DGCN: LEARNING GRAPH REPRESENTATIONS VIA DENSE CONNECTIONS A PREPRINT

Khairi Abidi, Wissem Inoubli, Engelbert Mephu Nguifo

DGCN: LEARNING GRAPH REPRESENTATIONS VIA DENSE CONNECTIONS

A PREPRINT

Khairi Abidi
Department of Computer Science
University of Jendouba
Jendouba, Tunisia
khairi.abidi@fsjegj.edu.tn

Wissem Inoubli
Department of Computer Science
University of Lorraine
CNRS, LORIA, France
wissem.inoubli@loria.fr

Engelbert Mephu Nguifo
Department of Computer Science
University Clermont Auvergne
Clermont Ferrand, France
engelbert.mephu_nguifo@uca.fr

February 15, 2023

ABSTRACT

In the last decades, learning over graph data has become one of the most challenging tasks in deep learning. The generally proposed Graph Neural Network (GNN) framework computes a hidden state for every node in the graph by applying nonlinear transformations to its neighborhood. Then updates the node of interest's hidden state. Nevertheless, the node features can contain discriminative information. That can get lost over GNN layers. Here we present a new variant of GNN architecture where we combine node features and GNN activations to learn nodes representations in the graph. We conduct extensive experiments on two graph prediction tasks (node classification and link prediction) and shows that our method can match and outperforms state-of-the-art results

***K*eywords** Graph Neural Networks · Graph Representation Learning · Node Classification · Link Prediction

1 Introduction

Graphs are data structures that represent objects and relations between them, such as citations networks, biological networks, and social networks. Graph kernel [1], [2] approaches have been the most efficient machine learning methods for graph classification However, because these kernels are handcrafted and constantly require domain expertise to be constructed, they cannot acquire useful node feature representation. And since, learning node representations has become one of the most challenging issues in graph machine learning. A wide variety of random walk-based node embedding algorithms [3], [4], [5] have been proposed. However, these methods are ineffective due to their transductivity, scalability..

Representation Learning [6], [7] on graphs use a neighborhood aggregation approach that has proven effective in a variety of graph prediction problems. Because of this aggregation, neural networks only learn node/graph representations based on the graph structure itself, and node level features that can contain meaningful and discriminative information are ignored. As a result, the network's later layers cannot incorporate low-level information embedded in raw node features. In this paper, we approach the problem of learning node representations (i,e; hidden state h_v, for each node $v \in \mathrm{V}$) while incorporating low-level information about nodes. Message Passing Neural Networks (MPNNs) [8] carry out this problem by encoding the local neighborhood structure of a given node v in a message m_v computed from its nearby nodes and uses this message and the previous hidden state to update the current hidden state as shown in equation 1.In MPNNs the initial node state $h_v^0 = x_v$, where x_v is the node feature vector, we make the assumption that x_v would be forgotten since it will transformed and merged with structural information computed in the aggregation process.

$$
\begin{aligned}
m_v^k &= MSG(\{h_u^k; \forall u \in \mathrm{N}(v)\}) \\
h_v^{k+1} &= UPDATE(h_v^k, m_v^k)
\end{aligned}
\tag{1}
$$

In this work, we present a novel method for learning graph representation, where we concatenate the transformed node features and node's local structure encoded by a GNN model.

In summary, the contributions of this paper are:

1. Propose a novel graph neural network method for learning node representation.
2. Implement the proposed architecture on top of GCN [9] framework.
3. Demonstrate that our method can achieve state-of-the-art results on a node classification and link prediction tasks.

The remaining of this paper is organized as follows. section 2 presents background on graph neural networks, section 3 presents the proposed method, section 4 sets experiments, section 5 results and discussion, finally, we conclude the paper in section 6.

2 Background

Graph Neural Networks

Graph Neural Networks are a class of Deep Learning architectures that operates on graph structured data. GNNs applies non linear transformations over the graph with the purpose of learning an embedding vector h_v for every node v in the graph. These nodes representations/embeddings could be used in downstream applications such as node classification, link prediction, or graph classification. Message Passing Neural Networks [8] is a framework for designing Graph Neural Networks. An MPNN layer is made up of two essential components: the aggregation/message function and the node update function. For computing the node representation/embedding, firstly, aggregate a function of nodes hidden states and edges of all neighbouring nodes, then, update the hidden state with the received message and the previous hidden state as stated in equation 1, the initial hidden state h_v^0 is set to be the node feature vector. The aggregation/message function could be an permutation invariant function (e.g., sum, mean, max, min, ...), as well as it can be approximated by multilayer perceptrons [10].

Next, we introduce and present GCN [9], GAT [11], and SGC [12].

Graph Convolution Networks (GCN)

GCN [9] is one of the most well-known GNN architectures. GCN learns node representations by propagating information from direct neighbors, applying linear transformation and pointwise nonlinearity. The information propagation procedure consists of aggregating information from the direct neighborhood. Next, as a multilayer perceptron, GCN [9] applies linear transformation followed by pointwise nonlinearity. By stacking k GCN layers, each node aggregates information from nodes k-hops away. The GCN [9] propagation rule is defined as follows:

Information Propagation

$$
\hat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} \tag{2}
$$

$$
H^l = \sigma(\hat{A}H^{l-1}W^l),\ with\ H^0 = X \tag{3}
$$

Where $\hat{A}$ is the normalized Adjacency matrix with added self-loops, $\tilde{A} = A + I$, $\tilde{D}_{ii} = \sum_j^{|V|} \tilde{A}_{ij}$ is the degree matrix, H^l and H^{l+1} are the previous and the new hidden state matrix, respectively, W^l is a trainable weight matrix for layer l, and σ denotes any non-linear activation function. The message passing form of equation 3 can be written as follows:

$$
\begin{aligned}
m_u^l &= \sum_{v \in \mathcal{N}(v)} \frac{1}{\sqrt{|\mathcal{N}_v|}\sqrt{|\mathcal{N}_u|}} h_v^{l-1} \\
h_u^l &= \sigma(m_u^l W^l + b^l)
\end{aligned}
\tag{4}
$$

That is, each node receive a message m_v which is the averaged hidden states of its neighbors $\mathcal{N}(v) = \{\mathcal{N}(v) \cup v\}$. Then, a linear transformation followed by nonlinear activation is applied to generate the new hidden state.

Simplifying Graph Convolution (SGC)

SGC [12] postulates that GNNs inhert unnecessary computations from deep neural networks,[12] removes the nonlinear activations and squashes weight matrices into one leanrable matrix W. Hence, a two-layer GCN [9] (Equation 5) is transformed into a linear model by removing the non-linearity and squashing the matrices W^1 and W^2 into W. The resulting model (Equation 6) is a simple logistic regression classifier.

$$\hat{Y} = softmax(\tilde{A}ReLU(\tilde{A}XW^1)W^2) \tag{5}$$

$$\hat{Y} = softmax(\tilde{A}^2XW) \tag{6}$$

Feature Propagation Unlike GCN [9], which aggregates k-hop information by stacking k GCN layers (Equation 3). The k-hop information in [12] can be aggregated by raising the adjacency matrix to the k-th power ([13] *Lemma 5.2*).

$$\tilde{X} = \tilde{A}^k X$$

$\tilde{X}$ is a preprocessing step to collect the k-hop neighbors information.

Graph Attention Networks (GAT)

GAT [11] spreads the attention mechanism [14] to graph neural networks. The attention is computed explicitly pairwise between nodes sharing the same edge, and thus can be learned by backpropagating the error, ***i.e.***, for each edge e_{ij} a coefficient $\alpha_{e_{ij}}$ is computed and used to update the hidden state. To depict the edge weight the coefficients $\alpha_{e_{ij}}$ are normalized using *softmax*.

Masked Attention

$$\begin{aligned} e_{ij} &= a(Wh_i \parallel Wh_j) \\ \alpha_{ij} &= softmax(e_{ij}) \end{aligned} \tag{7}$$

In order to obtain an expressive power, at least one linear transformation is needed Wh_i, then, for each edge $\{(i,j);\ j \in \mathcal{N}(i)\}$ a feature vector is generated by concatenating the corresponding transformed node features $Wh_i||Wh_j$. Next, another linear transformation $a : F' \times F' \rightarrow F'$ is applied to the edge features to produce the edge weights (attention coefficients) e_{ij}. Finally, the attention coefficients are normalized by the *softmax* function.

Message Passing and MultiHead Attention

$$\begin{aligned} m_i &= \sum_{j \in \mathcal{N}_i} \alpha_{ij} h_j \\ h_i &= \|_{k=1}^{K}\ \sigma(m_i W + b) \end{aligned} \tag{8}$$

The message received is a linear combination of the attention coefficient α_{ij} and the hidden state of the neighbor h_j. a K parallel attention heads (Equation 8) are computed then passed, each through an MLP layer (linear transformation + nonlinearity). Finally, the attention heads are concatenated $\|$ to produce the new hidden state.

3 Proposed Method

Problem formulation

We consider an undirected (or directed) simple graph $G = (\mathcal{V}, \mathcal{E}, X)$, where $\mathcal{V}$ is the set of nodes $v \in \mathcal{V}$, $\mathcal{E}$ is the set of edges $e_{uv} \in \mathcal{E}$, and $X = \{x_1, \ldots, x_{|\mathrm{V}|}\}$, with $x_i \in \mathbb{R}^d$ is the set of attributes / features of the nodes. We also consider the problem of learning $h_v \in \mathbb{R}^f$ a vector representation of node v. We motivate the selection of our architecture by the idea that the node attributes may contain discriminating information that can be useful for learning higher node representations and improving model performance.

In most GNNs architectures [9], [11], graph topological information is extracted through a message passing scheme [8], and node attributes are neglected. That is, a GNN Layer is aggregating a message m_v from its neighbors and then update its hidden state h_v, even when initializing the initial state $h_v^0 = x_v$ as the node feature vector, we assume that

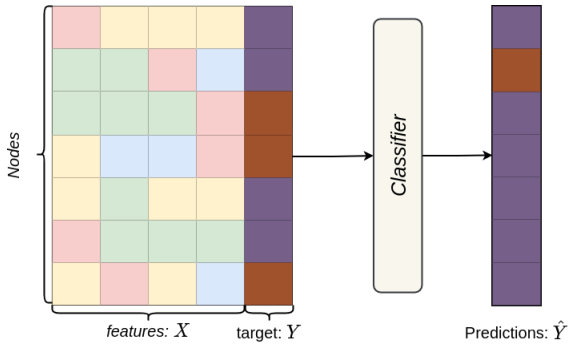

Figure 1: documents classification using raw features.

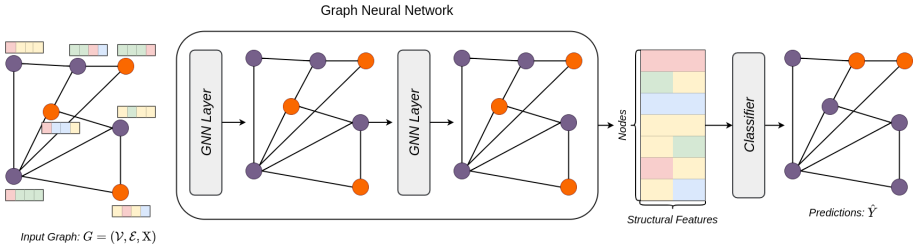

Figure 2: documents classification using topological corpus structure.

this information will be forgotten or simply erased by the structural information encoded in m_v. However, in numerous applications; such as social networks, product recommendation, *etc*, the node features x_v is as important as structural information learned by GNN.

Motivation

In this section, we provide motivation for the choice of our architecture using a straightforward node classification problem. Assume we have a collection of N documents, where each document i has a four-dimensional feature vector $x_i \in \mathbb{R}^4$ and a binary label $y \in \{0, 1\}$, as well as an adjacency matrix $A \in \mathbb{R}^{N \times N}$ encoding the linkages between documents. In other words, $A_{ij} = 1$ If document i links to document j, we assume that the adjacency matrix is symmetric, that is, if document j links to j, then document j links to i. Each document is represented by a vector holding the frequency of the four most frequently occurring words in the corpus. Colors reflect frequencies in this ascending order: blue, green, yellow, red. For example, red is affiliated with words that have high frequencies, whereas blue is associated with words that have low frequencies. For illustration, the second most common term in the corpus appears frequently in document 7 (Figure 1), whereas the fourth most frequent word in the corpus appears less frequently.

Figure 1 illustrates the classification of documents using only raw features / document content. As can be observed, the classifier favors documents with high frequency terms and assigns them label 1, whereas documents with less frequent terms, such as document 2 (2 green and one blue), are more likely to be given label 0. When using the corpus structure encoded in documents links and raw node features as initialization to a Graph Neural Network model. The model may extract node features based on their neighborhood information. However, this GNN feature may cause the model to overfit and favor structural information over content. As illustrated in Figure 2, the classifier labels document two as 0 because of its relationship with document 3, while documents four and seven are classified as 1 since they are only connected to nodes with label 1. As a result, neither node features nor graph structure alone would produce good results in this scenario. Next, we present our model, which combines node content/intrinsic information with structural information encoded by GNN to build node representations that can improve the classification (or more generally *the downstream task*).

Dense Graph Convolution Network

We propose Dense Graph Convolution Networks (DGCN) an architecture that combine node features and structural information to learn node representations (Figure 3). DGCN consists of three components, *projection module*, *the GNN module*, and a *dense connection* procedure.

Projection Module The projection module is responsible for learning hidden representations of the features of the node. To obtain sufficient expressive power, at least one linear transformation is needed. Therefore, we choose the projection module to be a linear transformation of node features.

$$p_v = W x_v^T \tag{9}$$

GNN Module In DGCN, the GNN module is carrying the message passing operation to update the hidden state. In our experiments, we choose GCN [15], as the graph encoder.

$$m_v = \sum_{u \in \mathcal{N}(v)} \frac{h_u}{\sqrt{|\mathcal{N}(v)| * |\mathcal{N}(u)|}} \tag{10}$$

$$h_v = ReLU(W m_v + b) \tag{11}$$

Dense Connection We define a dense connection as an operation combining the node hidden representation p_v and its hidden state h_v. Various combining operations are available, such as *sum*, *mean*, *dot product*, *hadamard product*, or *concatenation*. To enrich the learned hidden state of node v we choose to concatenate the hidden representation and the hidden state (Equation 12). That is, the output of the GCN layer will be concatenated with the projected node features p_v. To produce a hidden representation that encodes both node-level and structural information.

$$h_v = h_v \parallel p_v \tag{12}$$

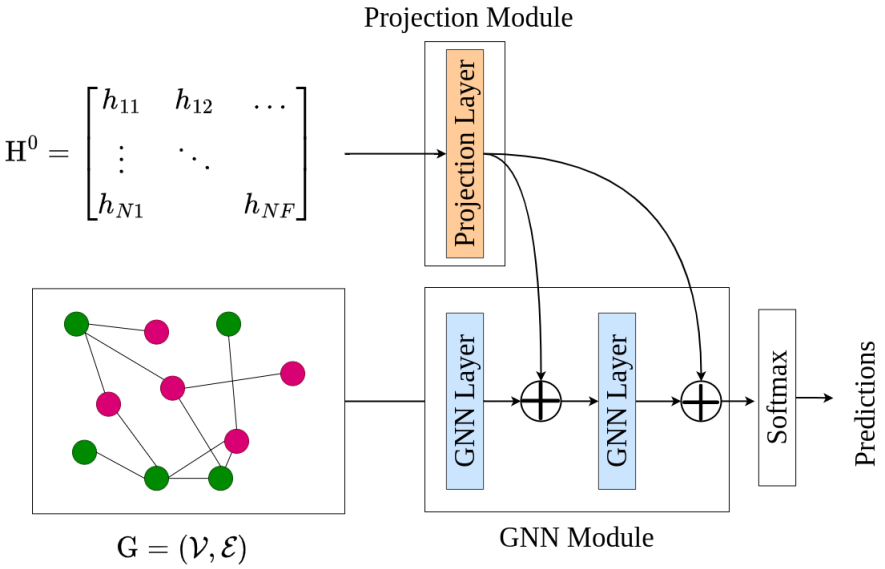

Figure 3: Dense Graph Neural Networks Architecture

First, the linear projection (Equation 9) is applied to the node features matrix X to produce the hidden representation P. Then, the graph structure encoded in $\hat{A} = \tilde{D}^{1/2}\tilde{A}\tilde{D}^{1/2}$ and the initial hidden state $H^0 = X$ are fed to the GNN encoder [9]. Finally, the output of each GNN layer H^l is concatenated with the projected features P, $H^l = H^l \parallel P$.

The DGCN algorithm (*i.e, forward pass*) is described in algorithm 1. which assumes that the model has already been trained and that the weights are fixed.

Algorithm 1: DGCN forward pass algorithm

Data: Graph $G = (\mathcal{V}, \mathcal{E})$; Input features $\{x_1, \ldots, x_{|\mathcal{V}|}\}$; Depth K; Weight matrices $W^k, \forall k \in \{1, \ldots, K\}$; Projection matrice W_p
Result: Vector embeddings $z_v, \forall v \in \mathcal{V}$;
$h_0 \leftarrow x_v, \forall v \in \mathcal{V}$;
$p_v = W^p x_v^T, \forall v \in \mathcal{V}$;
for $k \in \{1, \ldots, K\}$ **do**
  for $v \in \mathcal{V}$ **do**
    $m_v^k = \sum_{u \in \mathcal{N}(v)} \frac{h_u}{\sqrt{|\mathcal{N}(v)| * |\mathcal{N}(u)|}}$;
    $h_v^k = ReLU(W^k . m_v^k)$;
    $h_v^k = h_v^k \parallel p_v$;
  end
end
$z_v = h_v^K, \forall v \in \mathcal{V}$

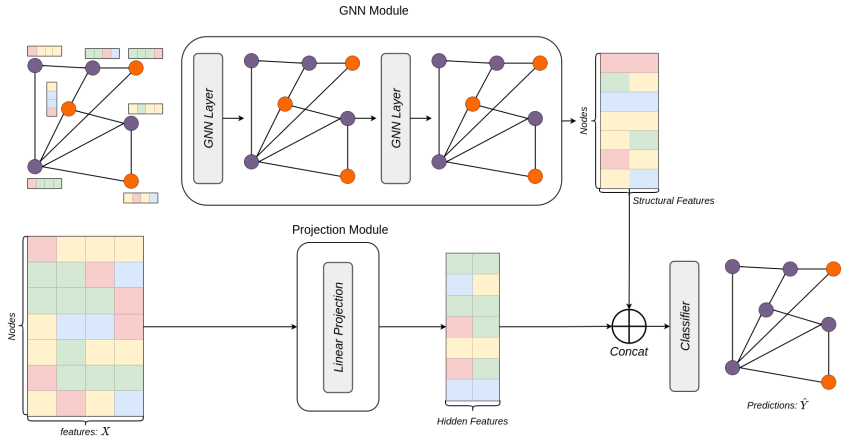

Figure 4: documents classification using document content and network information.

DGCN Analysis

As mentioned earlier in Section 3 (Motivation) and illustrated by the example, neither node features nor graph structure would give satisfactory classification results. For this reason, DGCN is utilizing structural information while keeping track of intrinsic document information. Furthermore, the intrinsic node information is passed to a linear layer (the projection module) for feature selection. Finally, the selected features are joined with the structural information (in Figure 4, we omit the concatenation after each layer for simplicity).

The overall DGCN architecture is shown in Figure 4, where input documents with their relations are supplied to the GNN Module, along with node features used to establish hidden states. Furthermore, intrinsic node features are passed via the linear projection layer to extract latent features, which will be concatenated with structural information later. That is, the GNN models produce structure-based hidden features; particularly, the seventh document has features that are similar to those of its neighbors and will be classed as 1, because it is only connected to nodes with labels 1. However, its hidden content features produces a low frequency feature vector (blue color), leading the classifier to classify it as class 0.

4 Experiments

Our experiments assess the performance of DGCN in node classification and link prediction tasks.

Dataset	# Nodes	# Edges	# Node Atrributes	# Classes
Cora	2708	10556	1433	7
Amazon Photos	7650	238162	745	8
Amazon Computers	13752	491722	767	10
Facebook	22470	342004	128	4
LastFM	22470	342004	128	4

Table 1: Node classification datasets statistics

Datasets

To evaluate our model's performance in a node classification task. We used Cora citation network [16], LastFMAsia [17], Facebook [18], as well as Amazon Computers and Amazon Photos [19], [20]. The *Pytorch Geometric* [21] Python software provides all of the datasets described above. The statistics of the datasets are summarized in Table 1.

Citation networks are graphs with nodes representing papers and edges representing the existence of citation between two papers. The node features are bag-of-words vectors derived from the abstract section of the paper. The category of the paper determines the node labels (e.g, Machine learning, Graph Theory, Statistics, ...)

Amazon Computers and Amazon Photos are Amazon co-purchase graph segments [19], [20]. The edges indicate that two products are frequently purchased together, the node attributes are bag-of-words-encoded product reviews, and node labels are provided by the product category.

LastFMAsia [17], is a social network of users from Asia. Nodes represent users of the LastFM music streaming service, and links among them are friendships.

Facebook [18] is a page-page graph of verified Facebook sites. Nodes correspond to official Facebook pages, links to mutual likes between sites. Node features are extracted from the site descriptions.

Splitting Strategy

Node Classification

In the node classification task, we split the graph according to [16], *i.e,* 20 nodes for training per class, 500 nodes for validation, and 1000 nodes for testing.

Dataset	# Training nodes	# Validation nodes	# Testing nodes
Cora	140	500	1000
LastFM	80	500	1000
Facebook	80	500	1000
Amazon Computers	200	500	1000
Amazon Photo	160	500	1000

Table 2: Number of training, validation, and testing nodes on each dataset.

Link Prediction

We turn the link prediction task into a binary classification problem, we train a model to predict whether an edge (i, j) exists or not. The fake edges are randomly sampled with a ratio of 1, *i.e,* for each positive edge, and we sample *one* negative edge. We divide the set of edges $\mathcal{E}$ into three subsets $\mathcal{E}_{train}$, $\mathcal{E}_{valid}$, and $\mathcal{E}_{test}$, being, 50%, 40%, and 10% of the original edges set $\mathcal{E}$, respectively.

Dataset	# Training edges	# Validation edges	# Testing edges
Cora	10558	8444	2110
LastFM	342006	273602	68400
Facebook	342006	273602	68400
Amazon Computers	491724	393376	98344
Amazon Photo	238164	190528	47632

Table 3: Number of training, validation, and testing edges on each dataset. each split includes positive and negative edges.

Metrics

To assess the performance of our model against baseline methods [15], [22], [23] in node classification, we choose accuracy as a performance measure. In link prediction, we use AUROC as a metric.

Accuracy is a metric that describes how the model performs across all classes. It is helpful when all classes are equally important. It is determined by dividing the number of *right predictions* by the total number of predictions (Equation 13).

$$accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{13}$$

Area Under the Receiver Operating Characteristics (AUROC) is a measure of a classifier's ability to distinguish between classes and is used to describe the ROC curve . The greater the AUC, the better the model's performance in differentiating between positive and negative classes.

Baselines

We compare DGCN to GCN [15], GAT [22], and SGC [23]. For all models, we use the same set of hyper-parameters stated in the original papers. All models hyperparameters were optimized on the Cora dataset. The following GCN [15] hyperparameters: *hidden units* $= 16$, *number of layers* $= 2$, *activation* is $ReLU$, *dropout rate*$= 0.5$, *learning rate* $= 0.1$, and *weight decay* $= 0.0005$. GAT [22] hyperparameters are the following: *hidden units* $= 8$, *attention heads* $= 8$, *dropout rate* $= 0.6$, *layer activation* is ELU, *attention activation* is $LeakyReLU$ with *negative slope* $= 0.2$, *learning rate* $= 0.005$, *weight decay* $= 0.0005$. SGC [23] hyperparameters are: $K = 2$, *learning rate* $= 0.2$, and *weight decay* $= 0.0000130$.

Configuration

First, we optimize the hyperparameters of our model (*projection dim*, *hidden units*, *learning rate*, and *weight decay*) on Cora dataset using Optuna [24] Python package for 30 trial. Then, we use the hyperparameters of our model in all other experiments on the remaining datasets.

In link prediction, we used an *encoder-predictor* architecture, *i.e* the graph, along with node features, are fed to a GNN encoder to compute the hidden states $\{h_v; \forall v \in \mathcal{V}\}$. Then, for each edge (u, v) a feature vector $e_{uv} = h_u \parallel h_v$ is constructed by concatenating node hidden states of corresponding nodes u and v. Finally, the edge feature e_{uv} is passed through a one layer MLP with one output to predict whether the edges exists or not.

For all experiments, we train all models for a maximum of 300 epochs using Adam [25]. We report the mean accuracy over 10 runs.

5 Experimental Results

5.1 Node Classification Experiments

Model	Cora	LastFM	Computers	Photos	Facebook
GCN	80,65±0.59	52,55±0.52	71,70±4.18	82,74±3.25	14,88±3.39
SGC	80,12±0.34	71,76±0.54	**94,88**±0.28	**96,95**±0.35	22,60±0.39
GAT	**83,50**±0.64	73,20 ±1.25	89,48±0.66	93,85±0.40	**74,64**±1.39
DGCN (ours)	80,30±0.97	**75,82**±0.89	91,00±1.38	93,50±1.67	71,37±1.16

Table 4: Caption

Table 4 shows the accuracy of node classification on Cora, LastFM, Amazon-Photos, Amazon-Computers, and Facebook. Our model closely matches the results of SGC [23], and GAT [22] on Cora, Amazon-Photos. And outperforms GCN [15] by orders of magnitude on all datasets (except Cora). When node features are dense, though, the projection layer can learn useful hidden representations, which improves the hidden state information encoded by GNN - in concordance with our expectations. As a result, the classifier benefits from node-level and structure knowledge. as demonstrated by our model outperforming GCN and GAT on LastFM and Amazon-Computers.

Performance with varying the number of layers

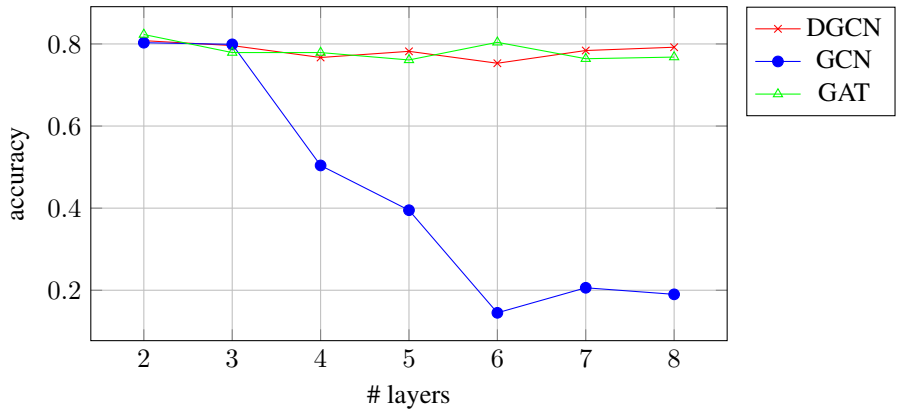

Figure 5: Performance (in terms of accuracy) with varying the number of layers Cora dataset

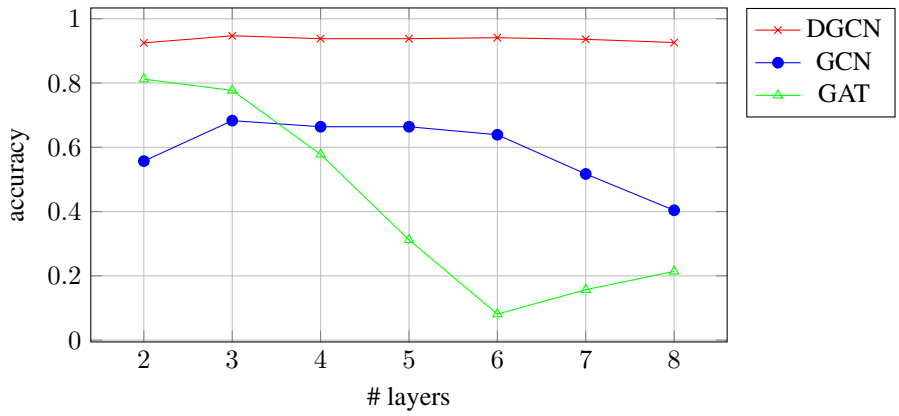

Figure 6: Performance (in terms of accuracy) with varying the number of layers on LastFM dataset.

The oversmoothing problem [26] is inherent in GNN. When more layers are added to a GNN model (GCN [15] or GAT [22]), each node aggregates information from the same set of k-hop neighbors (especially when k is large). As a result, different nodes will have similar representations, and the classifier will be unable to differentiate between them, resulting in poor performance. Figure 5 shows that even a sparse features can assist the model (DGCN) to alleviate the oversmoothing issue, while other methods (GCN and GAT) suffers clearly from performance deterioration. On the LastFM dataset (Figure 6), our model benefits from the additional layers, achieving 93% (+ 1%) and 94% (+ 2%) on 5 and 6 layers, respectively.

5.1.1 Performance with varying the number of training nodes

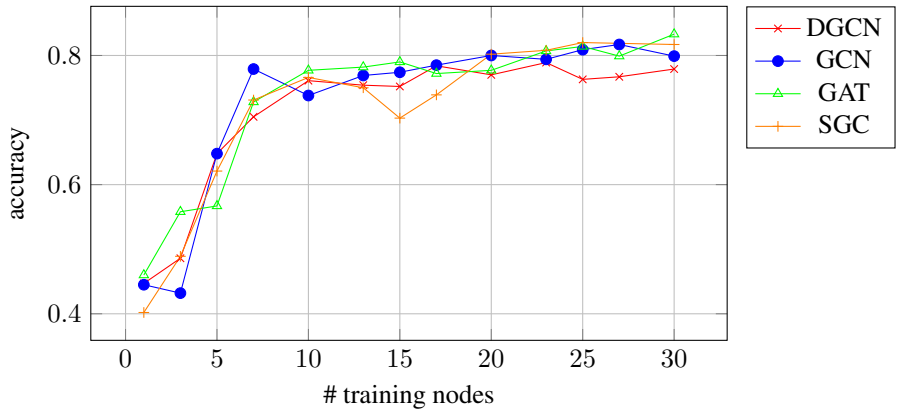

Figure 7: Performance (in terms of accuracy) with varying the number of training node on Cora dataset.

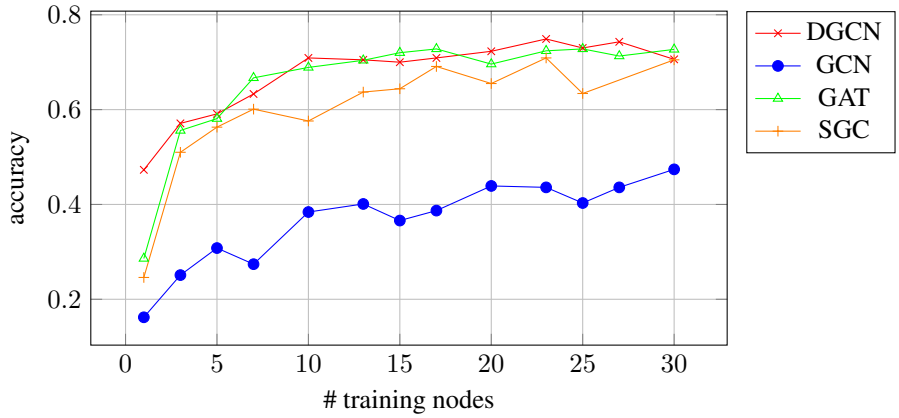

Figure 8: Performance (in terms of accuracy) with varying the number of training node on LastFM dataset.

To evaluate the robustness of our model against GCN [15], GAT [22], and SGC [23]. We vary the number of training nodes per class. Then, train the model to fix the weights. Finally, report the accuracy on the test nodes. Figure 7 shows that our model nearly achieves the same performance of GCN, GAT, and SGC on the Cora citation network; we explain this to the sparsity of the matrix X. Neverthless, when node features are dense and a small set of training nodes is available (1 nodes per class), which is the case for almost all real-world applications, DGCN is outperforming baselines by significant margin ($+24\%$ on average).

Performance with different combining operations

Dataset	Operation		
	sum	*average*	*concat*
Cora	78,80±2.83	79,96±0.40	**80,30**±0.97
LastFM	67,81±1.80	72,63±0.26	**75,82**±0.89
Photo	87,24±0.14	97,92±0.15	**93,50**±1.67
Computer	84,87±0.52	80,05±0.63	**91,00**±1.38
Facebook	69,55±0.23	**71,60**±0.71	71,37±0.21

Table 5: Performs in terms of accuracy of different dense connections operations

As stated in section 3, concatenation is the best dense operation choice in our model. We validate our assumption by performing several dense operations (*sum*, *average*, *concat*) on node classification datasets and reporting the test accuracy. On three datasets (Cora, LastFM, and Computers), the concatenation outperforms other approaches and matches the *avg* score on Facebook. In the concatenation operation, the node-level information are kept and injected to structural information forming a node representation with both node and graph information, while in other operations, node intrinsic information are *merged* with structural information.

Link Prediction Experiments

Model	**Cora**	**LastFM**	**Computers**
GCN	59,59±0.21	74,82±0.21	78,83±0.12
SGC	**61,95**±0.15	77,32±0.01	76,24±0.05
GAT	53,19±2.53	65,83 ±0.72	43,14±2.88
DGCN (ours)	53,68±1.78	**80,15**±0.24	**79,50**±0.80

Table 6: Link prediction results. Bests are bold.

Table 6 summarizes our link prediction task outcomes. A general conclusion we can draw from the results is that node-level information is beneficial to the classifier. Furthermore, we confirm that sparse features are ineffective and potentially affect DGCN performance. By combining node attributes with learnt hidden representation, DGCN gains $+5.33\%$ over GCN on LastFM and $+0.68\%$ on Amazon Computers, whereas SGC, a simpler model, outperforms GCN, GAT, and DGCN by a large margin on the Cora dataset.

6 Conclusion and future works

Motivated by the idea that nodes may contain significant information as their connections. We proposed DGCN, a novel method for learning graph representation by combining node-level features with hidden representation learned by GCN. We compared our proposed method with state-of-the-art models in node classification and link prediction tasks and confirm our assumptions.

Several potential DGCN improvements and additions, such as adopting an attention-based strategy for combining node-level features with hidden representation, could be addressed as future work. An interesting option to pursue would be to investigate more advanced projection architecture. Examine model inductivity in graph classification tasks. Finally, the model would be expanded to include edge features.

Acknowledgments

References

[1] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.

[2] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In David van Dyk and Max Welling, editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR.

[3] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[4] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[6] Jonathan Masci, Emanuele Rodolà, Davide Boscaini, Michael M. Bronstein, and Hao Li. Geometric deep learning. In Niloy J. Mitra, editor, *SIGGRAPH ASIA Courses*, pages 1:1–1:50. ACM, 2016.

[7] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[8] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[9] Max Welling and Thomas N Kipf. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016.

[10] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[11] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[12] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[13] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[15] Max Welling and Thomas N Kipf. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016.

[16] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[17] Benedek Rozemberczki and Rik Sarkar. Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models. In *Proceedings of the 29th ACM international conference on information & knowledge management*, pages 1325–1334, 2020.

[18] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[19] Julian McAuley, Christopher Targett, and Anton van den Hengel. Image-based recommendations on styles and substitutes.

[20] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

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

[22] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

[23] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[24] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3438–3445, 2020.