Node Dependent Local Smoothing for Scalable Graph Learning

Wentao Zhang[1], **Mingyu Yang**[1], **Zeang Sheng**[1], **Yang Li**[1]
Wen Ouyang[2], **Yangyu Tao**[2], **Zhi Yang**[1,3], **Bin Cui**[1,3,4]
[1]School of CS, Peking University [2]Tencent Inc.
[3] Key Lab of High Confidence Software Technologies, Peking University
[4]Institute of Computational Social Science, Peking University (Qingdao), China
[1]{wentao.zhang, ymyu, shengzeang18, liyang.cs, yangzhi, bin.cui}@pku.edu.cn
[2]{gdpouyang, brucetao}@tencent.com

Abstract

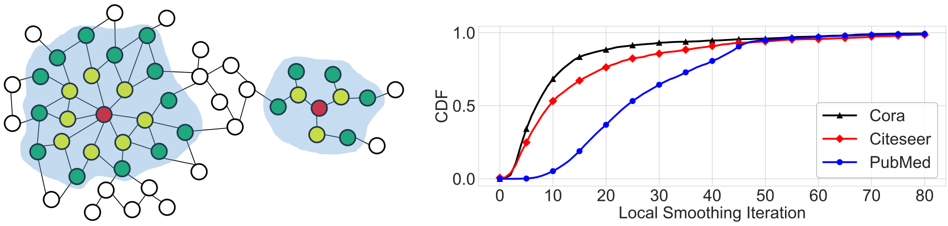
Recent works reveal that feature or label smoothing lies at the core of Graph Neural Networks (GNNs). Concretely, they show feature smoothing combined with simple linear regression achieves comparable performance with the carefully designed GNNs, and a simple MLP model with label smoothing of its prediction can outperform the vanilla GCN. Though an interesting finding, smoothing has not been well understood, especially regarding how to control the extent of smoothness. Intuitively, too small or too large smoothing iterations may cause *under-smoothing* or *over-smoothing* and can lead to sub-optimal performance. Moreover, the extent of smoothness is node-specific, depending on its degree and local structure. To this end, we propose a novel algorithm called node-dependent local smoothing (NDLS), which aims to control the smoothness of every node by setting a node-specific smoothing iteration. Specifically, NDLS computes influence scores based on the adjacency matrix and selects the iteration number by setting a threshold on the scores. Once selected, the iteration number can be applied to both feature smoothing and label smoothing. Experimental results demonstrate that NDLS enjoys high accuracy – state-of-the-art performance on node classifications tasks, flexibility – can be incorporated with any models, scalability and efficiency – can support large scale graphs with fast training.

1 Introduction

In recent years, Graph Neural Networks (GNNs) have received a surge of interest with the state-of-the-art performance on many graph-based tasks [2, 41, 12, 39, 33, 34]. Recent works have found that the success of GNNs can be mainly attributed to smoothing, either at feature or label level. For example, SGC [32] shows using smoothed features as input to a simple linear regression model achieves comparable performance with lots of carefully designed and complex GNNs. At the smoothing stage, features of neighbor nodes are aggregated and combined with the current node's feature to form smoothed features. This process is often iterated multiple times. The smoothing is based on the assumption that labels of nodes that are close to each other are highly correlated, therefore, the features of nodes nearby should help predict the current node's label.

One crucial and interesting parameter of neighborhood feature aggregation is the number of smoothing iterations k, which controls how much information is being gathered. Intuitively, an aggregation process of k iterations (or layers) enables a node to leverage information from nodes that are k-hop away [26, 38]. The choice of k is closely related to the structural properties of graphs and has a

35th Conference on Neural Information Processing Systems (NeurIPS 2021).



(a) Two nodes with different local structures

(b) The CDF of LSI in different graphs

Figure 1: (Left) The node in dense region has larger smoothed area within two iterations of propagation. (Right) The CDF of LSI in three citation networks.

significant impact on the model performance. However, most existing GNNs only consider the fixed-length propagation paradigm – a uniform k for all the nodes. This is problematic since the number of iterations should be *node dependent* based on its degree and local structures. For example, as shown in Figure 1(a), the two nodes have rather different local structures, with the left red one resides in the center of a dense cluster and the right red one on the periphery with few connections. The number of iterations to reach an optimal level of smoothness are rather different for the two nodes. Ideally, poorly connected nodes (e.g., the red node on the right) needs large iteration numbers to efficiently gather information from other nodes while well-connected nodes (e.g., the red node on the left) should keep the iteration number small to avoid *over-smoothing*. Though some learning-based approaches have proposed to adaptively aggregate information for each node through gate/attention mechanism or reinforcement learning [29, 21, 40, 27], the performance gains are at the cost of increased training complexity, hence not suitable for scalable graph learning.

In this paper, we propose a simple yet effective solution to this problem. Our approach, called node-dependent local smoothing (NDLS), calculates a node-specific iteration number for each node, referred to as local smooth iteration (LSI). Once the LSI for a specific node is computed, the corresponding local smoothing algorithm only aggregates the information from the nodes within a distance less than its LSI as the new feature. The LSI is selected based on influence scores, which measure how other nodes influence the current node. NDLS sets the LSI for a specific node to be the minimum number of iterations so that the influence score is ϵ-away from the *over-smoothing* score, defined as the influence score at infinite iteration. The insight is that each node's influence score should be at a reasonable level. Since the nodes with different local structures have different "smoothing speed", we expect the iteration number to be adaptive. Figure 1(b) illustrates Cumulative Distribution Function (CDF) for the LSI of individual nodes in real-world graphs. The heterogeneous and long-tail property exists in all the datasets, which resembles the characteristics of the degree distribution of nodes in real graphs.

Based on NDLS, we propose a new graph learning algorithm with three stages: (1) feature smoothing with NDLS (NDLS-F); (2) model training with smoothed features; (3) label smoothing with NDLS (NDLS-L). Note that in our framework, the graph structure information is only used in pre-processing and post-processing steps, i.e., stages (1) and (3) (See Figure 2). Our NDLS turns a graph learning problem into a vanilla machine learning problem with independent samples. This simplicity enables us to train models on larger-scale graphs. Moreover, our NDLS kernel can act as a drop-in replacement for any other graph kernels and be combined with existing models such as Multilayer Perceptron (MLP), SGC [32], SIGN [28], S^{2}GC [42] and GBP [6].

Extensive evaluations on seven benchmark datasets, including large-scale datasets like ogbn-papers100M [16], demonstrates that NDLS achieves not only the state-of-the-art node classification performance but also high training scalability and efficiency. Especially, NDLS outperforms APPNP [29] and GAT [30] by a margin of 1.0%-1.9% and 0.9%-2.4% in terms of test accuracy, while achieving up to $39\times$ and $186\times$ training speedups, respectively.

2 Preliminaries

In this section, we first introduce the semi-supervised node classification task and review the prior models, based on which we derive our method in Section 3. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$

nodes and $|\mathcal{E}| = m$ edges, the adjacency matrix (including self loops) is denoted as $\tilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$ and the feature matrix is denoted as $\mathbf{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2..., \boldsymbol{x}_n\}$ in which $\boldsymbol{x}_i \in \mathbb{R}^f$ represents the feature vector of node v_i. Besides, $\mathbf{Y} = \{\boldsymbol{y_1}, \boldsymbol{y_2}..., \boldsymbol{y_l}\}$ is the initial label matrix consisting of one-hot label indicator vectors. The goal is to predict the labels for nodes in the unlabeled set $\mathcal{V}_u$ with the supervision of labeled set $\mathcal{V}_l$.

GCN smooths the representation of each node via aggregating its own representations and the ones of its neighbors'. This process can be defined as

$$\mathbf{X}^{(k+1)} = \delta\left(\hat{\mathbf{A}}\mathbf{X}^{(k)}\mathbf{W}^{(k)}\right), \qquad \hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{r-1}\tilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-r}, \tag{1}$$

where $\hat{\mathbf{A}}$ is the normalized adjacency matrix, $r \in [0, 1]$ is the convolution coefficient, and $\widetilde{\mathbf{D}}$ is the diagonal node degree matrix with self loops. Here $\mathbf{X}^{(k)}$ and $\mathbf{X}^{(k+1)}$ are the smoothed node features of layer k and $k+1$ respectively while $\mathbf{X}^{(0)}$ is set to $\mathbf{X}$, the original feature matrix. In addition, $\mathbf{W}^{(k)}$ is a layer-specific trainable weight matrix at layer k, and $\delta(\cdot)$ is the activation function. By setting $r =$ 0.5, 1 and 0, the convolution matrix $\widetilde{\mathbf{D}}^{r-1}\tilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-r}$ represents the symmetric normalization adjacency matrix $\widetilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}$ [20], the transition probability matrix $\tilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1}$ [37], and the reverse transition probability matrix $\widetilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ [35], respectively.

SGC. For each GCN layer defined in Eq. 1, if the non-linear activation function $\delta(\cdot)$ is an identity function and $\mathbf{W}^{(k)}$ is an identity matrix, we get the smoothed feature after k-iterations propagation as $\mathbf{X}^{(k)} = \hat{\mathbf{A}}^k\mathbf{X}$. Recent studies have observed that GNNs primarily derive their benefits from performing feature smoothing over graph neighborhoods rather than learning non-linear hierarchies of features as implied by the analogy to CNNs [25, 10, 15]. By hypothesizing that the non-linear transformations between GCN layers are not critical, SGC [32] first extracts the smoothed features $\mathbf{X}^{(k)}$ then feeds them to a linear model, leading to higher scalability and efficiency. Following the design principle of SGC, piles of works have been proposed to further improve the performance of SGC while maintaining high scalability and efficiency, such as SIGN [28], S^{2}GC [42] and GBP [6].

Over-Smoothing [22] issue. By continually smoothing the node feature with infinite number of propagation in SGC, the final smoothed feature $\mathbf{X}^{(\infty)}$ is

$$\mathbf{X}^{(\infty)} = \hat{\mathbf{A}}^{\infty}\mathbf{X}, \qquad \hat{\mathbf{A}}^{\infty}_{i,j} = \frac{(d_i+1)^r(d_j+1)^{1-r}}{2m+n}, \tag{2}$$

where $\hat{\mathbf{A}}^{\infty}$ is the final smoothed adjacency matrix, $\hat{\mathbf{A}}^{\infty}_{i,j}$ is the weight between nodes v_i and v_j, d_i and d_j are the node degrees for v_i and v_j, respectively. Eq. (2) shows that as we smooth the node feature with an infinite number of propagations in SGC, the final feature is over-smoothed and unable to capture the full graph structure information since it only relates with the node degrees of target nodes and source nodes. For example, if we set $r = 0$ or 1, all nodes will have the same smoothed features because only the degrees of the source or target nodes have been considered.

3 Local Smoothing Iteration (LSI)

The features after k iterations of smoothing is $\mathbf{X}^{(k)} = \hat{\mathbf{A}}^k\mathbf{X}$. Inspired by [35], we measure the influence of node v_j on node v_i by measuring how much a change in the input feature of v_j affects the representation of v_i after k iterations. For any node v_i, the influence vector captures the influences of all other nodes. Considering the h^{th} feature of $\mathbf{X}$, we define an influence matrix $I_h(k)$:

$$I_h(k)_{ij} = \frac{\partial\hat{\mathbf{X}}^{(k)}_{ih}}{\partial\hat{\mathbf{X}}^{(0)}_{jh}}. \tag{3}$$

$$I(k) = \hat{\mathbf{A}}^k, \tilde{I}_i = \hat{\mathbf{A}}^{\infty} \tag{4}$$

Since $I_h(k)$ is independent to h, we replace $I_h(k)$ with $I(k)$, which can be further represented as $I(k) = I_h(k),\ \forall h \in \{1, 2, .., f\}$, where f indicates the number of features of $\mathbf{X}$. We denote $I(k)_i$ as the i^{th} row of $I(k)$, and $\tilde{I}$ as $I(\infty)$. Given the normalized adjacency matrix $\hat{\mathbf{A}}$, we can have $I(k) = \hat{\mathbf{A}}^k$ and $\tilde{I} = \hat{\mathbf{A}}^\infty$. According to Eq. (2), $\tilde{I}$ converges to a unique stationary matrix independent of the distance between nodes, resulting in that the aggregated features of nodes are merely relative with their degrees (i.e., over-smoothing).

We denote $I(k)_i$ as the i^{th} row of $I(k)$, and it means the influence from the other nodes to the node v_i after k iterations of propagation. We introduce a new concept *local smoothing iteration* (parameterized by ϵ), which measures the minimal number of iterations k required for the influence of other nodes on node v_i to be within an ϵ-distance to the over-smoothing stationarity $\tilde{I}_i$.

Definition 3.1. Local-Smoothing Iteration *(LSI, parameterized by ϵ) is defined as*

$$K(i, \epsilon) = \min\{k : ||\tilde{I}_i - I(k)_i||_2 < \epsilon\}, \tag{5}$$

where $||\cdot||_2$ is two-norm, and ϵ is an arbitrary small constant with $\epsilon > 0$.

Here ϵ is a graph-specific parameter, and a smaller ϵ indicates a stronger smoothing effect. The ϵ-distance to the over-smoothing stationarity $\tilde{I}_i$ ensures that the smooth effect on node v_i is sufficient and bounded to avoid over-smoothing. As shown in Figure 1(b), we can have that the distribution of LSI owns the *heterogeneous and long-tail property*, where a large percentage of nodes have much smaller LSI than the rest. Therefore, the required LSI to approach the stationarity is heterogeneous across nodes. Now we discuss the connection between LSI and node local structure, showcasing nodes in the sparse region (e.g., both the degrees of itself and its neighborhood are low) can greatly prolong the iteration to approach over-smoothing stationarity. This heterogeneity property is not fully utilized in the design of current GNNs, leaving the model design in a dilemma between unnecessary iterations for a majority of nodes and insufficient iterations for the rest of nodes. Hence, by adaptively choosing the iteration based on LSI for different nodes, we can significantly improve model performance.

Theoretical Properties of LSI. We now analyze the factors determining the LSI of a specific node. To facilitate the analysis, we set the coefficient $r = 0$ for the normalized adjacency matrix $\hat{\mathbf{A}}$ in Eq. (1), thus $\hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$. The proofs of following theorems can be found in Appendix A.1.

Theorem 3.1. *Given feature smoothing $\mathbf{X}^{(k)} = \hat{\mathbf{A}}^k\mathbf{X}$ with $\hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$, we have*

$$K(i, \epsilon) \leq \log_{\lambda_2}\left(\epsilon\sqrt{\frac{\tilde{d}_i}{2m+n}}\right), \tag{6}$$

where λ_2 is the second largest eigenvalue of $\hat{\mathbf{A}}$, $\tilde{d}_i$ denotes the degree of node v_i plus 1 (i.e., $\tilde{d}_i = d_i + 1$), and m, n denote the number of edges and nodes respectively.

Note that $\lambda_2 \leq 1$. Theorem 3.1 shows that the upper-bound of the LSI is positively correlated with the scale of the graph (m, n), the sparsity of the graph (small λ_2 means strong connection and low sparsity, and vice versa), and negatively correlated with the degree of node v_i.

Theorem 3.2. *For any nodes i in a graph $\mathcal{G}$,*

$$K(i, \epsilon) \leq \max\{K(j, \epsilon), j \in N(i)\} + 1, \tag{7}$$

where $N(i)$ is the set of node v_i's neighbours.

Theorem 3.2 indicates that the difference between two neighboring nodes' LSIs is no more than 1, therefore the nodes with a super-node as neighbors (or neighbor's neighbors) may have small LSIs. That is to say, the sparsity of the local area, where a node locates, also affects its LSI positively. Considering Theorems 3.1 and 3.2 together, we can have a union upper-bound of $K(i, \epsilon)$ as

$$K(i, \epsilon) \leq \min\left\{\max\{K(j, \epsilon), j \in N(i)\} + 1, \log_{\lambda_2}\left(\epsilon\sqrt{\frac{\tilde{d}_i}{2m+n}}\right)\right\}. \tag{8}$$

4 NDLS Pipeline

The basic idea of NDLS is to utilize the LSI heterogeneity to perform a node-dependent aggregation over a neighborhood within a distance less than the specific LSI for each node. Further, we propose

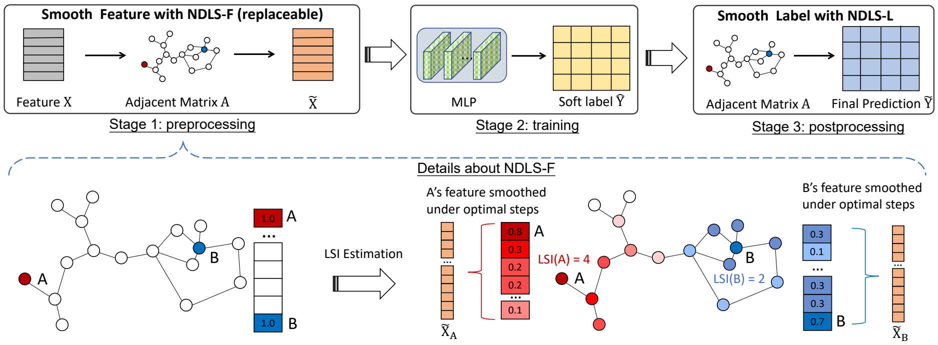


Figure 2: Overview of the proposed NDLS method, including (1) feature smoothing with NDLS (NDLS-F), (2) model training with smoothed features, and (3) label smoothing with NDLS (NDLS-L). NDLS-F and NDLS-L correspond to pre-processing and post-processing steps respectively.

a simple pipeline with three main parts (See Figure 2): (1) a node-dependent local smoothing of the feature (NDLS-F) over the graph, (2) a base prediction result with the smoothed feature, (3) a node-dependent local smoothing of the label predictions (NDLS-L) over the graph. Note this pipeline is not trained in an end-to-end way, the stages (1) and (3) in NDLS are only the pre-processing and post-processing steps, respectively. Furthermore, the graph structure is only used in the pre/post-processing NDLS steps, not for the base predictions. Compared with prior GNN models, this key design enables higher scalability and a faster training process.

Based on the graph structure, we first compute the node-dependent *local smoothing iteration* that maintains a proper distance to the over-smoothing stationarity. Then the corresponding local smoothing kernel only aggregates the information (feature or prediction) for each node from the nodes within a distance less than its LSI value. The combination of NDLS-F and NDLS-L takes advantage of both label smoothing (which tends to perform fairly well on its own without node features) and the node feature smoothing. We will see that combining these complementary signals yields state-of-the-art predictive accuracy. Moreover, our NDLS-F kernel can act as a drop-in replacement for graph kernels in other scalable GNNs such as SGC, S^{2}GC, GBP, etc.

4.1 Smooth Features with NDLS-F

Once the node-dependent LSI $K(i,\epsilon)$ for a specific node i is obtained, we smooth the initial input feature $\mathbf{X}_i$ of node i with node-dependent LSI as:

$$\widetilde{\mathbf{X}}_i(\epsilon) = \frac{1}{K(i,\epsilon)+1}\sum_{k=0}^{K(i,\epsilon)} \mathbf{X}_i^{(k)}. \tag{9}$$

To capture sufficient neighborhood information, for each node v_i, we average its multi-scale features $\{\mathbf{X}_i^{(k)} \mid k \le K(i,\epsilon)\}$ obtained by aggregating information within k hops from the node v_i.

The matrix form of the above equation can be formulated as

$$\widetilde{\mathbf{X}}(\epsilon) = \sum_{k=0}^{\max_i K(i,\epsilon)} \mathbf{M}^{(k)}\mathbf{X}^{(k)}, \qquad \mathbf{M^{(k)}}_{ij} = \begin{cases} \frac{1}{K(i,\epsilon)+1}, & i=j \quad and \quad k \le K(i,\epsilon) \\ 0, & \text{otherwise} \end{cases}, \tag{10}$$

where $\mathbf{M^{(k)}}$ is a set of diagonal matrix.

4.2 Simple Base Prediction

With the smoothed feature $\widetilde{\mathbf{X}}$ according to Eq. 9, we then train a model to minimize the loss – $\sum_{v_i \in \mathcal{V}_l} \ell\left(\boldsymbol{y}_i, f(\widetilde{\mathbf{X}}_i)\right)$, where $\widetilde{\mathbf{X}}_i$ denotes the i^{th} row of $\widetilde{\mathbf{X}}$, ℓ is the cross-entropy loss function, and $f(\widetilde{\mathbf{X}}_i)$ is the predictive label distribution for node v_i. In NDLS, the default f is a MLP model and

$\hat{\mathbf{Y}} = f(\widetilde{\mathbf{X}})$ is its soft label predicted (softmax output). Note that, many other models such as Random Forest [24] and XGBoost [7] could also be used in NDLS (See more results in Appendix A.2).

4.3 Smooth Labels with NDLS-L

Similar to the feature propagation, we can also propagate the soft label $\hat{\mathbf{Y}}$ with $\hat{\mathbf{Y}}^{(k)} = \hat{\mathbf{A}}^k\hat{\mathbf{Y}}$. Considering the influence matrix of softmax label $J_h(k)$.

$$J_h(k)_{ij} = \frac{\partial \hat{\mathbf{Y}}^{(k)}_{ih}}{\partial \hat{\mathbf{Y}}^{(0)}_{jh}}. \tag{11}$$

According to the definition above we have that

$$J_h(k) = I_h(k), \forall h \in \{1, 2, .., f\}. \tag{12}$$

Therefore, local smoothing can be further applied to address over-smoothing in label propagation. Concretely, we smooth an initial soft label $\hat{\mathbf{Y}}_i$ of node v_i with NDLS as follows

$$\widetilde{\mathbf{Y}}_i(\epsilon) = \frac{1}{K(i,\epsilon)+1} \sum_{k=0}^{K(i,\epsilon)} \hat{\mathbf{Y}}^{(k)}_i. \tag{13}$$

Similarly, the matrix form of the above equation can be formulated as

$$\widetilde{\mathbf{Y}}(\epsilon) = \sum_{k=0}^{\max\limits_i K(i,\epsilon)} \mathbf{M}^{(k)}\hat{\mathbf{Y}}^{(k)}, \tag{14}$$

where $\mathbf{M}^{(\mathbf{k})}$ follows the definition in Eq. (10).

5 Comparison with Existing Methods

Decoupled GNNs. The aggregation and transformation operations in coupled GNNs (i.e., GCN [18], GAT [30] and JK-Net [35]) are inherently intertwined in Eq. (1), so the propagation iterations L always equals to the transformation iterations K. Recently, some decoupled GNNs (e.g., PPNP [20], PPRGo [1], APPNP [20], AP-GCN [29] and DAGNN [25]) argue the entanglement of these two operations limits the propagation depth and representation ability of GNNs, so they first do the transformation and then smooth and propagate the predictive soft label with higher depth in an end-to-end manner. Especially, AP-GCN and DAGNN both use a learning mechanism to learn propagation adaptively. Unfortunately, all these coupled and decoupled GNNs are hard to scale to large graphs – *scalability issue* since they need to repeatedly perform an expensive recursive neighborhood expansion in multiple propagations of the features or soft label predicted. NDLS addresses this issue by dividing the training process into multiple stages.

Sampling-based GNNs. An intuitive method to tackle the recursive neighborhood expansion problem is sampling. As a node-wise sampling method, GraphSAGE [14] samples the target nodes as a mini-batch and samples a fixed size set of neighbors for computing. VR-GCN [5] analyzes the variance reduction on node-wise sampling, and it can reduce the size of samples with an additional memory cost. In the layer level, Fast-GCN [3] samples a fixed number of nodes at each layer, and ASGCN [17] proposes the adaptive layer-wise sampling with better variance control. For the graph-wise sampling, Cluster-GCN [8] clusters the nodes and only samples the nodes in the clusters, and GraphSAINT [37] directly samples a subgraph for mini-batch training. We don't use sampling in NDLS since the sampling quality highly influences the classification performance.

Linear Models. Following SGC [32], some recent methods remove the non-linearity between each layer in the forward propagation. SIGN [28] allows using different local graph operators and proposes to concatenate the different iterations of propagated features. S^{2}GC [42] proposes the simple spectral graph convolution to average the propagated features in different iterations. In addition, GBP [6] further improves the combination process by weighted averaging, and all nodes in the same layer share the same weight. In this way, GBP considers the smoothness in a layer perspective way. Similar

Table 1: Algorithm analysis for existing scalable GNNs. n, m, c, and f are the number of nodes, edges, classes, and feature dimensions, respectively. b is the batch size, and k refers to the number of sampled nodes. L corresponds to the number of times we aggregate features, K is the number of layers in MLP classifiers. For the coupled GNNs, we always have $K = L$.

Type	Method	Preprocessing and postprocessing	Training	Inference	Memory
Node-wise sampling	GraphSAGE	-	$\mathcal{O}(k^L nf^2)$	$\mathcal{O}(k^L nf^2)$	$\mathcal{O}(bk^L f + Lf^2)$
Layer-wise sampling	FastGCN	-	$\mathcal{O}(kLnf^2)$	$\mathcal{O}(kLnf^2)$	$\mathcal{O}(bkLf + Lf^2)$
Graph-wise sampling	Cluster-GCN	$\mathcal{O}(m)$	$\mathcal{O}(Lmf + Lnf^2)$	$\mathcal{O}(Lmf + Lnf^2)$	$\mathcal{O}(bLf + Lf^2)$
Linear model	SGC	$\mathcal{O}(Lmf)$	$\mathcal{O}(nf^2)$	$\mathcal{O}(nf^2)$	$\mathcal{O}(bf + f^2)$
	S^{2}GC	$\mathcal{O}(Lmf)$	$\mathcal{O}(nf^2)$	$\mathcal{O}(nf^2)$	$\mathcal{O}(bf + f^2)$
	SIGN	$\mathcal{O}(Lmf)$	$\mathcal{O}(Knf^2)$	$\mathcal{O}(Knf^2)$	$\mathcal{O}(bLf + Kf^2)$
	GBP	$\mathcal{O}(Lnf + L\frac{\sqrt{m \lg n}}{\varepsilon})$	$\mathcal{O}(Knf^2)$	$\mathcal{O}(Knf^2)$	$\mathcal{O}(bf + Kf^2)$
Linear model	NDLS	$\mathcal{O}(Lmf + Lmc)$	$\mathcal{O}(Knf^2)$	$\mathcal{O}(Knf^2)$	$\mathcal{O}(bf + Kf^2)$

Table 2: Overview of datasets and task types (T/I represents Transductive/Inductive).

Dataset	#Nodes	#Features	#Edges	#Classes	#Train/Val/Test	Type	Description
Cora	2,708	1,433	5,429	7	140/500/1,000	T	citation network
Citeseer	3,327	3,703	4,732	6	120/500/1,000	T	citation network
Pubmed	19,717	500	44,338	3	60/500/1,000	T	citation network
Industry	1,000,000	64	1,434,382	253	5K/10K/30K	T	short-form video network
ogbn-papers100M	111,059,956	128	1,615,685,872	172	1,207K/125K/214K	T	citation network
Flickr	89,250	500	899,756	7	44K/22K/22K	I	image network
Reddit	232,965	602	11,606,919	41	155K/23K/54K	I	social network

to these works, we also use a linear model for higher training scalability. The difference lies in that we consider the smoothness from a node-dependent perspective and each node in NDLS has a personalized aggregation iteration with the proposed local smoothing mechanism.

Table 1 compares the asymptotic complexity of NDLS with several representative and scalable GNNs. In the stage of the preprocessing, the time cost of clustering in Cluster-GCN is $\mathcal{O}(m)$ and the time complexity of most linear models is $\mathcal{O}(Lmf)$. Besides, NDLS has an extra time cost $\mathcal{O}(Lmc)$ for the postprocessing in label smoothing. GBP conducts this process approximately with a bound of $\mathcal{O}(Lnf + L\frac{\sqrt{m \lg n}}{\varepsilon})$, where ε is a error threshold. Compared with the sampling-based GNNs, the linear models usually have smaller training and inference complexity, i.e., higher efficiency. Memory complexity is a crucial factor in large-scale graph learning because it is difficult for memory-intensive algorithms such as GCN and GAT to train large graphs on a single machine. Compared with SIGN, both GBP and NDLS do not need to store smoothed features in different iterations, and the feature storage complexity can be reduced from $\mathcal{O}(bLf)$ to $\mathcal{O}(bf)$.

6 Experiments

In this section, we verify the effectiveness of NDLS on seven real-world graph datasets. We aim to answer the following four questions. **Q1:** Compared with current SOTA GNNs, can NDLS achieve higher predictive accuracy and why? **Q2:** Are NDLS-F and NDLS-L better than the current feature and label smoothing mechanisms (e.g., the weighted feature smoothing in GBP and the adaptive label smoothing in DAGNN)? **Q3:** Can NDLS obtain higher efficiency over the considered GNN models? **Q4:** How does NDLS perform on sparse graphs (i.e., low label/edge rate, missing features)?

6.1 Experimental Setup

Datasets. We conduct the experiments on (1) six publicly partitioned datasets, including four citation networks (Citeseer, Cora, PubMed, and ogbn-papers100M) in [18, 16] and two social networks (Flickr and Reddit) in [37], and (2) one short-form video recommendation graph (Industry) from our industrial cooperative enterprise. The dataset statistics are shown in Table 2 and more details about these datasets can be found in Appendix A.3.

Baselines. In the transductive setting, we compare our method with (1) the coupled GNNs: GCN [18], GAT [30] and JK-Net [35]; (2) the decoupled GNNs: APPNP [20], AP-GCN [29],

Table 3: Results of transductive settings. OOM means “out of memory”.

Type	Models	Cora	Citeseer	PubMed	Industry	ogbn-papers100M
Coupled	GCN	81.8±0.5	70.8±0.5	79.3±0.7	45.9±0.4	OOM
	GAT	83.0±0.7	72.5±0.7	79.0±0.3	46.8±0.7	OOM
	JK-Net	81.8±0.5	70.7±0.7	78.8±0.7	47.2±0.3	OOM
Decoupled	APPNP	83.3±0.5	71.8±0.5	80.1±0.2	46.7±0.6	OOM
	AP-GCN	83.4±0.3	71.3±0.5	79.7±0.3	46.9±0.7	OOM
	PPRGo	82.4±0.2	71.3±0.5	80.0±0.4	46.6±0.5	OOM
	DAGNN (Gate)	84.4±0.5	73.3±0.6	80.5±0.5	47.1±0.6	OOM
	DAGNN (NDLS-L)*	84.4±0.6	73.6±0.7	80.9±0.5	47.2±0.7	OOM
Linear	MLP	61.1±0.6	61.8±0.8	72.7±0.6	41.3±0.8	47.2±0.3
	SGC	81.0±0.2	71.3±0.5	78.9±0.5	45.2±0.3	63.2±0.2
	SIGN	82.1±0.3	72.4±0.8	79.5±0.5	46.3±0.5	64.2±0.2
	S^{2}GC	82.7±0.3	73.0±0.2	79.9±0.3	46.6±0.6	64.7±0.3
	GBP	83.9±0.7	72.9±0.5	80.6±0.4	46.9±0.7	65.2±0.3
Linear	NDLS-F+MLP*	84.1±0.6	73.5±0.5	81.1±0.6	47.5±0.7	65.3±0.5
	MLP+NDLS-L*	83.9±0.6	73.1±0.8	81.1±0.6	46.9±0.7	64.6±0.4
	SGC+NDLS-L*	84.2±0.2	73.4±0.5	81.1±0.4	47.1±0.6	64.9±0.3
	NDLS*	**84.6±0.5**	**73.7±0.6**	**81.4±0.4**	**47.7±0.5**	**65.6±0.3**

DAGNN (Gate) [25], and PPRGo [1]; (3) the linear-model-based GNNs: MLP, SGC [32], SIGN [28], S^{2}GC [42] and GBP [6]. In the inductive setting, the compared baselines are sampling-based GNNs: GraphSAGE [14], FastGCN [3], ClusterGCN [8] and GraphSAINT [37]. Detailed descriptions of these baselines are provided in Appendix A.4.

Implementations. To alleviate the influence of randomness, we repeat each method ten times and report the mean performance. The hyper-parameters of baselines are tuned by OpenBox [23] or set according to the original paper if available. Please refer to Appendix A.5 for more details.

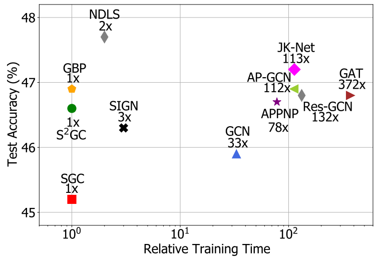


Figure 3: Performance along with training time on the Industry dataset.

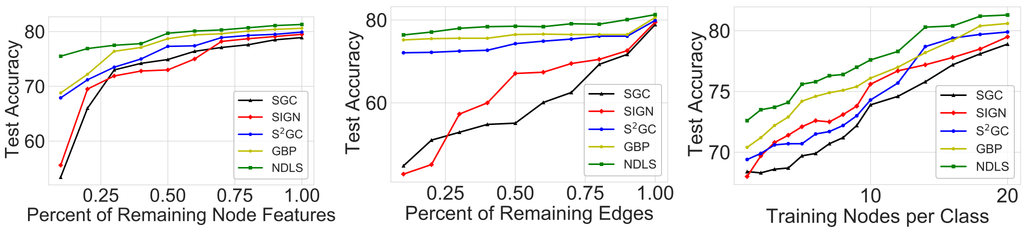
Table 4: Results of inductive settings.

Models	Flickr	Reddit
GraphSAGE	50.1±1.3	95.4±0.0
FastGCN	50.4±0.1	93.7±0.0
ClusterGCN	48.1±0.5	95.7±0.0
GraphSAINT	51.1±0.1	96.6±0.1
NDLS-F+MLP*	51.9±0.2	96.6±0.1
GraphSAGE+NDLS-L*	51.5±0.4	96.3±0.0
NDLS*	**52.6±0.4**	**96.8±0.1**

6.2 Experimental Results.

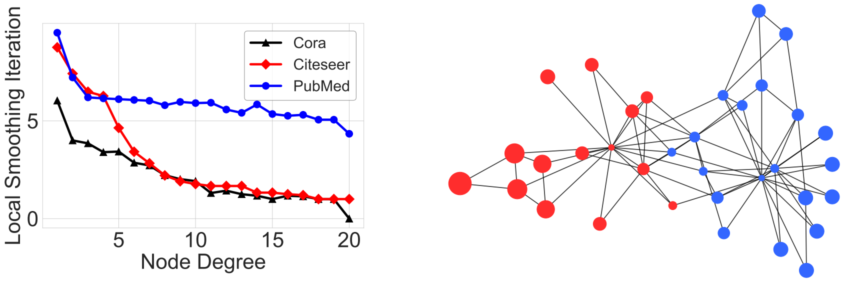
End-to-end comparison. To answer **Q1**, Table 3 and 4 show the test accuracy of considered methods in transductive and inductive settings. In the inductive setting, NDLS outperforms one of the most competitive baselines – GraphSAINT by a margin of 1.5% and 0.2% on Flickr and Reddit. NDLS exceeds the best GNN model among all considered baselines on each dataset by a margin of 0.2% to 0.8% in the transductive setting. In addition, we observe that with NDLS-L, the model performance of MLP, SGC, NDLS-F+MLP, and GraphSAGE can be further improved by a large margin. For example, the accuracy gain for MLP is 21.8%, 11.3%, 8.4%, and 5.6% on Cora, Citseer, PubMed, and Industry, respectively. To answer **Q2**, we replace the gate mechanism in the vanilla DAGNN with NDLS-L and refer to this method as DAGNN (NDLS-L). Surprisingly, DAGNN (NDLS-L) achieves at least comparable or (often) higher test accuracy compared with AP-GCN and DAGNN (Gate), and it shows that NDLS-L performs better than the learned mechanism in label smoothing. Furthermore, by replacing the original graph kernels with NDLS-F, NDLS-F+MLP outperforms both S^{2}GC and GBP on all compared datasets. This demonstrates the effectiveness of the proposed NDLS.

Training Efficiency. To answer **Q3**, we evaluate the efficiency of each method on a real-world industry graph dataset. Here, we pre-compute the smoothed features of each linear-model-based



(a) Feature Sparsity (b) Edge Sparsity (c) Label Sparsity

Figure 4: Test accuracy on PubMed dataset under different levels of feature, edge and label sparsity.



(a) LSI along with the node degree (b) The visualization of LSI

Figure 5: (Left) LSI distribution along with the node degree in three citation networks. (Right) The visualization of LSI in Zachary's karate club network. Nodes with larger radius have larger LSIs.

GNN, and the time for pre-processing is also included in the training time. Figure 3 illustrates the results on the industry dataset across training time. Compared with linear-model-based GNNs, we observe that (1) both the coupled and decoupled GNNs require a significantly larger training time; (2) NDLS achieves the best test accuracy while consuming comparable training time with SGC.

Performance on Sparse Graphs. To reply **Q4**, we conduct experiments to test the performance of NDLS on feature, edge, and label sparsity problems. For feature sparsity, we assume that the features of unlabeled nodes are partially missing. In this scenario, it is necessary to calculate a personalized propagation iteration to "recover" each node's feature representation. To simulate edge sparsity settings, we randomly remove a fixed percentage of edges from the original graph. Besides, we enumerate the number of nodes per class from 1 to 20 in the training set to measure the effectiveness of NDLS given different levels of label sparsity. The results in Figure 4 show that NDLS outperforms all considered baselines by a large margin across different levels of feature, edge, and label sparsity, thus demonstrating that our method is more robust to the graph sparsity problem than the linear-model-based GNNs.

Interpretability. As mentioned by **Q1**, we here answer why NDLS is effective. One theoretical property of LSI is that the value correlates with the node degree negatively. We divide nodes into several groups, and each group consists of nodes with the same degree. And then we calculate the average LSI value for each group in the three citation networks respectively. Figure 5(a) depicts that nodes with a higher degree have a smaller LSI, which is consistent with Theorem 3.1. We also use NetworkX [13] to visualize the LSI in Zachary's karate club network [36]. Figure 5(b), where the radius of each node corresponds to the value of LSI, shows three interesting observations: (1) nodes with a larger degree have smaller LSIs; (2) nodes in the neighbor area have similar LSIs; (3) nodes adjacent to a super-node have smaller LSIs. The first observation is consistent with Theorem 3.1, and the latter two observations show consistency with Theorem 3.2.

7 Conclusion

In this paper, we present node-dependent local smoothing (NDLS), a simple and scalable graph learning method based on the local smoothing of features and labels. NDLS theoretically analyzes what influences the smoothness and gives a bound to guide how to control the extent of smoothness for different nodes. By setting a node-specific smoothing iteration, each node in NDLS can smooth its feature/label to a local-smoothing state and then help to boost the model performance. Extensive experiments on seven real-world graph datasets demonstrate the high accuracy, scalability, efficiency, and flexibility of NDLS against the state-of-the-art GNNs.

Broader Impact

NDLS can be employed in areas where graph modeling is the foremost choice, such as citation networks, social networks, chemical compounds, transaction graphs, road networks, etc. The effectiveness of NDLS when improving the predictive performance in those areas may bring a broad range of societal benefits. For example, accurately predicting the malicious accounts on transaction networks can help identify criminal behaviors such as stealing money and money laundering. Prediction on road networks can help avoid traffic overload and save people's time. A significant benefit of NDLS is that it offers a node-dependent solution. However, NDLS faces the risk of information leakage in the smoothed features or labels. In this regard, we encourage researchers to understand the privacy concerns of NDLS and investigate how to mitigate the possible information leakage.

Acknowledgments and Disclosure of Funding

This work is supported by NSFC (No. 61832001, 6197200), Beijing Academy of Artificial Intelligence (BAAI), PKU-Baidu Fund 2019BD006, and PKU-Tencent Joint Research Lab. Zhi Yang and Bin Cui are the corresponding authors.

References

[1] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2464–2473, 2020.

[2] L. Cai and S. Ji. A multi-scale approach for graph link prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3308–3315, 2020.

[3] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[4] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[5] J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 941–949, 2018.

[6] M. Chen, Z. Wei, B. Ding, Y. Li, Y. Yuan, X. Du, and J. Wen. Scalable graph neural networks via bidirectional propagation. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[7] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794, 2016.

[8] W. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. Hsieh. 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 257–266, 2019.

[9] F. R. Chung and F. C. Graham. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

[10] G. Cui, J. Zhou, C. Yang, and Z. Liu. Adaptive graph encoder for attributed graph embedding. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 976–985, 2020.

[11] H. Dihe. An introduction to markov process in random environment [j]. *Acta Mathematica Scientia*, 5, 2010.

[12] Q. Guo, X. Qiu, X. Xue, and Z. Zhang. Syntax-guided text generation via graph neural network. *Sci. China Inf. Sci.*, 64(5), 2021.

[13] A. Hagberg, P. Swart, and D. S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[14] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.

[15] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 639–648, 2020.

[16] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec. OGB-LSC: A large-scale challenge for machine learning on graphs. *CoRR*, abs/2103.09430, 2021.

[17] W. Huang, T. Zhang, Y. Rong, and J. Huang. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4563–4572, 2018.

[18] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[19] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[20] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[21] K.-H. Lai, D. Zha, K. Zhou, and X. Hu. Policy-gnn: Aggregation optimization for graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 461–471, 2020.

[22] Q. Li, Z. Han, and X.-M. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[23] Y. Li, Y. Shen, W. Zhang, Y. Chen, H. Jiang, M. Liu, J. Jiang, J. Gao, W. Wu, Z. Yang, C. Zhang, and B. Cui. Openbox: A generalized black-box optimization service. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 3209–3219, 2021.

[24] A. Liaw, M. Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[25] M. Liu, H. Gao, and S. Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 338–348, 2020.

[26] X. Miao, N. M. Gürel, W. Zhang, Z. Han, B. Li, W. Min, S. X. Rao, H. Ren, Y. Shan, Y. Shao, Y. Wang, F. Wu, H. Xue, Y. Yang, Z. Zhang, Y. Zhao, S. Zhang, Y. Wang, B. Cui, and C. Zhang. Degnn: Improving graph neural networks with graph decomposition. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 1223–1233, 2021.

[27] X. Miao, W. Zhang, Y. Shao, B. Cui, L. Chen, C. Zhang, and J. Jiang. Lasagne: A multi-layer graph convolutional network framework via node-aware deep architecture. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[28] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. M. Bronstein, and F. Monti. SIGN: scalable inception graph neural networks. *CoRR*, abs/2004.11198, 2020.

[29] I. Spinelli, S. Scardapane, and A. Uncini. Adaptive propagation graph convolutional network. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[30] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[31] K. Wang, Z. Shen, C. Huang, C.-H. Wu, Y. Dong, and A. Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020.

[32] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 6861–6871, 2019.

[33] S. Wu, W. Zhang, F. Sun, and B. Cui. Graph neural networks in recommender systems: A survey. *CoRR*, abs/2011.02260, 2020.

[34] S. Wu, Y. Zhang, C. Gao, K. Bian, and B. Cui. Garg: Anonymous recommendation of point-of-interest in mobile networks by graph convolution network. *Data Science and Engineering*, 5(4):433–447, 2020.

[35] K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 5449–5458, 2018.

[36] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[37] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna. Graphsaint: Graph sampling based inductive learning method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[38] W. Zhang, Y. Jiang, Y. Li, Z. Sheng, Y. Shen, X. Miao, L. Wang, Z. Yang, and B. Cui. ROD: reception-aware online distillation for sparse graphs. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pages 2232–2242, 2021.

[39] W. Zhang, X. Miao, Y. Shao, J. Jiang, L. Chen, O. Ruas, and B. Cui. Reliable data distillation on graph convolutional network. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1399–1414, 2020.

[40] W. Zhang, Z. Sheng, Y. Jiang, Y. Xia, J. Gao, Z. Yang, and B. Cui. Evaluating deep graph neural networks. *CoRR*, abs/2108.00955, 2021.

[41] X. Zhang, H. Liu, Q. Li, and X. Wu. Attributed graph clustering via adaptive graph convolution. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 4327–4333, 2019.

[42] H. Zhu and P. Koniusz. Simple spectral graph convolution. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

A Appendix

A.1 Proofs of Theorems

We represent the adjacency matrix and the diagonal degree matrix of graph $\mathcal{G}$ by A and D respectively, represent D+I and A+I by $\tilde{D}$ and $\tilde{A}$. Then we denote $\tilde{D}^{-1}\tilde{A}$ as a transition matrix P. Suppose P is connected, which means the graph is connected, for any initial distribution π_0, let

$$\tilde{\pi}(\pi_0) = \lim_{k\to\infty} \pi_0 P^k, \tag{15}$$

then according to [11], for any initial distribution π_0

$$\tilde{\pi}(\pi_0)_i = \frac{1}{n}\sum_{j=1}^{n} P_{ji}, \tag{16}$$

where $\tilde{\pi}_i$ denotes the i^{th} component of $\tilde{\pi}(\pi_0)$, and n denotes the number of nodes in graph. If matrix P is unconnected, we can divide P into connected blocks. Then for each blocks(denoted as B_g), there always be

$$\tilde{\pi}(\pi_0)_i = \frac{1}{n_g}\sum_{j\in B_g} P_{ji} * \sum_{j\in B_g} \pi_{0j}, \tag{17}$$

where n_g is the number of nodes in B_g. To make the proof concise, we will assume matrix P is connected, otherwise we can perform the same operation inside each block. Therefore, $\tilde{\pi}$ is independent to π_0, thus we replace $\tilde{\pi}(\pi_0)$ by $\tilde{\pi}$.

Definition A.1 (Local Mixing Time). *The local mixing time (parameterized by ϵ) with an initial distribution is defined as*

$$T(\pi_0,\epsilon) = \min\{t : ||\tilde{\pi} - \pi_0 P^t||_2 < \epsilon\}, \tag{18}$$

where "$||\cdot||_2$" symbols two-nor m.

In order to consider the impact of each node to the others separately, let π_0 = e_i, where e_i is a one-hot vector with the i^{th} component equal to 1, and the other components equal to 0. According to [9] we have lemma A.1.

Lemma A.1.

$$|(e_i P^t)_j - \tilde{\pi}_j| \leq \sqrt{\frac{\tilde{d_j}}{\tilde{d_i}}}\lambda_2^t, \tag{19}$$

where λ_2 is the second large eigenvalue of P and $\tilde{d_i}$ denotes the degree of node v_i plus 1 (to include itself).

$$\tilde{d_i} = d_i + 1, \quad \tilde{d_j} = d_j + 1,$$

Theorem A.2.

$$T(e_i,\epsilon) \leq \log_{\lambda_2}(\epsilon\sqrt{\frac{\tilde{d_i}}{2m+n}}), \tag{20}$$

where m and n denote the number of edges and nodes in graph $\mathcal{G}$ separately.

$$\tilde{d_i} = d_i + 1,$$

Proof. [9] shows that when π_0 = e_i,

$$|(e_i P^t)_j - \tilde{\pi}_j| \leq \sqrt{\frac{\tilde{d_j}}{\tilde{d_i}}}\lambda_2^t, \tag{21}$$

where $(e_i P^t)_j$ symbols the j^{th} element of $e_i P^t$. We denote $e_i P^t$ as $\pi_i(t)$, then

$$\begin{aligned} ||\tilde{\pi} - \pi_i(t)||_2^2 &= \sum_{j=1}^{n}(\tilde{\pi}_j - \pi_i(t)_j)^2 \\ &\leq \frac{\sum\limits_{j=1}^{n}\tilde{d_j}}{\tilde{d_i}}\lambda_2^{2t} = \frac{2m+n}{\tilde{d_i}}\lambda_2^{2t}, \end{aligned} \tag{22}$$

which means

$$||\tilde{\pi} - \pi_i(t)||_2 \leq \sqrt{\frac{2m+n}{\tilde{d}_i}}\lambda_2^t. \tag{23}$$

Now let

$$\epsilon = \sqrt{\frac{2m+n}{\tilde{d}_i}}\lambda_2^t,$$

there exists

$$T(e_i, \epsilon) \leq \log_{\lambda_2}(\epsilon\sqrt{\frac{\tilde{d}_i}{2m+n}}).$$

□

Next consider the real situation in SGC with $n \times m$-dimension matrix $X(0)$ as input, where n is the number of nodes, m is the number of features. We apply P as the normalized adjacent matrix.(The definition of P is the same as $\tilde{\mathbf{A}}$ in main text). In feature propagation we have

$$X(t) = P^t X(0),$$

Now consider the h^{th} feature of X, we define an $n \times n$ influence matrix

$$I_{hij}(t) = \frac{\partial X(t)_{ih}}{\partial X(0)_{jh}}, \tag{24}$$

Because $I_h(k)$ is independent to h, we replace $I_h(k)$ by $I(k)$, which can be formulated as

$$I(k) = I_h(k), \quad \forall h \in \{1, 2, .., f\}, \tag{25}$$

where f symbols the number of features of X.

Definition A.2 (Local Smoothing Iteration). *The Local Smoothing Iteration (parameterized by ϵ) is defined as*

$$K(i, \epsilon) = \min\{k : ||\tilde{I}_i - I_i(k)||_2 < \epsilon\}. \tag{26}$$

According to Theorem A.2, there exists

Theorem A.3 (Theorem 3.1 in main text). *When the normalized adjacent matrix is P,*

$$K(i, \epsilon) \leq \log_{\lambda_2}(\epsilon\sqrt{\frac{\tilde{d}_i}{2m+n}}). \tag{27}$$

Proof. From equation (9) we can derive that

$$||e_i P^\infty - e_i P^k||_2 \leq \sqrt{\frac{2m+n}{\tilde{d}_i}}\lambda_2^k.$$

Because

$$I_i(k) = P_i^k = e_i P^k \quad I_i(\infty) = P_i^\infty = e_i P^\infty,$$

we have

$$||I_i(\infty) - I_i(k)||_2 \leq \sqrt{\frac{2m+n}{\tilde{d}_i}}\lambda_2^k.$$

Now let

$$\epsilon = \sqrt{\frac{2m+n}{\tilde{d}_i}}\lambda_2^k,$$

there exists

$$K(i, \epsilon) \leq \log_{\lambda_2}(\epsilon\sqrt{\frac{\tilde{d}_i}{2m+n}}).$$

□

Therefore, we expand Theorem A.3 to the propagation in SGC or our method. What is remarkable, Theorem A.3 requires P, which is equal to $\widetilde{D}^{-1}\tilde{A}$ as the normalized adjacent matrix.

From Theorem A.3 we can conclude that the node which has a lager degree may need more steps to propagate. At the same time, we have another bond of local mixing time as following.

Theorem A.4. *For each node v_i in graph $\mathcal{G}$, there always exits*

$$T(e_i, \epsilon) \leq \max\{T(e_j, \epsilon), j \in N(i)\} + 1. \tag{28}$$

where N(i) is the set of node v_i's neighbours.

Proof.

$$\begin{aligned} |||\tilde{\pi} - e_i P^{t+1}||_2 &= \frac{1}{|N(i)|} \sum_{j \in N(i)} ||\tilde{\pi} - e_j P^t||_2 \\ &\leq \max_{j \in N(i)} ||\tilde{\pi} - e_j P^t||_2. \end{aligned} \tag{29}$$

Therefore, when

$$\max_{j \in N(i)} ||\tilde{\pi} - e_j P^t||_2 \leq \epsilon,$$

there exists

$$||\tilde{\pi} - e_i P^{t+1}||_2 \leq \epsilon.$$

Thus we can derive that

$$T(e_i, \epsilon) \leq \max\{T(e_j, \epsilon), j \in N(i)\} + 1.$$

□

As we extend Theorem A.2 to Theorem A.3, according to Theorem A.4, there always be

Theorem A.5 (Theorem 3.2 in main text). *For each node v_i in graph $\mathcal{G}$, there always exits*

$$K(i, \epsilon) \leq \max\{K(j, \epsilon), j \in N(i)\} + 1. \tag{30}$$

A.2 Results with More Base Models

Our proposed NDLS consists of three stages: (1) feature smoothing with NDLS (NDLS-F), (2) model training with smoothed features, and (3) label smoothing with NDLS (NDLS-L). In stage (2), the default option of the base model is a Multilayer Perceptron (MLP). Besides MLP, many other models can also be used in stage (2) to generate soft labels. To verify it, here we replace the MLP in stage (2) with popular machine learning models Random Forest [24] and XGBoost [7], and measure their node classification performance on PubMed dataset. The experiment results are shown in Table 3 where Random Forest and XGBoost are abbreviated as *RF* and *XGB* respectively.

Compared to the vanilla model, both Random Forest and XGBoost achieve significant performance gain with the addition of our NDLS. With the help of NDLS, Random Forest and XGBoost outperforms their base models by 6.1% and 7.5% respectively. From Table 3, we can observe that both NDLS-F and NDLS-L can contribute great performance boost to the base model, where the gains are at least 5%. When all equipped with both NDLS-F and NDLS-L, XGBoost beat the default MLP, achieving a test accuracy of 81.6%. Although Random Forest – 80.5% – cannot outperform the other two models, it is still a competitive model.

The above experiment demonstrates that the base model selection in stage (2) is rather flexible in our NDLS. Both traditional machine learning methods and neural networks are promising candidates in the proposed method.

A.3 Dataset Description

Cora, **Citeseer**, and **Pubmed**[1] are three popular citation network datasets, and we follow the public training/validation/test split in GCN [18]. In these three networks, papers from different topics are

[1] https://github.com/tkipf/gcn/tree/master/gcn/data

Table 5: Results of different base models on PubMed.

Base Models	Models	Accuracy	Gain
MLP	Base	72.7±0.6	-
	+ NDLS-F	81.1±0.6	+ 8.4
	+ NDLS-L	81.1±0.6	+ 8.4
	+ NDLS (both)	**81.4**±**0.4**	+ 8.7
RF	Base	74.4±0.2	-
	+ NDLS-F	80.3±0.1	+ 5.9
	+ NDLS-L	80.0±0.2	+ 5.6
	+ NDLS (both)	**80.5**±**0.4**	+ 6.1
XGB	Base	74.1±0.2	-
	+ NDLS-F	81.0±0.3	+ 6.9
	+ NDLS-L	79.8±0.2	+ 5.7
	+ NDLS (both)	**81.6**±**0.3**	+ 7.5

considered as nodes, and the edges are citations among the papers. The node attributes are binary word vectors, and class labels are the topics papers belong to.

Reddit is a social network dataset derived from the community structure of numerous Reddit posts. It is a well-known inductive training dataset, and the training/validation/test split in our experiment is the same as the one in GraphSAGE [14].

Flickr originates from NUS-wide [2] and contains different types of images based on the descriptions and common properties of online images. The public version of Reddit and Flickr provided by GraphSAINT[3] is used in our paper.

Industry is a short-form video graph, collected from a real-world mobile application from our industrial cooperative enterprise. We sampled 1,000,000 users and videos from the app, and treat these items as nodes. The edges in the generated bipartite graph represent that the user clicks the short-form videos. Each user has 64 features and the target is to category these short-form videos into 253 different classes.

ogbn-papers100M is a directed citation graph of 111 million papers indexed by MAG [31]. Among its node set, approximately 1.5 million of them are arXiv papers, each of which is manually labeled with one of arXiv's subject areas. Currently, this dataset is much larger than any existing public node classification datasets.

A.4 Compared Baselines

The main characteristic of all baselines are listed below:

- **GCN** [18]: GCN is a novel and efficient method for semi-supervised classification on graph-structured data.
- **GAT** [30]: GAT leverages masked self-attention layers to specify different weights to different nodes in a neighborhood, thus better represent graph information.
- **JK-Net** [35]: JK-Net is a flexible network embedding method that could gather different neighborhood ranges to enable better structure-aware representation.
- **APPNP** [19]: APPNP uses the relationship between graph convolution networks (GCN) and PageRank to derive improved node representations.
- **AP-GCN** [29]: AP-GCN uses a halting unit to decide a receptive range of a given node.

[2] http://lms.comp.nus.edu.sg/research/NUS-WIDE.html

[3] https://github.com/GraphSAINT/GraphSAINT

Table 6: URLs of baseline codes.

Type	Baselines	URLs
Coupled	GCN	https://github.com/rusty1s/pytorch_geometric
	GAT	https://github.com/rusty1s/pytorch_geometric
Decoupled	APPNP	https://github.com/rusty1s/pytorch_geometric
	PPRGo	https://github.com/TUM-DAML/pprgo_pytorch
	AP-GCN	https://github.com/spindro/AP-GCN
	DAGNN	https://github.com/divelab/DeeperGNN
Sampling	GraphSAGE	https://github.com/williamleif/GraphSAGE
	GraphSAINT	https://github.com/GraphSAINT/GraphSAINT
	FastGCN	https://github.com/matenure/FastGCN
	Cluster-GCN	https://github.com/benedekrozemberczki/ClusterGCN
Linear	SGC	https://github.com/Tiiiger/SGC
	SIGN	https://github.com/twitter-research/sign
	S^{2}GC	https://github.com/allenhaozhu/SSGC
	GBP	https://github.com/chennnM/GBP
	NDLS	https://github.com/zwt233/NDLS

- **DAGNN** [25]: DAGNN proposes to decouple the representation transformation and propagation, and show that deep graph neural networks without this entanglement can leverage large receptive fields without suffering from performance deterioration.
- **PPRGo** [1]: utilizes an efficient approximation of information diffusion in GNNs resulting in significant speed gains while maintaining state-of-the-art prediction performance.
- **GraphSAGE** [14]: GraphSAGE is an inductive framework that leverages node attribute information to efficiently generate representations on previously unseen data.
- **FastGCN** [4]: FastGCN interprets graph convolutions as integral transforms of embedding functions under probability measures.
- **Cluster-GCN** [8]: Cluster-GCN is a novel GCN algorithm that is suitable for SGD-based training by exploiting the graph clustering structure.
- **GraphSAINT** [37]: GraphSAINT constructs mini-batches by sampling the training graph, rather than the nodes or edges across GCN layers.
- **SGC** [32]: SGC simplifies GCN by removing nonlinearities and collapsing weight matrices between consecutive layers.
- **SIGN** [28]: SIGN is an efficient and scalable graph embedding method that sidesteps graph sampling in GCN and uses different local graph operators to support different tasks.
- **S^{2}GC** [42]: S^{2}GC uses a modified Markov Diffusion Kernel to derive a variant of GCN, and it can be used as a trade-off of low-pass and high-pass filter which captures the global and local contexts of each node.
- **GBP** [6]: GBP utilizes a localized bidirectional propagation process from both the feature vectors and the training/testing nodes

Table 6 summarizes the github URLs of the compared baselines. Following the original paper, we implement JK-Net by ourself since there is no official version available.

A.5 Implementation Details

Hyperparameter details. In stage (1), when computing the Local Smoothing Iteration, the maximal value of k in equation (12) is set to 200 and the optimal ϵ value is get by means of a grid search from {0.01, 0.03, 0.05}. In stage (2), we use a simple two-layer MLP to get the base prediction. The hidden size is set to 64 in small datasets – Cora, Citeseer and Pubmed. While in larger datasets – Flicker, Reddit, Industry and ogbn-papers100M, the hidden size is set to 256. As for the dropout percentage and the learning rate, we use a grid search from {0.2, 0.4, 0.6, 0.8} and {0.1,

Table 7: Performance comparison between C&S and NDLS-L

Methods	Cora	Citeseer	PubMed	ogbn-papers100M
MLP+C&S	87.2	76.6	88.3	63.9
MLP+NDLS-L	**88.1**	**78.3**	**88.5**	**64.6**

Table 8: Performance comparison under varied label rate on the Cora dataset.

Methods	2%	5%	10%	20%	40%	60%
MLP+S	63.1	77.8	82.6	84.2	85.4	86.4
MLP+C&S	62.8	76.7	82.8	84.9	86.4	87.2
MLP+NDLS-L	**77.4**	**83.9**	**85.3**	**86.5**	**87.6**	**88.1**

Table 9: Performance comparison after combining the node-dependent idea with C&S.

Methods	Cora	Citeseer	PubMed
MLP+C&S	76.7	70.8	76.5
MLP+C&S+nd	**79.9**	**71.1**	**78.4**

0.01, 0.001} respectively. In stage (3), during the computation of the Local Smoothing Iteration, the maximal value of k is set to 40. The optimal value of ϵ is obtained through the same process in stage (1).

Implementation environment. The experiments are conducted on a machine with Intel(R) Xeon(R) Gold 5120 CPU @ 2.20GHz, and a single NVIDIA TITAN RTX GPU with 24GB memory. The operating system of the machine is Ubuntu 16.04. As for software versions, we use Python 3.6, Pytorch 1.7.1 and CUDA 10.1.

A.6 Comparison and Combination with Correct&Smooth

Similar to our NDLS-L, Correct and Smooth (C&S) also applies post-processing on the model prediction. Therefore, we compare NDLS-L with C&S below.

Adaptivity to node. C&S adopts a propagation scheme based on Personalized PageRank (PPR), which always maintains certain input information to slow down the occurrence of over-smoothing. The expected number of smoothing iterations is controlled by the restart probability, which is a constant for all nodes. Therefore, C&S still falls into the routine of fixed smoothing iteration. Instead, NDLS-L employs node-specific smoothing iterations. We compare each method's performance (test accuracy, %) under the same data split as in the C&S paper (60%/20%/20% on three citation networks, official split on ogbn-papers100M), and the experimental results in Table 7 show that NDLS-L outperforms C&S in different datasets.

Sensitivity to label rate. During the "Correct" stage, C&S propagates uncertainties from the training data across the graph to correct the base predictions. However, the uncertainties might not be accurate when the number of training nodes is relatively small, thus even degrading the performance. To confirm the above assumption, we conduct experiments on the Cora dataset under different label rates, and the experimental results are provided in Table 8. As illustrated, the result of C&S drops much faster than NDLS-L's when the label rate decreases. What's more, MLP+S (removing the "Correct" stage) outperforms MLP+C&S when the label rate is low as expected.

Compared with C&S, NDLS is more general in terms of smoothing types. C&S can only smooth label predictions. Instead, NDLS can smooth both node features and label predictions and combine them to boost the model performance further.

Table 10: Efficiency comparison on the PubMed dataset.

	SGC	S^{2}GC	GBP	NDLS	SIGN	JK-Net	DAGNN	GCN	ResGCN	APPNP	GAT
Time	1.00	1.19	1.20	1.50	1.59	11.42	14.39	20.43	20.49	28.88	33.23
Accuracy	78.9	79.9	80.6	**81.4**	79.5	78.8	80.5	79.3	78.6	80.1	79.0

Table 11: Performance comparison on the ogbn-arxiv dataset.

	MLP	MLP+C&S	GCN	SGC	SIGN	DAGNN	JK-Net	S^{2}GC	GBP	NDLS	GAT
Accuracy	55.50	71.58	71.74	71.72	71.95	72.09	72.19	72.21	72.45	73.04	**73.56**

Node Adaptive C&S. The node-dependent mechanism in our NDLS can easily be combined with C&S. The two stages of C&S both contain a smoothing process using the personalized PageRank matrix, where a coefficient controls the remaining percentage of the original node feature. Here, we can precompute the smoothed node features after the same smoothing step yet under different values like 0.1, 0.2, ..., 0.9. After that, we adopt the same strategy in our NDLS: for each node, we choose the first in the ascending order that the distance from the smoothed node feature to the stationarity is less than a tuned hyperparameter. By this means, the smoothing process in C&S can be carried out in a node-dependent way.

We also evaluate the performance of C&S combined with the node-dependent idea (represented as C&S+nd) on the three citation networks under official splits, and the experimental results in Table 9 show that C&S combined with NDLS consistently outperforms the original version of C&S.

A.7 Training Efficiency Study

we measure the training efficiency of the compared baselines on the widely used PubMed dataset. Using the training time of SGC as the baseline, the relative training time and the corresponding test accuracy of NDLS and the baseline methods are shown in Table 10. Compared with other baselines, NDLS can get the highest test accuracy while maintaining competitive training efficiency.

A.8 Experiments on ogbn-arxiv

We also conduct experiments on the ogbn-arxiv dataset. The experiment results (test accuracy, %) are provided in Table 11. Although GAT outperforms NDLS on ogbn-arxiv dataset, it is hard to scale to large graphs like ogbn-papers100M dataset. Note that MLP+C&S on the OGB leaderboard makes use of not only the original node feature but also diffusion embeddings and spectral embeddings. Here we remove the latter two embeddings for fairness, and the authentic MLP+C&S achieves 71.58% on the ogbn-arxiv dataset.