CONVOLUTIONAL NETWORKS ON ENHANCED MESSAGE-PASSING GRAPH IMPROVE SEMI-SUPERVISED CLASSIFICATION WITH FEW LABELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Efficient message propagation is critical to node classification in sparse graph with few labels that remains largely unaddressed until now. Recently popularized Graph Convolutional Networks (GCNs) lack the ability to propagate message to distant nodes because of over-smoothing. Besides, GCNs with numerous parameters suffer from overfitting when labeled nodes are scarce. We attack this problem via building GCNs on Enhanced Message-Passing Graph (EMPG). The key idea is node classification can benefit from various variants of the original graph that are more efficient for message propagation, based upon the assumption that each variant is a potential structure as more nodes are properly labeled. Specifically, we first map nodes to a latent space through graph embedding that captures structure information. Considering node attributes together, we construct EMPG by adding connections between nodes in close proximity in the latent space. With the help of added connections, EMPG allows a node to propagate message to the right nodes at distance, so that GCNs on EMPG need not stack multiple layers and therefore avoid over-smoothing. However, adding connections may cause message propagation saturation or lead to overfitting. Seeing EMPG as an accumulation of the potential variants of the original graph, we apply dropout to EMPG and train GCNs on various dropout graphs. The features learned from different dropout EMPGs are aggregated to compute the final prediction. Experiments demonstrate a significant improvement on node classification in sparse graph with few labels.

1 INTRODUCTION

Graphs are emerging as a prevalent form of data representation to capture complex structures and relationships between a set of objects in many fields. A problem that comes up often but remains largely unaddressed is node classification in sparse graph with few labels. The earlier work was done by using structural information only. Recently, attention has shifted to graph neural networks (GNN) that model structure and attributes jointly, including GCN (Kipf & Welling, 2017), SGC (Wu et al., 2019), GAT (Velickovic et al., 2018), AGNN (Thekumparampil et al., 2018), TAGCN (Du et al., 2017), DGCN (Zhuang & Ma, 2018), and GIL (Xu et al., 2020) to name some.

Most GCNs work by a message passing scheme. A Graph Convolutional Layer (GCL) can be viewed as a message propagating step, where each node sends its feature representation, a "message", to its neighbors; and then updates its feature representation by aggregating all "messages" received from its neighbors. Different aggregation and update functions give rise to a new form of GCN. Due to this flexibility, the class of message passing networks have been widely used, such as publication citation networks (Kipf & Welling, 2017), social networks (Chen et al., 2018), applied chemistry (Liao et al., 2019), and natural language processing (Yao et al., 2019).

Despite their success, limitations exist. For example, GCN, which updates node state by aggregating messages from one-hop neighbors, lacks the ability to receive long-range messages. This suggests it only works on graphs where nodes from the same class tend to be connected directly. However, in many practical graph data, the nodes of the same class that possess high structural similarity may be far apart from each other (Ribeiro et al., 2017). For example, the literatures on the same topic but published by independent research groups may be separate in the citation network, as shown in

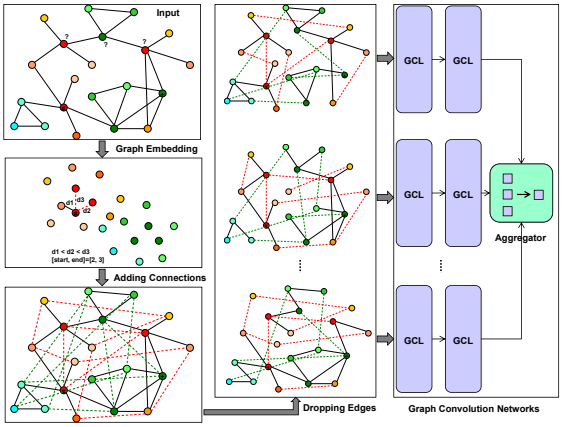

Figure 1: The framework of our model.

the upper left corner of Fig.1. In cases like this, the performance of the GCNs that lack the ability to capture long-range messages will drop, especially where labeled nodes are scarce and graph is sparse (Jia & Benson, 2020; Chien et al., 2021). It is necessary to enhance message propagation such that the features and structural information of nodes at long distances can be accessed.

One straightforward way to expand message propagation is to stack more layers to build deep GCNs. Theoretically, a $k-$layer network can propagate messages from a node to the nodes at $k-$hop distance. However, increasing layers tends to cause over-smoothing (Li et al., 2018), because repeatedly applying Laplacian smoothing would mix node features and make them indistinguishable. Besides, deep networks may cause over-crashing (Alon & Yahav, 2020) and become difficult to train.

Another way to boost message propagation is to add connections. To develop deep networks, Xu et al. (2018) introduced jumping connections into feature aggregation mechanism to realize multi-layer message passing. Li et al. (2019) used residual connections to extract knowledge from long-distance nodes. On the other hand, some works attempted to combine dense connections with shallow GCNs. For example, Kampffmeyer et al. (2019) exploited the hierarchical structure of knowledge graph by a weighted dense connection scheme. SGC (Wu et al., 2019) captures higher-order information by applying the $k-$th power of graph convolution matrix in a single layer. Xu et al. (2020) used between-node paths to propagate messages. Despite the success, adding connections introduces additional parameters and further complicates models, which is most likely to cause overfitting and finally decrease accuracy, especially when labeled nodes are scarce. Moreover, as will be shown below, dense connections may introduce noise and cause message propagation saturation.

To combat the challenges, we propose a scheme for generating Enhanced Message-Passing Graph (EMPG) and train GCNs on dropout EMPGs, such that long-distance messages can be aggregated effectively in shallow GCNs, and the impact of label scarcity and graph sparsity can be avoided. Unlike the jumping connections used to link the output of low layers to the input of high layers in deep GCNs (Xu et al., 2018; Chen et al., 2020), we add connections between nodes according to their relationship in a latent space where the input graph is embedded. These added connections allow propagating messages between long-range nodes without increasing layers. Our scheme of adding connections is different from that of Kampffmeyer et al. (2019), where they directly connected the ancestors and descendants of a node to exploit the hierarchical structure of knowledge graph. In contrast, our scheme is based on node embedding, considers graph structure and links a node to distant nodes with structural similarity. Pei et al. (2020) extracted neighbors for aggregation in embedding space also. The major difference lies in the utilization of dense connections. We employed dropout to avoid side effects of dense connections, whereas they used a bi-level aggregation scheme to update node features and combated complexity by controlling the number of virtual nodes.

Adding connections introduces additional parameters and complicates GCNs, which is most likely to lead to overfitting when labeled nodes are scarce (Zhang et al., 2017; Ying, 2019). Furthermore, dense connections may bring noise in message propagation and cause message propagation saturation. Various techniques have been developed to tackle overfitting, an incomplete list includes early

stopping (Makarova et al., 2021; Perin et al., 2020), data augmentation (Devries & Taylor, 2017; Rice et al., 2020), adding statistical noise to inputs (Theunissen et al., 2019) and regularization (Tian et al., 2020; Wu et al., 2017). Dropout that was first introduced by Hinton et al. (2012) and subsequently proved to be a stochastic regularization technique by Srivastava et al. (2014) is effective. Dropout can be applied to nodes (Chen et al., 2018) or edges (Rong et al., 2020). In contrast, we apply dropout on EMPG. Seeing EMPG as an accumulation of potential variants of the original graph, we generate a series of variants by dropping EMPG edges out randomly, which is equivalent to augmenting data. Besides, it reduces noise and prevents message propagation saturation.

Our contributions are summarized as follows. i) We propose a dense connection scheme based on graph embedding for enhancing message propagation over long-range nodes. ii) We incorporate dropout with EMPG and implement GCNs on dropout EMPG successfully, which can avoid over-smoothing and prevent overfitting. iii) We demonstrate the performance of our model in contrast with other state-of-the-art methods on the task of node classification in sparse graph with few labels.

2 PRELIMINARIES AND MOTIVATION

Notations. Let $\mathcal{G} = (V, E)$ denote the undirected graph with nodes $v_i \in V$ and edges $(v_i, v_j) \in E$. $N = |V|$ is the number of nodes. The adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N\times N}$ represents the weight associated to edge (v_i, v_j) by element $A_{i,j}$. The node features are denoted as $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \in \mathbb{R}^{N\times D}$ where a $D-$dimensional feature vector $\boldsymbol{x}_i$ is associated with node v_i. A label matrix $\boldsymbol{Y} \in \{0, 1\}^{N\times C}$ is also given, where C is the number of classes. The node set V is divided into a labeled node set V_L and an unlabeled node set V_U. Usually, $|V_L| \ll |V|$. Our goal is to learn a classifier $f_{\boldsymbol{A},\boldsymbol{X},\boldsymbol{Y},v_i\in V_U}(v_i) \to y$ to predict the label of the nodes $v_i \in V_U$ based on $\boldsymbol{A}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$.

GCNs. The original GCN model was developed by Kipf & Welling (2017). The feed forward propagation in GCN is recursively conducted as:

$$\boldsymbol{H}^{(k+1)} = ReLU(\hat{\boldsymbol{A}}\boldsymbol{H}^{(k)}\boldsymbol{W}^{(k)})$$

where $\boldsymbol{H}^{(k)} = \{\boldsymbol{h}_1^{(k)}, \ldots, \boldsymbol{h}_N^{(k)}\}$ are the hidden vectors of the $k-$th layer with $\boldsymbol{h}_i^{(k)}$as the hidden feature for node v_i; $\hat{\boldsymbol{A}} = \hat{\boldsymbol{D}}^{(-1/2)}\left(\boldsymbol{A} + \boldsymbol{I}\right)\hat{\boldsymbol{D}}^{(-1/2)}$ is the re-normalization of the adjacency matrix, and $\hat{\boldsymbol{D}}$ is the corresponding degree matrix of $(\boldsymbol{A} + \boldsymbol{I})$; $\boldsymbol{W}^{(k)} \in \mathbb{R}^{\boldsymbol{D}^{(k)}\times\boldsymbol{D}^{(k-1)}}$ is the filter matrix in the $k-$th layer, where $\boldsymbol{D}^{(k)}$ refers to the size of the $k-$th layer. The GCN model of Kipf & Welling (2017) contains only two convolutional layers. The loss function $\mathcal{L}$ used to train GCN is defined as the cross-entropy error over all labeled nodes, i.e., $\mathcal{L} := -\sum_{v_i\in V_L}\sum_{C=1}^{C}\boldsymbol{Y}_{ic}Ln\boldsymbol{Z}_{ic}$.

The success of GCN has attracted more attention and different variants have emerged. For example, Wu et al. (2019) argued that nonlinearity between GCN layers is not crucial and demonstrated that the majority of the benefits arises from local weighting of neighboring features and proposed SGC:

$$\boldsymbol{H} = \hat{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}^{(0)}\boldsymbol{W}^{(1)}\ldots\boldsymbol{W}^{(k)} = \hat{\boldsymbol{A}}^{(k)}\boldsymbol{X}\widetilde{\boldsymbol{W}}$$

Besides, APPNP (Klicpera et al., 2019a) with $k-$hop aggregation is defined as:

$$\boldsymbol{H}^{(k+1)} = (1-\alpha)\hat{\boldsymbol{A}}\boldsymbol{H}^{(k)} + \alpha\boldsymbol{H}^{(0)}$$

In deep GCNs, the output of previous layers is used in combination. For example, Xu et al. (2018) combined all previous representations $[\boldsymbol{H}^{(1)}, \ldots, \boldsymbol{H}^{(k)}]$ to learn the final representation. Li et al. (2019) defined a vertex-wise concatenation function $\mathcal{T}(\ )$ to fuse the input graph $\mathcal{G}_0$ with all the intermediate GCN layer outputs:

$$\mathcal{G}_{l+1} = \mathcal{T}(\mathcal{F}(\mathcal{G}_k, \mathcal{W}_k), \ldots, \mathcal{F}(\mathcal{G}_0, \mathcal{W}_0), \mathcal{G}_0)$$

Similarly, Sun et al. (2021) combined the outputs from different orders of neighbors by AdaBoost:

$$\boldsymbol{H} = Adaboost(f_\theta^{(k)}(\hat{\boldsymbol{A}} \cdot (\hat{\boldsymbol{A}}^{(k-1)}\boldsymbol{X})))$$

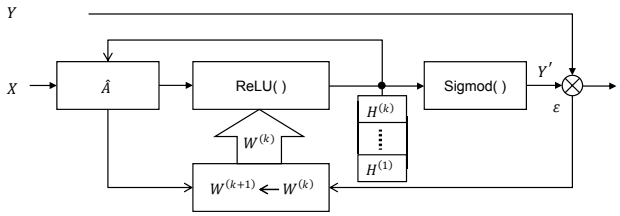

Figure 2: The organization of GCN and the recursive training process.

Motivation. We now plot the block diagram in Fig.2 that depicts the organization of GCN and the recursive process to train it. We observe that the main difference between the models described before lies in the way of using the representations $[\boldsymbol{H}^{(1)}, \ldots, \boldsymbol{H}^{(k)}]$ of each layer to learn the final representation. These works focus either on increasing the number of layers k, or on finding a way to combine $\boldsymbol{H}^{(1)}, \ldots, \boldsymbol{H}^{(k)}$. Instead, we put our effort into making the block $\hat{\boldsymbol{A}}$ more efficient for message propagation, so that the message from a node can reach the right nodes at long distances. In the GCN of Kipf & Welling (2017), $\hat{\boldsymbol{A}} = \hat{\boldsymbol{D}}^{(-1/2)} (\boldsymbol{A} + \boldsymbol{I}) \hat{\boldsymbol{D}}^{(-1/2)}$ is generated from the graph with a self-loop attached to each node. The attached self-loop can be regarded as a way of enhancing node message. In our work, we choose the nodes with structural similarity and link them directly, by which we construct a new graph called EMPG $\mathcal{G}^{'} = (V, E^{'})$. EMPG allows messages to propagate to the appropriate nodes at long distances. So, it does not need to stack many layers into GCN. As shown in Fig.1, we first map the graph $\mathcal{G} = (V, E)$ to a continuous latent space via node embedding, and then add links between the nodes that are with structural similarity to generate EMPG $\mathcal{G}^{'} = (V, E^{'})$. Seeing EMPG as an accumulation of potential variants of the original graph, we remove some edges from EMPG $\mathcal{G}^{'} = (V, E^{'})$ randomly and generate dropout EMPG $\mathcal{G}^{'}_{drop} = (V, E^{'}_{drop})$, which can be regarded as a alternative structure of the original graph. So, we augment the training data and reduce the potential risk of over-fitting. Simultaneously, by dropping edges from EMPG, we mitigate noise in message propagation and avoid message propagation saturation caused by dense links. Subsequently, we train GCNs on different dropout EMPG $\mathcal{G}^{'}_{drop} = (V, E^{'}_{drop})$ and combine them together to produce the final prediction.

3 Model implementation

3.1 Generating EMPG based on graph embedding

Graph embedding. Graph embedding transforms nodes $v_i \in V$ in graph $\mathcal{G} = (V, E)$ to low-dimensional representations $\boldsymbol{z}_i \in \mathbb{R}^d$ and effectively captures the graph topology, which is a critical step to generate EMPG. d is a small number of latent dimensions. There exist multiple methods (Perozzi et al., 2014; Yang et al., 2016; Velickovic et al., 2019) for embedding graph nodes into a latent space. Among them, DeepWalk (Perozzi et al., 2014) relies on truncated random walk and uses a skip-gram model to generate embeddings. We adopt DeepWalk to embed nodes, since it preserves the topology patterns of the graph well. Other embedding methods can also be tried out.

Add connections. According to the node representations in the latent space, we extract the structural neighborhoods $\mathcal{N}_z(v)$ for each node $v \in V$, which contains the nodes that are similar to the node v in terms of the structure, no matter whether or not they are directly linked to the node v in the graph $\mathcal{G} = (V, E)$. We define a distance function $dis(\cdot, \cdot)$ in the latent space $\mathbb{R}^d$ as:

$$dis(\boldsymbol{z}_v, \boldsymbol{z}_u) \rightarrow r \in R$$

where, r represents the structural proximity between nodes v and u in the latent space. We do not set a proximity threshold to extract the structural neighborhoods $\mathcal{N}_z(v)$, like Pei et al. (2020) did. Instead, we sort r from small to large and include the nodes in a certain range of proximity into $\mathcal{N}_z(v)$. The range is denoted as $[start, end]$ and the structural neighborhoods $\mathcal{N}_z(v)$ is defined as:

$$\mathcal{N}_z(v) = \{u | u \in V, start \leq \#dis(\boldsymbol{z}_v, \boldsymbol{z}_u) \leq end\}$$

where, $\#dis\left(\boldsymbol{z}_v, \boldsymbol{z}_u\right)$ means the position of the distance $dis\left(\boldsymbol{z}_v, \boldsymbol{z}_u\right)$ in the sorted queue.

Subsequently, we construct EMPG $\mathcal{G}^{'} = (V, E^{'})$ using $\mathcal{N}_z\left(v\right)$. If $u \in \mathcal{N}_z\left(v\right)$ and $(u, v) \notin E$, we add an edge (u, v) to $\mathcal{G} = (V, E)$. That is, $E^{'} = (u, v) \cup E$. In the EMPG $\mathcal{G}^{'} = (V, E^{'})$, a node can propagate its message to other nodes at long distances from it but with similar structure. The added edges do not change the distribution of node degree, since the number of edges added to each node is nearly same, which is about $(start - end)$. The way of adding links will not lead to the appearance of large-degree nodes also. The nodes with higher degree are more likely to suffer from over-smoothing in a multi-layer GCN model, since the repeatedly applying Laplacian smoothing will converge to be proportional to the square root of node degree (Chen et al., 2020; Li et al., 2018).

Influence range of message propagation. To watch the influence of message propagation on node classification accuracy, we define influence range of message propagation. Given a node $v_i \in V$ and a $k-$layer GCN, the node v_i can receive the messages from the nodes at $k-$hop distance in the graph. The influence that the node v_i receives from other nodes is defined as

$$influence(v_i) = influence^{pos}(v_i) - influence^{neg}(v_i)$$

where, $influence^{pos}(v_i) = \sum_{j \in V} \sum_{l \in C} \hat{A}^k_{i,j} \cdot \delta\left(Y_{i,l} = Y_{j,l}\right)$ and $influence^{neg}(v_i) = \sum_{j \in V} \sum_{l \in C} \hat{A}^k_{i,j} \cdot \delta\left(Y_{i,l} \neq Y_{j,l}\right)$ represent the influence from the nodes with the same label of v_i and from the nodes with different label of v_i respectively. $\delta\left(\star\right) = 1$ if the condition $\star$ is satisfied; Otherwise, $\delta\left(\star\right) = 0$. Furthermore, we define the influence range of message propagation, which is the ratio of the number of nodes received more messages from the nodes with the same label to the total of nodes:

$$influence\ range = \frac{\sum_{v_i \in V} \delta\left(influence\left(v_i\right) > 0\right)}{|V|}$$

3.2 Constructing GCNs on dropout EMPG

Adding edges helps message propagate to long-distance nodes without increasing the network layers. However, dense connections change the original structure and bring noise. What is worse, they are most likely to cause message propagation saturation. Moreover, the GCNs constructed on EMPG directly prone to overfit to the few training data, since each layer has numerous trainable parameters. A reasonable understanding of EMPG is to regard it as an accumulation of potential variants of the original graph, so we apply dropout to EMPG to extract the variants.

Dropout was first introduced by Hinton et al. (2012) as a way to train deep neural networks in which a collection of hidden neurons is stochastically "dropped out" at each iteration of a training procedure. It has been proven effective in controlling overfitting. Dropout can be understood as a regularizer that implicitly minimizes the expectation of a stochastic loss function based on predictions from random subnetworks. Dropout can also be seen as averaging over many neural networks with shared weights (Baldi & Sadowski, 2013). Here, dropout is used as a data argumentation technique.

We apply dropout to the edges of EMPG $\mathcal{G}^{'} = (V, E^{'})$. Each edge $(v_i, v_j) \in E$ is removed with a probability p, independent of others. A dropout EMPG is denoted by $\mathcal{G}^{'}_{drop} = (V, E^{'}_{drop})$, whose adjacency matrix $\boldsymbol{A}^{'}_{drop}$ can be expressed as $\boldsymbol{A}^{'}_{drop} = \boldsymbol{R} * \boldsymbol{A}^{'}$, where $\boldsymbol{R} \in \{0, 1\}^{|V| \times |V|}$ is a random matrix with $\boldsymbol{R}_{ij} \sim Bernoulli(1 - p)$ and $*$ denotes an element-wise product. As shown in Fig. 1, GCNs are constructed on various dropout EMPG $\mathcal{G}^{'}_{drop} = (V, E^{'}_{drop})$. For the $i-$th channel, we perform the re-normalization trick on $\boldsymbol{A}^{'}_{i\_drop}$, leading to $\hat{\boldsymbol{A}}^{'}_{i\_drop} = \hat{\boldsymbol{D}}^{(-1/2)}(\boldsymbol{A}^{'}_{i\_drop} + \boldsymbol{I})\hat{\boldsymbol{D}}^{(-1/2)}$. The feed forward propagation in the $i-$th channel GCN is recursively conducted as

$$\boldsymbol{H}^{(k+1)}_i = ReLU(\hat{\boldsymbol{A}}^{'}_{i\_drop}\boldsymbol{H}^{(k)}_i\boldsymbol{W}^{(k)}_i)$$

The features $\boldsymbol{H}^{(k)}_i$ obtained from different channels are aggregated to compute final prediction, that is, $\boldsymbol{H}^{(k)} = \sum_i \boldsymbol{H}^{(k)}_i$. Each $\boldsymbol{H}^{(k)}_i$ is computed independently and each re-normalized adjacency matrix $\hat{\boldsymbol{A}}^{'}_{i\_drop}$ can be computed in advance, thus reducing significantly training time.

Table 1: Dataset statistics

DATASET	NODES	EDGES	CLASSES	FEATURES	E-DEN	N-DEG
Cora	2708	5429	7	1433	0.148%	4.01
Citeseer	3327	4732	6	3703	0.086%	2.84

4 RELATED WORK

The efforts devoted to improving message propagation in GCNs generally fall into two categories:

i) Efforts towards building deep GCNs. A straightforward solution to realize long-range message propagation is to deepen GCNs. However, a challenge in deep GCNs is over-smoothing, which was first discussed in Li et al. (2018). To combat this problem, Xu et al. (2018) employed skip connections for multi-hop message passing to build deep JK-network. Li et al. (2019) used residual connections and dilated convolutions to deepen GCNs. Klicpera et al. (2019a;b) proposed a propagation scheme based on personalized Pagerank to aggregate information from a larger and adjustable neighbors. DAGNN (Liu et al., 2020) incorporates information from large receptive fields through the entanglement of representation transformation and propagation. GCNII (Chen et al., 2020) prevents over-smoothing by residual connections and identity mapping. Zhao & Akoglu (2020) added a normalization layer into GCNs, by which they could stack more convolutional layers. Sun et al. (2021) proposed a deep model AdaGCN by using AdaBoost to combine features of different-order neighbors. These works have produced promising results. However, stacking multiple layers increases model parameters and complexity and makes training harder in the semi-supervised setting. Furthermore, deep models with too many parameters are very prone to overfitting with label scarcity.

ii) Efforts based on dense graph connectivity scheme. On the other hand, there are attempts to improve message propagation with shallow networks. For example, Kampffmeyer et al. (2019) proposed DGP that links distant nodes directly to enhance message propagation. Zhuang & Ma (2018) proposed DGCN that jointly considers the local- and global-consistency information. Thekumparampil et al. (2018) and Velickovic et al. (2018) used attention mechanisms to improve message aggregation. SGC (Wu et al., 2019) uses $k-$th power of graph convolution matrix in a single layer to capture high-order information. Xu et al. (2020) proposed GIL that considers between-node path reachability in learning process. Pei et al. (2020) selected structural neighbors in a latent space of graph embedding for aggregation to extract long-range structural information. These models produced better results compared to the works without enhancing message propagation. Moreover, these models are usually computationally efficient because the number of layers is small.

Our method belongs to the second type. In contrast to these works, the major differences lie in the way of adding and using dense connections. We add connections according to graph embedding and keep the distribution of node degree unchanged after edge densification. Furthermore, we employ dropout to leverage the strengths of dense connections and avoid their weakness.

5 EXPERIMENTS AND DISCUSSIONS

5.1 DATASETS AND EXPERIMENTAL SETUP

We experimented with two citation datasets: Cora and Citeseer. Their statistics are reported in Table 1. Note that the average edge density and the average node degree of Citeseer are much lower than those of Cora. For each dataset, we split all instances into three parts: 1% to 5% labeled data in each class were randomly selected for training, 500 for validation, and 1000 for the test. We built a two-layer SGC using PyTorch and trained it for 600 epochs using Adam with learning rate 2e-2. The L2 regularization parameter was set to 5e-4. In addition, we used step decay schedule to drop the learning rate by 0.97 half every 60 epochs. We employed the same experimental setup when observing the influence of a certain factor on the model performance. However, we performed delicate parameter selection when trying to push node classification accuracy.

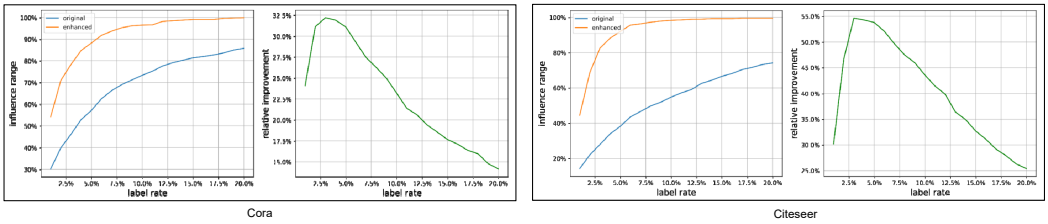

Figure 3: Comparisons of the influence range in the original graph (blue) and the enhanced graph (orange) as the training label rate increases on Cora (left) and Citeseer (right).

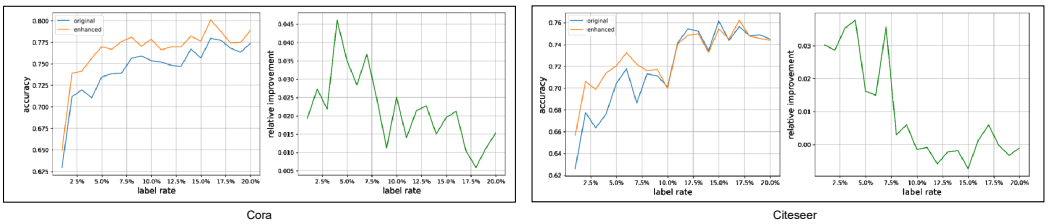

Figure 4: Accuracy obtained in the original graph (blue) and the enhanced graph (orange) as the training label rate increases on Cora (left) and Citeseer (right).

5.2 Experimental results

Enhancement of influence range. The aim of adding connections is to enlarge the influence range of message propagation. Figure 3 compares the influence range in the original graph (blue) and the enhanced graph (orange) with the increasing label rate on Cora (left) and Citeseer (right) respectively. We can observe that the influence range in the enhanced graph is always larger than in the original graph. This is as expected, since the added connections provide more paths for message propagation. When only few labeled nodes are given, the added connections lead to the rapid increase in influence range. The green curves indicate the enhancement of influence range. We can observe that the enhancement increases fast initially and drops gradually as the training label rate increases over 3%. When the label rate continuously rises above 10%, the label messages propagate to nearly the entire EMPG, reaching saturation. In addition, we can find that the enhancement on Citeseer is larger than on Cora. The reason is that the edge density of Citeseer is much lower than that of Cora. The added connections play a more important role in message propagation on Citeseer.

Accuracy of node classification. We further investigate how the accuracy of node classification benefits from the enhancement of influence range. As illustrated in Fig.4, for Cora, the accuracy obtained on EMPG (orange) is consistently better than on the original graph (blue). For Citeseer, when very few labeled nodes are given ($< 3\%$), the rise of accuracy is evident. When the label rate increases from 3% to 10%, the accuracy on EMPG is still better but the gap drops. When the label rate increases over 10%, the improvement is very limited and sometimes the accuracy may get worse. The green curves in Fig.4 indicate the rise of accuracy. Compared to the enhancement of influence range shown in Fig.3, we can easily observe that, for both datasets, the tendency of accuracy improvement is consistent with the enhancement of influence range, which confirms the effect of dense connections. Furthermore, once the influence range increases close to saturation (label rate $> 10\%$), the accuracy increases a little bit or even becomes worse.

Table 2 compares the node classification accuracy of our model and the recent state-of-the-art models metioned in section 1. The results of the compared models are taken from the relative references. All experiments are run on the standard benchmark data split used in most literatures (Yang et al., 2016). Table 2 shows that our model achieves a performance comparable to or exceeding the performance of the competitive models on Cora. However, our model outperforms the compared models by obvious margins on Citeseer. The result suggests that our model is more suitable for node classification in extremely sparse graphs with low label rate like Citeseer.

Table 2: Summary of results in terms of classification accuracy (in percent)

DATASET	DeepWalk	GCN	SGC	GAT	AGNN	TAGCN	DGCN	OURS
Cora	67.2	81.5	81.0	83.0	83.1	83.3	83.5	83.5
Citeseer	43.2	70.3	71.9	72.5	71.7	71.4	72.6	73.3

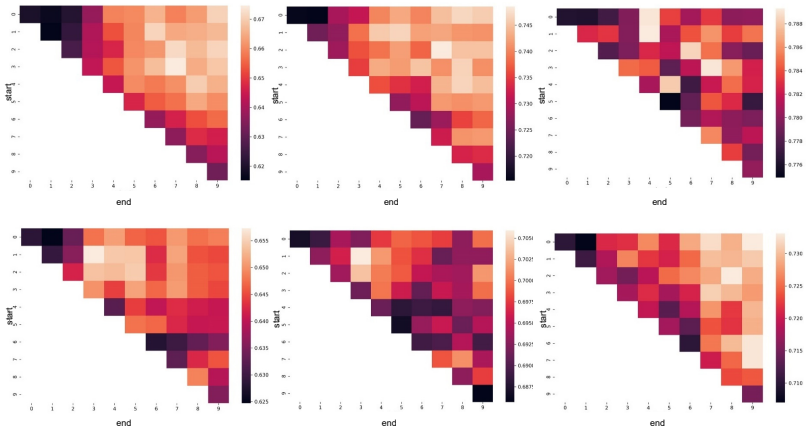

Figure 5: Accuracy for varying $start$ and end on Cora (up) and Citeseer (down) with training label rate 1% (left), 2% (middle) and 5% (right).

Effect of densification strength. The pair of parameters $[start, end]$ control the number of added connections, which reflect the densification strength. Figure 5 shows the accuracy when $start$ and end change from 0 to 9 on Cora (up) and Citeseer (down) with training label rate 1% (left), 2% (middle) and 5% (right). The block color represents the value of accuracy, with lighter colors indicating higher values and darker colors indicating lower ones. We find that, in general, good classification performance is obtained when $start$ is set a little less than the average node degree and end is set around the double of the average node degree. This is understandable, as the nodes with the closest proximity may have been linked by edges in the original graph. On the other hand, adding more than double edges will bring more noise and make message propagation saturate soon.

Effect of dropout rate. Dropout rate p is a tunable parameter that indicates the probability of removing the edges of EMPG $\mathcal{G}^{'} = (V, E^{'})$. We increased p from 0 to 0.9 with an increment of 0.1 and examined how the accuracy depends on it. Figure 6 shows the validation accuracy (blue) and the test accuracy (yellow) for varying p on Cora (up) and Citeseer (down) with training label rate 1% (left), 2% (middle) and 5% (right). We can observe that as p increases, both accuracies on Cora continue to increase till p reaches over 0.8, no matter with training label rate 1% and 2%. For the training label rate 5%, p could be set to a larger value. Meanwhile, both accuracies on Citeseer drop continuously with the increasing p for the training label rate 1% and 2%. For the training label rate 5%, both accuracies on Citeseer increase till $p = 0.4$ and then take a turn for the worse.

Considering the average node degree and the edge density of both datasets, the densification strength and the training label rate jointly, we may be able to give some clues to the complex phenomenon. These factors are working together to determine message propagation. No matter which factor changes, if it expands influence range, the accuracy will increase. Otherwise, the accuracy will drop. Adding connections enhances message propagation but may lead to message propagation saturation with high label rate. Meanwhile, dropping edges reduces noise, prevents message propagation saturation, augments training data and mitigates overfitting. The two seemingly contradictory operations play in different roles, which actually complement one another to raise accuracy.

Assessment of robustness. To assess model robustness, we randomly selected 10% to 50% samples from training dataset and changed their label. Figure 7 shows the accuracy obtained in the original graph (yellow) and EMPG (blue) for varying noise level on Cora (up) and Citeseer (down), given

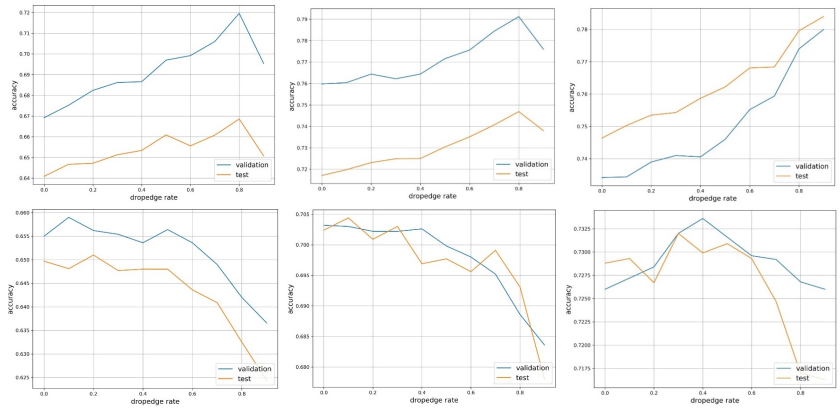

Figure 6: Validation accuracy (blue) and test accuracy (yellow) for varying dropout rate p on Cora (up) and Citeseer (down) with training label rate 1% (left), 2% (middle) and 5% (right).

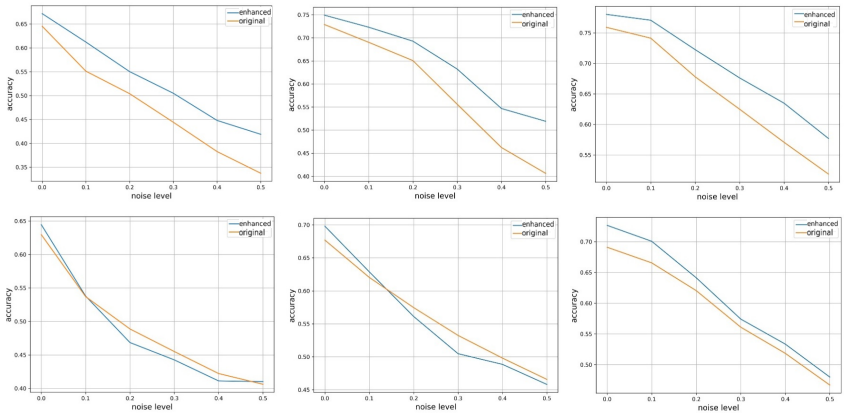

Figure 7: Accuracy obtained in original (yellow) and enhanced graph (blue) for varying noise level on Cora (up) and Citeseer (down), given training label rate 1% (left), 2% (middle) and 5% (right).

three training label rate 1% (left), 2% (middle) and 5% (right). It is clear that the accuracy decreases as noise level rises. However, for Cora, the accuracy obtained on EMPG is consistently better, and the gap is obvious and enlarges as noise level increases. For Citeseer, for the low training label rate 1% and 2%, the accuracy obtained on EMPG fluctuates up and down around that obtained on the original graph. For the label rate 5%, the accuracy obtained on EMPG is always better, but the gap decreases as the noise level increases. In short, the model built on EMPG is more robust.

6 Conclusion and Future Work

In this work, we proposed a new GCN framework to address semi-supervised node classification in sparse graph with few labels, whose distinguishing feature is a dense connection scheme based on graph embedding, by which a node message can propagate over long distances and towards the right nodes without the need of stacking multiple layers. This is very useful for avoiding problems like over-smoothing and computational complexity in deep networks. By incorporating with dropout technique, our model mitigates the negative effects of dense connections such as message propagation saturation and augments training data, thus prevents overfitting and raises accuracy. Experiments on benchmark datasets demonstrate the effectiveness of the proposed method. In future work, we plan to explore techniques for adding connections adaptively and dynamically. It will be worthwhile to model the relationship among graph properties, densification strength and message propagation range. We will also apply the proposed method to more real applications.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *CoRR*, abs/2006.05205, 2020.

Pierre Baldi and Peter J. Sadowski. Understanding dropout. In *NIPS*, pp. 2814–2822, 2013.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR (Poster)*. OpenReview.net, 2018.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *CoRR*, abs/2007.02133, 2020.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*. OpenReview.net, 2021.

Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.

Jian Du, Shanghang Zhang, Guanhang Wu, José M. F. Moura, and Soummya Kar. Topology adaptive graph convolutional networks. *CoRR*, abs/1710.10370, 2017.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

Junteng Jia and Austin R. Benson. Outcome correlation in graph neural network regression. *CoRR*, abs/2002.08274, 2020.

Michael Kampffmeyer, Yinbo Chen, Xiaodan Liang, Hao Wang, Yujia Zhang, and Eric P. Xing. Rethinking knowledge graph propagation for zero-shot learning. In *CVPR*, pp. 11487–11496. Computer Vision Foundation / IEEE, 2019.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR (Poster)*. OpenReview.net, 2019a.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, pp. 13333–13345, 2019b.

Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 9266–9275. IEEE, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. *CoRR*, abs/1801.07606, 2018.

Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard S. Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *ICLR (Poster)*. OpenReview.net, 2019.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. *CoRR*, abs/2007.09296, 2020.

Anastasia Makarova, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias W. Seeger, and Cédric Archambeau. Overfitting in bayesian optimization: an empirical study and early-stopping solution. *CoRR*, abs/2104.08166, 2021.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *CoRR*, abs/2002.05287, 2020.

Guilherme Perin, Ileana Buhan, and Stjepan Picek. Learning when to stop: a mutual information approach to fight overfitting in profiled side-channel analysis. Cryptology ePrint Archive, Report 2020/058, 2020. `https://ia.cr/2020/058`.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *KDD*, pp. 701–710. ACM, 2014.

Leonardo F. R. Ribeiro, Pedro H. P. Savarese, and Daniel R. Figueiredo. struc2vec: Learning node representations from structural identity, 2017. URL http://arxiv.org/abs/1704.03165.

Leslie Rice, Eric Wong, and J. Zico Kolter. Overfitting in adversarially robust deep learning. *CoRR*, abs/2002.11569, 2020.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Ke Sun, Zhanxing Zhu, and Zhouchen Lin. Adagcn: Adaboosting graph convolutional networks into deep models. In *ICLR*. OpenReview.net, 2021.

Kiran Koshy Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *CoRR*, abs/1803.03735, 2018.

Marthinus W. Theunissen, Marelie H. Davel, and Etienne Barnard. Insights regarding overfitting on noise in deep learning. In *FAIR*, volume 2540 of *CEUR Workshop Proceedings*, pp. 49–63. CEUR-WS.org, 2019.

Kai Tian, Yi Xu, Jihong Guan, and Shuigeng Zhou. Network as regularization for training deep neural networks: Framework, model and performance. In *AAAI*, pp. 6013–6020. AAAI Press, 2020.

Petar Velickovic, Guillem Cucurull, A. Casanova, A. Romero, P. Liò, and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2018.

Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Bingzhe Wu, Zhichao Liu, Zhihang Yuan, Guangyu Sun, and Charles Wu. Reducing overfitting in deep convolutional neural networks using redundancy regularizer. In *ICANN (2)*, volume 10614 of *Lecture Notes in Computer Science*, pp. 49–55. Springer, 2017.

Felix Wu, Tianyi Zhang, Amauri H. Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. *CoRR*, abs/1902.07153, 2019.

Chunyan Xu, Zhen Cui, Xiaobin Hong, Tong Zhang, Jian Yang, and Wei Liu. Graph inference learning for semi-supervised classification. *CoRR*, abs/2001.06137, 2020.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5449–5458. PMLR, 2018.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings., 2016. URL http://arxiv.org/abs/1603.08861.

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *AAAI*, pp. 7370–7377. AAAI Press, 2019.

Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022, 2019.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*. OpenReview.net, 2017.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *ICLR*. OpenReview.net, 2020.

Chenyi Zhuang and Qiang Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *WWW*, pp. 499–508. ACM, 2018.