Rethinking Graph Regularization for Graph Neural Networks

Han Yang, Kaili Ma, James Cheng
The Chinese University of Hong Kong
hyang@cse.cuhk.edu.hk, klma@cse.cuhk.edu.hk, jcheng@cse.cuhk.edu.hk

Abstract

The graph Laplacian regularization term is usually used in semi-supervised representation learning to provide graph structure information for a model $f(X)$. However, with the recent popularity of graph neural networks (GNNs), directly encoding graph structure A into a model, i.e., $f(A, X)$, has become the more common approach. While we show that graph Laplacian regularization brings little-to-no benefit to existing GNNs, and propose a simple but non-trivial variant of graph Laplacian regularization, called Propagation-regularization (P-reg), to boost the performance of existing GNN models. We provide formal analyses to show that P-reg not only infuses extra information (that is not captured by the traditional graph Laplacian regularization) into GNNs, but also has the capacity equivalent to an infinite-depth graph convolutional network. We demonstrate that P-reg can effectively boost the performance of existing GNN models on both node-level and graph-level tasks across many different datasets.

1 Introduction

Semi-supervised node classification is one of the most popular and important problems in graph learning. Many effective methods (Zhu, Ghahramani, and Lafferty 2003; Zhou et al. 2003; Belkin, Niyogi, and Sindhwani 2006; Ando and Zhang 2007) have been proposed for node classification by adding a regularization term, e.g., Laplacian regularization, to a feature mapping model $f(X) : \mathbb{R}^{N\times F} \to \mathbb{R}^{N\times C}$, where N is the number of nodes, F is the dimensionality of a node feature, C is the number of predicted classes, and $X \in \mathbb{R}^{N\times F}$ is the node feature matrix. One known drawback of these methods is that the model f itself only models the features of each node in a graph and does not consider the relation among the nodes. They rely on the regularization term to capture graph structure information based on the assumption that neighboring nodes are likely to share the same class label. However, this assumption does not hold in many real world graphs as relations between nodes in these graphs could be complicated, as pointed out in (Kipf and Welling 2017). This motivates the development of early *graph neural network* (*GNN*) models such as *graph convolutional network* (*GCN*) (Kipf and Welling 2017).

Many GNNs (Kipf and Welling 2017; Veličković et al. 2018; Wu et al. 2019; Hamilton, Ying, and Leskovec 2017; Pei et al. 2020; Li et al. 2015) have been proposed, which encode graph structure information directly into their model as $f(A, X) : (\mathbb{R}^{N\times N}, \mathbb{R}^{N\times F}) \to \mathbb{R}^{N\times C}$, where $A \in \mathbb{R}^{N\times N}$ is the adjacency matrix of a graph. Then, they simply train the model by minimizing the supervised classification loss, without using graph regularization. However, in this work we ask the question: *Can graph regularization also boost the performance of existing GNN models as it does for traditional node classification models?*

We give an affirmative answer to this question. We show that existing GNNs already capture graph structure information that traditional graph Laplacian regularization can offer. Thus, we propose a new graph regularization, *Propagation-regularization* (*P-reg*), which is a variation of graph Laplacian-based regularization that provides new supervision signals to nodes in a graph. In addition, we prove that P-reg possesses the equivalent power as an infinite-depth GCN, which means that P-reg enables each node to capture information from nodes farther away (as a deep GCN does, but more flexible to avoid over-smoothing and much less costly to compute). We validate by experiments the effectiveness of P-reg as a general tool for boosting the performance of existing GNN models. We believe our work could inspire a new direction for GNN framework designs.

2 Propagation-Regularization

The notations used in this paper and their description are also given in Appendix A. We use a 2-layer GCN model f_1 as an example of GNN for convenience of presentation. A GCN model f_1 can be formulated as $f_1(A, X) = \hat{A}(\sigma(\hat{A}XW_0))W_1$, where $W_0 \in \mathbb{R}^{F\times H}$ and $W_1 \in \mathbb{R}^{H\times C}$ are linear mapping matrices, and H is the size of hidden units. $\hat{A} = D^{-1}A$ is the normalized adjacency matrix, where $D \in \mathbb{R}^{N\times N}$ is the diagonal degree matrix with $D_{ii} = \sum_{j=1}^{N} A_{ij}$ and $D_{ij} = 0$ if $i \neq j$. σ is the activation function. f_1 takes the graph structure and node features as input, then outputs $Z = f_1(A, X) \in \mathbb{R}^{N\times C}$. Denote $P_{ij} = \frac{\exp(Z_{ij})}{\sum_{k=1}^{C}\exp(Z_{ik})}$ for $i = 1, \ldots, N$ and $j = 1, \ldots, C$ as the *softmax* of the output logits. Here, $P \in \mathbb{R}^{N\times C}$ is the predicted class posterior probability for all nodes. By fur-

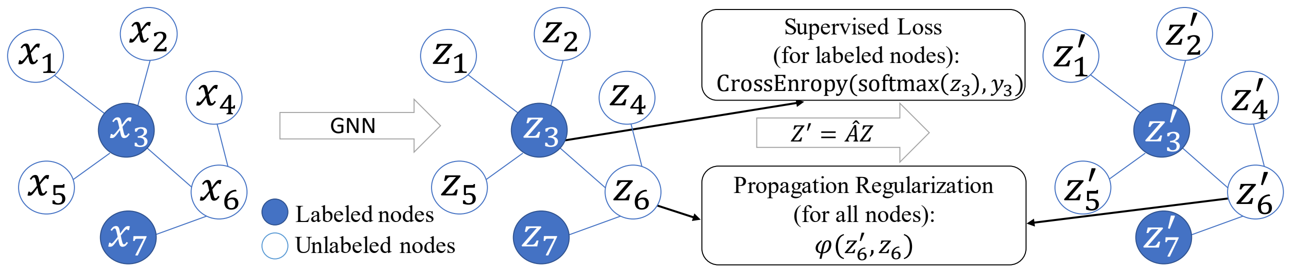


Figure 1: An overview of Propagation-regularization

ther propagating the output Z of f_1, we obtain $Z' = \hat{A}Z \in \mathbb{R}^{N\times C}$. The corresponding softmax probability of Z' is given as $Q_{ij} = \frac{\exp(Z'_{ij})}{\sum_{k=1}^{C}\exp(Z'_{ik})}$ for $i = 1, \ldots, N$ and $j = 1, \ldots, C$.

The **Propagation-regularization** (**P-reg**) is defined as follows:

$$\mathcal{L}_{P\text{-}reg} = \frac{1}{N}\phi\left(Z, \hat{A}Z\right), \tag{1}$$

where $\hat{A}Z$ is the further propagated output of f_1 and ϕ is a function that measures the *difference* between Z and $\hat{A}Z$. We may adopt typical measures such as Square Error, Cross Entropy, Kullback–Leibler Divergence, etc., for ϕ, and we list the corresponding function below, where $(\cdot)_i^\top$ denotes the i-th row vector of the matrix $(\cdot)$.

ϕ	Squared Error	Cross Entropy	KL Divergence
	$\frac{1}{2}\sum_{i=1}^{N}\left\|(\hat{A}Z)_i^\top - (Z)_i^\top\right\|_2^2$	$-\sum_{i=1}^{N}\sum_{j=1}^{C} P_{ij}\log Q_{ij}$	$\sum_{i=1}^{N}\sum_{j=1}^{C} P_{ij}\log\frac{P_{ij}}{Q_{ij}}$

Then the GNN model f_1 can be trained through the composition loss:

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{cls} + \mu\mathcal{L}_{P\text{-}reg} \\ &= -\frac{1}{M}\sum_{i\in S_{\text{train}}}\sum_{j=1,\ldots,C} Y_{ij}\log\left(P_{ij}\right) + \mu\frac{1}{N}\phi(Z, \hat{A}Z),\end{aligned} \tag{2}$$

where S_{train} is the set of training nodes and $M = |S_{\text{train}}|$, N is the number of all nodes, Y_{ij} is the ground-truth one-hot label, i.e., $Y_{ij} = 1$ if the label of node v_i is j and $Y_{ij} = 0$ otherwise. The *regularization factor* $\mu \geq 0$ is used to adjust the influence ratio of the two loss terms $\mathcal{L}_{cls}$ and $\mathcal{L}_{P\text{-}reg}$.

The computation overhead incurred by Eq. (1) is $\mathcal{O}(|\mathcal{E}|C + NC)$, where $|\mathcal{E}|$ is the number of edges in the graph. Computing $\hat{A}Z$ has time complexity $\mathcal{O}(|\mathcal{E}|C)$ by sparse-dense matrix multiplication or using the message passing framework (Gilmer et al. 2017). Evaluating $\phi(Z, \hat{A}Z)$ has complexity $\mathcal{O}(NC)$. An overview of P-reg is shown in Figure 1. We give the annealing and thresholding version of P-reg in Appendix C.

3 Understanding P-reg through Laplacian Regularization and Infinite-Depth GCN

We first examine how P-reg is related to graph Laplacian regularization and infinite-depth GCN, which helps us better understand the design of P-reg and its connection to GNNs. The proofs of all lemmas and theorems are given in Appendix B.

3.1 Equivalence of Squared-Error P-Reg to Squared Laplacian Regularization

Consider using squared error as ϕ in P-reg, i.e., $\mathcal{L}_{P\text{-}SE} = \frac{1}{N}\phi_{SE}(Z,\hat{A}Z) = \frac{1}{2N}\sum_{i=1}^{N}\|(\hat{A}Z)_i^\top - (Z)_i^\top\|_2^2$. As stated by Smola and Kondor (2003), the graph Laplacian $\langle Z, \Delta Z\rangle$ can serve as a tool to design regularization operators, where $\Delta = D - A$ is the Laplacian matrix and $\langle\cdot,\cdot\rangle$ means the inner product of two matrices. A class of regularization functions on a graph can be defined as:

$$\langle Z, RZ\rangle := \left\langle Z, r\left(\tilde{\Delta}\right)Z\right\rangle,$$

where $\tilde{\Delta} = I - D^{-1}A$ is the normalized Laplacian matrix. $r(\tilde{\Delta})$ is applying the scalar valued function $r(\lambda)$ to eigenvalues of $\tilde{\Delta}$ as $r(\tilde{\Delta}) := \sum_{i=1}^{N} r\left(\lambda_i\right)u_i u_i^\top$, where λ_i and u_i are i-th pair of eigenvalue and eigenvector of $\tilde{\Delta}$.

Theorem 3.1. *The squared-error P-reg is equivalent to the regularization with matrix $R = \tilde{\Delta}^\top\tilde{\Delta}$.*

Theorem 3.1 shows that the squared-error P-reg falls into the traditional regularization theory scope, such that we can enjoy its nice properties, e.g., constructing a Reproducing Kernel Hilbert Space $\mathcal{H}$ by dot product $\langle f, f\rangle_{\mathcal{H}} = \left\langle f, \tilde{\Delta}^\top\tilde{\Delta}f\right\rangle$, where $\mathcal{H}$ has kernel $\mathcal{K} = \left(\tilde{\Delta}^\top\tilde{\Delta}\right)^{-1}$. We refer readers to (Smola and Kondor 2003) for other interesting properties of spectral graph regularization and (Chung 1997) for properties of Laplacian matrix.

3.2 Equivalence of Minimizing P-Reg to Infinite-Depth GCN

We first analyze the behavior of infinite-depth GCN. Take the output $Z \in \mathbb{R}_+^{N\times C}$ of a GNN model as the input of an infinite-depth GCN. Let $G\left(A, Z\right)$ be a graph with adjacency matrix A and node feature matrix Z. A typical infinite-depth GCN applied to a graph $G\left(A, Z\right)$ can be represented as $\left(\hat{A}\sigma\left(\cdots\hat{A}\sigma\left(\hat{A}\sigma\left(\hat{A}ZW_0\right)W_1\right)W_2\cdots\right)W_\infty\right)$. We use *ReLU* as the $\sigma(\cdot)$ function and neglect all linear mappings, i.e., $W_i = I \quad \forall i \in \mathbb{N}$. Since $Z \in \mathbb{R}_+^{N\times C}$, and $\text{ReLU}(\cdot) = \max(0,\cdot)$, the infinite-depth GCN can be simplified to $\left(\hat{A}^\infty Z\right)$. This simplification shares similarities with SGC (Wu et al. 2019).

Lemma 3.1. *Applying graph convolution infinitely to a graph $G\left(A, Z\right)$ with self-loops (i.e., $A := A + I$) produces* *the same output vector for each node, i.e., define $\tilde{Z} = \hat{A}^{\infty} Z$, then $\tilde{z}_1 = \ldots = \tilde{z}_N$, where $\tilde{z}_i$ is the i-th row vector of $\tilde{Z}$.*

Lemma 3.1 shows that an infinite-depth GCN makes each node capture and represent the information of the whole graph.

We next establish the equivalence between minimizing P-reg and an infinite-depth GCN.

Lemma 3.2. *Minimizing the squared-error P-reg of node feature matrix Z produces the same output vector for each node, i.e., $\tilde{Z} \in \arg\min_Z \left\| \hat{A}Z - Z \right\|_F^2$, where $\tilde{z}_1 = \cdots = \tilde{z}_N$.*

From Lemmas 3.1 and 3.2, the behaviors of applying infinite graph convolution and minimizing the squared-error P-reg are identical. Next, we show that iteratively minimizing P-reg of $G(A, Z)$ has the same effect as recursively applying graph convolution to $G(A, Z)$ step by step. We can rewrite $\|\hat{A}Z - Z\|_F^2 = \|\left(D^{-1}A - I\right)Z\|_F^2 = \sum_{i=1}^{N} \|(\frac{1}{d_i}\sum_{j\in\mathcal{N}(v_i)} z_j) - z_i\|_2^2$, where d_i is the degree of v_i and $\mathcal{N}(v_i)$ is the set of neighbors of v_i. If we minimize $\sum_{i=1}^{N} \|(\frac{1}{d_i}\sum_{j\in\mathcal{N}(v_i)} z_j) - z_i\|_2^2$ in an iterative manner, i.e., $z_i^{(k+1)} = \frac{1}{d_i}\sum_{j\in\mathcal{N}(v_i)} z_j^{(k)}$ where $k \in \mathbb{N}$ is the iteration step, then it is exactly the same as applying a graph convolution operation to Z, i.e., $\hat{A}Z$, which can be rewritten into a node-centric representation: $z_i^{(k+1)} = \frac{1}{d_i}\sum_{j\in\mathcal{N}(v_i)} z_j^{(k)}$. Thus, when $k \to +\infty$, the two formulas converge into the same point. The above analysis motivates us to establish the equivalence between minimizing different versions of P-reg and an infinite-depth GCN, giving the following theorem:

Theorem 3.2. *When neglecting all linear mappings in GCN, i.e., $W_i = I \ \forall i \in \mathbb{N}$, minimizing P-reg (including the three versions of ϕ using Squared Error, Cross Entropy and KL Divergence defined in Section 2) is equivalent to applying graph convolution infinitely to a graph $G(A, Z)$, i.e., define $\tilde{Z} = \hat{A}^{\infty} Z$, then $\tilde{Z}$ is the optimal solution to $\arg\min_Z \phi(Z, \hat{A}Z)$.*

In Theorem 3.2, the optimal solution to minimizing P-reg corresponds to the case when $\mu \to +\infty$ in Eq.(2).

3.3 Connections to Other Existing Work

P-reg is conceptually related to *Prediction/Label Propagation* (Zhu, Ghahramani, and Lafferty 2003; Zhou et al. 2003; Wang and Zhang 2007; Karasuyama and Mamitsuka 2013; Liu et al. 2019; Li et al. 2019b; Wang and Leskovec 2020), *Training with Soft Targets* such as distillation-based (Hinton, Vinyals, and Dean 2015; Zhang et al. 2019) or entropy-based (Miller et al. 1996; Jaynes 1957; Szegedy et al. 2016; Pereyra et al. 2017; Müller, Kornblith, and Hinton 2019), and other *Graph Regularization* (Civin, Dunford, and Schwartz 1960; Ding, Tang, and Zhang 2018; Feng et al. 2019; Deng, Dong, and Zhu 2019; Verma et al. 2019; Stretcu et al. 2019) using various techniques such as Laplacian operator, adversarial training, generative adversarial networks, or co-training. We discuss these related works in details in Appendix D.

4 Why P-Reg can improve existing GNNs

We next answer why P-reg can boost the performance of existing GNN models. From the graph regularization perspective, P-reg provides extra information for GNNs to improve their performance. From the infinite-depth GCN perspective, using P-reg enhances the current shallow-layered GNNs with the power of a deeper GCN as P-reg can simulate a GCN of any layers (even a *fractional* layer) at low cost.

4.1 Benefits of P-Reg from the Graph Regularization Perspective

4.1.1 Graph Regularization Provides Extra Supervision Information

Usually, the more training nodes are provided, the more knowledge a model can learn and thus the higher accuracy the model can achieve. This can be verified in Figure 2a: as we keep adding nodes randomly to the training set, we generally obtain higher accuracy for the GCN/GAT models.

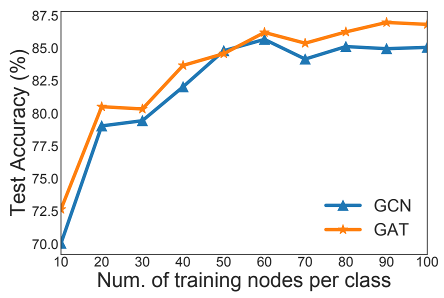
However, in real-world graphs, the training nodes are usually only a small fraction of all nodes. Graph regularization, as an unsupervised term applied on a model's output in training, can provide extra supervision information for the model to learn a better representation for the nodes in a graph. Especially in semi-supervised learning, regularization can bridge the gap between labeled data and unlabeled data, making the supervised signal spread over the whole dataset. The effectiveness of regularization in semi-supervised learning has been shown in (Zhu, Ghahramani, and Lafferty 2003; Zhou et al. 2003; Belkin, Niyogi, and Sindhwani 2006; Ando and Zhang 2007).

4.1.2 Limitations of Graph Laplacian Regularization

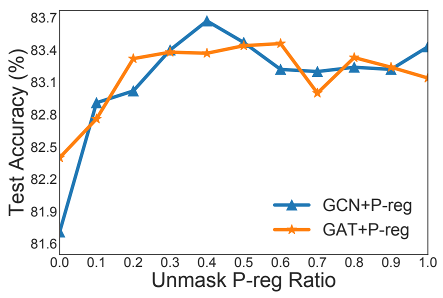
Unfortunately, **graph Laplacian regularization can hardly provide extra information that existing GNNs cannot capture.** The original graph convolution operator g_θ is approximated by the truncated Chebyshev polynomials as $g_\theta * Z = \sum_{k=0}^{K} \theta_k T_k(\tilde{\Delta})Z$ (Hammond, Vandergheynst, and Gribonval 2011) in GNNs, where K means to keep K-th order of the Chebyshev polynomials, and θ_k is the parameters of g_θ as well as the Chebyshev coefficients. $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ is the Chebyshev polynomials, with $T_0(x) = 1$ and $T_1(x) = x$. K is equal to 1 for today's spatial GNNs such as GCN (Kipf and Welling 2017), GAT (Veličković et al. 2018), GraphSAGE (Hamilton, Ying, and Leskovec 2017) and so on. Thus, the 1-order Laplacian information is captured by popular GNNs, making the Laplacian regularization useless to them. This is also verified in Figure 2c, which shows that the Laplacian regularization does not improve the test accuracy of GCN and GAT.

4.1.3 How P-Reg Addresses the Limitations of Graph Laplacian Regularization

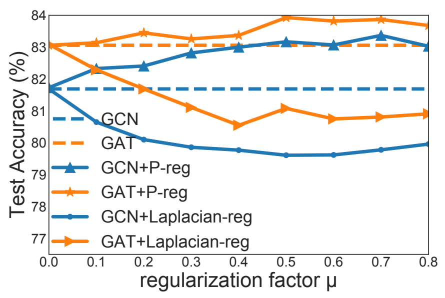
P-reg provides new supervision information for GNNs. Although P-reg shares similarities with Laplacian regularization, one key difference is that the Laplacian regularization $\mathcal{L}_{lap} = \sum_{(i,j)\in\mathcal{E}} \left\| Z_i^\top - Z_j^\top \right\|_2^2$ is *edge-centric*, while P-reg $\mathcal{L}_{P\text{-}reg} = \phi(Z, \hat{A}Z) = \sum_{i=1}^{N} \phi(Z_i^\top, (\hat{A}Z)_i^\top)$ is *node-centric*. The node-centric P-reg can be regarded as using the



(a)

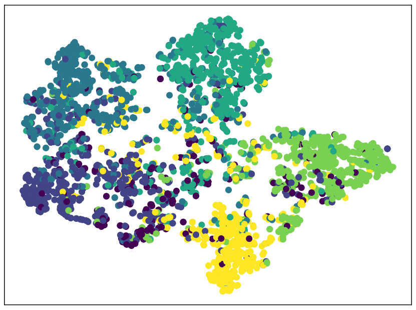


(b)

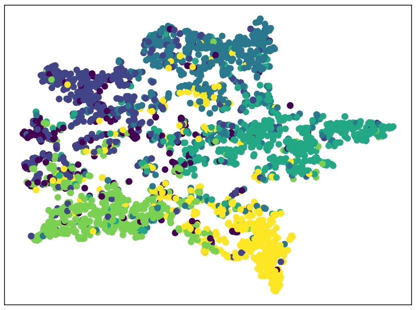


(c)

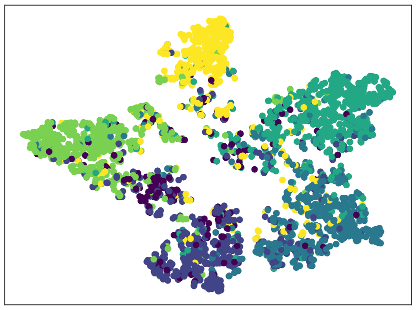
Figure 2: The effects of increasing *(a) the number of training nodes, (b) the number of nodes applied with P-reg, (c) regularization factor on the accuracy of GCN and GAT,* on the CORA dataset



(a) GCN



(b) GCN+Laplacian-reg ($\mu = 0.5$)



(c) GCN+P-reg ($\mu = 0.5$)

Figure 3: The t-SNE visualization of GCN outputs on the CiteSeer dataset (best viewed in color)

aggregated predictions of neighbors as the supervision target for each node. Based on the model's prediction vectors $Z \in \mathbb{R}^{N\times C}$, the average of the neighbors' predictions of node v_i, i.e., $(\hat{A}Z)_i^\top = \sum_{k\in\mathcal{N}(v_i)} z_k$, serves as the soft supervision targets of the model's output z_i. This is similar to taking the *voting result* of each node's neighbors to supervise the nodes. Thus, P-reg provides additional categorical information for the nodes, which cannot be obtained from Laplacian regularization as it simply pulls the representation of edge-connected nodes closer.

Empirical verification. The above analysis can be verified from the node classification accuracy in Figure 2b. As P-reg is node-centric, we can add a mask to a node v_i so that unmasking v_i means to apply ϕ to v_i. Define the unmask ratio as $\alpha = \frac{|S_{\text{unmask}}|}{N}$, where S_{unmask} is the set of nodes with ϕ applied. The masked P-reg can be defined as $\mathcal{L}_{P\text{-}reg\text{-}m} = \frac{1}{|S_{\text{unmask}}|}\sum_{i\in S_{\text{unmask}}} \phi(Z_i^\top, (AZ)_i^\top)$. If the unmask ratio α is 0, the model is basically vanilla GCN/GAT. As the unmask ratio increases, P-reg is applied on more nodes and Figure 2b shows that the accuracy generally increases. After sufficient nodes are affected by P-reg, the accuracy tends to be stabilized. The same result is also observed on all the datasets in our experiments. The result thus verifies that we can consider $\hat{A}Z$ as the supervision signal of Z, and the more supervision we have, the higher is the accuracy.

In Figure 2c, we show that P-reg improves the test accuracy of both GCN and GAT, while applying Laplacian regularization even harms their performance. We also plot the t-SNE visualization in Figure 3. Compared with pure GCN and GCN with Laplacian regularization, Figure 3c shows that by applying P-reg, the output vectors of GCN are more condensed within each class and the classes are also more clearly separated from each other.

4.2 Benefits of P-Reg from the Deep GCN Perspective

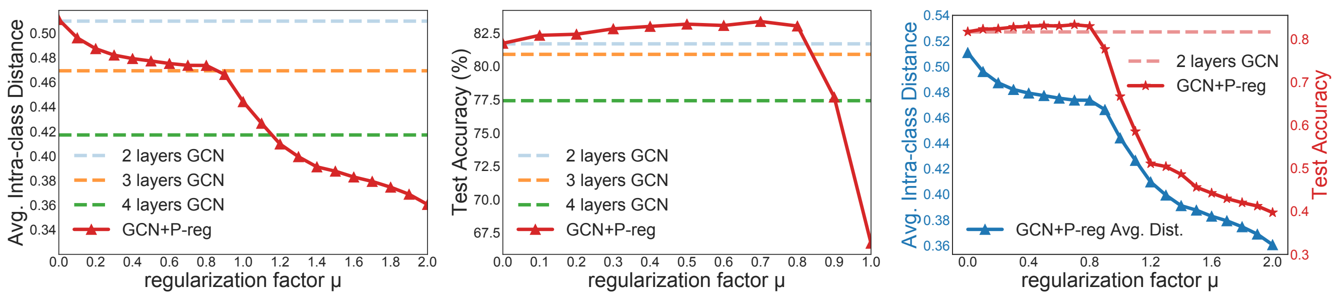
4.2.1 Deeper GCN Provides Information from Farther Nodes

Xu et al. (2018) proved that for a K-layer GCN, the influence distribution[1] I_x for any node x in a graph G is equivalent to the k-step random walk distribution on G starting at node x. Thus, the more layers a GCN has, the more information a node x can obtain from nodes farther away from x in the graph G. Some works such as (Li et al. 2019a; Rong et al. 2020; Huang et al. 2020; Zhao and Akoglu 2020) provide methods to make GNN models deep.

4.2.2 Limitations of Deep GCNs

More layers is not always better. The more layers a GCN

[1]The influence distribution I_x is defined as the normalized influence score $I(x, y)$, where the influence score $I(x, y)$ is the sum of $\left[\delta h_x^{(k)}/\delta h_y^{(0)}\right]$, where $h_x^{(k)}$ is the k-th layer activation of node x.



(a) Avg. intra-class distance ω

(b) Test accuracy

(c) Relation between ω and accuracy

Figure 4: Empirical observation of connections between P-reg and deep GCNs (best viewed in color)

has, the representation vectors of nodes produced become more prone to be similar. As shown in Lemma 3.1, an infinite-depth GCN even gives the same output vectors for all nodes. This is known as the over-smoothing problem of GNNs and there are studies (NT and Maehara 2019; Oono and Suzuki 2020) focusing on this issue, which claim that a GCN can be considered as a low-passing filter and exponentially loses its expressive power as the number of layers increases. Thus, more layers may make the representation vectors of nodes in-discriminable and thus can be detrimental to the task of node classification.

Computational issues. First, generally for all deep neural networks, gradient vanishing/exploding issues will appear when the neural networks go deeper. Although it could be mitigated by using residual skip connections, the training of deeper neural networks will inevitably become both harder and slower. Second, as GNNs grow deeper, the size of a node's neighborhood expands exponentially (Huang et al. 2018), making the computational cost for both gradient back-propagation training and feed-forwarding inference extremely high.

4.2.3 How P-Reg Addresses the Limitations of Deep GCNs

P-reg can effectively balance information capturing and over-smoothing at low cost. Although an infinite-depth model outputting the same vector for each node is not desired, only being able to choose the number of layers L from a discrete set $\mathbb{N}_+$ is also not ideal. Instead, a balance between the size of the receptive field (the farthest nodes that a node can obtain information from) and the preservation of discriminating information is critical for generating useful node representation. To this end, the regularization factor $\mu \in \mathbb{R}$ in Eq. (2) plays a vital role in enforcing a flexible strength of regularization on the GNN. When $\mu = 0$, the GNN model is basically the vanilla model without P-reg. On the contrary, the model becomes an infinite-depth GCN when $\mu \to +\infty$. Consequently, using P-reg with a *continuous* μ can be regarded as using a GCN with a *continuous* rather than discrete number of layers. Adjusting μ empowers a GNN model to balance between information capturing and over-smoothing.

P-reg provides a controllable receptive field for a GNN model and this is achieved at only a computation cost of $\mathcal{O}(NC + |\mathcal{E}|\, C)$, which is basically equivalent to the overhead of just adding one more layer of GCN[2]. We also do not need to worry about gradient vanishing/exploding and neighborhood size explosion. P-reg also has zero overhead for GNN inference.

Empirical verification. Define $\omega = \frac{1}{N}\sum_{k=1}^{C}\sum_{i\in S_k}\|z_i - c_k\|_2$ as the average intra-class Euclidean distance for a trained GNN's output Z, where z_i is the learned representation of node v_i (the i-th row of Z), S_k denotes all nodes of class k in the graph, and $c_k = \frac{1}{|S_k|}\sum_{i\in S_k} z_i$. We trained GCNs with 2, 3 and 4 layers and GCNs+P-reg with different μ. The results on the CORA graph are reported in Figure 4, where all y-axis values are the average values of 10 random trials. Similar patterns are also observed on other datasets in our experiments.

Figure 4a shows that as a GCN has more layers, ω becomes smaller (as indicated by the 3 dashed horizontal lines). Also, as μ increases, ω for GCN+P-reg becomes smaller. The result is consistent with our analysis that a larger μ corresponds to a deeper GCN. Figure 4b further shows that as μ increases, the test accuracy of GCN+P-reg first improves and then becomes worse. The best accuracy is achieved when $\mu = 0.7$. By checking Figure 4a, $\mu = 0.7$ corresponds to a point lying between the lines of the 2-layer and 3-layer GCNs. This implies that the best accuracy could be achieved with an l-layer GCN, where $2 < l < 3$. While a *real* l-layer GCN does not exist, applying P-reg to GCN produces the equivalent result as from an l-layer GCN. Finally, Figure 4c shows the relation between ω and the accuracy. The high accuracy region corresponds to the flat area of ω. This could be explained as the flat area of ω achieves a good balance between information capturing and over-smoothing.

5 Experimental Results

We evaluated P-reg on node classification, graph classification and graph regression tasks. We report the results of P-reg here using Cross Entropy as ϕ if not specified, and the results of different choices of the ϕ function are reported in Appendix F due to the limited space. In addition to the node classification task for both random splits of 7 graph

[2]One layer of GCN has a computation cost of $\mathcal{O}(|\mathcal{E}|\, C)$. Since $2|\mathcal{E}| \geq N$ holds for all undirected connected graphs, we have $\mathcal{O}(NC + |\mathcal{E}|\, C) = \mathcal{O}(3\,|\mathcal{E}|\, C) = \mathcal{O}(|\mathcal{E}|\, C)$.

Table 1: Accuracy improvements brought by P-reg and other techniques using **random splits**

		CORA	CiteSeer	PubMed	CS	Physics	Computers	Photo
GCN	Vanilla	79.47±0.89	69.45±0.99	76.26±1.30	91.48±0.42	93.66±0.41	80.73±1.52	89.24±0.76
	Label Smoothing	79.70±1.23	70.08±1.48	76.83±1.20	91.65±0.49	93.67±0.41	81.31±1.46	90.03±0.81
	Confidence Penalty	79.70±1.08	69.69±1.22	76.57±0.97	91.55±0.45	93.67±0.41	80.70±1.50	89.26±0.76
	Laplacian Regularizer	79.41±0.65	69.46±1.03	76.29±1.13	91.51±0.42	93.65±0.40	81.20±1.69	89.52±0.57
	P-reg	**82.83**±1.16	**71.63**±2.17	**77.37**±1.50	**92.58**±0.32	**94.41**±0.74	**81.66**±1.42	**91.24**±0.75
GAT	Vanilla	79.47±1.77	69.28±1.56	75.61±1.56	91.15±0.36	93.08±0.55	81.00±2.02	89.55±1.20
	Label Smoothing	80.30±1.71	69.42±1.41	76.53±1.28	91.09±0.34	93.25±0.51	81.92±1.98	90.55±0.86
	Confidence Penalty	79.89±1.73	69.87±1.35	76.44±1.53	90.91±0.36	93.13±0.53	78.86±2.07	88.58±1.96
	Laplacian Regularizer	80.23±1.90	69.50±1.42	**76.80**±1.57	90.86±0.39	93.09±0.52	82.38±2.00	90.58±1.15
	P-reg	**82.97**±1.19	**70.00**±1.89	76.39±1.46	**91.92**±0.20	**94.28**±0.29	**83.68**±2.24	**91.31**±1.06
MLP	Vanilla	57.47±2.46	56.96±2.10	68.36±1.35	86.87±1.01	89.43±0.67	62.61±1.81	76.26±1.40
	Label Smoothing	59.00±1.54	57.98±1.76	68.75±1.26	87.90±0.57	89.54±0.61	62.55±2.21	76.12±1.11
	Confidence Penalty	57.20±2.59	56.89±2.15	68.16±1.36	86.97±1.04	89.54±0.63	62.75±2.15	76.19±1.34
	Laplacian Regularizer	60.30±2.47	58.62±2.40	68.67±1.39	86.96±0.75	89.50±0.48	62.60±1.99	76.23±1.09
	P-reg	**64.41**±4.56	**61.09**±2.13	**70.09**±1.75	**90.87**±1.90	**91.57**±0.69	**68.93**±3.28	**79.71**±3.72

Table 2: Comparison of P-reg with the state-of-the-art methods using the **standard split**

	CORA	CiteSeer	PubMed
GCN	81.67±0.60	71.57±0.46	79.17±0.37
GAT	83.20±0.72	71.29±0.84	77.99±0.47
APPNP	83.12±0.44	72.00±0.48	80.10±0.26
GMNN	83.18±0.85	72.77±1.43	**81.63**±0.36
Graph U-Nets	81.31±1.47	67.74±1.48	77.76±0.84
GraphAT	82.46±0.60	73.49±0.38	79.10±0.20
BVAT	83.37±1.01	73.83±0.54	77.97±0.84
GraphMix	83.54±0.72	73.81±0.85	80.76±0.84
GAM	82.28±0.48	72.74±0.62	79.60±0.63
DeepAPPNP	83.78±0.33	71.37±0.66	79.73±0.22
GCN+P-reg	83.38±0.86	**74.83**±0.17	80.11±0.45
GAT+P-reg	**83.89**±0.31	72.86±0.65	78.39±0.27

datasets (Yang, Cohen, and Salakhutdinov 2016; McAuley et al. 2015; Shchur et al. 2019) and the standard split of 3 graph datasets reported in Section 5.1 and 5.2, we also evaluated P-reg on graph-level tasks on the OGB dataset (Hu et al. 2020) in Section 5.3. Our implementation is based on PyTorch (Paszke et al. 2019) and we used the Adam (Kingma and Ba 2014) optimizer with learning rate equal to 0.01 to train all the models. Additional details about the experiment setup are given in Appendix E.

5.1 Improvements on Node Classification Accuracy

We evaluated the node classification accuracy for 3 models on 7 popular datasets using random splits. The train/validation-/test split of all the 7 datasets are 20 nodes/30 nodes/all the remaining nodes per class, as recommended by Shchur et al. (2019). We conducted each experiment on 5 random splits and 5 different trials for each random split. The mean value and standard deviation are reported in Table 1 based on the 5×5 experiments for each cell. The regularization factor μ is determined by grid search using the validation accuracy. The search space for μ is 20 values evenly chosen from $[0, 1]$. The μ is the same for each cell (model $\times$ dataset) in Table 1.

We applied P-reg on GCN, GAT and MLP, respectively. GCN is a typical convolution-based GNN, while GAT is attention based. Thus, they are representative of a broad range of GNNs. MLP is multi-layer perception without modeling the graph structure internally. In addition, we also applied label smoothing, confidence penalty, and Laplacian regularization to the models, respectively, in order to give a comparative analysis on the effectiveness of P-reg. Label smoothing and confidence penalty are two general techniques used to improve the generalization capability of a model. Label smoothing (Szegedy et al. 2016; Müller, Kornblith, and Hinton 2019) has been adopted in many state-of-the-art deep learning models such as (Huang et al. 2019; Real et al. 2019; Vaswani et al. 2017; Zoph et al. 2018) in computer vision and natural language processing. It softens the one-hot hard targets $y_c = 1,\ y_i = 0\ \forall i \neq c$ into $y_i^{LS} = (1-\alpha)\, y_i + \alpha/C$, where c is the correct label and C is the number of classes. Label smoothing Confidence penalty (Pereyra et al. 2017) adds the negative entropy of the network outputs to the classification loss as a regularizer, $\mathcal{L}^{CP} = \mathcal{L}_{cls} + \beta \sum_i p_i \log p_i$, where p_i is the predicted class probability of the i-th sample.

Table 1 shows that P-reg significantly improves the accuracy of both GCN and GAT, as well as MLP. General techniques such as label smoothing and confidence penalty also show their capability to improve the accuracy on most datasets but the improvements are relatively small compared with the improvements brought by P-reg. The improvements

Table 3: The scores (with standard deviation) of graph-level tasks of P-reg applied on GCN/GIN on the OGB datasets (ROC-AUC scores: higher is better. RMSE scores: lower is better.)

	moltox21 (ROC-AUC)↑	molhiv (ROC-AUC)↑	molbbbbp (ROC-AUC)↑	molesol (RMSE)↓	molfreesolv (RMSE)↓
GCN	74.75±0.55	**76.08**±1.41	67.99±1.18	1.124±0.034	2.564±0.148
GCN+P-reg	75.86±0.75	76.03±1.25	69.37±1.03	1.141±0.038	2.406±0.147
GCN+Virtual	77.42±0.67	75.67±1.54	66.92±0.85	1.020±0.056	2.164±0.129
GCN+Virtual+P-reg	**77.58**±0.39	75.81±1.61	**69.67**±1.62	**0.946**±0.041	**2.082**±0.119
GIN	74.89±0.66	75.47±1.22	68.03±2.72	1.150±0.057	2.971±0.263
GIN+P-reg	74.90±0.59	76.77±1.41	67.69±2.43	1.137±0.028	2.529±0.216
GIN+Virtual	77.04±0.83	76.33±1.52	68.37±2.08	0.991±0.052	2.172±0.192
GIN+Virtual+P-reg	**77.51**±0.68	**77.12**±1.14	**70.25**±1.86	**0.965**±0.069	**2.050**±0.144

by P-reg are also consistent in all the cases except for GAT on the PubMed dataset. Although Laplacian regularization improves the performance of MLP on most datasets, it has marginal improvement or even negative effects on GCN and GAT, which also further validates our analysis in Section 4.1.

5.2 Comparison with the State-of-the-Art Methods (using the Standard Split)

We further compared GCN+P-reg and GAT+P-reg with the state-of-the-art methods. Among them, APPNP (Klicpera, Bojchevski, and Günnemann 2019), GMNN (Qu, Bengio, and Tang 2019) and Graph U-Nets(Gao and Ji 2019) are newly proposed state-of-the-art GNN models, and GraphAT (Feng et al. 2019), BVAT (Deng, Dong, and Zhu 2019) and GraphMix (Verma et al. 2019) use various complicated techniques to improve the performance of GNNs. GraphAT (Feng et al. 2019) and BVAT (Deng, Dong, and Zhu 2019) incorporate the adversarial perturbation[3] into the input data. GraphMix (Verma et al. 2019) adopts the idea of co-training (Blum and Mitchell 1998) to use a parameters-shared fully-connected network to make a GNN more generalizable. It combines many other semi-supervised techniques such as Mixup (Zhang et al. 2017), entropy minimization with Sharpening (Grandvalet and Bengio 2005), Exponential Moving Average of predictions (Tarvainen and Valpola 2017) and so on. GAM (Stretcu et al. 2019) uses co-training of GCN with an additional agreement model that gives the probability that two nodes have the same label. The DeepAPPNP (Rong et al. 2020; Huang et al. 2020) we used contains $K = 64$ layers with restart probability $\alpha = 0.1$ and DropEdge rate 0.2. If more than one method/variant is proposed in the respective papers, we report the best performance we obtained for each work using their official code repository or PyTorch-geometric (Fey and Lenssen 2019) benchmarking code.

Table 2 reports the test accuracy of node classification using the standard split, on the CORA, CiteSeer and PubMed datasets. We report the mean and standard deviation of the accuracy of 10 different trials. The search space for μ is 20 values evenly chosen from $[0, 1]$. P-reg outperforms the state-of-the-art methods on CORA and CiteSeer, while its performance is among the best three on PubMed. This result shows that P-reg's performance is very competitive because P-reg is a simple regularization with only a single hyper-parameter, unlike in the other methods where heavy tricks, adversarial techniques or complicated models are used.

[3]Adversarial perturbation is the perturbation along the direction of the model gradient $\frac{\partial f}{\partial x}$.

5.3 Experiments of Graph-Level Tasks

We also evaluated the performance of P-reg on graph-level tasks on the OGB (Open Graph Benchmark) (Hu et al. 2020) dataset. We reported the mean and standard deviation of the scores for the graph-level tasks of 10 different trials in Table 3. The ROC-AUC scores of the graph classification task on the *moltox21, molhiv, molbbbbp* datasets are the higher the better, while the RMSE scores of the graph regression task on the *molesol, molfreesolv* datasets are the lower the better. We use the code of GCN, Virtual GCN, GIN, Virtual GIN provided in the OGB codebase. Here *Virtual* means that the graphs are augmented with virtual nodes (Gilmer et al. 2017). GIN (Xu et al. 2019) represents the state-of-the-art backbone GNN model for graph-level tasks. P-reg was added directly to those models, without modifying any other configurations. The search space for μ is 10 values evenly chosen from $[0, 1]$.

Table 3 shows that P-reg can improve the performance of GCN and GIN, as well as Virtual GCN and Virtual GIN, in most cases for both the graph classification task and the graph regression task. In addition, even if *virtual node* already can effectively boost the performance of GNN models in the graph-level tasks, using P-reg can still further improve the performance. As shown in Table 3, Virtual+P-reg makes the most improvement over vanilla GNNs on most of the datasets.

6 Conclusions

We presented P-reg, which is a simple graph regularizer designed to boost the performance of existing GNN models. We theoretically established its connection to graph Laplacian regularization and its equivalence to an infinite-depth GCN. We also showed that P-reg can provide new supervision signals and simulate a deep GCN at low cost while avoiding over-smoothing. We then validated by experiments that compared with existing techniques, P-reg is a significantly more effective method that consistently improves the performance of popular GNN models such as GCN, GAT and GIN.

7 Acknowledgments

We thank the reviewers for their valuable comments. This work was partially supported GRF 14208318 from the RGC and ITF 6904945 from the ITC of HKSAR, and the National Natural Science Foundation of China (NSFC) (Grant No. 61672552).

References

Ando, R. K.; and Zhang, T. 2007. Learning on graph with Laplacian regularization. In *Advances in neural information processing systems*, 25–32.

Belkin, M.; Niyogi, P.; and Sindhwani, V. 2006. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research* 7(Nov): 2399–2434.

Blum, A.; and Mitchell, T. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on computational learning theory*, 92–100.

Chung, F. R. 1997. *Spectral Graph Theory*. Number 92 in CBMS Regional Conference Series. Conference Board of the Mathematical Sciences. ISBN 9780821889367.

Civin, P.; Dunford, N.; and Schwartz, J. T. 1960. *Linear Operators. Part I: General Theory.* Pure and applied mathematics. Interscience Publishers.

Deng, Z.; Dong, Y.; and Zhu, J. 2019. Batch virtual adversarial training for graph convolutional networks. *arXiv:1902.09192* .

Ding, M.; Tang, J.; and Zhang, J. 2018. Semi-Supervised Learning on Graphs with Generative Adversarial Nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 913–922.

Feng, F.; He, X.; Tang, J.; and Chua, T.-S. 2019. Graph Adversarial Training: Dynamically Regularizing Based on Graph Structure. *IEEE Transactions on Knowledge and Data Engineering* .

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Gao, H.; and Ji, S. 2019. Graph U-Nets. In *International Conference on Machine Learning*, 2083–2092.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 1263–1272.

Grandvalet, Y.; and Bengio, Y. 2005. Semi-Supervised Learning by Entropy Minimization. In *Advances in neural information processing systems*, 529–536. Cambridge, MA, USA.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, 1024–1034.

Hammond, D. K.; Vandergheynst, P.; and Gribonval, R. 2011. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis* 30(2): 129–150.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv:1503.02531* .

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open graph benchmark: Datasets for machine learning on graphs. *arXiv:2005.00687* .

Huang, W.; Rong, Y.; Xu, T.; Sun, F.; and Huang, J. 2020. Tackling Over-Smoothing for General Graph Convolutional Networks. *arXiv:2008.09864* .

Huang, W.; Zhang, T.; Rong, Y.; and Huang, J. 2018. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems*, 4558–4567.

Huang, Y.; Cheng, Y.; Bapna, A.; Firat, O.; Chen, D.; Chen, M.; Lee, H.; Ngiam, J.; Le, Q. V.; Wu, Y.; et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, 103–112.

Jaynes, E. T. 1957. Information theory and statistical mechanics. *Physical review* 106(4): 620.

Karasuyama, M.; and Mamitsuka, H. 2013. Manifold-based similarity adaptation for label propagation. In *Advances in neural information processing systems*, 1547–1555.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv:1412.6980* .

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *International Conference on Learning Representations*.

Li, G.; Müller, M.; Thabet, A.; and Ghanem, B. 2019a. DeepGCNs: Can GCNs Go as Deep as CNNs? In *The IEEE International Conference on Computer Vision (ICCV)*.

Li, Q.; Wu, X.-M.; Liu, H.; Zhang, X.; and Guan, Z. 2019b. Label efficient semi-supervised learning via graph filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9582–9591.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2015. Gated graph sequence neural networks. *arXiv:1511.05493* .

Liu, Y.; Lee, J.; Park, M.; Kim, S.; Yang, E.; Hwang, S.; and Yang, Y. 2019. Learning to Propagate Labels: Transductive Propagation Network for Few-shot Learning. In *International Conference on Learning Representations*.

McAuley, J.; Targett, C.; Shi, Q.; and Van Den Hengel, A. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 43–52.

Miller, D.; Rao, A. V.; Rose, K.; and Gersho, A. 1996. A global optimization technique for statistical classifier design. *IEEE Transactions on Signal Processing* 44(12): 3108–3122.

Müller, R.; Kornblith, S.; and Hinton, G. E. 2019. When does label smoothing help? In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 4694–4703.

NT, H.; and Maehara, T. 2019. Revisiting Graph Neural Networks: All We Have is Low-Pass Filters. *arXiv:1905.09550* .

Oono, K.; and Suzuki, T. 2020. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 8024–8035.

Pei, H.; Wei, B.; Chang, K. C.-C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*.

Pereyra, G.; Tucker, G.; Chorowski, J.; Kaiser, Ł.; and Hinton, G. 2017. Regularizing Neural Networks by Penalizing Confident Output Distributions. *arXiv:1701.06548* .

Qu, M.; Bengio, Y.; and Tang, J. 2019. GMNN: Graph Markov Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML*, 5241–5250.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 4780–4789.

Rong, Y.; Huang, W.; Xu, T.; and Huang, J. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.

Shchur, O.; Mumme, M.; Bojchevski, A.; and Günnemann, S. 2019. Pitfalls of graph neural network evaluation. *arXiv:1811.05868* .

Smola, A. J.; and Kondor, R. 2003. Kernels and regularization on graphs. In *Learning theory and kernel machines*, 144–158. Springer.

Stretcu, O.; Viswanathan, K.; Movshovitz-Attias, D.; Platanios, E.; Ravi, S.; and Tomkins, A. 2019. Graph Agreement Models for Semi-Supervised Learning. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8713–8723.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

Tarvainen, A.; and Valpola, H. 2017. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in neural information processing systems*, 1195–1204.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, 5998–6008. Curran Associates, Inc.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.

Verma, V.; Qu, M.; Lamb, A.; Bengio, Y.; Kannala, J.; and Tang, J. 2019. GraphMix: Regularized Training of Graph Neural Networks for Semi-Supervised Learning. *arXiv:1909.11715* .

Wang, F.; and Zhang, C. 2007. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering* 20(1): 55–67.

Wang, H.; and Leskovec, J. 2020. Unifying graph convolutional neural networks and label propagation. *arXiv:2002.06755* .

Wu, F.; Souza, A.; Zhang, T.; Fifty, C.; Yu, T.; and Weinberger, K. 2019. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 6861–6871.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations*.

Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.-i.; and Jegelka, S. 2018. Representation learning on graphs with jumping knowledge networks. *arXiv:1806.03536* .

Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. *arXiv:1603.08861* .

Zhang, H.; Cisse, M.; Dauphin, Y. N.; and Lopez-Paz, D. 2017. mixup: Beyond empirical risk minimization. *arXiv:1710.09412* .

Zhang, L.; Song, J.; Gao, A.; Chen, J.; Bao, C.; and Ma, K. 2019. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE International Conference on Computer Vision*, 3713–3722.

Zhao, L.; and Akoglu, L. 2020. PairNorm: Tackling Oversmoothing in GNNs. In *International Conference on Learning Representations*.

Zhou, D.; Bousquet, O.; Lal, T. N.; Weston, J.; and Schölkopf, B. 2003. Learning with Local and Global Consistency. In *Advances in Neural Information Processing Systems (NIPS)*.

Zhu, X.; Ghahramani, Z.; and Lafferty, J. 2003. Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions. In *Proceedings of the 20th International Conference on International Conference on Machine Learning (ICML)*, 912–919.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697–8710.

Supplementary Materials For "Rethinking Graph Regularization for Graph Neural Networks"

A Notations

Table 4: Notations

G	a graph
$\mathcal{V}$	the vertex set of a graph
$\mathcal{E}$	the edge set of a graph
$G(A, X)$	a graph G with its adjacency matrix A and node feature matrix X
v_i	the i-th node in a graph
$\mathcal{N}(v_i)$	the neighborhood of v_i
A	$A \in \mathbb{R}^{N\times N}$, the adjacency matrix of a graph
D	the diagonal degree matrix, $D_{ii} = \sum_{k=1}^{C} A_{ik},\ D_{ij} = 0$ if $i \neq j$.
$\hat{A}$	the normalized adjacency matrix of A, $\hat{A} = D^{-1}A$
Δ	the graph Laplacian matrix, $\Delta = D - A$
$\tilde{\Delta}$	the normalized Laplacian matrix, $\tilde{\Delta} = I - D^{-1}A$
N	the number of nodes in a graph G
M	the size of the training set
F	the dimensionality of node features
C	the number of classes, as well as the dimensionality of model outputs
H	the dimensionality of hidden units in GNN
X	$X \in \mathbb{R}^{N\times F}$, node features
Z	$Z \in \mathbb{R}^{N\times C}$, model outputs
Z'	the propagated model output, $Z' = \hat{A}Z$
P	$P \in \mathbb{R}^{N\times C}$, *softmax* of Z, predicted posterior class probability
Q	$Q \in \mathbb{R}^{N\times C}$, *softmax* of Z'
Y	$Y \in \mathbb{N}^{N\times C}$, one-hot ground truth label, $Y_{ic} = 1$ if v_i is of class c, otherwise $Y_{ic} = 0$
x_i	$x_i \in \mathbb{R}^{F}$, the feature of v_i, $x_i = (X)_i^\top$
z_i	$z_i \in \mathbb{R}^{C}$, the learned representation of v_i, $z_i = (Z)_i^\top$
y_i	$y_i \in \mathbb{R}^{C}$, the i-th row vector of Y
p_i	$p_i \in \mathbb{R}^{C}$, the i-th row vector of P
$(\cdot)_i^\top$	the i-th row of $(\cdot)$
$\|S\|$	the size of a set S
$\mathcal{L}_{cls}$	the supervised classification loss, e.g., cross entropy loss
$\mathcal{L}_{P\text{-}reg}$	the Propagation-regularization term

B Proofs to Lemmas and Theorems

B.1 Proof of Theorem 3.1

Proof. The squared-error P-reg $\mathcal{L}_{P\text{-}SE}$ is given as follows:

$$\phi_{SE}(Z, \hat{A}Z) = \frac{1}{2}\sum_{i=1}^{N} \left\|(\hat{A}Z)_i^\top - (Z)_i^\top\right\|_2^2 = \frac{1}{2}\left\|\hat{A}Z - Z\right\|_F^2 = \frac{1}{2}\left\|\tilde{\Delta}Z\right\|_F^2 = \frac{1}{2}\left\langle Z, \tilde{\Delta}^\top\tilde{\Delta}Z\right\rangle.$$

Thus, from the traditional Laplacian-based regularization perspective, the squared-error P-reg, $\mathcal{L}_{P\text{-}SE}$, is equivalent to taking $R = r(\tilde{\Delta}) = \tilde{\Delta}^\top \tilde{\Delta}$ as the regularization matrix in the class of Laplacian-based regularization functions, i.e., $\langle Z, RZ \rangle$. □

B.2 Proof of Lemma 3.1

Proof. Chung (1997) proved that for a connected graph with self-loops ($A := A + I$), the largest eigenvalue λ_1 of the normalized adjacency matrix $\hat{A} = D^{-1}A$ is equal to 2, and the multiplicity of λ_1 is 1. The corresponding eigenvector is $u_1 = \mathbf{1} = (1, \ldots, 1)^\top$. And the smallest eigenvalue λ_n is 0.

Denote the j-th column vector of Z as $Z_j,\ j \in [1, C]$. By applying the power iteration method, we obtain $\mathbf{1} = u_1 = \lim_{k\to\infty} \frac{\hat{A}^k Z_j}{\|\hat{A}^k Z_j\|}$, and thus $\hat{A}^\infty Z_j = k\mathbf{1}\ \ \forall j \in [1, C]$, where $k \in \mathbb{R}\backslash\{0\}$ is a constant. Thus, we know the column vectors of $\tilde{Z} = \hat{A}^\infty Z$ are all vector $\mathbf{1}$ multiplied by constants, which means that all the row vectors of $\tilde{Z}$ are the same. □

B.3 Proof of Lemma 3.2

Proof. Chung (1997) proved that for the Laplacian matrix Δ of a connected graph G, 0 is an eigenvalue of multiplicity 1 for Δ, and its corresponding eigenvector is $\mathbf{1} = (1, \ldots, 1)^\top \in \mathbb{R}^N$. Then, if we take each column vector of $\tilde{Z}$ as $\tilde{Z}_j = k\mathbf{1}$, where $j \in [1, \ldots, C]$ and $k \in \mathbb{R}\backslash\{0\}$ is a constant (then all row vectors of $\tilde{Z}$ are the same), we have $\Delta\tilde{Z} = \mathbf{0}$. Thus, $\left\|\Delta\tilde{Z}\right\|_F^2 = \left\|D\tilde{Z} - A\tilde{Z}\right\|_F^2 = 0$, indicating $\phi_{SE}(Z, AZ) = \left\|\tilde{Z} - \hat{A}\tilde{Z}\right\|_F^2 = \left\|D^{-1}\left(D\tilde{Z} - A\tilde{Z}\right)\right\|_F^2 = \sum_{i=1}^N \frac{1}{D_{ii}} \left\|\left(\tilde{\Delta}Z\right)_i^\top\right\|_2^2 = 0$, which is the minimum value of ϕ_{SE}. Thus, such $\tilde{Z}$ with all row vectors $\tilde{z}_i\ \forall i \in [1, \ldots, N]$ being the same is an optimal solution to $\arg\min_Z \left\|\hat{A}Z - Z\right\|_F^2$. □

B.4 Proof of Theorem 3.2

Proof. Define $\tilde{Z} = \hat{A}^\infty Z$. We want to show that $\tilde{Z}$ is the optimal solution for all the three versions of P-reg $\phi(Z, \hat{A}Z)$.

Consider the Cross Entropy version first: $\arg\min_Z \phi_{CE}(Z, \hat{A}Z) = -\sum_{i=1}^N \sum_{j=1}^C P_{ij} \log Q_{ij}$.

By computing the gradient of P to Z_{ij}:

$$\sum_{m=1}^C \frac{\partial P_{im}}{\partial Z_{ij}} = \sum_{m=1}^C (-P_{ij}P_{im}) + P_{ij} = P_{ij}\left(1 - \sum_{m=1}^C P_{im}\right).$$

We can get $\sum_{m=1}^C \frac{\partial P_{im}}{\partial Z_{ij}} = 0$ because $\sum_{m=1}^C P_{im} = 1$.

Then we can compute the gradient of the Cross Entropy version of P-reg $\frac{\partial \mathcal{L}}{\partial Z_{ij}}$:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial Z_{ij}} &= \sum_{k\in\mathcal{N}(i)\cup i} \frac{\partial \mathcal{L}_k}{\partial Z_{ij}} \\
&= \frac{\partial \mathcal{L}_i}{\partial Z_{ij}} + \sum_{k\in\mathcal{N}(i)} \frac{\partial \mathcal{L}_k}{\partial Z_{ij}} \\
&= -\frac{1}{N}\sum_{m=1}^C \log\left(\frac{1}{d_i}\sum_{k\in\mathcal{N}(i)} P_{km}\right)\frac{\partial P_{im}}{\partial Z_{ij}} + \sum_{k\in\mathcal{N}(i)} \frac{\partial \mathcal{L}_k}{\partial Z_{ij}} \\
&= 0 + \sum_{k\in\mathcal{N}(i)} \frac{\partial \mathcal{L}_k}{\partial Z_{ij}} \\
&= -\frac{1}{N}\sum_{k\in\mathcal{N}(i)}\sum_{m=1}^C \left(P_{km}\frac{1}{\sum_{l\in\mathcal{N}(k)\backslash i} P_{lm} + P_{im}}\right)\frac{\partial P_{im}}{\partial Z_{ij}} \\
&= -\frac{1}{N}\sum_{k\in\mathcal{N}(i)}\left(\sum_{m=1}^C\left(\frac{P_{km}}{\sum_{l\in\mathcal{N}(k)\backslash i} P_{lm} + P_{im}}\right)(-P_{ij}P_{im}) + \left(\frac{P_{kj}}{\sum_{l\in\mathcal{N}(k)\backslash i} P_{lj} + P_{ij}}\right)P_{ij}\right).
\end{aligned}$$

Consider the situation $rank(Z) = 1$. For $P = \text{softmax}(Z)$, we know that all row vectors of P will be the same, i.e., $P_{1m} = \cdots = P_{Nm}\ \ \forall m \in [1, C]$. Taking this into the above $\frac{\partial \mathcal{L}}{\partial Z_{ij}}$, and considering $\sum_{m=1}^C P_{im} = 1$, we can get $\frac{\partial \mathcal{L}}{\partial Z_{ij}} =$

$-\frac{1}{N}\sum_{k\in\mathcal{N}(i)}\left(\sum_{m=1}^{C}\left(\frac{1}{d_k}\right)(-P_{ij}P_{im})+\left(\frac{1}{d_k}\right)P_{ij}\right)=-\frac{1}{N}\sum_{k\in\mathcal{N}(i)}\left(\left(\frac{1}{d_k}\right)P_{ij}\sum_{m=1}^{C}(1-P_{im})\right)=0$. Thus, the P-reg $\phi_{CE}(Z,\hat{A}Z)$ is optimized when $rank(Z)=1$. Taking $\tilde{Z}=\hat{A}^{\infty}Z$, from Lemma 3.1 we know $rank(\tilde{Z})=1$. Thus, $\tilde{Z}$ is the optimal solution for the Cross Entropy version of P-reg.

And for the Squared Error version $\arg\min_Z \phi_{SE}(Z,\hat{A}Z)=\frac{1}{2}\sum_{i=1}^{N}\left\|(\hat{A}Z)_i^{\top}-(Z)_i^{\top}\right\|_2^2$, taking $\tilde{Z}=\hat{A}^{\infty}Z$, we have $\tilde{Z}=\hat{A}\tilde{Z}$. Thus, $\frac{1}{2}\sum_{i=1}^{N}\left\|(\hat{A}\tilde{Z})_i^{\top}-\left(\tilde{Z}\right)_i^{\top}\right\|_2^2=0$, which is the minimum value for $\phi_{SE}(Z,\hat{A}Z)$. Thus, $\tilde{Z}$ is the optimal solution to $\phi_{SE}(Z,\hat{A}Z)$.

Similarly, consider the KL Divergence version $\arg\min_Z \phi_{KL}(Z,\hat{A}Z)=\sum_{i=1}^{N}\sum_{j=1}^{C}P_{ij}\log\frac{P_{ij}}{Q_{ij}}$. When taking $\tilde{Z}=\hat{A}^{\infty}Z$, we can get $P=Q$ because $rank(\tilde{Z})=1$. Thus, $\phi_{KL}(\tilde{Z},\hat{A}\tilde{Z})=0$, which is the minimum value of $\phi_{KL}(Z,\hat{A}Z)$. Thus $\tilde{Z}$ is the optimal solution to $\phi_{KL}(Z,\hat{A}Z)$. □

C Annealing and Thresholding of P-reg

C.1 Annealing of P-reg

In practice, we adopt a relative small μ to make the classification loss dominate in the early stage. After f_1 becomes a reasonably good predictor, the P-reg term will then dominate to further fine-tune the model parameters. A relatively small μ will implicitly make P-reg work in the later stage of training.

To explicitly enforce the model to converge first and then let P-reg further tune the model parameters gradually, we could have an annealing version of P-reg as:

$$\mathcal{L}=\text{CrossEntropy}(Z,Y_{train})+\mu^t\frac{1}{N}\phi\left(Z,\hat{A}Z\right),$$

where $0<\mu<1$, and $t=\frac{1}{epoch}$, $1\le epoch\le E$, E is the total epochs.

C.2 Thresholding of P-reg

As it is shown in the analysis in Section 4.2 of the main paper, we do not expect the P-reg term to be exact 0, i.e., $Z=\hat{A}Z$, since this causes over-smoothing and makes the representation for nodes in-discriminable.

Thus, we could strengthen P-reg by only computing its backward gradient when it is larger than a specified threshold. A hinge loss term can help achieve this:

$$\mathcal{L}=\text{CrossEntropy}\left(Z,Y_{train}\right)+\mu\max\left(0,\frac{1}{N}\phi\left(Z,\hat{A}Z\right)-\tau\right),$$

where $\tau>0$ is the threshold above which P-reg is applied.

D Related Work

Prediction/Label Propagation. In P-reg, $\phi(Z,\hat{A}Z)$, where $Z\in\mathbb{R}^{N\times C}$ is the output of a graph neural network $f(A,X)$ and can be used for prediction, $\hat{A}Z$ can be regarded as to further propagate the prediction vectors. In classical Graph Label Propagation algorithms (Zhu, Ghahramani, and Lafferty 2003; Zhou et al. 2003), they directly propagate the labels of nodes in an iterative manner to predict over unlabeled nodes, i.e., $Z_{t+1}=\alpha\hat{A}Z_t+(1-\alpha)Y$. And there are some adaptive variants (Wang and Zhang 2007; Karasuyama and Mamitsuka 2013; Liu et al. 2019) trying to build better graph structures from the node features for propagation. There are also some work (Li et al. 2019b; Wang and Leskovec 2020) attempting to unify graph neural networks with graph label propagation algorithms. Li et al. (2019b) proposed Generalized Label Propagation, which uses the closed-form solution to Laplacian regularized label propagation as the graph filter for node features, and Wang and Leskovec (2020) proposed to use the classification loss of label propagation as a regularizer to optimize the graph structure A to help graph convolution networks achieve higher node classification accuracy. In contrast, P-reg is a regularizer to minimize the gap between the trained graph neural network prediction outputs Z and the propagated prediction outputs $\hat{A}Z$, which is different from the above-mentioned works.

Soft Targets (Penalizing Label Confidence). Confidence Penalty (Miller et al. 1996) comes from the maximum entropy principle (Jaynes 1957). It adds the negative entropy of the network outputs to the classification loss as a regularizer, $\mathcal{L}^{CP}=-\sum_{i\in\text{train}}y_i\log p_i-\beta\left(-\sum_i p_i\log p_i\right)$, where p_i is the output class probability distillation for the i-th sample, thus penalizing the confidence of the network prediction. Label Smoothing (Szegedy et al. 2016) is a strategy to improve the classification accuracy by softening the one-hot hard targets $y_c=1,\ y_i=0\ \forall i\neq c$ into $y_i^{LS}=(1-\alpha)\,y_i+\alpha/C$, where c is the correct label and C is the number of classes. Label Smoothing has been adopted in many state-of-the-art models (Zoph et al. 2018; Real et al. 2019; Huang et al. 2019; Vaswani et al. 2017) in computer vision and natural language processing. Pereyra et al. (2017) and

Müller, Kornblith, and Hinton (2019) conducted extensive experiments on Label Smoothing and Confidence Penalty to show their effectiveness in improving the generalization of neural network models. Different from manually smoothing the targets, knowledge distillation (Hinton, Vinyals, and Dean 2015) trains a teacher model to generate soft targets for student models to learn. Using P-reg can be viewed as a self-distillation (Zhang et al. 2019) model, treating the trained 2-layer graph neural network model output as the soft supervision signal for all nodes' neighbors' aggregated output. And the whole model is trained in an end-to-end manner.

Graph Regularization. Apart from the traditional graph regularization theory based on Laplacian operators, which is to extend Laplacian regularizer by applying r to the spectrum of Laplacian Δ, i.e., $r(\Delta)$ (Civin, Dunford, and Schwartz 1960), some modern regularizers for graph neural networks have been proposed. GraphSGAN (Ding, Tang, and Zhang 2018) used generative adversarial networks to generate fake examples in low density region to help the semi-supervised learning model generalize better. GraphAT (Feng et al. 2019), as well as BVAT (Deng, Dong, and Zhu 2019), incorporates the adversarial training methods into the training of GNN models, which adds adversarial perturbation into the input data, where adversarial perturbation is the perturbation along to the direction of the model gradient $\frac{\partial f}{\partial x}$. GAM (Stretcu et al. 2019) co-trains GNN models with an additional agreement model which gives the probability that two nodes have the same label. GraphMix (Verma et al. 2019) also uses the idea of co-training (Blum and Mitchell 1998), by using a parameters-shared fully-connected network to make GNNs more generalizable. It also combined many semi-supervised techniques such as Mixup (Zhang et al. 2017), entropy minimization with Sharpening (Grandvalet and Bengio 2005), Exponential Moving Average of predictions (Tarvainen and Valpola 2017) and so on. These methods either need a pre-trained model to generated adversarial samples, or need additional gradient computation to modify the input data, or use too many semi-supervised learning tricks. In contrast, P-reg is a regularizer relying on no extra models or input modification, and P-reg has a low computation cost and memory overhead.

E More Details about the Experiments

E.1 Datasets

Table 5 lists some statistics of the datasets used in our node classification experiments. CORA, CiteSeer and PubMed are citation networks prepared by Yang, Cohen, and Salakhutdinov (2016). Shchur et al. (2019) provided the pre-processed Computers, Photo, CS and Photo datasets. The Computers and Photo datasets are Amazon co-purchase graphs from McAuley et al. (2015). The CS and Photo datasets are obtained from the Microsoft Academic Graph. All these datasets can be downloaded from the PyTorch-geometric (Fey and Lenssen 2019) built-in corresponding dataset loader.

Table 5: Node classification datasets

	Nodes	Edges	Features	Classes
CORA	2,708	5,278	1,433	7
CiteSeer	4,552	3,668	3,327	6
PubMed	19,717	44,324	500	3
CS	18,333	81,894	6,805	15
Physics	34,493	247,962	8,415	5
Computers	13,752	245,861	767	10
Photo	7,650	119,081	745	8

E.2 Models

In this section, we describe the details of the models we used in the node classification experiments.

The GCN has 2 graph convolutional layers, with the size of hidden units equal to 64. *ReLU* and a dropout layer with probability 0.5 are applied to the output of the first graph convolutional layer.

The GAT has 2 graph attention layers. The first layer has 8 heads with the size of hidden units equal to 16. The second graph attention layer has 1 head. Both layers' attention dropout probabilities are 0.6. A dropout layer is applied before each graph attention layer, with dropout probability 0.6. *ELU* is used as the activation function.

The MLP has 2 fully connected layers, with the size of hidden units equal to 16. *ReLU* is used as the activation function.

E.3 Training and Evaluating

We used the Adam (Kingma and Ba 2014) optimizer with learning rate equal to 0.01 to train all the models. For experiments on CORA, CiteSeer and PubMed, L_2 weight decay of 5×10^{-4} was applied, and for experiments on the CS, Physics, Computers and Photo datasets, no weight decay was applied. We used early stopping to determine the training epochs. If validation accuracy no longer increased for 200 epochs, the training was stopped. All these hyperparameters were fixed in all the experiments.

The search space for μ is 20 values evenly chosen from $[0, 1]$ based on the validation accuracy. μ remains the same for each cell (model $\times$ dataset) in Table 1.

To avoid bias and randomness, for the random split experiments, we evaluated each experiment on 5 randomly generated splits, and 5 different trials for each split. The training/validation/test splits for all datasets are 20/30/rest nodes per class. These settings follow the way of (Shchur et al. 2019).

For the standard split experiments using the state-of-the-art methods, the code we used is either the official code released by the authors or from the PyTorch-geometric (Fey and Lenssen 2019) implementation for benchmarking. Experimental results for all the models were from 10 random trials with initialization random seeds from 0 to 9.

E.4 Software and Hardware

P-reg in our experiments was implemented on PyTorch 1.5.0 (Paszke et al. 2019) and PyTorch-geometric 1.4.3 (Fey and Lenssen 2019). We ran our experiments on Ubuntu 18.04 LTS with CUDA 10.1. The hardware platform we used in the experiments has 40 cores Intel(R) Xeon(R) Silver 4114 CPU @ 2.20GHz, 256 GB Memory, and 4 Nvidia RTX 2080Ti graphics cards.

F Choices of ϕ

The node classification accuracy of GCN and GAT using the three versions of P-reg, i.e., Squared Error, Cross Entropy, and KL Divergence on the 7 datasets using random splits is reported in Table 6. The three versions of P-reg have similar performance, and we report the results of the Cross Entropy version of P-reg in the main paper because it shows better stability.

Table 6: The performance of P-reg with different ϕ functions

		CORA	CiteSeer	PubMed	CS	Physics	Computers	Photo
GCN	Vanilla	79.47±0.89	69.45±0.99	76.26±1.30	91.48±0.42	93.66±0.41	80.73±1.52	89.24±0.76
	Squared Error	82.85±1.08	71.58±1.86	77.43±1.63	92.59±0.33	94.21±0.69	81.80±1.53	91.10±0.73
	Cross Entropy	82.83±1.16	71.63±2.17	77.37±1.50	92.58±0.32	94.41±0.74	81.66±1.42	91.24±0.75
	KL Divergence	80.40±0.99	69.76±1.15	77.37±0.68	92.07±0.50	94.19±0.39	82.23±1.34	90.84±1.02
GAT	Vanilla	79.47±1.77	69.28±1.56	76.64±2.03	91.15±0.36	93.08±0.55	81.00±2.02	89.55±1.20
	Squared Error	82.96±1.45	70.60±1.50	76.62±1.99	91.83±0.22	94.14±0.55	76.89±1.23	86.53±2.03
	Cross Entropy	82.97±1.19	70.00±1.89	76.56±1.99	91.92±0.20	94.28±0.29	83.68±2.24	91.31±1.06
	KL Divergence	80.46±1.22	69.37±1.76	76.38±1.85	90.97±0.33	93.29±0.42	79.64±2.77	90.37±1.23