Lrn&Align: A Parameter-Free Method for Reducing Over-Smoothing in Graph Convolutional Networks

Haimin Zhang and Min Xu[1]

Abstract

Graph convolutional networks have today become the dominant approach for learning on graph-structured data. However, studies continually find that message-passing graph convolutional networks suffer from the over-smoothing problem, which is a core issue that prevents us from building deep graph convolutional network models. In this paper, we present Lrn&Align, a parameter-free method for reducing over-smoothing in graph convolutional networks. The idea of Lrn&Align is to layerwisely learn embeddings and then align the learned embeddings with those of the previous layer by reducing the angle between these embeddings. Through alignment, the over-smoothness of the learned embeddings is explicitly reduced. We evaluate our Lrn&Align method on six popular benchmark datasets on different graph domain tasks. The experimental results show that our Lrn&Align is a general method that improves the performance of a variety of graph convolutional network models, advancing the state of the art performance for graph representation learning.

1. Introduction

Graph-structured data are very commonly seen in the real world. Social networks, molecular structures and citation networks—all of these types of data can be described using graphs. It is crucial to design and develop models that are able to learn and and generalize from this kind of data. Recent years have seen a surge in studies on learning on graph-structured data, including techniques for deep graph embedding and generalizations of convolutional neural networks to non-Euclidean data (Hamilton, 2020). These advances have produced new state of the art results in a wide variety domains, including recommendation systems, drug discovery, 2D/3D computer vision, and question answering systems.

The major challenge in learning on graph-structured data is that they have an underlying representation that is in non-Euclidean spaces. Derived as a generalization of convolutions, the graph convolutional network framework is a general approach over graphs. The defining feature of a graph convolutional network is that it uses a form of message passing in which messages are exchanged between nodes and updated using neural networks (Gilmer et al., 2017). The message passing paradigm is at the core of current graph convolutional networks, but it also has serious drawbacks. The power of message-passing graph convolutional networks are inherently bounded by the Weisfeiler-Lehman isomorphism test (Xu et al., 2019; Morris et al., 2019). Empirically, studies have continually found that message-passing graph neural networks suffer from the problem of over-smoothing, and this issue of over-smoothing can be viewed as a consequence of the neighborhood aggregation operation (Hamilton, 2020).

The basic idea of over-smoothing is that the embeddings for all the nodes in the graph can converge to similar values after many iterations of message passing. This phenomenon is especially common in basic graph convolutional networks models that employ the self-loop update approach. The over-smoothing issue is problematical because node-specific information becomes lost if more graph convolutional layers are added to the model. Due to the over-smoothing issue, basic graph convolutional network models such as GCN (Kipf & Welling, 2016) are restricted to a small number of layers, e.g., 2 to 4. Further increasing the number of layers will lead to significantly reduced performance. This is different from convolutional neural networks, the performance of which can be considerably improved by using very deep layers.

Recent years have seen increasing studies in understanding and addressing the over-smoothing problem. Li et al. (2018) showed that graph convolution is a special form of Laplacian smoothing and proved that repeatedly applying Laplacian smoothing leads to node representations becoming very similar to one another. Zhao et al. (2020) proposed

[1]School of Electrical and Data Engineering, University of Technology Sydney. Correspondence to: Min Xu <min.xu@uts.edu.au>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

PariNorm, a normalization layer that ensures the total pairwise feature distance remains to be constant across layers, preventing node features from converging to similar values. Zhang et al. (2022) showed that over-smoothing can lead to both over-fitting and under-fitting in different domain tasks and introduced to stochastically scale features and gradients during training for preventing over-smoothing.

While these efforts have been made, over-smoothing is still a key issue in graph convolutional networks. It is significant to make further advances to tackle the issue of over-smoothing. As introduced above, over-smoothing occurs when more graph convolutiional layers are added to the model. In this paper we show through an intuitive example that each layer of graph convolution can make the node representations more smoothed than the previous layer. We propose Lrn&Align, denoting learn and align, to reduce this issue in the embedding generation process. The idea of Lrn&Align is to layerwisely learn embeddings and then align the learned embeddings with those of the previous layer through reducing the angle between these embeddings. By aligning the learned embeddings, the issue of over-smoothing is explicitly reduced.

The angles between learned embeddings with those of the previous layer are randomly reduced in alinement. Therefore our Lrn&Align method does not introduce additional trainable parameters or hyper-parameters. It can be applied on current message passing graph convolutional networks without the laborious parameter tuning procedure. We show through experiments that Lrn&Align improves the generalization performance of a variety of models including GCN (Kipf & Welling, 2017), GAT (Veličković et al., 2018), GatedGCN (Bresson & Laurent, 2017), SAN (Kreuzer et al., 2021), and GPS (Rampasek et al., 2022) which combines a graph convolutional network with transformer. We also show that Lrn&Align is effective on different graph domain tasks, including graph classification and node classification, advancing the state of the art performance for learning on graph-structured data.

The main contributions of this paper can be summarized as follows: **(1)** We propose a method referred to as Lrn&Align which tackles the issue of over-smoothing through layerwisely learning and aligning node embeddings. We show that Lrn&Align is a general method that improves the performance of a variety of message passing graph convolutional networks. **(2)** Lrn&Align is parameter-free method, it can be directly applied on current graph convolutional networks without laborious parameter tuning. **(3)** Lrn&Align has a high generalization performance on different graph domain tasks, advancing the state of the art results for graph representation learning on a variety of benchmark datasets.

2. Related Work

2.1. Graph Convolutional Networks

Current graph convolutional networks can be categorized into the spectral approach and the spatial approach (Wu et al., 2020). Spectral graph convolutional networks are based on spectral graph theory, and the graph convolutions are constructed via an extension of the Fourier transform to graphs. Bruna et al. (2014) explored generalizing of convolutional networks to the graph domain and constructed convolutions based on the eigendecomposition of the graph Laplacian. Following up this work, Defferrard et al. (2016) introduced to construct convolutions based on the Chebyshev expansion of the graph Laplacian. This approach eliminates the need for the graph Laplacian decomposition and results in spatially localized filters. Kipf et al. (2017) later elaborated on the concept of graph convolutions to define graph convolutional network models by stacking very simple graph convolutional layers.

Spatial graph convolutional networks directly construct convolutions on the graph and learns embeddings for the nodes by aggregating information from a local neighbourhood. Monti et al. (2017) proposed the mixture model network, referred to as MoNet, a framework which generalizes convolutional neural network architectures to graphs and manifolds. Bresson et al. (2017) proposed residual gated graph convnets, integrating edge gates, residual connections (He et al., 2016) and batch normalization (Ioffe & Szegedy, 2015) into the graph convolutional neural network model. Velickovic et al. (2018) introduced to apply the self-attention mechanism which assign an attention weight or importance value to each neighbor in the aggregation step to graph convolutional network models. Balcilar et al. (2021) demonstrated that both spectral and spatial graph convolutional networks are essentially message passing neural networks that use a type of message passing for embedding generation.

2.2. Addressing The Over-smoothing Problem

The problem of over-smoothing occurs when the information aggregated from the local neighbours is dominating the updated node embeddings. A natural way to reduce this issue is to use feature concatenations or skip connections (Hamilton, 2020), which are used in computer vision to build deep convolutional network architectures. Feature concatenation and skip-connection can preserve information learned by previous graph convolutional layers. Inspired by the gating methods used to improve recurrent neural networks, researchers also proposed gated updates in aggregating information from local neighbours (Li et al., 2015; Bresson & Laurent, 2017). These gated updates are very effective in building deep graph convolutional network architectures, e.g., 10 or more layers. Zhao *et al.* (2020)

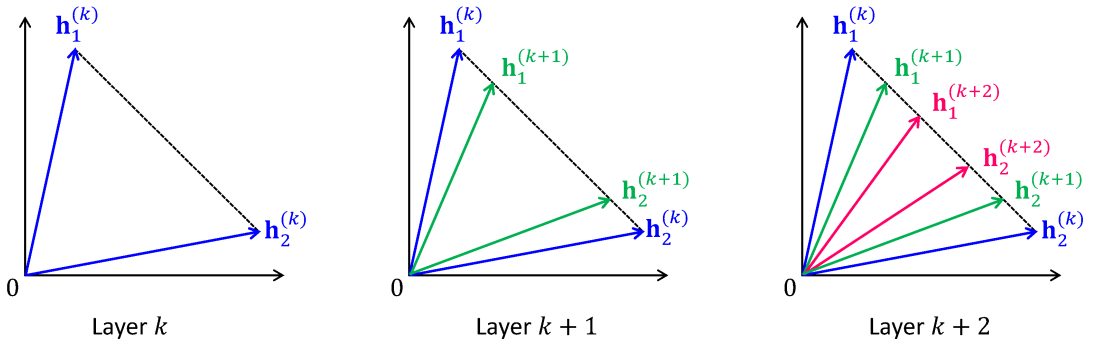

Figure 1. An illustrative example for understanding the over-smoothing issue. We consider a two node fully connected graph and use a GAT model that layerwisely learn embeddings using the equation $\mathbf{h}_i^{(k)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v^{(k-1)}$, wherein $\alpha_{u,v} > 0$ and $\sum_{v \in \mathcal{N}(u)} \alpha_{u,v} = 1$. We have simplified the model by removing the non-linearity and learnable parameter matrix. We show that the learned embeddings layerwisely become smoothed than the previous layer due to the convex combination of neighbourhood features.

proposed the PairNorm method to tackle oversmoothing by ensuring the total pairwise feature distance across layers to be constant. Zhang et al. (2022) proposed a regularization method, referred to as SSFG, that stochastically scales features and gradients in the training procedure. Empirically, SSFG can help to address both the overfitting issue and the underfitting issue for different graph convolutional network architectures. Chen et al. (2022) proposed a graph convolution operation, called graph implicit nonlinear diffusion, that can implicitly have access to infinite hops of neighbors while adaptively aggregating features with nonlinear diffusion to alleviate the over-smoothing problem.

3. Methodology

3.1. Preliminaries

Formally, a graph $G = (V, E)$ is defined by a set of nodes, or called vertices, V and and a set of edges E between these nodes. An edge going from node $u \in V$ to node $v \in V$ is denoted as (u, v). Conveniently, a graph can be represented using an adjacent matrix $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$, in which $\mathbf{A}_{u,v} = 1$ if $(u, v) \in E$ and $\mathbf{A}_{u,v} = 0$ otherwise. The degree matrix of G is a diagonal matrix and is denoted as $\mathbf{D} \in \mathbb{R}^{|V| \times |V|}$, in which $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. $\mathbf{X} \in \mathbb{R}^{|V| \times m}$ is the node-level attribute or feature information associated with the graph. The graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, and the symmetric normalized Laplacian is defined as $\mathbf{A}_{sym} = \mathbf{I}_n - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{I}_n$ is a $|V| \times |V|$ identity matrix.

The basic message passing framework utilizes a type of neural message passing, in which messages are exchanged between nodes and updated with neural networks (Gilmer et al., 2017). During each iteration of message-passing, a hidden representation $\mathbf{h}_v^{(k)}$ for each node $v \in V$ is updated according to the information aggregated from v's local neighbourhood. This message passing rule can be described as follows (Hamilton, 2020):

$$\mathbf{h}_v^{(k)} = f^{(k)}\left(\mathbf{h}_v^{(k-1)}, agg^{(k)}(\{\mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\})\right), \tag{1}$$

where f and agg are differentiable functions, i.e., neural networks, and $\mathcal{N}(v)$ is the set of v's neighbouring nodes. After running k iterations of message passing, every node embedding contains information about its k-hop neighborhood.

3.2. The Issue of Over-smoothing

The message passing rule is at the core of current graph convolutional networks and has become the dominant approach for representation learning on graphs. However, the message passing paradigm also has major bottlenecks. Studies continually find that message-passing graph convolutional networks suffer from the over-smoothing issue. Over-smoothing is core limitation in current graph convolutional networks, and this issue an be viewed as a consequence of the neighborhood aggregation operation, which is at the heart of the message passing paradigm.

The intuitive idea of over-smoothing is that after several iterations of message passing, the embeddings for all nodes in the graph converge to very similar values, washing away node-specific information. The issue of over-smoothing can be formalized through defining the influence of each node's input feature on the final layer embedding of all the other nodes in the graph. For any pair of nodes u and v we can quantify the influence of node u on node v in the graph convolutional network by examining the magnitude of the

corresponding Jacobian matrix (Xu et al., 2018):

$$I_K(u,v) = \mathbf{1}^\top \left(\frac{\partial \mathbf{h}_v^{(K)}}{\partial \mathbf{h}_u^{(0)}} \right) \mathbf{1}, \tag{2}$$

where $\mathbf{1}$ is a vector of ones. $I_K(u,v)$, which is the sum of the entries in the Jacobian matrix $\frac{\partial \mathbf{h}_v^{(K)}}{\partial \mathbf{h}_u^{(0)}}$, is a measure of how much the initial embedding of node u influences the final embedding of node v. Given the definition of influence, Xu et al. (2018) prove the following theorem:

Theorem 1. For any graph convolutional network model that uses a self-loop update approach and an aggregation function of the form below:

$$agg(\{\mathbf{h}_v, \forall v \in \mathcal{N}(u) \cup \{u\}\}) = \frac{1}{g_n(|\mathcal{N}(u) \cup \{u\}|)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \mathbf{h}_v, \tag{3}$$

where g_n a normalization function, we have the following:

$$I_K(u,v) \propto p_{G,K}(u|v), \tag{4}$$

where $p_{G,K}(u|v)$ denotes the probability of visiting node v on a length of K random walk starting from node u.

This theorem states that when we are using a K-layer graph convolutional network, the influence of node u and node v is proportional to the probability of reaching node v on a K-step random walk starting from node u. The consequence of this is that as $K \to \infty$ the influence of every node approaches the stationary distribution of random walks over the graph, therefore the information from local neighborhood is lost. Theorem 1 applies directly to models which use a self-loop update approach, but the result can also be extended in asymptotic sense for the basic message passing update in Equation 1.

3.3. Proposed Method: Lrn&Align

Over-smoothing is problematic because it makes the graph convolutional network models can only capture limited dependencies of the graph. As discussed above, the issue of over-smoothing occurs when we are using more layers of message passing, the learned node embeddings become over-smoothed and uninformative and we lose information of the input node features.

Theorem 1 provides theoretical analysis of the issue of over-smoothing. Intuitively, we can expect the learned embeddings for the nodes become smoothed layerwisely or asymptotically layerwisely. We show through an example that the embeddings become smoothed layervisely. As shown in Figure 1, consider we have a two node fully connected graph and use a GAT model (Veličković et al., 2018) to learn node embeddings. The GAT model updates the the embedding $\mathbf{h}_u^{(k)}$ according to a weighted sum of information from the neighbours as follows:

$$\mathbf{h}_i^{(k)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{W}^{(k)} \mathbf{h}_v^{(k-1)}. \tag{5}$$

We have simplified the model by removing the non-linearity. In the above Equation, $\alpha_{u,v}$ denotes the attention weight on neighbor $v \in \mathcal{N}(u)$ when aggregating information at node u, and $\mathbf{W}^{(k)}$ is a learnable parameter matrix. The weight $\alpha_{u,v}$ is defined using the softmax function as:

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top \left[\mathbf{W}^{(k)}\mathbf{h}_u \parallel \mathbf{W}^{(k)}\mathbf{h}_v\right]\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{a}^\top \left[\mathbf{W}^{(k)}\mathbf{h}_u \parallel \mathbf{W}^{(k)}\mathbf{h}_{v'}\right]\right)}, \tag{6}$$

where $\mathbf{a}$ is learnable vector, and $\parallel$ denotes the concatenation operation. With the softmax function, the attention weights are normalized to 1, i.e., $\sum_v \alpha_{u,v} = 1$. Therefore, the generated node embedding is a convex combination of the information from the neighbours. As shown in Figure 1, $\mathbf{h}_0^{(k+1)}$ and $\mathbf{h}_1^{(k+1)}$ are on the dash line between $\mathbf{h}_0^{(k)}$ and $\mathbf{h}_1^{(k)}$. Thus, each layer of the graph convolution makes the generated embeddings smoothed compared to the previous layer. As more layers are added the learned embeddings become over-smoothed and the information about neighbourhood become lost.

When the embeddings become smoothed, the average angle between pairs of the embeddings is reduced compared

Algorithm 1 The embedding generation process with our Lrn&Align method.

Input: Graph $G = (V, E)$; number of graph convolutional layers K; input node features $\{\mathbf{x}_v, \forall v \in V\}$
Output: Node embeddings $\mathbf{h}_v^{(K)}$ for all $v \in V$

$\mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v, \forall v \in V$
for $k = 1, ..., K$ **do**
  for $v \in \mathcal{V}$ **do**
    $\overline{\mathbf{h}}_v^{(k)} = f^{(k)}\left(\mathbf{h}_v^{(k-1)}, agg^{(k)}(\{\mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\})\right)$
    // Learn node embeddings using a graph convolution such as GCN and GAT.
  end for
  for $v \in \mathcal{V}$ **do**
    if model.training == True **then**
      $\lambda \sim U(0,1)$
    else
      $\lambda = 0.5$
    end if
    $\mathbf{h}_v^{(k)} = \lambda \cdot \frac{\mathbf{h}_v^{(k-1)}}{\|\mathbf{h}_v^{(k-1)}\|} \cdot \|\overline{\mathbf{h}}_v^{(k)}\| + (1-\lambda) \cdot \overline{\mathbf{h}}_v^{(k)}$ // Align the learned $\overline{\mathbf{h}}_v^{(k)}$ with the input $\overline{\mathbf{h}}_v^{(k-1)}$.
  end for
end for

to that of the previous layer. As demonstrated in Figure 1, the angle between $\mathbf{h}_1^{(k+1)}$ and $\mathbf{h}_2^{(k+1)}$ is reduced compared to the angle between $\mathbf{h}_1^{(k)}$ and $\mathbf{h}_2^{(k)}$, and the angle between $\mathbf{h}_1^{(k+2)}$ and $\mathbf{h}_2^{(k+2)}$ is reduced compared to the angle between $\mathbf{h}_1^{(k+1)}$ and $\mathbf{h}_2^{(k+1)}$. Based on the above analysis, we propose Lrn&Align that layerwisely aligns the learned embeddings with those of the previous layer to reduce the over-smoothing problem. In each layer, we first apply a general graph convolution to learn an intermediate embedding for each node:

$$\overline{\mathbf{h}}_v^{(k)} = f^{(k)}\left(\mathbf{h}_v^{(k-1)}, agg^{(k)}(\{\mathbf{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\})\right). \tag{7}$$

Then we align $\overline{\mathbf{h}}_v^{(k)}$ with $\mathbf{h}_v^{(k-1)}$ through reducing the angle between the two embeddings. We adopt a random alignment approach. Specifically, we first rescale $\mathbf{h}_v^{(k-1)}$ to have the same norm as $\overline{\mathbf{h}}_v^{(k)}$, then we apply a random interpolation between the two embedding, obtaining the following aligned embedding:

$$\mathbf{h}_v^{(k)} = \lambda \frac{\mathbf{h}_v^{(k-1)}}{\|\mathbf{h}_v^{(k-1)}\|}\|\overline{\mathbf{h}}_v^{(k)}\| + (1-\lambda)\overline{\mathbf{h}}_v^{(k)}, \tag{8}$$

where $\lambda \sim U(0,1)$ is sampled from the standard uniform distribution. By this way, we can keep the representational ability of $\overline{\mathbf{h}}_v^{(k)}$ in the aligned embedding while reducing the angle between the two embeddings. Because the expected value of λ is 0.5, i.e., $E[\lambda] = 0.5$, at test time λ is set to a fixed value of 0.5. Algorithm 1 shows the embedding generation algorithm with our Lrn&Align method.

Our Lrn&Align method is straightforward to understand. By aligning the learned embeddings with those of the previous layer, the over-smoothness of these learned embeddings is explicitly reduced. Because the embeddings before alignment are learned by the basic message-passing framework, our Lrn&Align is a general method that can be applied to different message passing graph convolutional networks to alleviate the over-smoothing problem. Our Lrn&Align does not introduce additional hyper-parameters or trainable weights, it can be directly applied without the laborious parameter tuning procedure.

4. Experiments

4.1. Datasets and Setup

We evaluate our Lrn&Align on four graph domain tasks: graph classification, node classification, multi-label graph classification and binary graph classification. The experiments are conducted on six benchmark datasets, which are briefly as follows.

- **MNIST and CIFAR10** (Dwivedi et al., 2020) are two datasets used for superpixel graph classification. The original images in MNIST (LeCun et al., 1998) and CIFAR10 (Krizhevsky et al., 2009) are converted to superpixel graphs using the SLIC technique (Achanta et al., 2012). Each superpixel represents a small region of homogeneous intensity in the original image.

- **PATTERN and CLUSTER** (Dwivedi et al., 2020). The two datasets are used for inductive node classification. The graphs in the two datasets are generated using the stochastic block model (Abbe, 2017). PATTERN is used for evaluating the model for recognizing specific predetermined subgraphs, and CLUSTER is used for identifying community clusters in the semi-supervised setting.

- **Peptides-Func** (Dwivedi et al., 2022) is a dataset of peptides molecular graphs. The nodes correspond to heavy (non-hydrogen) atoms of the peptides, and the edges represent the bonds between these atoms. The graphs are categorized into 10 classes based on the peptide function, e.g., antibacterial, antiviral, cell-cell communication. This dataset is used for evaluating the model multi-label graph classification.

- **OGBG-Molhiv** is a molecule graph dataset introduced in the open graph benchmark (OGB) (Hu et al., 2020). The nodes and edges in the graphs represent atoms and the chemical bonds between these atoms. The task on this dataset is binary class classification. That is we evaluate the model's ability to predict if the molecule can inhibit HIV virus replication.

The details of the six datasets, including the dataset size and split, can be found in Table 7 in the appendix section.

Following Dwivedi et al. (2020) and Rampasek et al. (2022), the following metrics are used for different tasks. For superpixel graph classification, we report the classification accuracy on test set. For the node classification task, the performance is measured by the weighted accuracy. For multi-label graph classification on Peptides-Func, the performance is measured by average precision (AP) across the categories. For the binary classification task on OGBG-Molhiv, the performance is measured by the area under the receiver operating characteristic curve (ROC-AUC). We closely follow the experimental setup as Dwivedi et al. (2020) and Rampasek et al. (2022) for training our model. We use the same train/validation/test split of each dataset and report the mean and standard deviation.

Table 1. Results for superpixel graph classification on MNIST and CIFAR10. We show that our Lrn&Align consistently improves the performance of the base graph convolutional network models. Residual connection and batch normalization, which are simple strategies that can help to alleviate over-smoothing, are applied to the GCN and GAT base models.

Model	MNIST				
	Mode	4 layers	8 layers	12 layers	16 layers
GCN	Training	97.196±0.223	99.211±0.421	99.862±0.043	99.697±0.029
GCN + Lrn&Align		88.311±0.262	92.450±0.170	94.283±0.192	95.505±0.154
GCN	Test	90.705±0.218	90.847±0.078	91.263±0.216	91.147±0.185
GCN + Lrn&Align		90.305±0.140	92.688±0.046	93.470±0.035	94.051±0.052
GAT	Training	99.994±0.008	100.00±0.000	100.00±0.000	100.00±0.000
GAT + Lrn&Align		96.853±0.236	98.492±0.294	99.146±0.104	99.189±0.158
GAT	Test	95.535±0.205	96.065±0.093	96.288±0.049	96.526±0.041
GAT + Lrn&Align		96.513±0.075	97.250±0.049	97.505±0.029	97.553±0.034
GatedGCN	Training	100.00±0.000	100.00±0.000	100.00±0.000	100.00±0.000
GatedGCN + Lrn&Align		99.713±0.094	99.933±0.048	99.849±0.020	99.813±0.023
GatedGCN	Test	97.340±0.143	97.950±0.023	98.108±0.021	98.132±0.022
GatedGCN + Lrn&Align		98.120±0.076	98.463±0.079	98.494±0.054	98.552±0.023

Model	CIFAR10				
	Mode	4 layers	8 layers	12 layers	16 layers
GCN	Training	69.523±1.948	77.546±0.813	81.073±1.224	84.279±0.656
GCN + Lrn&Align		59.798±0.324	65.405±0.603	66.711±0.338	70.919±0.522
GCN	Test	55.710±0.381	54.242±0.454	53.867±0.090	53.353±0.184
GCN + Lrn&Align		55.275±0.165	57.145±0.202	57.603±0.157l	57.736±0.162
GAT	Training	89.114±0.499	99.561±0.064	99.972±0.005	99.980±0.003
GAT + Lrn&Align		74.522±1.179	81.071±0.596	81.511±0.464	79.962±0.142l
GAT	Test	64.223±0.455	64.452±0.303	64.423±0.121	64.340±0.146
GAT + Lrn&Align		65.385±0.074	69.158±0.438	69.707±0.350	69.920±0.082
GatedGCN	Training	94.553±1.018	99.983±0.006	99.995±0.003	99.995±0.004
GatedGCN + Lrn&Align		77.784±0.799	83.552±0.570	86.779±0.520	90.903±0.785
GatedGCN	Test	67.312±0.311	69.808±0.421	68.417±0.262	70.007±0.165
GatedGCN + Lrn&Align		72.075±0.154	75.015±0.177	76.135±0.248	76.395±0.186

4.2. Experimental Results

4.2.1. Superpixel Graph Classification on MNIST and CIFAR10

The results for superpixel graph classification on MNIST and CIFAR10 are reported in Table 1. We experiment with three different base models: GCN, GAT and GatedGCN. We have also applied residual connection and batch normalization to the base models of GCN and GAT. Residual connection and batch normalization are simple strategies that can help reduce the over-smoothing issue and improve the numerical stability in optimization. GatedGCN employs the gated update approach for aggregating information from neighbors and also integrates residual connection and batch normalization. We see that the base models only slightly improve the performance or see a reduced performance as the number of layer increases from 4 to 16. Without residual connection and batch normalization, the performance would drop considerably with increased layers due to over-smoothing. By applying our Lrn&Align method, the performance of the base models consistently improves as the number of layers increases. Lrn&Align on GatedGCN with 16 layers yields a 6.388% performance improvement on CIFAR10, which is a 9.13% relative improvement. We also see that for the 4 layer GCN model, applying Lrn&Align could not improves the performance on the two datasets. This is because the model dose not suffer the over-smoothing issue at this layer.

The base models suffer serious over-fitting problem on the two datasets. For example, the GAT and GatedGCN with 8 or more layers archive nearly 100% accuracy on CIFAR10, but their test accuracies are all below 70.007%. By using our Lrn&Align method, we see that the training accuracy reduces while the task performance improves. This shows that through tackling the over-smoothing issue with Lrn&Align, the over-fitting problem is significantly reduced, and therefore the model performance improves. Figure 2 demonstrates the learning curves of the three based models with 16 layers on CIFAR10.

Table 2 compares the performance of our results with the recent work on MNIST and CIFAR10. EGT (Hussain et al., 2022), which integrates an additional edge channels into

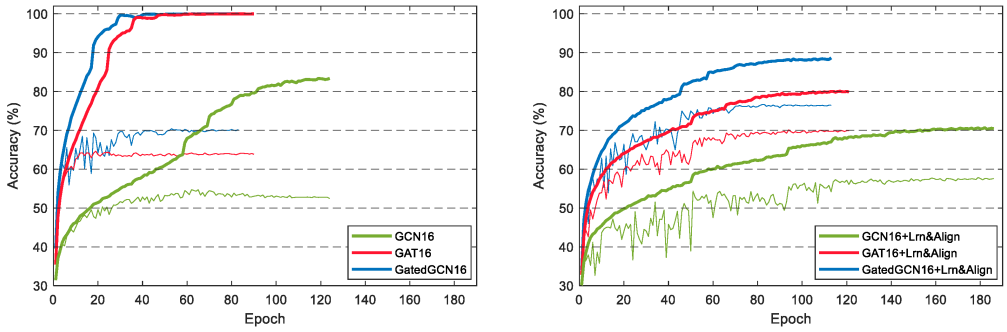

Figure 2. Learning curves on CIFAR10. Bold lines are training curves and thin lines are test curves. We show that our Lrn&Align method improves the generalization performance by reducing the issue of over-smoothing.

Table 2. Comparison with previous work on MNIST and CIFAR10 on the superpixel graph classification task.

Model	MNIST	CIFAR10
GCN	90.705±0.218	55.710±0.381
MoNet	90.805±0.032	54.655±0.518
GraphSAGE	97.312±0.097	65.767±0.308
GIN	96.485±0.252	55.255±1.527
PNA	97.94±0.12	70.35±0.63
DGN	–	72.838±0.417
CRaWl	97.944±0.050	69.013±0.259
GIN-AK+	–	72.19±0.13
EGT	98.173±0.087	68.702±0.409
GPS	98.051±0.126	72.298±0.356
GatedGCN+SSFG	97.985±0.032	71.938±0.190
GAT-16	95.535±0.205	64.223±0.455
GAT-16 + Lrn&Align	**97.553±0.034**	**69.920±0.082**
GatedGCN-16	97.340±0.143	67.312±0.311
GatedGCN-16 + Lrn&Align	**98.512±0.033**	**76.395±0.186**

Table 3. Experimental results PATTERN and CLUSTER on the node classification task.

Model	PATTERN	CLUSTER
GCN	71.892±0.334	68.498±0.976
GraphSAGE	50.492±0.001	63.844±0.110
GIN	85.387±0.136	64.716±1.553
GAT	78.271±0.186	70.587±0.447
MoNet	85.582±0.038	66.407±0.540
GatedGCN	85.568±0.088	73.840±0.326
K-Subgraph SAT	86.848±0.037	77.856±0.104
GatedGCN+SSFG	85.723±0.069	75.960±0.020
SAN	86.581±0.037	76.691±0.650
SAN + Lrn&Align	**86.770±0.067**	**77.847±0.073**
GPS	86.685±0.059	78.016±0.180
GPS + Lrn&Align	**86.858±0.010**	**78.592±0.052**

the Transformer model and uses the global self-attention to generate embeddings, achieves 98.173% accuracy on MNIST, which is the best among the previous models. Our model achieves 0.337% improved performance compared with EGT (Hussain et al., 2022). On CIFAR10, our model outperforms the previous best DGN (Beaini et al., 2021) by 3.557%, which is a large improvement. Our method also outperforms SSFG, which stochastically scale features and gradients for regularization graph network models but involve a laborious parameter tuning process. To the best of our knowledge, our model achieves the state of the art performance on the two datasets.

4.2.2. Node Classification on PATTERN and CLUSTER

Table 3 reports our results on PATTERN and CLUSTER for node classification. We experiment with two state of the art base architectures: SAN and GPS. SAN utilizes an invariant aggregation of Laplacian's eigenvectors for position encoding and also utilizes conditional attention for the real and virtual edges to improve the performance. A GPS layer is a hybrid layer that integrates a message passing graph convolutional layer and a Transformer layer. Again, our Lrn&Align improves the performance of the two base model and advances the state of the results on the two datasets. It improves the performance by 1.156% on SAN and 0.576% on GPS on CLUSTER. The model of GPS with our Lrn&Align outperforms all the baseline models on the two datasets. Our model achieves considerably improved performance compered with GCN, GAT and GraphSAGE.

4.2.3. Peptides-struct

Table 4 reports the results on Peptides-struct. This dataset was introduced to evaluate a model's ability to capture long-range dependencies in the graph. We experiment with GPS as the base model. As introduced above, a GPS layer integrates a Transformer layer with the message pass-

Table 4. Experimental results on Peptides-struct on the multi-label graph classification task.

Model	AP (↑)
GCN	0.5930±0.0023
GINE (Hu et al., 2019)	0.5498±0.0079
GatedGCN	0.5864±0.0077
GatedGCN+RWSE	0.6069±0.0035
Transformer + LapPE	0.6326±0.0126
SAN + LapPE	0.6384±0.0121
SAN + RWSE	0.6439±0.0075
GPS	0.6535±0.0041
GPS + Lrn&Align	**0.6630±0.0005**

Table 5. Experimental results on OGBG-molhiv on binary graph classification. The models are all trained from scratch.

Model	ROC-AUC (↑)
GCN	0.7599±0.0119
GIN (Xu et al., 2019)	0.7707±0.0149
PNA (Corso et al., 2020)	0.7905±0.0132
DeeperGCN (Li et al., 2020)	0.7858±0.0117
DGN (Beaini et al., 2021)	0.7970±0.0097
ExpC (Yang et al., 2022)	0.7799±0.0082
GIN-AK+ (Zhao et al., 2022)	0.7961±0.0119
SAN	0.7785±0.2470
GPS	0.7880±0.0101
GPS + Lrn&Align	**0.8021±0.0305**

ing graph convolutional network framework to capture the global independencies. Our Lrn&Align improves the average precision of GPS from 0.6535 to 0.6630, outperforming the baseline models including GatedGCN, Transformer (Vaswani et al., 2017) and SAN. The results show that Lrn&Align is also effective in improving the model performance for capturing long-range dependencies.

4.2.4. OGBG-MOLHIV

Table 5 reports the results on OGBG-molhiv. As with Rampasek et al. (2022) we only compare with models that are trained from scratch. We experiment using GPS as the base model. Our Lrn&Align improves the ROC-AUC of GPS from 0.6535 to 0.6630, which is a relative 1.45% improvement, outperforming all the baseline models.

We have shown that Lrn&Align is a general method for preventing the over-smoothing issue. It improves the task performance of different graph convolutional network models and on different tasks. We also see from the experimental results that Lrn&Align yields a small standard deviation for most settings compared with the base models. This suggests that Lrn&Align is also helpful for improving the stability of optimization.

Table 6. Importance of scaling embeddings of the previous layer in alignment.

Model	MNIST	CIFAR10
GAT-8		
w/o Lrn&Align	96.065±0.093	64.452±0.303
Lrn&Align w/o scaling	**96.977±0.021**	**66.212±0.182**
Lrn&Align + scaling	97.250±0.049	69.158±0.438
GatedGCN-8		
w/o Lrn&Align	97.950±0.023	69.808±0.421
Lrn&Align w/o scaling	**98.247±0.018**	**74.437±0.150**
Lrn&Align + scaling	98.463±0.079	75.015±0.177

4.2.5. ANALYSIS OF SCALING $\mathbf{h}_v^{(k-1)}$ IN ALIGNMENT

In Lrn&Align, we first scale $\mathbf{h}_v^{(k-1)}$ to $\frac{\mathbf{h}_v^{(k-1)}}{\|\mathbf{h}_v^{(k-1)}\|}\|\overline{\mathbf{h}}_v^{(k)}\|$ and then apply a random interpolation between the two features in aligning $\overline{\mathbf{h}}_v^{(k)}$ (see Equation 8). To show the importance of scaling, we conduct an experiment to compare the performance of our approach with that without scaling. We perform experiments using GAT-8 and GatedGCN-8 as base models on CIFAR10 and MNIST, and Table 6 reports the comparison results. We see that applying scaling improves the performance by. By scaling $\mathbf{h}_v^{(k-1)}$ to have the same norm as $\overline{\mathbf{h}}_v^{(k)}$, more information about $\overline{\mathbf{h}}_v^{(k)}$ is contained in the aligned representation, therefore this helps to improve the performance.

5. Conclusions

In this paper, we proposed Lrn&Align for reducing the over-smoothing problem in graph convolutional networks. The basic idea of Lrn&Align is to layerwisely generate embeddings and then align the generated embeddings with those of the previous layer. Our method is motivated by the intuition that learned embeddings for the nodes become smoothed layerwisely or asymptotically layerwisely. In our Lrn&Align, a random interpolation method is utilized for feature alignment. By aligning generated embeddings with those of the previous layer, the smoothness of these embeddings is reduced. Moreover, our Lrn&Align is a parameter-free method, and it can be directly applied current graph convolutional networks without hyper-parameter tuning or introducing additional trainable parameters. We experimentally evaluated Lrn&Align on six popular benchmark datasets on four graph domain tasks including graph classification, node classification, multi-label graph classification and binary graph classification. We presented extensive results demonstrating Lrn&Align is a general method that improves the performance for a variety of graph convolutional network models and advances the state of the art results for graph representation learning.

References

Abbe, E. Community detection and stochastic block models: recent developments. *The Journal of Machine Learning Research*, 18(1):6446–6531, 2017.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., and Süsstrunk, S. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

Balcilar, M., Guillaume, R., Héroux, P., Gaüzère, B., Adam, S., and Honeine, P. Analyzing the expressive power of graph neural networks in a spectral perspective. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

Beaini, D., Passaro, S., Létourneau, V., Hamilton, W., Corso, G., and Liò, P. Directional graph networks. In *International Conference on Machine Learning*, pp. 748–758. PMLR, 2021.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations*, 2014.

Chen, Q., Wang, Y., Wang, Y., Yang, J., and Lin, Z. Optimization-induced graph implicit nonlinear diffusion. In *International Conference on Machine Learning*, pp. 3648–3661. PMLR, 2022.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33: 13260–13271, 2020.

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

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Dwivedi, V. P., Rampasek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Hamilton, W. L. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33: 22118–22133, 2020.

Hussain, M. S., Zaki, M. J., and Subramanian, D. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 655–665, 2022.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

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

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR2017)*, 2017.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. *AAAI Conference on Artificial Intelligence*, 2018.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *International Conference on Learning Representations*, 2015.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5115–5124, 2017.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 4602–4609, 2019.

Rampasek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? 2019.

Yang, M., Wang, R., Shen, Y., Qi, H., and Yin, B. Breaking the expression bottleneck of graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

Zhang, H., Xu, M., Zhang, G., and Niwa, K. Ssfg: Stochastically scaling features and gradients for regularizing graph convolutional networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2020.

Zhao, L., Jin, W., Akoglu, L., and Shah, N. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=Mspk_WYKoEH.

A. Details of the Benchmark Datasets Used in Our Experiments.

The details of the six benchmark datasets used in the experiments are shown in the table below. We used the same train/validation/test split of each dataset as the previous work (Dwivedi et al., 2020; Rampasek et al., 2022).

Table 7. Details of the six benchmark datasets used in the experiments.

Dataset	Graphs	Nodes/graph	#Training	#Validation	#Test	Categories	Task
PATTERN	14K	44-188	10,000	2000	2000	2	Node classification
CLASTER	12K	41-190	10,000	1000	1000	6	
MNIST	70K	40-75	55,000	5000	10,000	10	Graph classification
CIFAR10	60K	85-150	45,000	5000	10,000	10	
Peptides-Func	15,535	150.9	70%	15%	15%	10	Multi-label graph classification
OGBG-Molhiv	41,127	25.5	80%	10%	10%	2	Binary graph classification