
16. Li, C.; Welling, M.; Zhu, J.; Zhang, B. Graphical generative adversarial networks. In Proceeding of the 32nd Conference on Neural Information Processing Systems, Montréal, QC, Canada, 3–8 December 2018; pp. 6072–6083.
17. Gharaee, Z.; Kowshik, S.; Stromann, O.; Felsberg, M. Graph representation learning for road type classification. *Pattern Recognit.* 2021, 120, 108174. [CrossRef]

18. Zhang, S.; Ni, W.; Fu, N. Differentially private graph publishing with degree distribution preservation. *Comput. Secur.* 2021, 106, 102285. [CrossRef]

19. Ruiz, L.; Gama, F.; Ribeiro, A. Gated graph recurrent neural networks. *IEEE Trans. Signal Process.* 2020, 68, 6303–6318. [CrossRef]

20. Bach, F.R.; Jordan, M.I. Learning graphical models for stationary time series. *IEEE Trans. Signal Process.* 2004, 52, 2189–2199. [CrossRef]

21. Liu, J.; Kumar, A.; Ba, J.; Kiros, J.; Swersky, K. Graph normalizing flows. *Adv. Neural Inf. Process. Syst.* 2019, 32, 5876.

22. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *Stat* 2017, 1050, 20.

23. Zhao, Y.; Zhou, H.; Zhang, A.; Xie, R.; Li, Q.; Zhuang, F. Connecting Embeddings Based on Multiplex Relational Graph Attention Networks for Knowledge Graph Entity Typing. *IEEE Trans. Knowl. Data Eng.* 2022, Early Access. [CrossRef]

24. Wu, J.; Pan, S.; Zhu, X.; Zhang, C.; Philip, S.Y. Multiple structure-view learning for graph classification. *IEEE Trans. Neural Networks Learn. Syst.* 2017, 29, 3236–3251. [CrossRef] [PubMed]

25. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. *arXiv* 2013, arXiv:1301.3781.