Progressive Supervision for Node Classification

Yiwei Wang[1] (✉), Wei Wang[1], Yuxuan Liang[1], Yujun Cai[2], and Bryan Hooi[1]

[1] School of Computing, National University of Singapore, Singapore
{y-wang,wangwei,yuxliang,bhooi}@comp.nus.edu.sg
[2] Nanyang Technological University, Singapore
yujun001@e.ntu.edu.sg

Abstract. Graph Convolution Networks (GCNs) are a powerful approach for the task of node classification, in which GCNs are trained by minimizing the loss over the final-layer predictions. However, a limitation of this training scheme is that it enforces every node to be classified from the fixed and unified size of receptive fields, which may not be optimal. We propose ProSup (Progressive Supervision), that improves the effectiveness of GCNs by training them in a different way. ProSup supervises all layers progressively to guide their representations towards the characteristics we desire. In addition, we propose a novel technique to reweight the node-wise losses, so as to guide GCNs to pay more attention to the nodes that are hard to classify. The hardness is evaluated progressively following the direction of information flows. Finally, ProSup fuses the rich hierarchical activations from multiple scales to form the final prediction in an adaptive and learnable way. We show that ProSup is effective to enhance the popular GCNs and help them to achieve superior performance on miscellaneous graphs.

Keywords: Graph Convolutional Networks · Progressive Supervision · Node Classification

1 Introduction

Node classification is a fundamental task on graph data, which aims to classify the nodes in an (attributed) graph [14]. For this task, Graph Convolutional Networks (GCNs) have achieved state-of-the-art performance [23]. Typically, GCNs follow a multi-layer structure (see Fig. 1(a)). Across layers, GCNs update node representations via the 'message-passing' mechanism, i.e., they aggregate the representations of each node and its neighbors to produce new ones at the next layer. Denote the subgraph contributing to a node's representation as its receptive field. From bottom to top, the receptive field expands gradually, which is generally a node's l-hop neighborhood at the lth layer [19].

For training GCNs, it is common to minimize the classification loss on the final-layer predictions. This training scheme is convenient, but not necessarily ideal for effectiveness. One limitation is that it enforces GCNs to classify all nodes from the unified size of receptive fields, but nodes can have diverse 'appropriate' receptive fields for classification [24]. In a social network, for example,

famous people include much noise with a small number of layers (hops), while freshmen may need more layers to include the relevant features. Another limitation is about the discriminativeness of learned features. [13] demonstrates that a classifier will perform better when trained on more discriminative features. However, minimizing the loss only on the final-layer predictions ignores the discriminativeness of hidden layers and may degrade the performance.

The central idea of this paper is to make predictions separately on each layer and progressively supervise hidden layers from bottom to top (see Fig. 1(b)). We encapsulate this idea in a new scheme, called ProSup (short for Progressive Supervision), that produces the side-outputs on hidden layers, and then fuse the multi-level, multi-scale activations to form the unified prediction. With our design, a node is classified at a side-output from an 'appropriate' receptive field instead of absorbing extra noise when propagating its representation to the final layer. Besides, our supervision over hidden layers acts as a kind of regularization. It regularizes hidden layers to produce discriminative and meaningful representations, rather than supervises only the final-layer representations. Note that ProSup can be incorporated into popular GCN architectures, e.g., GCN [11], LGCN [8], GraphSAGE [9], etc., for better effectiveness.

Each layer has some nodes easy to classify and some hard ones (see Fig. 2). To leverage the information of classification hardness, we propose a technique to progressively reweight the node-wise losses. Following the direction of information flows, we encourage GCNs to pay more attention to hard nodes. In principle, we give the nodes, that are classified incorrectly at a layer, larger weights at the next layer. This technique facilitates communication across GCN layers to mine the hard nodes progressively. In terms of gradient propagation, ProSup simultaneously minimizes the classification losses on the side-outputs and the fused prediction. The former propagates the local errors to the corresponding layer, while the latter directly supervises all layers globally. We use both of them to train the model holistically so as to produce discriminative representations and the finer fused prediction.

We evaluate ProSup on the node classification task using the Citeseer, Cora, Pubmed [14], Flickr [15], Yelp [25], and Reddit [9] datasets. Qualitatively, ProSup makes the class-specific representations more concentrated (see Fig. 4). We also observe quantitative improvements evaluated by test accuracy and F1-micro scores. Overall, ProSup improves the popular GCN [11], LGCN [8], GraphSAGE [9] and GraphSAINT [25] models by a significant margin, and enhances them to outperform the benchmark methods. As we analyze, our ProSup improves the effectiveness of GCNs without changing the time complexity.

2 Related Work

Due to the long history of Graph Neural Networks, we refer readers to [23] and [27] for a comprehensive review. The first work that proposes the convolution operation on graph data is [2]. More recently, [11] and [7] speed up the graph convolution operations by introducing localized filters based on Chebyshev ex-

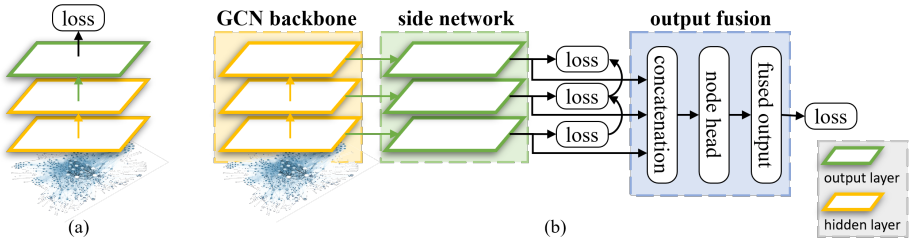
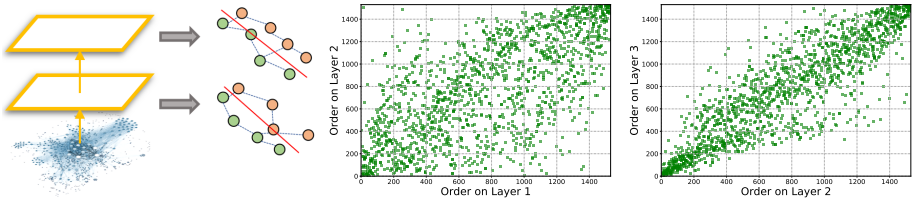


Fig. 1: **ProSup applied to node classification.** (a) Standard GCN takes an input graph and yields predictions on the final layer, where the loss is calculated. (b) ProSup supervises hidden layers progressively and fuses the outputs from multiple layers to produce the unified prediction. We minimize the classification losses on both the side-outputs and the unified output simultaneously.

pansion. Specifically, [11] has made breakthrough advancements in the task of node classification. As a result, the model proposed in [11] is generally denoted as the vanilla GCN, or GCN. After [11], numerous GCN methods are proposed for better performance on node classification. There are two main lines of research in this field. The first one is to propose new GCN architectures to improve the model capacity. The second is to propose mini-batch techniques for GCNs to achieve better scalability without loss of effectiveness.

To improve the model capacity, [21], [26], and [10] use the attention mechanism to better capture neighbor features by dynamically adjusting edge weights. Mixture Model Network (MoNet) [16] adopts a different approach to assign edge weights. It introduces node pseudo-coordinates to determine the relative position between a node and its neighbors, then defines a weight function to map the relative positions to edge weights. [28] utilizes the positive pointwise mutual information (PPMI) matrix to capture nodes co-occurrence information through random walks sampled from a graph. [12] combines PageRank with GCNs to enable efficient information propagation. [22] alternatively drives local network embeddings to capture global structural information by maximizing local mutual information. [4] proposes a non-uniform graph convolutional strategy, which learns different convolutional kernel weights for different neighboring nodes according to their semantic meanings. LGCN [8] ranks a node's neighbors based on node features. It assembles a feature matrix that consists of its neighborhood and sorts this feature matrix along each column. [18] proposes GMNN to model the dependency of object labels through statistical relational learning.

Another line of research focuses on scaling GCNs to large graphs efficiently and effectively. GraphSAGE [9] performs uniform node sampling on the previous layer neighbors. It enforces a pre-defined budget on the sample size, so as to bound the mini-batch computation complexity. [5] further restricts neighborhood size by requiring only two support nodes in the previous layer. Instead of sampling layers, ClusterGCN [6] and GraphSAINT [25] build mini-batches from subgraphs, so as to avoid the 'neighbor explosion' problem.



(a) Diverse 'Preferences' (b) Layer 1 - Layer 2 (c) Layer 2 - Layer 3

Fig. 2: **GCN layers have diverse 'preferences' on node classification.** (a) GCN layers produces various node representation, which lead to various optimal decision boundaries (red lines). (b, c) Descending order of the node-wise losses in `Citeseer`. (b) Each point represents a node. x-value is the node order on layer 1, and y is for layer 2. (c) is similar for layers 2 (x) and 3 (y). If the 'preferences' are the same, all the points in (b) and (c) concentrate on the line of $x = y$.

Our work is orthogonal to the two lines above in the sense that it neither alters the GCN architecture for better capacity nor introduces a new mini-batch technique. We propose a new scheme that takes a GCN architecture as the backbone to enhance its effectiveness without changing the time complexity. Meanwhile, existing mini-batch techniques can be applied to our scheme for better scalability. Our scheme is inspired by deeply-supervised nets [13], that perform deep supervision to 'guide' early classification results. We find that the favorable characteristics of feature representations provided by ProSup lead to more accurate predictions.

3 Methodology

In this section, we describe our proposed ProSup scheme for node classification. We first introduce the background and mathematical notations. Next, we introduce the modules of ProSup, including the GCN backbone, the side network, and output fusion, as illustrated in Fig. 1(b). Finally, we analyze why ProSup can improve the performance of GCNs.

3.1 Background and Motivation

We define a graph as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of nodes, and $\mathcal{E}$ is the set of edges. The input feature vector of node i is $\mathbf{x}_i$, and the neighborhood of node i is $\mathcal{V}_i = \{j \in \mathcal{V} | (i,j) \in \mathcal{E}\}$. Suppose the representation of node i at layer l is $\mathbf{h}_i^{(l)}$. Typically, a GCN layer obtains $\mathbf{h}_i^{(l)}$ as:

$$\mathbf{h}_i^{(l)} = \mathsf{AGGREGATE}\left(\mathbf{h}_i^{(l-1)}, \left\{\mathbf{h}_j^{(l-1)}, j \in \mathcal{V}_i\right\}, \mathbf{W}^{(l)}\right), \tag{1}$$

where $\mathbf{W}^{(l)}$ denotes the trainable weights at layer l, and AGGREGATE is an aggregation function defined by the specific GCN model. $\mathbf{h}_i^{(0)} = \mathbf{x}_i$ holds at the input layer.

For the task of node classification, GCNs learns the high-level semantic representations by stacking L layers and minimizing the loss, e.g. cross entropy [1], over the final-layer outputs, as presented in Fig. 1(a). Denote the subgraph contributing to a node's representation as its receptive field, which is generally a node's l-hop neighborhood at layer l [19]. However, it has been found that increasing L does not necessarily improve the effectiveness of GCNs, even though it expands the receptive fields for classification. Popular GCN models observe empirically optimal performance with a small L, e.g. 2 or 3 [11]. A reason is that nodes have diverse 'appropriate' receptive field sizes for effective classification [24]. Larger receptive fields can introduce extra noise, while smaller ones miss valuable information. In addition, each GCN layer has some nodes easy to classify and others hard. Hence, a GCN layer may be effective at classifying some nodes, but not others. This 'preference' varies with l, i.e., various receptive fields, as shown in Fig. 2. Thus, minimizing the classification loss on the final-layer outputs, i.e., enforcing GCNs to classify all the nodes based on the fixed and unified receptive field of L-hop neighborhoods, inevitably introduces noise. The larger L is, the more noise is introduced, which may lead to decreased accuracy.

In our work, we propose our progressive supervision (ProSup) scheme to enhance GCN models for node classification. As depicted in Fig.1(b), the core idea is to supervise the feature maps on hidden layers progressively, and fuse the side-outputs from multiple levels and multiple scales to produce the unified output in an adaptive and learnable way. We combine the rich hierarchical responses, that allow each node to be classified at its 'appropriate' receptive fields, from different layers. For training, we minimize the classification losses over both the side-outputs and the fused output simultaneously. Next, we introduce the details of ProSup.

3.2 GCN Backbone and Side Network

We take an existing GCN model as the backbone, e.g., GCN, GAT, LGCN, etc., as shown in Fig. 1(b). This backbone typically has a multi-layer structure, of which the aggregation mechanism is introduced in Sec. 3.1. We construct a side network parallel to the GCN backbone for conducting progressive supervision. Following the notations in Eq. (1), we denote node representations in layer l as $\mathbf{H}^{(l)} \in \mathbb{R}^{N \times d_l}$, where $N = |\mathcal{V}|$ is the number of nodes, and d_l is the dimension of representations in layer l. As defined in Eq. (1), the representation of node i, $\mathbf{h}_i^{(l)}$, is the ith row of $\mathbf{H}^{(l)}$. Accordingly, the input feature vectors $\mathbf{x}_i$ form the rows of $\mathbf{X} = \mathbf{H}^{(0)}$. We produce the side-outputs at layer l as:

$$\mathbf{A}^{(l)} = \mathbf{H}^{(l)}\mathbf{W}_{side}^{(l)}, \quad \hat{\mathbf{Y}}^{(l)} = \sigma\left(\mathbf{A}^{(l)}\right), \tag{2}$$

where $\mathbf{A}^{(l)}$, $\mathbf{W}_{side}^{(l)} \in \mathbb{R}^{d_l \times C}$, and $\hat{\mathbf{Y}}^{(l)} \in \mathbb{R}^{N \times C}$ are the activation, trainable weights, and the prediction respectively. C is the number of classes. $\sigma(\cdot)$ is the nonlinear activation function used to produce probabilities, which is softmax for single-label classification and sigmoid for multi-label classification [1].

Unlike existing GCN architectures that produce outputs only at their final layer, we build an output layer after each convolutional layer. The receptive fields of these side-outputs become larger as l increases. Thus, we enable hidden layers to produce predictions by themselves, instead of only contributing to the final-layer prediction indirectly through feed-forward propagation. As a result, the nodes that are easier to classify with a smaller receptive field are classified earlier, rather than absorbing more noise through further aggregations. In addition, the hidden layers are expected to produce more discriminative representations through our direct supervision.

3.3 Output Fusion and Loss Function

To produce the unified prediction, we construct node-wise features by combining (e.g., concatenating) the activations from multiple layers, $\left\{\mathbf{A}^{(l)}\right\}_l$. Given the node-wise feature representation, we make node-wide class predictions using a light-weight node head, implemented as a multi-layer perception (MLP). The node head shares weights across all nodes, analogous to PointNet [17]:

$$\mathbf{A} = \delta\left(\mathsf{CONCAT}\left(\left\{\mathbf{A}^{(l)}\right\}_l\right)\mathbf{W}_{fuse}^{(1)}\right)\mathbf{W}_{fuse}^{(2)}, \quad \hat{\mathbf{Y}} = \sigma\left(\mathbf{A}\right), \tag{3}$$

where $\mathbf{W}_{fuse}^{(1)} \in \mathbb{R}^{LC \times d_{fuse}}$, $\mathbf{W}_{fuse}^{(2)} \in \mathbb{R}^{d_{fuse} \times C}$ are the learnable weights for fusing the multi-level activations, while $\mathbf{A}$ is the fused activation for the unified output. $\delta(\cdot)$ is the activation function, which we set as ReLU. During inference, we take $\hat{\mathbf{Y}}$ as the unified prediction. During training, we supervise both the side-outputs and the fused output. The loss on side-outputs is:

$$\mathcal{L}_{side} = -\sum_l \sum_{i \in \mathcal{V}_L} \sum_{c=1}^{C} Y_{ic} \log \hat{Y}_{ic}^{(l)}, \tag{4}$$

where $\mathcal{V}_L$ is the set of node indices that have labels in the training set, and Y_{ic} is the binary ground-truth label value of node i. $Y_{ic} = 1$ indicates node i belongs to class c, and $Y_{ic} = 0$, otherwise.

As for the fused prediction $\hat{\mathbf{Y}}$, we calculate the loss:

$$\mathcal{L}_{fuse} = -\sum_{i \in \mathcal{V}_L} \sum_{c=1}^{C} Y_{ic} \log \hat{Y}_{ic}. \tag{5}$$

Putting these together, we minimize the following objective function:

$$\mathcal{L} = \mathcal{L}_{fuse} + \alpha \mathcal{L}_{side}, \tag{6}$$

where α is the hyper-parameter for balancing $\mathcal{L}_{fuse}$ and $\mathcal{L}_{side}$.

From the perspective of gradient propagation, minimizing $\mathcal{L}_{side}$ over a side-output propagates the local errors to the corresponding layer, while minimizing $\mathcal{L}_{fuse}$ propagates the global errors to all layers simultaneously. The former supervises each layer to generate semantically meaningful predictions and regularizes GCNs to learn consistently discriminative representations across hidden layers. This endows the final prediction with the higher quality given the better hidden representations. Depending on the receptive field size, the side-output of any single layer may be coarse. For example, lower layers have only localized features, while higher layers may give over-smooth outputs [20]. Thus, we fuse the (coarse) side-outputs to form the unified (fine) output. This process of combining multi-level and multi-scale activations is learnable and adaptive, by which the final prediction of each node can be obtained from an 'appropriate' receptive field.

3.4 Progressively Re-weighting Hard Nodes

At each layer, some nodes are easy to classify, while others are hard, as shown in Fig. 2. Giving all the nodes the same weight in the loss function $\mathcal{L}_{side}$ can lose the information on the classification hardness. To address this limitation, we design a new scheme to progressively reweight the hard nodes. As shown in Fig. 1, on the losses computed on side-outputs, from bottom to top, we conduct a node re-weighting operation layer by layer. This bottom-up hierarchical mechanism is inspired by the sequential information flow across hidden layers of GCNs. For example, the third layer of the side network needs the feature map from the second layer to compute the classification results and loss. If the lower layers can inform the upper layers which nodes are hard to classify, the higher layers will pay more attention to these hard nodes. Through this communication, hard examples can be mined and classified more effectively from bottom to top, resulting in a better fused prediction.

Specifically, given $\hat{\mathbf{Y}}^{(l)}$ as the output from the l th layer of the side network, we denote the differences of node i between the prediction on layer l and the ground truth label as:

$$\xi_i^{(l)} = \sum_{c=1}^{C} \left| \hat{Y}_{ic}^{(l)} - Y_{ic} \right| \tag{7}$$

Then, we rewrite the loss function in Eq. (4) to:

$$\begin{aligned} \mathcal{L}_{side-reweight} = & - \sum_{i \in \mathcal{V}_L} \sum_{c=1}^{C} Y_{ic} \log \hat{Y}_{ic}^{(1)} \\ & - \sum_{l=2}^{L} \sum_{i \in \mathcal{V}_L} \frac{|\mathcal{V}_L|}{\sum_{i \in \mathcal{V}_L} |\xi_i^{(l-1)}|} |\xi_i^{(l-1)}| \sum_{c=1}^{C} Y_{ic} \log \hat{Y}_{ic}^{(l)}, \end{aligned} \tag{8}$$

Algorithm 1 ProSup: progressive supervision for enhancing Graph Convolutional Networks (GCNs) on node classification.

Input: Graph $G = (\mathcal{V}, \mathcal{E})$, Feature Matrix $\mathbf{X}$, a GCN backbone with the aggregation function $\mathsf{AGGREGATE}(\cdot)$, hyper-parameter α for balancing losses, d_{fuse} for the node head, ground-truth labels $\mathbf{Y}$, labeled nodes in the training set $\mathbf{V}_L$.

Output: Fused prediction $\hat{\mathbf{Y}}$, trained parameters of the GCN backbone $\left\{\mathbf{W}^{(l)}\right\}_l$, and trained parameters of ProSup, $\left\{\mathbf{W}^{(l)}_{side}\right\}_l$, $\mathbf{W}^{(1)}_{fuse}$, and $\mathbf{W}^{(2)}_{fuse}$.

Initialize all parameters.
while $\mathcal{L}$ does not converge **do**
  $\mathbf{H}^{(0)} \leftarrow \mathbf{X}$
  for $l \leftarrow 1$ to L **do**
    for $i \leftarrow 1$ to $|\mathcal{V}|$ **do**
      $\mathbf{h}_i^{(l)} \leftarrow \mathsf{AGGREGATE}\left(\mathbf{h}_i^{(l-1)}, \left\{\mathbf{h}_j^{(l-1)}, j \in \mathcal{V}_i\right\}, \mathbf{W}^{(l)}\right)$
    end for
    $\mathbf{A}^{(l)} \leftarrow \mathbf{H}^{(l)}\mathbf{W}^{(l)}_{side}$
    $\hat{\mathbf{Y}}^{(l)} \leftarrow \sigma\left(\mathbf{A}^{(l)}\right)$
    for $i \leftarrow 1$ to $|\mathcal{V}_L|$ **do**
      $\xi_i^{(l)} \leftarrow \sum_{c=1}^{C}\left|\hat{Y}^{(l)}_{ic} - Y_{ic}\right|$
    end for
  end for
  $\mathbf{A} \leftarrow \delta\left(\mathsf{CONCAT}\left(\left\{\mathbf{A}^{(l)}\right\}_l\right)\mathbf{W}^{(1)}_{fuse}\right)\mathbf{W}^{(2)}_{fuse}$
  $\hat{\mathbf{Y}} \leftarrow \sigma(\mathbf{A})$
  Calculate $\mathcal{L}_{side}$ from Eq. (8)
  $\mathcal{L}_{fuse} \leftarrow -\sum_{i\in\mathcal{V}_L}\sum_{c=1}^{C} Y_{ic}\log\hat{Y}_{ic}$
  $\mathcal{L} \leftarrow \mathcal{L}_{fuse} + \alpha\mathcal{L}_{side}$
  Back-propagation for minimizing $\mathcal{L}$
end while

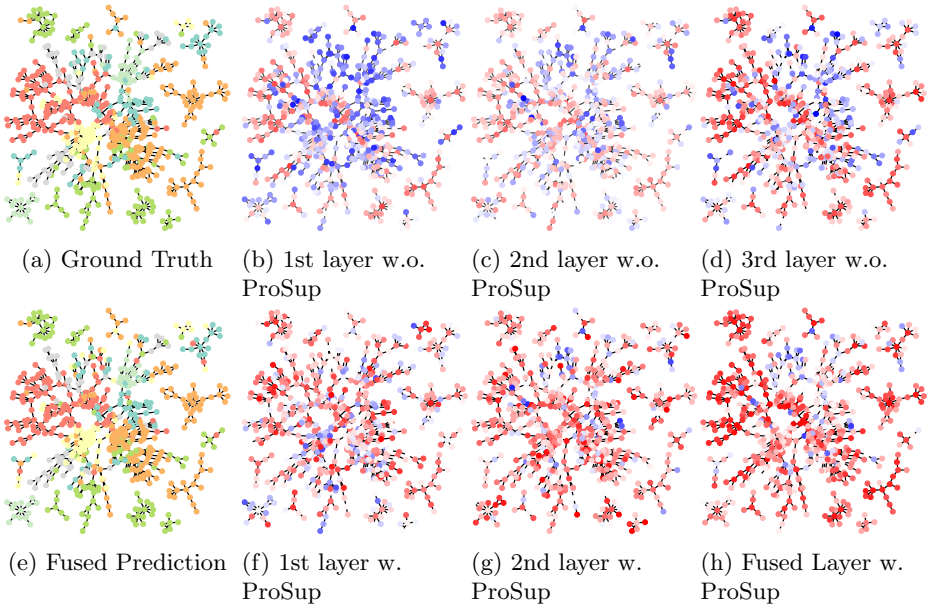
where $\frac{|\mathcal{V}_L|}{\sum_i|\xi_i^{(l-1)}|}|\xi_i^{(l-1)}|$ is the normalized node weight. $\frac{|\mathcal{V}_L|}{\sum_i|\xi_i^{(l-1)}|}$ helps to ensure that, $\forall\, l$, the average loss weight of all the labeled nodes in the training set is

$$\frac{1}{|\mathcal{V}_L|}\sum_{i\in\mathcal{V}_L}\frac{|\mathcal{V}_L|}{\sum_{i\in\mathcal{V}_L}|\xi_i^{(l-1)}|}|\xi_i^{(l-1)}| = \frac{|\mathcal{V}_L|}{|\mathcal{V}_L|}\sum_{i\in\mathcal{V}_L}\frac{|\xi_i^{(l-1)}|}{\sum_{i\in\mathcal{V}_L}|\xi_i^{(l-1)}|} = 1.$$

Thus, with the progressive re-weighting scheme, we leverage the predictions of a shallower side network to weight loss terms in a deeper side network. This facilitates communication among hidden layers and provide better activations $\{\mathbf{A}^l\}_l$ for producing the final prediction effectively. We provide pseudo-code for ProSup in Alg. 1.

3.5 Discussion

ProSup has several advantages over the classical GCN training scheme that minimizes the loss on the final-layer outputs. With ProSup, different nodes have



(a) Ground Truth (b) 1st layer w.o. ProSup (c) 2nd layer w.o. ProSup (d) 3rd layer w.o. ProSup

(e) Fused Prediction (f) 1st layer w. ProSup (g) 2nd layer w. ProSup (h) Fused Layer w. ProSup

Fig. 3: **ProSup enhances GCN to provide better predictions.** In the first column, colors denote classes of nodes in the `Cora` dataset. (a) shows the ground truth labels. (e) is the fused prediction given by GCN with ProSup. In the other columns, colors denote the predicted score on the correct class. Colors closer to red mean that the score is closer to 1 (correct). Otherwise, the blue color corresponds to the score of 0 (incorrect). ProSup enhances GCN to provide hidden layers with better predictions, improving the final prediction.

adaptively 'appropriate' receptive fields for classification, since the corresponding hidden layers can produce the outputs by themselves rather than pass the representations to the next layers, which may aggregate extra noise. Second, a discriminative classifier trained on highly discriminative features will perform better than that on less discriminative features [13]. By training the classifier constructed at each hidden layer of a GCN, we are able to progressively influence the weight update process of hidden layers to favor highly discriminative feature maps. This is a source of supervision that acts within the GCN at each layer: we expect the final classification predictions to be more reliable than the case of relying on backpropagation from the final layer alone.

Moreover, ProSup can also be seen as a form of regularization, as GCN learns consistently discriminative features among hidden layers instead of overfitting the training data by minimizing the classification loss solely on the final layer. Finally, in terms of explainability, ProSup provides a classifier for each hidden layer to endow their representations with clear semantic information, rather than treating GCNs as a black box with complicated and non-interpretable input-output relations. An example of node classification by GCN with ProSup is

Table 1: Statistics of the utilized datasets. 'm' stands for multi-label classification, while 's' for single-label.

Dataset	`Citeseer`	`Cora`	`Pubmed`	`Flickr`	`Yelp`	`Reddit`
#Nodes	3,327	2,708	19,717	89,250	716,847	232,965
#Edges	4,732	5,429	44,338	899,756	6,977,410	11,606,919
#Features	3,703	1,433	500	500	300	602
#Classes	7 (s)	6 (s)	3(s)	7 (s)	100 (m)	41 (s)

presented in Fig. 3. The predictions on a hidden layer of GCN without ProSup are obtained by training a fully-connected layer using the learned representations.

4 Complexity Analysis

We train the model in the end-to-end style. The time complexity of GCN is $\mathcal{O}\left(|\mathcal{E}|\sum_{l=1}^{L} d_l + |\mathcal{V}|\sum_{l=1}^{L} d_{l-1}d_l\right)$. With the side network, we have the time complexity as $\mathcal{O}\left(|\mathcal{V}|\sum_{l=1}^{L} d_l C\right)$. In the output fusion module, we have the complexity $\mathcal{O}\left(|\mathcal{V}|LCd_{fuse}\right)$. Taking all the computation into consideration, we have the complexity of $\mathcal{O}\left(|\mathcal{E}|\sum_{l=1}^{L} d_l + |\mathcal{V}|\left(\sum_{l=1}^{L} d_l d_{l-1} + C\left(d_l + d_{fuse}\right)\right)\right)$. The complexity is linear to $|\mathcal{E}|$ and $|\mathcal{V}|$, same as in GCNs. Note that $C < d_l, \forall\ l$ holds generally. Thus, using ProSup to improve the effectiveness of GCNs does not increase their time complexity.

5 Experiment

In this section, we present the empirical improvements over various GCN architectures achieved by ProSup. We report the experimental results under both the transductive and inductive settings. In addition, we visualize the learned representations of GCN with ProSup compared with the GCN without ProSup. Finally, we conduct ablation studies to show the influence of different components of ProSup, as well as the sensitivity with respect to the hyper-parameters of ProSup.

We use standard benchmark datasets: Cora, Citeseer, Pubmed [14], Flickr [15], Yelp [25], and Reddit [9] for evaluation. The former three are citation networks, where each node is a document and each edge is a citation link. In Flickr, each node represents one image. An edge is built between two images if they share some common properties (e.g. same geographic location, same gallery, etc.). The Yelp dataset contains a social network, where an edge means that the connected users are friends. Reddit is collected from an online discussion forum where users comment in different topical communities. Two posts (nodes) are connected if some user comments on both posts. Each dataset contains an

Table 2: Test Accuracy (%) of transductive node classification. #Layers indicates the best-performing number of layers among 1 to 8. We conduct 100 trials with random weight initialization. The mean and standard derivations are reported.

Method	#Layers	`Citeseer`	#Layers	`Cora`	#Layers	`Pubmed`
GCN [11]	2	77.1±1.4	2	88.3±0.8	3	86.4±1.1
GAT [21]	2	76.3±0.8	3	87.6±0.5	3	85.7±0.7
JKNet-MaxPool [24]	1	77.4±0.6	6	89.5±0.8	3	86.5±0.9
JKNet-Concat [24]	1	78.1±0.9	6	89.1±1.2	4	86.9±1.3
JKNet-LSTM [24]	1	74.7±0.8	1	86.1±0.9	1	85.8±1.2
LGCN [8]	2	77.5±1.1	2	89.0±1.2	2	86.5±0.6
GMNN [18]	2	77.4±1.5	2	88.7±0.8	2	86.7±1.0
ProSup + GCN	3	**79.3±1.2**	4	**90.8±0.7**	4	**88.2±0.8**
ProSup + LGCN	3	**79.6±0.7**	3	**91.2±0.8**	3	**88.5±0.5**

unweighted adjacency matrix and bag-of-words features. The statistics of these datasets are summarized in Table 1.

In the transductive setting, we have access to the features of all nodes but only the labels of nodes in the training set for training. In the inductive setting, both the features and labels of the nodes in the validation/testing set are unavailable during training.

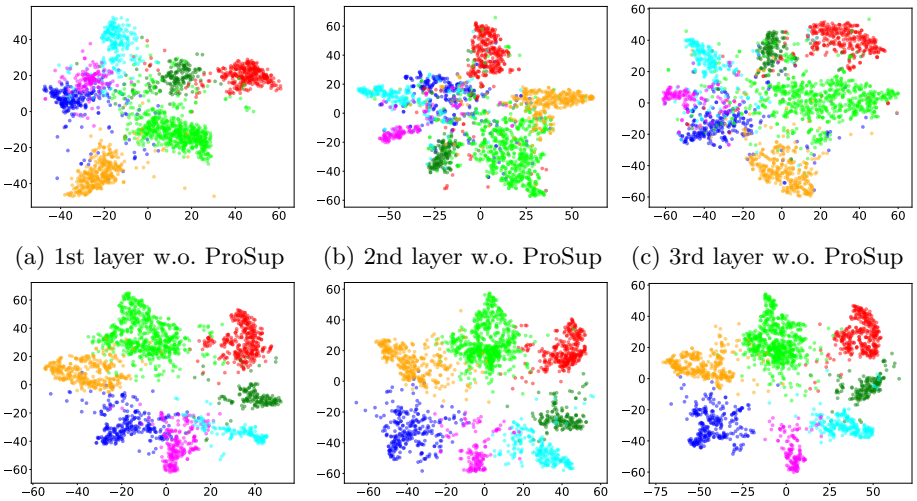
For the hyper-parameters of the benchmarks, e.g. the number of hidden units, the optimizer, the learning rate, we set them as suggested by their authors. For the hyper-parameters of our ProSup, we set $d_{fuse} = d_{L-1}$ for the dimensionality of the output fusion module, and $\alpha = 1$ for the weight of the loss from side-outputs by default.

5.1 Transductive Node Classification

In the transductive settings, we take the popular GCN architectures of GCN [11], GAT [21], LGCN [8], JKNet [24], and GMNN [18] as the baselines for comparison. We split nodes in each graph into 60%, 20%, 20% for training, validation, and testing. We make 10 random splits and conduct the experiments for 100 trials with random weight initialization for each split.

We vary the number of layers from 1 to 8 for each model and choose the best performing number with respect to the validation set. The results are reported in Table 2. We observe that ProSup improves the test accuracy of GCN by 2.9% on Citeseer, 2.8% on Cora, 2.1% on Pubmed, and LGCN by 2.7% on Citeseer, 2.5% on Cora, and 2.3% on Pubmed respectively. As a result, ProSup enhances GCN and LGCN to outperform all the benchmark methods.

Taking a closer look, we observe that ProSup increases the number of layers that GCN and LGCN need to achieve their best performance. The reason is that more layers inevitably introduce extra noise for learning the final-layer representations. However, ProSup enables hidden layers to make outputs by themselves,



(a) 1st layer w.o. ProSup (b) 2nd layer w.o. ProSup (c) 3rd layer w.o. ProSup

(d) 1st layer w. ProSup (e) 2nd layer w. ProSup (f) Fused layer w. ProSup

Fig. 4: The learnt representations of the nodes in the `Cora` dataset (visualized by t-SNE [3]). Colors denote the ground-truth class labels. ProSup (lower row) makes class-specific representations more concentrated (than the upper row).

so the node-wise predictions are made with the 'appropriate' receptive fields. Besides, in terms of the fitting capacity, more layers provide a larger fitting capacity. In this case, minimizing the loss solely on the final-layer prediction leads to higher risks of overfitting, since it ignores the characteristics of representations of hidden layers. ProSup decreases this risk by regularizing GCNs to learn consistently discriminative features among its hidden layers.

Fig. 4 presents the learned representations obtained by a 3-layer GCN and a 3-layer GCN with ProSup. We can observe that the hidden layers supported by ProSup learn more discriminative features consistently, thanks to the direct supervision applied to them. These highly discriminative features potentially help to produce better class predictions than less discriminative features.

5.2 Inductive Node Classification

In the inductive settings, we use the datasets Flickr, Yelp, Reddit with the fixed partition [25] for evaluation. These datasets are too large to be handled well by the full-batch implementations of GCN architectures. Hence, we use GraphSAGE [9] and GraphSAINT [25] as the benchmarks for comparison, which are more scalable. We implement ProSup with GraphSAGE-mean and GraphSAINT-GCN to observe whether ProSup can improve the performance of GCNs under the inductive setting.

We vary the number of layers of each method from 1 to 8 for each model and choose the best performing model with respect to the validation set. The

Table 3: Test F1-micro score (%) of inductive node classification. #Layers indicates the best-performing number of layers among 1 to 8. We report mean and standard derivations of 100 trials with random weight initialization. We implement ProSup with GraphSAGE-mean and GraphSAINT-GCN.

Method	#Layers	`Flickr`	#Layers	`Yelp`	#Layers	`Reddit`
GraphSAGE-mean	3	50.1±1.1	2	63.4±0.6	3	95.3±0.1
GraphSAGE-LSTM	3	50.3±1.3	3	63.2±0.8	2	95.1±0.1
GraphSAGE-pool	3	50.0±0.8	3	63.1±0.5	2	95.2±0.1
ProSup + GraphSAGE	4	**51.9±0.9**	3	**64.7±0.7**	4	**96.1±0.1**
GraphSAINT-GCN	3	51.1±0.2	2	65.3±0.3	3	96.6±0.1
GraphSAINT-GAT	2	50.5±0.1	2	65.1±0.2	3	95.8±0.0
GraphSAINT-JKNet	4	51.3±0.5	3	65.3±0.4	4	97.0±0.1
ProSup + GraphSAINT	3	**52.8±0.2**	3	**66.2±0.2**	4	**97.3±0.1**

results are reported in Table 3. GraphSAGE-mean/LSTM/pool denotes that GraphSAGE uses mean, LSTM, and max-pooling as the aggregator respectively. GraphSAINT-GCN/GAT/JKNet denote GraphSAINT using GCN, GAT, and JKNet as the base architecture respectively. We observe that ProSup improves the test F1-micro scores of GraphSAGE-mean by 3.6% on Flickr, 2.1% on Yelp, 0.8% on Reddit, and GraphSAINT-GCN by 3.3% on Flickr, 1.4% on Yelp, and 0.3% on Reddit respectively. As a result, ProSup enhances them to outperform the benchmark methods. Overall, the above results validate that ProSup is effective in improving the performance of the popular GCN models under both transductive and inductive settings.

5.3 Ablation Study

We conduct a number of ablations to analyze ProSup. First, we investigate the effects of the side network and the output fusion module. We compare the results on the Flickr dataset of GraphSAINT and its variants combined with our proposed techniques in Table 4. To obtain the classification results on hidden layers of GraphSAINT-GCN, we train a fully-connected layer over the learned hidden representations at every GCN layer. We observe that, with our side network, the prediction results are better on every layer, which demonstrates that our progressive supervision can act as a form of regularization, which not only makes the hidden representations more discriminative but also improves the quality of the final-layer predictions. Moreover, with our module of output fusion, the prediction performance of hidden layers is improved further, thanks to the additional global supervision. Finally, the fusion module fuses the coarse predictions from hidden layers to form the finer final prediction.

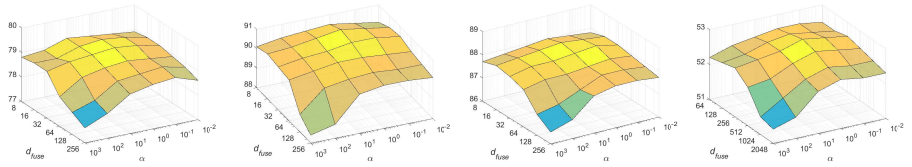
Table 5 presents the performance of GCNs with and without our technique of progressively re-weighting the hard nodes among hidden layers. Empirically,

Table 4: F1-micro score (%) of inductive node classification on Flickr.

	GCN	GCN w. side network	GCN w. ProSup
Layer 1	47.3	51.3	51.7
Layer 2	48.6	51.6	51.9
Layer 3	**51.1**	**52.0**	52.2
Fused Output	-	-	**52.8**

Table 5: Test Accuracy (%) of GCN on Citeseer, Cora, Pubmed and F1-micro score (%) of GraphSAGE-mean on Flickr, Yelp, Reddit.

	`Citeseer`	`Cora`	`Pubmed`	`Flickr`	`Yelp`	`Reddit`
GCN + ProSup w.o Reweighting	78.9	90.5	88.0	51.1	64.3	95.9
GCN + ProSup w. Reweighting	**79.3**	**90.8**	**88.2**	**51.9**	**64.7**	**96.1**



(a) `Citeseer` (b) `Cora` (c) `Pubmed` (d) `Flickr`

Fig. 5: Test accuracy (% on z-axis) of GCN with ProSup on `Citeseer`, `Cora`, `Pubmed` and F1-micro scores of GraphSAINT-GCN with ProSup on Flickr versus values of hyper-parameters d_{fuse} (x-axis) and α (y-axis).

this technique consistently improves the classification performance of GCN and GraphSAGE over all datasets. This demonstrates that it is valuable to leverage the hardness information implied by the predicted scores through different layers. The re-weighting technique encourages communication among hidden layers so as to improve the final predictions.

Finally, we evaluate how sensitive our ProSup is to the selection of hyper-parameter values: d_{fuse} to control the dimensionality of the node head in the output fusion module, and α to adjust the weight of side-output loss. We visualize the results in Fig. 5. As we can see, the performance of GCN with ProSup is relatively smooth when parameters are within certain ranges. However, extremely large values of d_{fuse} and α result in low performance on all datasets, which should be avoided in practice. Moreover, increasing α from 0.01 to 1 improves the test accuracy/F1-micro scores on all datasets, demonstrating that the supervision progressively applied on hidden layers plays an important role in improving the performance of GCNs.

6 Conclusion

In this paper, we have developed a new graph convolutional network based node classifier and demonstrated its superior performance on datasets comprising documents, images, friendships, and online discussion posts. Our scheme builds on the idea of progressively supervising the hidden layers of GCNs. During training, every hidden layer participates in the loss calculation directly to learn discriminative features. In addition, We propose a reweighting technique to deal with hard nodes by giving them larger loss weights in a progressive way. This encourages GCNs to pay more attention to hard nodes and thus make predictions more effectively. Our method shows the effectiveness of predicting node-wise labels by combining multi-scale and multi-level responses in an adaptive and learnable way. Moreover, ProSup does not incur any change in the time complexity of GCNs, as we analyze. An interesting future direction is to extend our progressive supervision algorithm to other tasks on graph data, such as graph classification, link prediction, personalized recommendation, etc.

Acknowledgments

This paper is supported by NUS ODPRT Grant R252-000-A81-133 and Singapore Ministry of Education Academic Research Fund Tier 3 under MOEs official grant number MOE2017-T3-1-007.

References

1. Bishop, C.M.: Pattern recognition and machine learning. springer (2006)
2. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203 (2013)
3. Buja, A., Cook, D., Swayne, D.F.: Interactive high-dimensional data visualization. Journal of computational and graphical statistics **5**(1), 78–99 (1996)
4. Cai, Y., Ge, L., Liu, J., Cai, J., Cham, T.J., Yuan, J., Thalmann, N.M.: Exploiting spatial-temporal relationships for 3d pose estimation via graph convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2272–2281 (2019)
5. Chen, J., Zhu, J., Song, L.: Stochastic training of graph convolutional networks with variance reduction. arXiv preprint arXiv:1710.10568 (2017)
6. Chiang, W.L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.J.: Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 257–266 (2019)
7. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems. pp. 3844–3852 (2016)
8. Gao, H., Wang, Z., Ji, S.: Large-scale learnable graph convolutional networks. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1416–1424 (2018)

9. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in neural information processing systems. pp. 1024–1034 (2017)
10. Haonan, L., Huang, S.H., Ye, T., Xiuyan, G.: Graph star net for generalized multi-task learning. arXiv preprint arXiv:1906.12330 (2019)
11. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
12. Klicpera, J., Bojchevski, A., Günnemann, S.: Predict then propagate: Graph neural networks meet personalized pagerank. arXiv preprint arXiv:1810.05997 (2018)
13. Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. In: Artificial intelligence and statistics. pp. 562–570 (2015)
14. London, B., Getoor, L.: Collective classification of network data. Data Classification: Algorithms and Applications **399** (2014)
15. McAuley, J., Leskovec, J.: Image labeling on a network: using social-network metadata for image classification. In: European conference on computer vision. pp. 828–841. Springer (2012)
16. Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model cnns. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 5115–5124 (2017)
17. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)
18. Qu, M., Bengio, Y., Tang, J.: Gmnn: Graph markov neural networks. arXiv preprint arXiv:1905.06214 (2019)
19. Quan, P., Shi, Y., Lei, M., Leng, J., Zhang, T., Niu, L.: A brief review of receptive fields in graph convolutional networks. In: IEEE/WIC/ACM International Conference on Web Intelligence-Volume 24800. pp. 106–110. ACM (2019)
20. Rong, Y., Huang, W., Xu, T., Huang, J.: Dropedge: Towards deep graph convolutional networks on node classification. In: International Conference on Learning Representations (2019)
21. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
22. Veličković, P., Fedus, W., Hamilton, W.L., Liò, P., Bengio, Y., Hjelm, R.D.: Deep graph infomax. arXiv preprint arXiv:1809.10341 (2018)
23. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596 (2019)
24. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. arXiv preprint arXiv:1806.03536 (2018)
25. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Graphsaint: Graph sampling based inductive learning method. arXiv preprint arXiv:1907.04931 (2019)
26. Zhang, J., Shi, X., Xie, J., Ma, H., King, I., Yeung, D.Y.: Gaan: Gated attention networks for learning on large and spatiotemporal graphs. arXiv preprint arXiv:1803.07294 (2018)
27. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. arXiv preprint arXiv:1812.08434 (2018)
28. Zhuang, C., Ma, Q.: Dual graph convolutional networks for graph-based semi-supervised classification. In: Proceedings of the 2018 World Wide Web Conference. pp. 499–508 (2018)