A UNIFIED VIEW ON GRAPH NEURAL NETWORKS AS GRAPH SIGNAL DENOISING

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) have risen to prominence in learning representations for graph structured data. A single GNN layer typically consists of a feature transformation and a feature aggregation operation. The former normally uses feed-forward networks to transform features, while the latter aggregates the transformed features over the graph. Numerous recent works have proposed GNN models with different designs in the aggregation operation. In this work, we establish mathematically that the aggregation processes in a group of representative GNN models including GCN, GAT, PPNP, and APPNP can be regarded as (approximately) solving a graph denoising problem with a smoothness assumption. Such a unified view across GNNs not only provides a new perspective to understand a variety of aggregation operations but also enables us to develop a unified graph neural network framework UGNN. To demonstrate its promising potential, we instantiate a novel GNN model, ADA-UGNN, derived from UGNN, to handle graphs with adaptive smoothness across nodes. Comprehensive experiments show the effectiveness of ADA-UGNN.

1 INTRODUCTION

Graph Neural Networks (GNNs) have shown great capacity in learning representations for graph-structured data and thus have facilitated many down-stream tasks such as node classification (Kipf & Welling, 2016; Veličković et al., 2017; Ying et al., 2018a; Klicpera et al., 2018) and graph classification (Defferrard et al., 2016; Ying et al., 2018b). As traditional deep learning models, a GNN model is usually composed of several stacking GNN layers. Given a graph $\mathcal{G}$ with N nodes, a typical GNN layer typically contains a feature transformation and an aggregation operation as:

$$\text{Feature Transformation: } \mathbf{X}'_{in} = f_{trans}(\mathbf{X}_{in}); \quad \text{Feature Aggregation: } \mathbf{X}_{out} = f_{agg}(\mathbf{X}'_{in}; \mathcal{G}) \quad (1)$$

where $\mathbf{X}_{in} \in \mathbb{R}^{N \times d_{in}}$ and $\mathbf{X}_{out} \in \mathbb{R}^{N \times d_{out}}$ denote the input and output features of the GNN layer with d_{in} and d_{out} as the corresponding dimensions, respectively. Note that the non-linear activation is not included in Eq. (1) to ease the discussion. The feature transformation operation $f_{trans}(\cdot)$ transforms the input of $\mathbf{X}_{in}$ to $\mathbf{X}'_{in} \in \mathbb{R}^{N \times d_{out}}$ as its output; and the feature aggregation operation $f_{agg}(\cdot; \mathcal{G})$ updates the node features by aggregating the transformed node features via the graph $\mathcal{G}$.

In general, different GNN models share similar feature transformations (often, a single feed-forward layer), while adopting different designs for aggregation operation. We raise a natural question – is there an intrinsic connection among these feature aggregation operations and their assumptions? The significance of a positive answer to this question is two-fold. Firstly, it offers a new perspective to create a uniform understanding on representative aggregation operations. Secondly, it enables us to develop a general GNN framework that not only provides a unified view on multiple existing representative GNN models, but also has the potential to inspire new ones. In this paper, we aim to build the connection among the representative feature aggregation operations including GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2017), PPNP and APPNP (Klicpera et al., 2018). In particular, we mathematically establish that the aggregation operations in these models can be unified as the process of exactly, and sometimes approximately, addressing a graph signal denoising problem with Laplacian regularization (Shuman et al., 2013). This connection suggests that these aggregation operations share a unified goal: to ensure feature smoothness of connected nodes. With this understanding, we propose a general GNN framework, UGNN, which not only provides a straightforward, unified view for many existing aggregation operations, but also suggests various promising directions to build new aggregation operations suitable for distinct applications. To demonstrate its potential, we build an instance of UGNN called ADA-UGNN, which is suited for handling varying smoothness properties across nodes, and conduct experiments to show its effectiveness.

2 REPRESENTATIVE GRAPH NEURAL NETWORKS

In this section, we introduce notations for graphs and briefly summarize several representative GNN models. A graph can be denoted as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ and $\mathcal{E}$ are its corresponding node and edge sets. $\mathcal{G}$ can be represented as an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, with N the number of nodes in the graph. The Laplacian matrix of the graph $\mathcal{G}$ is denoted as $\mathbf{L}$. It is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is a diagonal degree matrix corresponding to $\mathbf{A}$. There are also normalized versions of the Laplacian matrix such as $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ or $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$. In this work, we sometimes adopt different Laplacians to establish connections between GNNs and the graph denoising problem, clarifying in the text. Each node is associated with a d-dimensional signal $\mathbf{X} \in \mathbb{R}^{N\times d}$. In this section, we generally use $\mathbf{X}_{in} \in \mathbb{R}^{N\times d_{in}}$ and $\mathbf{X}_{out} \in \mathbb{R}^{N\times d_{out}}$ to denote input and output features of GNN layers. Next, we describe a few representative GNN models.

2.1 GRAPH CONVOLUTIONAL NETWORKS (GCN)

Following Eq. (1), a single layer in GCN (Kipf & Welling, 2016) can be written as follows:

$$\text{Feature Transformation: } \mathbf{X}'_{in} = \mathbf{X}_{in}\mathbf{W}; \quad \text{Feature Aggregation: } \mathbf{X}_{out} = \tilde{\mathbf{A}}\mathbf{X}'_{in}, \tag{2}$$

where $\mathbf{W} \in \mathbb{R}^{d_{in}\times d_{out}}$ is a feature transformation matrix, and $\tilde{\mathbf{A}}$ is a normalized adjacency matrix which includes a self-loop, defined as follows:

$$\tilde{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}, \quad \text{with} \quad \hat{\mathbf{A}} = \mathbf{A} + \mathbf{I} \quad \text{and} \quad \mathbf{D} = \text{diag}(\sum_j \hat{\mathbf{A}}_{1,j}, \ldots, \sum_j \hat{\mathbf{A}}_{N,j}). \tag{3}$$

In practice, multiple GCN layers can be stacked, where each layer takes the output of its previous layer as input. Non-linear activation functions are included between consecutive layers.

2.2 GRAPH ATTENTION NETWORKS (GAT)

Graph Attention Networks (GAT) adopts the same feature transformation operation as GCN in Eq. (2). The feature aggregation operation (written node-wise) for a node v_i is as:

$$\mathbf{X}_{out}[i,:] = \sum_{v_i \in \tilde{\mathcal{N}}(v_i)} \alpha_{ij}\mathbf{X}'_{in}[j,:], \quad \text{with} \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{v_k \in \tilde{\mathcal{N}}(v_i)} \exp(e_{ik})}. \tag{4}$$

where $\tilde{\mathcal{N}}(v_i) = \mathcal{N}(v_i) \cup \{v_i\}$ denotes the neighbors (self-inclusive) of node v_i, and $\mathbf{X}_{out}[i,:]$ is the i-th row of the matrix $\mathbf{X}_{out}$, i.e. the output node features of node v_i. In this aggregation operation, α_{ij} is a learnable attention score to differentiate the importance of distinct nodes in the neighborhood. Specifically, α_{ij} is a normalized form of e_{ij}, which is modeled as:

$$e_{ij} = \text{LeakyReLU}\left(\left[\mathbf{X}'_{in}[i,:] \| \mathbf{X}'_{in}[j,:]\right]\mathbf{a}\right) \tag{5}$$

where $[\cdot\|\cdot]$ denotes the concatenation operation and $\mathbf{a} \in \mathbb{R}^{2d}$ is a learnable vector. Similar to GCN, a GAT model usually consists of multiple stacked GAT layers.

2.3 PERSONALIZED PROPAGATION OF NEURAL PREDICTIONS (PPNP)

Personalized Propagation of Neural Predictions (PPNP) (Klicpera et al., 2018) introduces an aggregation operation based on Personalized PageRank (PPR). Specifically, the PPR matrix is defined as $\alpha(\mathbf{I} - (1-\alpha)\tilde{\mathbf{A}})^{-1}$, where $\alpha \in (0,1)$ is a hyper-parameter. The ij-th element of the PPR matrix specifies the influence of node v_i on node v_j. The feature transformation operation is modeled as Multi-layer Perception (MLP). The PPNP model can be written in the form of Eq. (1) as follows:

$$\begin{aligned} &\text{Feature Transformation: } \mathbf{X}'_{in} = \text{MLP}(\mathbf{X}_{in}); \\ &\text{Feature Aggregation: } \mathbf{X}_{out} = \alpha(\mathbf{I} - (1-\alpha)\tilde{\mathbf{A}})^{-1}\mathbf{X}'_{in}. \end{aligned} \tag{6}$$

Unlike GCN and GAT, PPNP only consists of a single feature aggregation layer, but with a potentially deep feature transformation. Since the matrix inverse in Eq. (6) is costly, (Klicpera et al., 2018) also introduces a practical, approximated version of PPNP, called APPNP, where the aggregation operation is performed in an iterative way as:

$$\mathbf{X}^{(k)}_{out} = (1-\alpha)\tilde{\mathbf{A}}\mathbf{X}^{(k-1)}_{out} + \alpha\mathbf{X}'_{in} \quad k = 1, \ldots K, \tag{7}$$

where $\mathbf{X}^{(0)}_{out} = \mathbf{X}'_{in}$ and $\mathbf{X}^{(K)}_{out}$ is the output of the aggregation operation. As proved in (Klicpera et al., 2018), $\mathbf{X}^{(K)}_{out}$ converges to the solution obtained by PPNP, i.e., $\mathbf{X}_{out}$ in Eq. (6).

3 GNNs AS GRAPH SIGNAL DENOISING

In this section, we aim to establish the connections between the introduced GNN models and a graph signal denoising problem with Laplacian regularization. We first introduce the problem.

Problem 1 (Graph Signal Denoising with Laplacian Regularization)**.** *Suppose that we are given a noisy signal* $\mathbf{X} \in \mathbb{R}^{N\times d}$ *on a graph* $\mathcal{G}$*. The goal of the problem is to recover a clean signal* $\mathbf{F} \in \mathbb{R}^{N\times d}$*, assumed to be smooth over* $\mathcal{G}$*, by solving the following optimization problem:*

$$\arg\min_{\mathbf{F}} \ \mathcal{L} = \|\mathbf{F} - \mathbf{X}\|_F^2 + c \cdot tr(\mathbf{F}^\top \mathbf{L}\mathbf{F}), \tag{8}$$

Note that the first term guides $\mathbf{F}$ to be close to $\mathbf{X}$, while the second term $tr(\mathbf{F}^\top \mathbf{L}\mathbf{F})$ is the Laplacian regularization that guides the smoothness of $\mathbf{F}$ over the graph. $c > 0$ is a balancing constant. Assuming we adopt the unnormalized version of Laplacian matrix with $\mathbf{L} = \mathbf{D} - \mathbf{A}$, the second term in Eq. (8) can be written in an edge-centric way or a node-centric way as:

$$\text{edge-centric: } c \sum_{(v_i,v_j)\in\mathcal{E}} \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2 \,, ; \quad \text{node-centric: } \frac{1}{2} c \sum_{v_i\in\mathcal{V}} \sum_{v_j\in\tilde{\mathcal{N}}(v_i)} \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2 \,. \tag{9}$$

Clearly, from the edge-centric view, the regularization term measures the *global smoothness* of $\mathbf{F}$, which is small when connected nodes share similar features. On the other hand, we can view the term $\sum_{v_j\in\tilde{\mathcal{N}}(v_i)} \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2$ as a *local smoothness* measure for node v_i as it measures the difference between node v_i and all its neighbors. The regularization term can then be regarded as a summation of local smoothness over all nodes. Note that similar definitions can also be made to other types of Laplacian matrices. In the following subsections, we demonstrate the connections between aggregation operations in various GNN models and the graph signal denoising problem.

3.1 CONNECTION TO PPNP AND APPNP

In this subsection, we establish the connection between the graph signal denoising problem (8) and the aggregation propagations in PPNP and APPNP in Theorem 1 and Theorem 2, respectively. Their proofs can be found in Appendix A.1 and Appendix A.2, respectively.

Theorem 1. *When we adopt the normalized Laplacian matrix* $\mathbf{L} = \mathbf{I} - \tilde{\mathbf{A}}$*, with* $\tilde{\mathbf{A}}$ *defined in Eq. (3), the feature aggregation operation in PPNP (Eq. (6)) can be regarded as exactly solving the graph signal denoising problem (8) with* $\mathbf{X}'_{in}$ *as the input noisy signal and* $c = \frac{1}{\alpha} - 1$*.*

Theorem 2. *When we adopt the normalized Laplacian matrix* $\mathbf{L} = \mathbf{I} - \tilde{\mathbf{A}}$*, the feature aggregation operation in APPNP (Eq. (7)) approximately solves the graph signal denoising problem (8) by iterative gradient descent with* $\mathbf{X}'_{in}$ *as the input noisy signal,* $c = \frac{1}{\alpha} - 1$ *and stepsize* $b = \frac{1}{2+2c}$*.*

These two connections provide a new explanation on the hyper-parameter α in PPNP and APPNP from the graph signal denoising perspective. Specifically, a smaller α indicates a larger c, which means the obtained $\mathbf{X}_{out}$ is enforced to be smoother over the graph.

3.2 CONNECTION TO GCN

We draw the connection between the GCN model (Kipf & Welling, 2016) and the graph signal denoising problem in Theorem 3. The proof can be found in Appendix A.3.

Theorem 3. *When we adopt the normalized Laplacian matrix* $\mathbf{L} = \mathbf{I} - \tilde{\mathbf{A}}$*, the feature aggregation operation in GCN Eq. (2) can be regarded as solving the graph signal denoising problem (8) using one-step gradient descent with* $\mathbf{X}'_{in}$ *as the input noisy signal and stepsize* $b = \frac{1}{2c}$*.*

With this connection, it is easy to verify that a GCN model with multiple GCN layers can be regarded as solving the graph signal denoising problem multiple times with different noisy signals. Specifically, each layer of a GCN model corresponds to a graph signal denoising problem, where the input noisy signal is the output from the previous layer after the feature transformation of the current layer. Note that there are earlier works (NT & Maehara, 2019; Zhao & Akoglu, 2019) drawing connection between GCN and the optimization problem in Eq. (8), where the aggregation operation in GCN is shown to be the first-order approximation of the exact solution.

3.3 CONNECTION TO GAT

To establish the connection between graph signal denoising and GAT (Veličković et al., 2017), in this subsection, we adopt an unnormalized version of the Laplacian. It is defined based on the adjacency

matrix with self-loop $\hat{\mathbf{A}}$, i.e. $\mathbf{L} = \hat{\mathbf{D}} - \hat{\mathbf{A}}$ with $\hat{\mathbf{D}}$ denoting the diagonal degree matrix of $\hat{\mathbf{A}}$. Then, the denoising problem in Eq. (8) can be rewritten from a node-centric view as:

$$\arg\min_{\mathbf{F}} \ \mathcal{L} = \sum_{v_i \in \mathcal{V}} \|\mathbf{F}[i,:] - \mathbf{X}[i,:]\|_2^2 + \frac{1}{2} \sum_{v_i \in \mathcal{V}} c \cdot \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2 \,, \tag{10}$$

where $\tilde{\mathcal{N}}(v_i) = \mathcal{N}(v_i) \cup \{v_i\}$ denotes the neighbors (self-inclusive) of node v_i. In Eq. (10), the constant c is shared by all nodes, which indicates that the same level of *local smoothness* is enforced to all nodes. However, nodes in a real-world graph can have varied levels of local smoothness. Hence, for nodes with low local smoothness, we should impose a relatively smaller c, while for those nodes with higher local smoothness, we need a larger c. Hence, instead of a unified c as in Eq. (10), we could consider a node-dependent c_i for each node v_i. Then, the optimization problem in Eq. (10) can be adjusted as:

$$\arg\min_{\mathbf{F}} \ \mathcal{L} = \sum_{v_i \in \mathcal{V}} \|\mathbf{F}[i,:] - \mathbf{X}[i,:]\|_2^2 + \frac{1}{2} \sum_{v_i \in \mathcal{V}} c_i \cdot \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2 \tag{11}$$

We next show that the aggregation operation in GAT is closely connected to an approximate solution of problem (11) with the help of the following theorem. The proof can be found in Appendix A.4.

Theorem 4. *With adaptive stepsize* $b_i = 1/\left(\sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\right)$ *for each node* v_i*, the process of taking one step of gradient descent from* $\mathbf{X}$ *to solve problem (11) can be described as follows:*

$$\mathbf{F}[i,:] \leftarrow \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} b_i (c_i + c_j) \mathbf{X}[j,:]. \tag{12}$$

Eq. (12) resembles the aggregation operation of GAT in Eq. (4) if we treat $b_i(c_i + c_j)$ as the attention score α_{ij}. In this case, b can be regarded as a normalization term that enforces the summation of weights to 1. We further compare $b_i(c_i + c_j)$ with α_{ij} by investigating the formulation of e_{ij} in Eq. (5). Eq. (5) can be rewritten as:

$$e_{ij} = \text{LeakyReLU}\left(\mathbf{X}'_{in}[i,:]\mathbf{a}_1 + \mathbf{X}'_{in}[j,:]\mathbf{a}_2\right) \tag{13}$$

where $\mathbf{a}_1$ and $\mathbf{a}_2$ are column vectors, which can be concatenated. Note that the aggregation operations in GAT and in Eq. (12) are similar if the unnormalized attention score e_{ij} in GAT and $(c_i + c_j)$ in Eq. (12) have similar designs. Specifically, $\mathbf{X}_i'^{\top}\mathbf{a}_1$ and $\mathbf{X}_j'^{\top}\mathbf{a}_2$ can be regarded as the approximations of c_i and c_j, respectively. However, the normalization in Eq. (12) is achieved via summation rather than a softmax as in GAT. By viewing the attention mechanism in GAT from the perspective of Eq. (12), namely that c_i actually indicates a notion of local smoothness for node v_i, we can develop other ways to parameterize c_i. For example, instead of directly using the node features of v_i as an indicator of local smoothness like GAT, we can consider the neighborhood information. In fact, we adopt this idea to design a new aggregation operation in Section 5.

4 UGNN: A UNIFIED GNN FRAMEWORK VIA GRAPH SIGNAL DENOISING

In the previous section, we established that the aggregation operations in PPNP, APPNP, GCN and GAT are intimately connected to the graph signal denoising problem with (generalized) Laplacian regularization. In particular, from this perspective, all their aggregation operations aim to ensure feature smoothness: either a global smoothness over the graph as in PPNP, APPNP and GCN, or a local smoothness for each node as in GAT. This understanding allows us to develop a unified feature aggregation operation by posing the following, more general graph signal denoising problem:

Problem 2 (Generalized UGNN Graph Signal Denoising Problem)**.**

$$\arg\min_{\mathbf{F}} \ \mathcal{L} = \|\mathbf{F} - \mathbf{X}\|_F^2 + r(\mathcal{C}, \mathbf{F}, \mathcal{G}), \tag{14}$$

where $r(\mathcal{C}, \mathbf{F}, \mathcal{G})$ *denotes a flexible regularization term to enforce some prior over* $\mathbf{F}$*.*

Note that we overload the notation $\mathcal{C}$ here: it can function as a scalar (like a global constant in GCN), a vector (like node-wise constants in GAT) or even a matrix (edge-wise constants) if we want to give flexibility to each node pair. Different choices of $r(\cdot)$ imply different feature aggregation operations. Besides PPNP, APPNP, GCN and GAT, there are aggregation operations in more GNN models that can be regarded as solving Problem 2 with different regularization terms such as PairNorm (Zhao & Akoglu, 2019) and DropEdge (Rong et al., 2019) (more details can be found in Appendix B). The above motioned regularization terms are all related to the Laplacian regularization. Other regularization terms can also be adopted, which may lead to novel designs of GNN layers. For example, if

we aim to enforce that the clean signal is piece-wise linear, we can adopt $r(\mathcal{C}, \mathbf{F}, \mathcal{G}) = \mathcal{C} \cdot \|\mathbf{LF}\|_1$ designed for trend filtering (Tibshirani et al., 2014; Wang et al., 2016).

With these discussions, we propose a unified framework (UGNN) to design GNN layers from the graph signal processing perspective as: (1) Design a graph regularization term $r(\mathcal{C}, \mathbf{F}, \mathcal{G})$ in Problem 2 according to specific applications; (2) Feature Transformation: $\mathbf{X}'_{in} = f_{trans}(\mathbf{X}_{in})$; and (3) Feature Aggregation: Solving Problem 2 with $\mathbf{X} = \mathbf{X}'_{in}$ and the designed $r(\mathcal{C}, \mathbf{F}, \mathcal{G})$. To demonstrate the potential of UGNN, next we introduce a new GNN model ADA-UGNN by instantiating UGNN with $r(\mathcal{C}, \mathbf{F}, \mathcal{G})$ enforcing adaptive local smoothness across nodes. Note that we introduce ADA-UGNN with node classification as the downstream task.

5 ADA-UGNN: ADAPTIVE LOCAL SMOOTHING WITH UGNN

From the graph signal denoising perspective, PPNP, APPNP and GCN enforces global smoothness by penalizing the difference with a constant c for all nodes. However, real-world graphs may consist of multiple groups of nodes which have different behaviors in connecting to similar neighbors. For example, Section 6.1 shows several graphs with varying distributions of local smoothness (as measured by label homophily): summarily, not all nodes are highly label-homophilic, and some nodes have considerably "noisier" neighborhoods than others. Moreover, as suggested by (Wu et al., 2019; Jin et al., 2020), adversarial attacks on graphs tend to promote such label noise in graphs by connecting nodes from different classes and disconnecting nodes from the same class, rendering resultant graphs with varying local smoothness across nodes. Under these scenarios, a constant c might not be optimal and adaptive (i.e. non-constant) smoothness to different nodes is desired.

As shown in Section 3.3 by viewing GAT's aggregation as a solution to regularized graph signal denoising, GAT unintentionally adopts an adaptive $\mathcal{C}$ for different nodes, which facilitates adaptive local smoothness. However, in GAT, the graph denoising problem is solved by a single step of gradient descent, which might still be suboptimal. Furthermore, when modeling the local smoothness factor c_i in Eq. (12), GAT only uses features of node v_i as input, which may not be optimal since by understanding c_i as local smoothness, it should be intrinsically related to the *neighborhood* of node v_i. In this section, we adapt this notion directly into the UGNN framework by introducing a new regularization term, and develop a resulting GNN model (ADA-UGNN) which aims to enforce adaptive local smoothness to nodes in a different manner to GAT. We then utilize an iterative gradient descent method to approximate the optimal solution for Problem 2 with the following regularization term:

$$r(\mathcal{C}, \mathbf{F}, \mathcal{G}) = \frac{1}{2} \cdot \sum_{v_i \in \mathcal{V}} \mathcal{C}_i \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \left\| \frac{\mathbf{F}[i,:]}{\sqrt{d_i}} - \frac{\mathbf{F}[j,:]}{\sqrt{d_j}}. \right\|_2^2 \tag{15}$$

where $\mathcal{C}$ can be construed as a vector, and $\mathcal{C}_i$ indicates the smoothness factor of node v_i. Note that, the above regularization term can be regarded as a generalized version of the regularization term used in PPNP, APPNP, and GCN. Similar to PPNP and APPNP, ADA-UGNN only consists of a single GNN layer. However, ADA-UGNN assumes adaptive local smoothness. We next describe the feature transformation and aggregation operations of ADA-UGNN for node classification, and show how to derive the model via UGNN.

5.1 FEATURE TRANSFORMATION

Similar to PPNP and APPNP, we adopt MLP for the feature transformation. Specifically, for a node classification task, the dimension of the output of the feature transformation $\mathbf{X}'_{in}$ is the number of classes in the graph.

5.2 FEATURE AGGREGATION

We use iterative gradient descent to solve Problem 2 with the regularization term in Eq. (15) The iterative gradient descent steps are stated in the following theorem and its proof can be found at Appendix A.5.

Theorem 5. *With adaptive stepsize* $b_i = 1/\left(2 + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\mathcal{C}_i + \mathcal{C}_j)/d_i\right)$ *for each node* v_i*, the iterative gradient descent steps to solve Problem 2 with the regularization term in Eq. (15) is as follows:*

$$\mathbf{F}^{(k)}[i,:] \leftarrow 2b\mathbf{X}[i,:] + b_i \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\mathcal{C}_i + \mathcal{C}_i) \frac{\mathbf{F}^{(k-1)}[j,:]}{\sqrt{d_i d_j}}; \quad k = 1, \ldots. \tag{16}$$

where $\mathbf{F}^{(0)}[i,:] = \mathbf{X}[i,:]$.

The iterative steps in Eq. (16) is guaranteed for convergence as stated in the following theorem and its proof can be found in Appendix A.6.

Theorem 6. *The iterative steps in Eq. (16) is guaranteed to converge to the optimal solution of Problem 2 with Eq. (15) as regularization term.*

Following the iterative solution in Eq. (16), we model the aggregation operation (for node v_i) for ADA-UGNN as follows:

$$\mathbf{X}_{out}^{(k)}[i,:] \leftarrow 2b_i\mathbf{X}'_{in}[i,:] + b_i \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\mathcal{C}_i + \mathcal{C}_j)\frac{\mathbf{X}_{out}^{(k-1)}[j,:]}{\sqrt{d_i d_j}}; \quad k = 1, \dots K, \tag{17}$$

where $\mathcal{C}_i$ can be considered as hyper-parameters to control the level of "local smoothness" for node v_i and b_i can be calculated from $\{\mathcal{C}_j | v_j \in \tilde{\mathcal{N}}(v_i)\}$ as $b_i = 1/\left(2 + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\mathcal{C}_i + \mathcal{C}_j)/d_i\right)$. Treating $\mathcal{C}_i$ as hyper-parameters for all nodes is impractical, since there are , in total N of them and we do not have their prior knowledge. Thus, we model $\mathcal{C}_i$ as a function of the information of the neighborhood of node v_i as follows:

$$\mathcal{C}_i = s \cdot \sigma\left(h_1\left(h_2\left(\left\{\mathbf{X}'_{in}[j,:] | v_j \in \tilde{\mathcal{N}}(v_i)\right\}\right)\right)\right), \tag{18}$$

where $h_2(\cdot)$ is a function to transform the neighborhood information of node v_i to a vector, while $h_1(\cdot)$ further transforms it to a scalar. $\sigma(\cdot)$ denotes the sigmoid function, which maps the output scalar from $h_1(\cdot)$ to $(0,1)$ and s can be treated as a hyper-parameter controlling the upper bound of $\mathcal{C}_i$. $h_1(\cdot)$ can be modeled as a single layer fully-connected neural network. There are different designs for $h_2(\cdot)$ such as channel-wise variance or mean (Corso et al., 2020). In this paper, we adopt channel-wise variance as the $h_2(\cdot)$ function. In this case, the calculation of $\mathcal{C}_i$ in Eq. (18) only involves H parameters, with H denoting number of classes in the dataset. APPNP can be regarded a special case of ADA-UGNN, where $h_2(\cdot)$ is modeled as a constant function producing 1 as the output for all nodes. For the node classification task, the representation $\mathbf{X}_{out}^{(K)}$, which is obtained after K iterations as in Eq. (17), is directly softmax normalized row-wise and its i-th row indicates the discrete class distribution of node v_i.

6 EXPERIMENT

In this section, we evaluate how the proposed ADA-UGNN handles graphs with varying local smoothness. We conduct node classification experiments on natural graphs, and also evaluate the model's robustness under adversarial attacks. We note that our main goal in proposing/evaluating ADA-UGNN is to demonstrate the promise of deriving new aggregations as solutions of denoising problems, rather than state-of-the-art performance.

6.1 NODE CLASSIFICATION

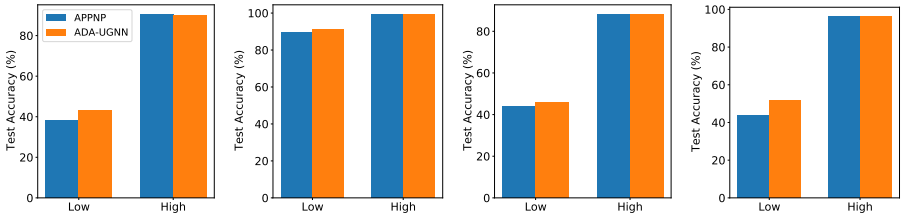
In this section, we conduct the node classification task. We first introduce the datasets and the experimental settings in Section 6.1.1 and then present the results in Section 6.1.2.

6.1.1 DATASETS AND EXPERIMENTAL SETTINGS

We conduct the node classification task on 8 datasets from various domains including citation, social, co-authorship and co-purchase networks. Specifically, we use three citation networks including CORA, CITESEER, and PUBMED (Sen et al., 2008); one social network, BLOGCATALOG (Huang et al., 2017); two co-authorship networks including COAUTHOR-CS and COAUTHOR-PH (Shchur et al., 2018); and two co-purchase networks including AMAZON-COMP and Amazon Photos (Shchur et al., 2018). Descriptions and detail statistics about these datasets can be found in Appendix C.1. To provide a sense of the local smoothness properties of these datasets, in addition to the summary statistics, we also illustrate the *local label smoothness* distributions in Appendix C.1.1: here, we define the local label smoothness of a node as the the ratio of nodes in its neighborhood that share the same label (see formal definition in Eq. 38 in Appendix C.1.1). Notably, the variety in local label smoothness within several real-world datasets – also observed in (Shah, 2020) – clearly motivates the importance of the adaptive smoothness assumption in ADA-UGNN. For the citation networks, we use the standard split as provided in (Kipf & Welling, 2016; Yang et al., 2016). For BLOGCATALOG, we adopt the split provided in (Zhao et al., 2020). For both the citation networks and BLOGCATALOG, the experiments are run with 30 random seeds and the average results are reported. For co-authorship and co-purchase networks, we utilize 20 labels per class for training, 30 nodes per class for validation and the remaining nodes for test. This process is repeated 20 times, which results in 20 different training/validation/test splits. For each split, the experiment is repeated for 20 times with different initialization. The average results over 20×20 experiments are reported. We compare our methods with the methods introduced in Section 2 including GCN, GAT and APPNP. Note that we do not include PPNP as it is difficult to scale for most of the datasets due to the

Table 1: Node Classification Accuracy on Various Datasets

Dataset	GCN	GAT	APPNP	ADA-UGNN
CORA	81.75±0.8	82.56±0.8	84.49±0.6	84.59±0.8
CITESEER	70.13±1.0	70.77±0.8	71.97±0.6	72.05±0.5
PUBMED	78.56±0.5	78.88±0.5	79.92±0.5	79.70±0.4
BLOGCATALOG	71.38±2.7	72.90±1.2	92.43±0.9	93.33±0.3
AMAZON-COMP	82.79±1.3	83.01±1.5	82.99±1.6	83.40±1.3
AMAZON-PHOTO	89.60±1.5	90.33±1.2	91.38±1.2	91.44±1.2
COAUTHOR-CS	91.55±0.6	90.95±0.7	91.69±0.4	92.33±0.5
COAUTHOR-PH	93.23±0.7	92.86±0.7	93.84±0.5	93.92±0.6



(a) CORA (b) BLOGCATALOG (c) AMAZON-COMP (d) COAUTHOR-CS

Figure 1: Accuracy for nodes with low and high local label smoothness.

calculation of inverse in Eq. 6. For all methods, we tune the hyperparameters from the following options: 1) learning rate: $\{0.005, 0.01, 0.05\}$; 2) weight decay $\{5e{-}04, 5e{-}05, 5e{-}06, 5e{-}07, 5e{-}08\}$; and 3) dropout rate: $\{0.2, 0.5, 0.8\}$. For APPNP and our method we further tune the number of iterations K and the upper bound s for c_i in Eq. 18 from the following range: 1) K: $\{5, 10\}$; and s: $\{1, 9, 19\}$. Note that we treat APPNP as a special case of our proposed method with $h_2(\cdot) = 1$.

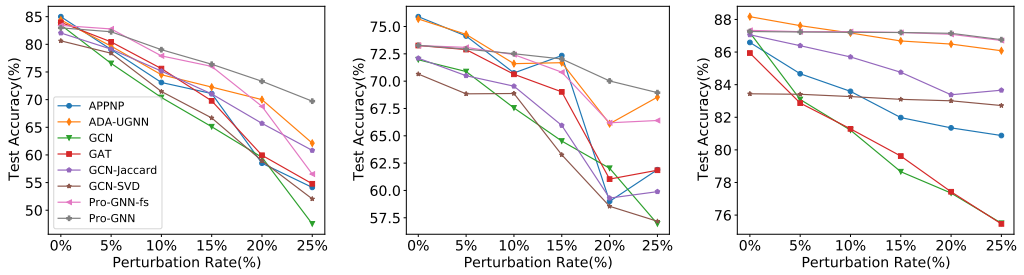
6.1.2 PERFORMANCE COMPARISON

The performance comparison is shown in Table 1. First, GAT outperforms GCN in most datasets. It indicates that modeling adaptive local smoothness is helpful. Second, APPNP/ADA-UGNN outperform GCN/GAT in most settings, suggesting that iterative gradient descent may offer advantages to single-step gradients, due to their better ability to achieve a solution closer to the optimal. Third, and most notably, the proposed ADA-UGNN achieves consistently better performance than GCN/GAT, and outperforms or matches the state-of-the-art APPNP across datasets. Notice that in some datasets such as CORA, CITESEER, and COAUTHOR-PH, the improvements of the proposed model compared with APPNP are marginal. Figure 3 in Appendix C.1.1 shows that these datasets have extremely skewed local label smoothness distributions, with the majority of nodes having perfect, 1.0, label homophily (they are only connected to other nodes of the same label). APPNP shines in such cases, since its assumption of $h_2(\cdot) = 1$ is ideal for these nodes (designating maximal local smoothness). Conversely, our model has the challenging task of learning $h_2(\cdot)$ – in such skewed cases, learning $h_2(\cdot)$ may be quite challenging and unfruitful. On the other hand, for datasets with higher diversity in local label smoothness across nodes such as BLOGCATALOG and AMAZON-COMP, the proposed ADA-UGNN achieves more significant improvements.

To further validate, we partition the nodes in the test set of each dataset into two groups: (1) *high smoothness:* those with local label smoothness >0.5, and (2) *low smoothness:* those with ≤ 0.5, and evaluate accuracy for APPNP and the proposed ADA-UGNN for each group. The results for CORA, BLOGCATALOG, AMAZON-COMP and COAUTHOR-CS are presented in Figure 1 while the results for the remaining datasets can be found in Figure 4 in Appendix C.2. Figure 1 clearly shows that ADA-UGNN consistently improves performance for low-smoothness nodes in most datasets, while keeping comparable (or marginally worse) performance for high-smoothness nodes. In cases where many nodes have low-level smoothness (like BLOGCATALOG or AMAZON-COMP), our method can notably improve overall performance.

6.2 ROBUSTNESS UNDER ADVERSARIAL ATTACKS

Adversarial attacks on graphs tend to connect nodes from different classes and remove edges between nodes from the same classWu et al. (2019); Jin et al. (2020), producing graphs with varying local label smoothness after attack (we demonstrate this in Appendix C.3). To further demonstrate that ADA-UGNN can handle graphs with varying local label smoothness better than alternatives, we conduct experiments to show its robustness under adversarial attacks. Specifically, we adopt Mettack (Zügner & Günnemann, 2019) to perform the attacks. Mettack produces non-targeted attacks which aim to impair test set node classification performance by strategically adding or removing edges from the victim graph. We utilize the attacked graphs (5%-25% perturb rate) from (Jin et al., 2020) and follow the same setting, i.e., each method is run with 10 random seeds and the



(a) CORA (b) CITESEER (c) PUBMED

Figure 2: Robustness under adversarial attacks (node classification accuracy).

average performance is reported. These attacked graphs are generated from CORA, CITESEER and PUBMED, respectively and only the largest connected component is retained in each graph. Furthermore, the training, validation and test split ratio is $10/10/80\%$, which is different from the standard splits we use in Section 6.1. Thus, the performances reported in this section is not directly comparable with those in the previous section. We compare our method both with standard GNNs discussed in Section 2 (GCN, GAT, APPNP), but also with recent state-of-the-art defense techniques against adversarial attacks including GCN-Jaccard (Wu et al., 2019), GCN-SVD (Entezari et al., 2020), Pro-GNN-fs and Pro-GNN (Jin et al., 2020). The detailed description of these methods can be found at Appendix C.4. Results under varying perturbation rates (attack intensities) are shown in Figure 2. Again, we observe that GAT outperforms GCN, suggesting the appeal of an adaptive local smoothness assumption. Here, our method (orange) substantially outperforms GCN, GAT and APPNP by a large margin, especially in scenarios with high perturbation rate. Moreover, the proposed ADA-UGNN is also even more robust than several specially designed adversarial defense methods, like GCN-Jaccard and GCN-SVD, which are based on pre-processing the adversarial attack graphs to obtain cleaner ones, thanks to its adaptive smoothness assumption. Compared with Pro-GNN-fs, our method performs comparably or even better in a few settings, especially when perturbation rate is high. Furthermore, in these settings, the performance of our method is even closer to Pro-GNN, which is the current state-of-the art adversarial defense technique. Note that, Pro-GNN-fs and Pro-GNN involves learning cleaner adjacency matrices of the attacked graphs, and thus has $O(M)$ parameters (M denotes the number of edges in a graph), while our proposed model has far less parameters. Specifically, we have $O(d_{in} \cdot d_{out})$ for feature transformation and H parameters for modelling $h_1(\cdot)$ with H denoting the number of labels.

7 RELATED WORKS

There are mainly two streams of work in developing GNN models, i.e, spectral-based and spatial-based. When designing spectral-based GNNs, graph convolution (Shuman et al., 2013), defined based on spectral theory, is utilized to design graph neural network layers together with the feature transformation and non-linearity (Bruna et al., 2013; Henaff et al., 2015; Defferrard et al., 2016). These designs of the spectral-based graph convolution are tightly related with graph signal processing, and they can be regarded as graph filters. Low-pass graph filters can usually be adopted to denoise graph signals (Chen et al., 2014). In fact, most algorithms discussed in our work can be regarded as low-pass graph filters. With the emergence of GCN (Kipf & Welling, 2016), which can be regarded as a simplified spectral-based and also a spatial-based graph convolution operator, numerous spatial-based GNN models have since been developed (Hamilton et al., 2017; Veličković et al., 2017; Monti et al., 2017; Gao et al., 2018).

Graph signal denoising is to infer a cleaner graph signal given a noisy signal, and can be usually formulated as a graph regularized optimization problem (Chen et al., 2014). Recently, several works connect GCN with graph signal denoising with Laplacian regularization NT & Maehara (2019); Zhao & Akoglu (2019), where they found the aggregation process in GCN models can be regarded as the first-order approximation of the optimal solution of the denoising problem. On the other hand, GNNs are also utilized to develop novel algorithms for graph denoising (Chen et al., 2020). Unlike these works, our paper details how a family of GNN models can be unified with a graph signal denoising perspective, and demonstrates its promise for new architecture design.

8 CONCLUSION

In this paper, we show how various representative GNN models including GCN, PPNP, APPNP and GAT can be unified mathematically as natural instances of graph denoising problems. Specifically, the aggregation operations in these models can be regarded as exactly or approximately addressing such denoising problems subject to Laplacian regularization. With these observations, we propose a general framework, UGNN, which enables the design of new GNN models from the denoising perspective via regularizer design. As an example demonstrating the promise of this paradigm, we instantiate the UGNN framework with a regularizer addressing adaptive local smoothness across nodes, a property prevalent in several real-world graphs, and proposed and evaluated a suitable new GNN model, ADA-UGNN.

REFERENCES

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Siheng Chen, Aliaksei Sandryhaila, José MF Moura, and Jelena Kovacevic. Signal denoising on graphs via graph filtering. In *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 872–876. IEEE, 2014.

Siheng Chen, Yonina C Eldar, and Lingxiao Zhao. Graph unrolling networks: Interpretable neural networks for graph signal denoising. *arXiv preprint arXiv:2006.01301*, 2020.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pp. 3844–3852, 2016.

Negin Entezari, Saba A Al-Sayouri, Amirali Darvishzadeh, and Evangelos E Papalexakis. All you need is low (rank) defending against adversarial attacks on graphs. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 169–177, 2020.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD*, pp. 1416–1424, 2018.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pp. 1024–1034, 2017.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Xiao Huang, Jundong Li, and Xia Hu. Label informed attributed network embedding. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pp. 731–739, 2017.

Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. Graph structure learning for robust graph neural networks. *arXiv preprint arXiv:2005.10203*, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on CVPR*, pp. 5115–5124, 2017.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Hoang NT and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Neil Shah. Scale-free, attributed and class-assortative graph generation to facilitate introspection of graph neural networks. *KDD Mining and Learning with Graphs*, 2020.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

Ryan J Tibshirani et al. Adaptive piecewise polynomial estimation via trend filtering. *The Annals of Statistics*, 42(1):285–323, 2014.

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

Yu-Xiang Wang, James Sharpnack, Alexander J Smola, and Ryan J Tibshirani. Trend filtering on graphs. *The Journal of Machine Learning Research*, 17(1):3651–3691, 2016.

Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples on graph data: Deep insights into attack and defense. *arXiv preprint arXiv:1903.01610*, 2019.

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

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD*, pp. 974–983, 2018a.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*, pp. 4800–4810, 2018b.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. *arXiv preprint arXiv:2006.06830*, 2020.

Daniel Zügner and Stephan Günnemann. Adversarial attacks on graph neural networks via meta learning. *arXiv preprint arXiv:1902.08412*, 2019.

A PROOFS

A.1 PROOF OF THEOREM 1

Theorem 1. *When we adopt the normalized Laplacian matrix $\mathbf{L} = \mathbf{I} - \tilde{\mathbf{A}}$, where $\tilde{\mathbf{A}}$ is the normalized adjacency matrix in Eq. (3), the feature aggregation operation in PPNP (Eq. (6)) can be regarded as exactly solving the graph signal denoising problem (8) with $\mathbf{X}'_{in}$ as the input noisy signal and $c = \frac{1}{\alpha} - 1$.*

Proof. Note that the objective in Eq. (8) is convex. Hence, its closed-form solution $\mathbf{F}^*$ to exactly solve the graph signal denosing problem can be obtained by setting its derivative to $\mathbf{0}$ as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{F}} = 2(\mathbf{F} - \mathbf{X}) + 2c\mathbf{L}\mathbf{F} = 0 \Rightarrow \mathbf{F}^* = (\mathbf{I} + c\mathbf{L})^{-1}\mathbf{X} \tag{19}$$

Given $\mathbf{L} = \mathbf{I} - \tilde{\mathbf{A}}$, $\mathbf{F}^*$ can be reformulated as:

$$\mathbf{F}^* = (\mathbf{I} + c\mathbf{L})^{-1}\mathbf{X} = \left(\mathbf{I} + c\left(\mathbf{I} - \tilde{\mathbf{A}}\right)\right)^{-1}\mathbf{X} = \frac{1}{1+c}\left(\mathbf{I} - \frac{c}{1+c}\tilde{\mathbf{A}}\right)^{-1}\mathbf{X} \tag{20}$$

The feature aggregation operation in Eq. (6) is equivalent to the closed-form solution in Eq. (20) when we set $\alpha = \frac{1}{1+c}$ and $\mathbf{X} = \mathbf{X}'_{in}$. This completes the proof. □

A.2 PROOF OF THEOREM 2

Theorem 2. *Under the same Laplacian matrix setting as Theorem 1, the feature propagation operation in APPNP (Eq. (7)) is equivalent to approximately solve the graph signal denoising problem (8) by iterative gradient descent (K iterations) with $\mathbf{X}'_{in}$ as the input noisy signal, $c = \frac{1}{\alpha} - 1$ and stepsize $b = \frac{1}{2+2c}$.*

Proof. To solve the graph signal denoising problem (8), we take iterative gradient method with the stepsize b. Specifically, the k-th step gradient descent on problem (8) gives

$$\begin{aligned}\mathbf{F}^{(k)} &= \mathbf{F}^{(k-1)} - b \cdot \left.\frac{\partial \mathcal{L}}{\partial \mathbf{F}}\right|_{\mathbf{F}=\mathbf{F}^{(k-1)}} \\ &= (1 - 2b - 2bc)\mathbf{F}^{(k-1)} + 2b\mathbf{X} + 2bc\tilde{\mathbf{A}}\mathbf{F}^{(k-1)}\end{aligned} \tag{21}$$

where $\mathbf{F}^{(0)} = \mathbf{X}$. When we set the stepsize b as $\frac{1}{2+2c}$, we have the following iterative step:

$$\mathbf{F}^{(k)} = \frac{1}{1+c}\mathbf{X} + \frac{c}{1+c}\tilde{\mathbf{A}}\mathbf{F}^{(k-1)}, k = 1, \dots K, \tag{22}$$

which is equivalent to the iterative aggregation operation of the APPNP model in Eq. (7) with $\mathbf{X} = \mathbf{X}'_{in}$ and $\alpha = \frac{1}{1+c}$. This completes the proof. □

A.3 PROOF OF THEOREM 3

Theorem 3. *Under the same Laplacian matrix setting as Theorem 1, the feature aggregation operation in GCN Eq. (2) can be regarded as solving the graph signal denoising problem (8) using one-step gradient descent with $\mathbf{X}'_{in}$ as the input noisy signal and stepsize $b = \frac{1}{2c}$.*

Proof. The gradient with respective to $\mathbf{F}$ at $\mathbf{X}$ is

$$\left.\frac{\partial \mathcal{L}}{\partial \mathbf{F}}\right|_{\mathbf{F}=\mathbf{X}} = 2c\mathbf{L}\mathbf{X}. \tag{23}$$

Hence, one-step gradient descent for the graph signal denoising problem (8) can be described as:

$$\mathbf{F} = \mathbf{X} - b\left.\frac{\partial \mathcal{L}}{\partial \mathbf{F}}\right|_{\mathbf{F}=\mathbf{X}} = \mathbf{X} - 2bc\mathbf{L}\mathbf{X} = (1 - 2bc)\mathbf{X} + 2bc\tilde{\mathbf{A}}\mathbf{X}. \tag{24}$$

When stepsize $b = \frac{1}{2c}$ and $\mathbf{X} = \mathbf{X}'_{in}$, we have $\mathbf{F} = \tilde{\mathbf{A}}\mathbf{X}'_{in}$, which is the same as the aggregation operation of GCN. □

A.4 PROOF OF THEOREM 4

Theorem 4. *With adaptive stepsize $b_i = 1/\left(\sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\right)$ for each node v_i, the process of taking one step of gradient descent from $\mathbf{X}$ to solve problem (11) can be described as follows:*

$$\mathbf{F}[i,:] \leftarrow \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} b_i(c_i + c_j)\mathbf{X}[j,:]. \tag{25}$$

Proof. The gradient of optimization problem in Eq. (11) with respect to $\mathbf{F}$ focusing on a node v_i can be formulated as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{F}[i,:]} = 2\left(\mathbf{F}[i,:] - \mathbf{X}[i,:]\right) + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\left(\mathbf{F}[i,:] - \mathbf{F}[j,:]\right), \tag{26}$$

where c_j in the second term appears since v_i is also in the neighborhood of v_j. Then, the gradient at $\mathbf{X}$ is as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{F}[i,:]}(\mathbf{F}[i,:] = \mathbf{X}[i,:]) = \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\left(\mathbf{X}[i,:] - \mathbf{X}[j,:]\right). \tag{27}$$

Thus, taking a single step of gradient descent starting from $\mathbf{X}$ with stepsize b can be described as follows:

$$\begin{aligned}\mathbf{F}[i,:] &= \mathbf{X}[i,:] - b \cdot \left.\frac{\partial \mathcal{L}}{\partial \mathbf{F}[i,:]}\right|_{\mathbf{F}[i,:]=\mathbf{X}[i,:]} \\ &= \Big(1 - b \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\Big)\mathbf{X}[i,:] + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} b\,(c_i + c_j)\,\mathbf{X}[j,:]\end{aligned} \tag{28}$$

Given $b = 1/\left(\sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\right)$, Eq. (28) can be rewritten as:

$$\mathbf{F}[i,:] \leftarrow \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} b(c_i + c_j)\mathbf{X}[j,:], \tag{29}$$

which completes the proof. □

A.5 Proof of Theorem 5

Theorem 5. *With adaptive stepsize* $b_i = 1/\left(2 + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\mathcal{C}_i + \mathcal{C}_j)/d_i\right)$ *for each node* v_i*, the iterative gradient descent steps to solve Problem 2 with the regularization term in Eq. (15) is as follows:*

$$\mathbf{F}^{(k)}[i,:] \leftarrow 2b\mathbf{X}[i,:] + b_i \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\mathcal{C}_i + \mathcal{C}_i)\frac{\mathbf{F}^{(k-1)}[j,:]}{\sqrt{d_i d_j}}; \quad k = 1, \ldots \tag{30}$$

where $\mathbf{F}^{(0)}[i,:] = \mathbf{X}[i,:]$.

Proof. The gradient of the optimization problem 2 with the regularization term in Eq. (15) with respect to $\mathbf{F}$ (focusing on node v_i) is as follows:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{F}[i,:]} = 2(\mathbf{F}[i,:] - \mathbf{X}[i,:]) + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \frac{c_i + c_j}{\sqrt{d_i}}\left(\frac{\mathbf{F}[i,:]}{\sqrt{d_i}} - \frac{\mathbf{F}[j,:]}{\sqrt{d_j}}\right), \tag{31}$$

where c_j in the second term appears since v_i is also in the neighborhood of v_j. The iterative gradient descent steps with adaptive stepsize b_i can be formulated as follows:

$$\mathbf{F}^{(k)}[i,:] \leftarrow \mathbf{F}^{(k-1)}[i,:] - b_i \cdot \left.\frac{\partial \mathcal{L}}{\partial \mathbf{F}[i,:]}\right|_{\mathbf{F}[i,:]=\mathbf{F}^{(k-1)}[i,:]}; \quad k = 1, \ldots \tag{32}$$

With the gradient in Eq. (31), the iterative steps in Eq. (32) can be rewritten as:

$$\begin{aligned}\mathbf{F}^{(k)}[i,:] \leftarrow &(1 - 2b_i - b_i \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} \frac{c_i + c_j}{d_i})\mathbf{F}^{(k-1)}[i,:] + 2b_i\mathbf{X}[i,:] \\ &+ b_i \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\frac{\mathbf{F}^{(k)}[j,:]}{\sqrt{d_i d_j}}; \quad k = 1, \ldots\end{aligned} \tag{33}$$

Given $b_i = 1/\left(2 + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)/d_i\right)$, the iterative steps in Eq. (33) can be re-written as follows:

$$\mathbf{F}^{(k)}[i,:] \leftarrow 2b\mathbf{X}[i,:] + b_i \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)\frac{\mathbf{F}^{(k-1)}[j,:]}{\sqrt{d_i d_j}}; \quad k = 1, \ldots, \tag{34}$$

with $\mathbf{F}^{(0)}[i,:] = \mathbf{X}[i,:]$, which completes the proof. □

A.6 PROOF OF THEOREM 6

Theorem 6. *The iterative steps in Eq. (16) is guaranteed to converge to the optimal solution of Problem 2 with Eq. (15) as regularization term.*

Proof. By taking the second derivative with respect to $\mathbf{F}[i,:]$, we obtain the Hessian matrix as:

$$\frac{\partial \mathcal{L}^2}{\partial \mathbf{F}[i,:]^2} = 2\mathbf{I} + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\frac{c_i + c_j}{d_i})\mathbf{I} \tag{35}$$

which implies the Lipschitz constant of the gradient in Eq. (31) is $2 + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (\frac{c_i + c_j}{d_i})$. To guarantee convergence, the stepsize b_i for node i should be smaller than $2/\left(2 + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)/d_i\right)$ (Nesterov, 2013). The stepsize we adopt in Theorem 5 is $b_i = 1/\left(2 + \sum_{v_j \in \tilde{\mathcal{N}}(v_i)} (c_i + c_j)/d_i\right)$, hence the convergence is guaranteed. □

B CONNECTIONS TO PAIRNORM AND DROPEDGE

PairNorm and DropEdge, which are two recently proposed GNN enhancements for developing deeper GNN models, are corresponding to the following regularization terms:

$$\text{PairNorm:} \quad \sum_{(v_i,v_j)\in\mathcal{E}} \mathcal{C}_p \cdot \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2 - \sum_{(v_i,v_j)\notin\mathcal{E}} \mathcal{C}_n \cdot \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2, \tag{36}$$

$$\text{DropEdge:} \quad \sum_{(v_i,v_j)\in\mathcal{E}} \mathcal{C}_{ij} \cdot \|\mathbf{F}[i,:] - \mathbf{F}[j,:]\|_2^2, \text{ where } \mathcal{C}_{ij} \in \{0,1\}. \tag{37}$$

For PairNorm, $\mathcal{C}$ consists of $\mathcal{C}_p, \mathcal{C}_n > 0$ and the regularization term ensures connected nodes to be similar while disconnected nodes to be dissimilar. For DropEdge, $\mathcal{C}$ is a sparse matrix having the same shape as adjacency matrix. For each edge (v_i, v_j), its corresponding $\mathcal{C}_{ij}$ is sampled from a Bernoulli distribution with mean $1 - q$, where q is a pre-defined dropout rate.

C EXPERIMENTS

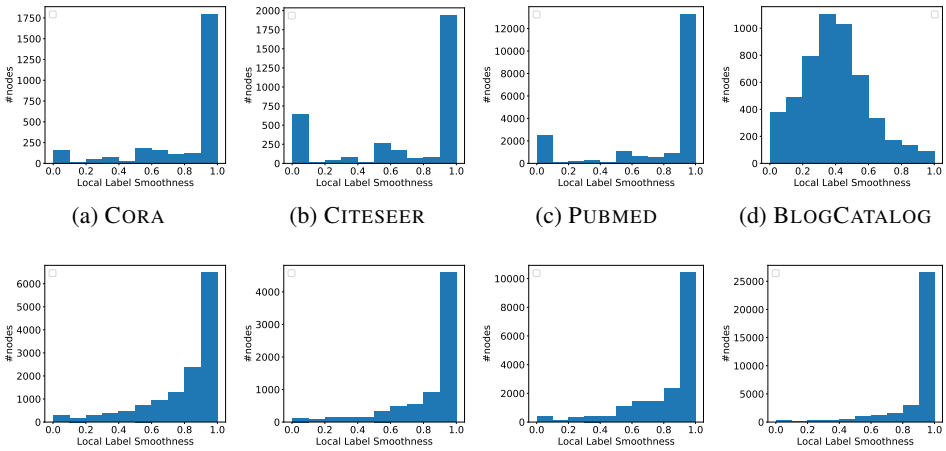
C.1 DATASETS

	#Nodes	#Edges	#Labels	#Features
CORA	2708	13264	7	1433
CITESEER	3327	12431	6	3703
PUBMED	19717	108365	3	500
BLOGCATALOG	5196	348682	6	8189
AMAZON-COMP	13381	504937	10	767
AMAZON-PHOTO	7487	245573	8	745
COAUTHOR-CS	18333	182121	15	6805
COAUTHOR-PH	34493	530417	5	8415

Table 2: Dataset summary statistics.

In this section, we provide information of the datasets we used in the experiments as follows:

- **Citation Networks:** CORA, CITESEER and PUBMED are widely adopted benchmarks of GNN models. In these graphs, nodes represent documents and edges denote the citation links between them. Each node is associated bag-of-words features of its corresponding document and also a label indicating the research field of the document.
- **Blogcatalog:** BLOGCATALOG is an online blogging community where bloggers can follow each other. The BLOGCATALOG graph consists of blogger as nodes while their social relations as edges. Each blogger is associated with some features generated from key words of his/her blogs. The bloggers are labeled according to their interests.



(a) CORA (b) CITESEER (c) PUBMED (d) BLOGCATALOG

(e) AMAZON-COMP (f) AMAZON-PHOTO (g) COAUTHOR-CS (h) COAUTHOR-PH

Figure 3: Distribution of local label smoothness (homophily) on different graph datasets: note the non-homogeneity of smoothness values.

- **Co-purchase Graph:** AMAZON-COMP and AMAZON-PHOTO are co-purchase graphs, where nodes represent items and edges indicate that two items are frequently bought together. Each item is associated with bag-of-words features extract from its corresponding reviews. The labels of items are given by the category of them.
- **Co-authorship Graphs:** COAUTHOR-CS and COAUTHOR-PH are co-authorship graphs, where nodes are authors and edges indicating the co-authorship between authors. Each author is associated with some features representing the keywords of his/her papers. The label of an author indicates the his/her most active research field.

Some statistics of these graphs are shown in Table 2.

C.1.1 LOCAL LABEL SMOOTHNESS OF DATASETS

We further present the distribution of local label smoothness in these datasets. For a node v_i we formally define the local label smoothness as follows

$$\text{ls}(v_i) = \frac{\sum_{v_j \in \mathcal{N}(v_i)} \mathbf{1}\{l(v_i) = l(v_j)\}}{|\mathcal{N}(v_i)|} \tag{38}$$

where $l(v_i)$ denotes the label of node v_i and $\mathbf{1}\{a\}$ is an indicator function, which takes 1 as output only when a is true, otherwise 0. The distributions of local label smoothness for all 8 datasets are presented in Figure 3.

C.2 NODE CLASSIFICATION ACCURACY FOR NODES WITH LOW-LEVEL AND HIGH-LEVEL LOCAL LABEL SMOOTHNESS

The performance of nodes with low local label smoothness and high local label smoothness in CITESEER, PUBMED, AMAZON-PHOTO and COAUTHOR-PH are presented in Figure 4.

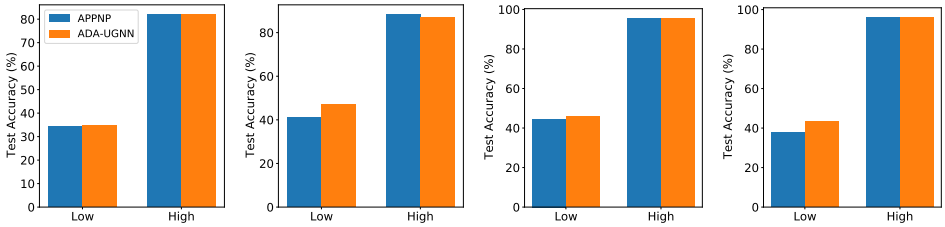
C.3 LOCAL SMOOTHNESS DISTRIBUTION OF ATTACKED GRAPH

Graph adversarial attacks tend to connect nodes from different classes while disconnect nodes from the same class, which typically leads to more diverse distributions of local smoothness level. We present the distributions of the graphs generated by Mettack Zügner & Günnemann (2019) with different perturbation rate for CORA, CITESEER and PUBMED in Figure 5, Figure 6 and Figure 7, respectively.

C.4 BASELINES FOR ADVERSARIAL DEFENSE

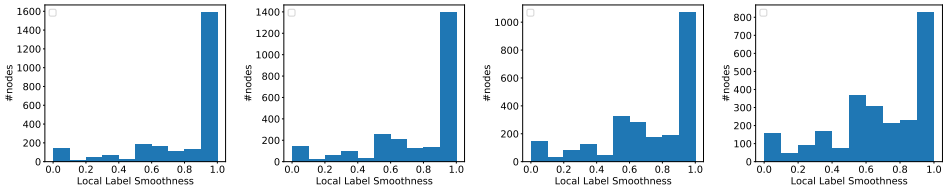
In this section, we list the descriptions of the defense algorithms we adopt in Section 6.2 as follows:

- **GCN-Jaccard (Wu et al., 2019):** GCN-Jaccard aims to pre-process a given attacked graph by removing those edges added by the attackers. Specifically, Jaccard smilarlity is utilized to measure



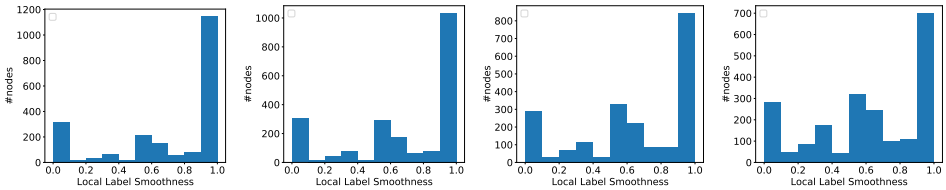
(a) CITESEER (b) PUBMED (c) AMAZON-PHOTO (d) COAUTHOR-PH

Figure 4: Accuracy with low label smoothness and high label smoothness nodes. Note the consistent improvement in low smoothness cases, enabled by adaptive local smoothing.



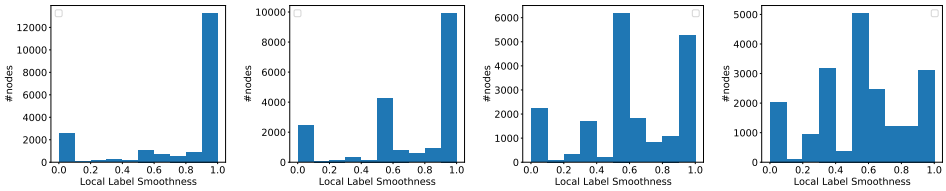
(a) 0% (b) 5% (c) 15% (d) 25%

Figure 5: Distribution of local label smoothness on CORA with various attack perturbation rates.



(a) 0% (b) 5% (c) 15% (d) 25%

Figure 6: Distribution of local label smoothness on CITESEER with various attack perturbation rates.



(a) 0% (b) 5% (c) 15% (d) 25%

Figure 7: Distribution of local label smoothness on PUBMED with various attack perturbation rates.

the feature similarity between connected pairs of nodes. The edges between node pairs with low-similarity are removed by the algorithm. This pre-processed graph is then utilized for the node classification task.

- **GCN-SVD Entezari et al. (2020):** GCN-SVD is also a pre-process method. It use SVD to decompose the adjacency matrix of a given perturbed graph and then obtain its low-rank approximation. The low-rank approximation is believed to be cleaner as graph adversarial attacks are observed to be high-rank in (Entezari et al., 2020).
- **Pro-GNN (Jin et al., 2020):** Pro-GNN tries to learn a cleaner graph while training the node classification model at the same time. Specifically, it treats the adjacency as parameters, which is optimized during the training stage. Several different constraints are enforced to this learnable adjacency matrix, including: 1) the learned adjacency matrix should be close to the original adjacency matrix; 2) the learned adjacency matrix should be low-rank; and 3) the learned adjacency matrix should ensure

feature smoothness. **Pro-GNN-fs** is a variant of Pro-GNN where the third constraint, i.e. feature smoothness, is not enforced.