SSFG: Stochastically Scaling Features and Gradients for Regularizing Graph Convolutional Networks

Haimin Zhang®, Min Xu®, Member, IEEE, Guoqiang Zhang®, Member, IEEE, and Kenta Niwa®, Senior Member, IEEE

Abstract—Graph convolutional networks (GCNs) have been successfully applied in various graph-based tasks. In a typical graph convolutional layer, node features are updated by aggregating neighborhood information. Repeatedly applying graph convolutions can cause the oversmoothing issue, i.e., node features at deep layers converge to similar values. Previous studies have suggested that oversmoothing is one of the major issues that restrict the performance of GCNs. In this article, we propose a stochastic regularization method to tackle the oversmoothing problem. In the proposed method, we stochastically scale features and gradients (SSFG) by a factor sampled from a probability distribution in the training procedure. By explicitly applying a scaling factor to break feature convergence, the oversmoothing issue is alleviated. We show that applying stochastic scaling at the gradient level is complementary to that applied at the feature level to improve the overall performance. Our method does not increase the number of trainable parameters. When used together with ReLU, our SSFG can be seen as a stochastic ReLU activation function. We experimentally validate our SSFG regularization method on three commonly used types of graph networks. Extensive experimental results on seven benchmark datasets for four graph-based tasks demonstrate that our SSFG regularization is effective in improving the overall performance of the baseline graph networks. The code is available at https://github.com/vailatuts/SSFG-regularization.

Index Terms—Graph convolutional networks (GCNs), oversmoothing issue, stochastic regularization.

I. INTRODUCTION

DATA are organized in graph structures in various domains. Social networks, citation networks, molecular structures, and protein–protein interactions, can be modeled using graphs. Developing powerful graph learning algorithms is important for many real-world applications, such as recommendation systems [1], link prediction [2], knowledge graphs [3], community-based question answering [4], and drug discovery [5]. Motivated by the success of deep convolutional networks, recent years have seen considerable interest in generalizing deep learning techniques to the graph domain.

Compared with images and sequence data, graphs have a much complex topographical structure. The nodes in a graph can have a very different number of neighbors, and there is no fixed node ordering for a graph. Early methods for learning on graphs are primarily based on recurrent neural networks. These methods involve a process that iteratively propagates node features until the node features reach a stable point. In recent years, graph convolutional networks (GCNs) that leverage graph convolutions have become the dominant approach for graph learning. Graph convolutions update node features by aggregating neighborhood information. Compared with recurrent-based methods, GCNs are much efficient in learning on graph-structured data.

While recently significant progress has been made in graph networks, overfitting and underfitting are major issues that restrict the performance of graph networks. Overfitting is more commonly seen in graph learning tasks, resulting in the model not generalizing well on unseen samples. This issue usually comes with overparameterization. An illustration of this phenomenon is shown in Fig. 1; we see that the training accuracies for superpixel graph classification on CIFAR10 and MNIST [6]; the results are obtained using a four-layer GatedGCN.

Fig. 1. Illustration that shows the overfitting issue with graph networks for superpixel graph classification on CIFAR10 and MNIST [6]; the results are obtained using a four-layer GatedGCN.
Underfitting can be caused by issues such as vanishing and exploding gradients. Li et al. [7] identified the oversmoothing issue in training graph networks. The oversmoothing issue comes with repeatedly applying graph convolutions, resulting in node features across different classes converging to similar values at deeper layers. As a result, state-of-the-art graph networks usually adopt a small number of layers [8], e.g., 2–4. Further increasing the number of layers will lead to reduced performance.

Regularization has been commonly used to improve the generalization performance of neural networks. For graph networks, however, commonly used regularization techniques, such as L2 regularization and dropout [9], can only slightly improve the generalization performance [10]. Rong et al. [11] proposed the DropEdge method as a variant of dropout for regularizing graph networks. DropEdge randomly removes a number of edges from the input graph at each training epoch. It works as a data augmentation method and a message-passing reducer. This method is more effective to improve performance than a dropout. Recently, increasing research attention has been focused on the oversmoothing issue [12]–[15]. Zhao and Akoglu [12] proposed 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. Feng et al. [13] proposed random propagation as a data augmentation method to mitigate oversmoothing. This method enables the model to perform high-order feature propagation, reducing the risk of node features becoming oversmooth.

While increasing research efforts have been devoted to this issue, the implication of the oversmoothing issue, i.e., whether it leads to overfitting or underfitting, still remains unknown. In this article, we propose a stochastic regularization method to address the oversmoothing issue. In our method, we stochastically scale features and gradients (SSFG) in the training procedure. Our idea is to explicitly apply stochastic scaling factors to node features to mitigate the feature convergence issue. The factors are sampled from a distribution transformed from the beta distribution. Applying stochastic scaling to gradients in backward propagation further increases randomness; this is complementary to that applied in forward propagation to improve the overall performance. We show that, by tackling oversmoothing with our SSFG regularization method, both the overfitting issue and the underfitting issue can be reduced.

Our SSFG regularization method can be seen as a variant of the dropout method. Unlike dropout, we preserve all neurons and stochastically dropout or add back a portion of the node feature in forward propagation. Our method can also be seen as a stochastic rectified linear unit (ReLU) function [16] when used together with ReLU. It generalizes the standard ReLU by using stochastic slopes in forward and backward propagations. We validate our SSFG regularization method on three commonly used GCNs, i.e., GraphSAGE [17], graph attention networks (GATs) [10], and gated graph convnets (GatedGCNs) [18], and conduct experiments on seven benchmark datasets for four graph-based tasks, i.e., graph classification, node classification, link prediction, and graph regression. Extensive experimental results demonstrate that our SSFG regularization is effective in improving the overall performance of the three baseline graph networks.

The contributions of this article can be summarized as follows.

1) We propose a stochastic regularization method for GCNs. In our method, we SSFG in the training procedure. As far as we know, this is the first research on regularizing graph networks at both the feature level and the gradient level. Our SSFG regularization does not require additional trainable parameters. We show that both the overfitting issue and the underfitting issue can be addressed by using the proposed regularization method.

2) We experimentally evaluate our SSFG regularization on three types of commonly used graph networks, i.e., GraphSAGE, GAT, and GatedGCN. Extensive experimental results on five benchmark datasets for four graph-based tasks demonstrate that our regularization effectively improves the overall performance of the three baseline graph networks.

II. RELATED WORK

A. Graph Convolutional Networks

GCNs have become the dominant approach for learning on graph-structured data. Existing studies on GCNs can be categorized into two approaches: the spectral-based approach and the spatial-based approach [19]. The spectral-based approach works with the spectral representation of graphs, while the spatial-based approach directly defines convolutions on graph nodes that are spatially close.

Burana et al. [20] proposed the first spectral-based graph network, in which convolutions are defined in the Fourier domain on the eigen-decomposition of the graph Laplacian. Defferrard et al. [21] later proposed Chebyshev spectral networks (ChebNets) to address the limitation of non-spatially localized filters in Burana’s work. ChebNets approximate the filters using Chebyshev expansion of the graph Laplacian, resulting in spatially localized filters. Kips and Welling [22] further introduced an efficient layerwise propagation rule based on the first-order approximation of spectral convolutions. In spectral-based graph networks, the learned filters depend on the graph structure; Therefore, a model trained on a specific graph cannot be applied to other graph structures.

Unlike the spectral-based approach, the spatial-based approach defines convolutions in the spatial domain and updates node features by aggregating neighborhood information. To deal with variable-sized neighbors, several sampling-based methods have been proposed for efficient graph learning. These methods include the nodewise sampling-based method [17], the layerwise sampling-based method [23], and the adaptive layerwise sampling-based method [24]. Velčković et al. [10] proposed GATs, in which the self-attention mechanism is used to compute attention weights in feature aggregation. Zhang et al. [25] further proposed gated attention networks that apply self-attention
to the outputs of multiattention heads to improve the performance. Bresson and Laurent [18] proposed residual GatedGCNs, integrating edge gates, residual learning [26], and batch normalization [27] into graph networks.

The oversmoothing problem occurs in both spectral- and spatial-based graph networks. Li et al. [7] showed that graph convolution is a special form of Laplacian smoothing and proved that repeatedly applying Laplacian smoothing leads to node features converging to similar values. Min et al. [28] proposed to augment conventional GCNs with geometric scattering transforms, which enables bandpass filtering of graph signals to reduce oversmoothing. More recently, Zhao and Akoglu [12] proposed the PairNorm method to tackle oversmoothing by ensuring the total pairwise feature distance across layers to be constant. Chen et al. [29] introduced to add a MADGap-based regularizer and use adaptive edge optimization to address oversmoothing.

B. Regularization Methods

Regularization methods have been widely used to improve the generalization performance of neural networks. Convolutional regularization methods include early stopping, i.e., terminating the training procedure when the performance on a validation set stops to improve, Lasso regularization, weight decay, and soft weight sharing [30]. Srivastava et al. [9] introduced dropout as a stochastic regularization method to prevent overfitting in neural networks. The key idea of dropout is to randomly dropout neurons from the neural network in the training procedure. Dropout can be seen as perturbing the feature outputted by a layer by setting the randomly selected feature points to zero. The idea behind dropout has also been adopted in graph networks. For example, in GraphSAGE [17], a fix-sized number of neighbors are sampled for each node in feature aggregation. This facilitates fast training and is also helpful to improve the overall performance. The method of node sampling in GraphSAGE can be seen as using random subgraphs of the original graph for training. Rong et al. [11] proposed the DropEdge method that randomly drops out edges in each training epoch. This method works as a data augmentor and also a message-passing reducer. It helps to reduce the convergence speed of oversmoothing. Feng et al. [13] recently proposed graph random networks that use a random propagation strategy to perform the high-order feature propagation to prevent oversmoothing.

III. METHODOLOGY

In this section, we first introduce the notations and the oversmoothing issue in graph networks. Then, we present the proposed SSFG regularization method for tackling the oversmoothing issue, demonstrating its relationship to dropout and ReLU. Finally, we introduce the use of our SSFG regularization on three types of commonly used graph networks.

A. Notations

Let a graph with \(N\) nodes be denoted \(G = (V, E, X)\), where \(V = \{v_1, v_2, \ldots, v_N\}\) is the node set, \(E \subseteq V \times V\) is the edge set, and \(X \in \mathbb{R}^{N \times C}\) is the feature matrix associated with the nodes. \(D = \text{diag}(\text{deg}_1, \ldots, \text{deg}_N) \in \mathbb{R}^{N \times N}\) is the degree matrix, and \(A \in \mathbb{R}^{N \times N}\), where \(A_{ij}\) equals to 1 if \(v_i\) is connected to \(v_j\) or 0 otherwise, is the adjacent matrix of \(G\). We consider the nodes of \(G\) to be self-connected, and then, the structure of \(G\) can be represented as \(\hat{A} = A + I\), where \(I\) is the identity matrix. \(\hat{D} = \hat{D} + I\) is the augmented degree matrix, and \(\hat{A}_{\text{sym}} = \hat{D}^{-1/2} \hat{A} \hat{D}^{-1/2}\) is the symmetrically normalized adjacency matrix of \(\hat{A}\).

B. Oversmoothing Issue

While graph networks have achieved state-of-the-art performance for various graph-based tasks, these models are mostly restricted to shallow layers, e.g., 2–4. Oversmoothing is one of the major issues that restrict the depth of graph networks. Oversmoothing comes with the nature of graph

Fig. 2. Illustration of the SSFG regularization method and its comparison to dropout. Unlike dropout, all neurons are preserved in our SSFG method. The input feature is stochastically scaled in forward propagation, and the gradient of the output feature is also stochastically scaled in backward propagation. The scaling factors are sampled from a distribution transformed from the beta function.
convolutions that update node features by aggregating neighborhood information. Repeatedly applying graph convolutions results in node features across different categories converging to similar values regardless of input features, eliminating the discriminative information from these features.

From the view of Zhao and Akoglu [12], oversmoothing can be understood as follows. Let $X_j \in \mathbb{R}^n$ denote the $j$th column of $X$, i.e., the input feature of the $j$th node; we can obtain the following:

$$\lim_{k \to +\infty} \tilde{A}^k_{\text{sym}} \cdot X_j = \pi_j, \quad j = 1, 2, \ldots, N. \tag{1}$$

The normalized representation $\pi = (\pi_j / \| \pi_j \|_1)$ satisfies $\pi_i = ((\text{deg}_i)^{1/2} / \sum (\text{deg}_i)^{1/2})$. Note that $\pi$ is only a function of the graph structure, regardless of the input feature. Li et al. [8] point out that oversmoothing leads to the vanishing gradient problem, making graph networks difficult to optimize.

While the oversmoothing issue has been widely discussed, the implication of this issue, i.e., whether it leads to overfitting or underfitting, still remains unsolved. The studies of Rong et al. [11], Zhao and Akoglu [12], and Yang et al. [15] regard overfitting and oversmoothing as separate issues. Yang et al. [15] further suggest that overfitting is a major issue that affects graph networks compared to oversmoothing.

In this work, we show that, by tackling the oversmoothing issue with our SSFG regularization, both the underfitting issue and the overfitting issue can be alleviated. This indicates that oversmoothing can lead to both the overfitting issue and the underfitting issue.

C. SSFG Regularization

Dropout randomly dropout neurons, as well as their connections, from the neural network during training. Applying dropout to a neural network can be seen as training many subnetworks of the original network and using the ensemble of these subnetworks to make predictions at test time [9]. Although the idea behind dropout has been applied in graph networks [11], [13], these studies do not directly address the oversmoothing issue.

We introduce SSFG regularization to address the oversmoothing issue. In SSFG, we stochastically scale features and gradients in the training procedure. Our idea is to explicitly apply a random scaling factor to break feature convergence. Specifically, we multiply each node feature by a factor sampled from a probability distribution in forward propagation. In backward propagation, the gradient of each node feature is also multiplied by a factor sampled from the probability distribution.

Fig. 3. (a) Illustration of our method for sampling scaling factors [see (1)]; note that $p_{\text{Beta}(a=0.5, b=0.5)}(\lambda_2 = 1 + \delta) = p_{\text{Beta}(a=0.5, b=0.5)}(\lambda_1 = 1 - \delta)$, and $\lambda_2 = (1/\lambda_1)$.

(b) Probability distribution functions of (1) and the shifted beta distribution; the probability distribution function of (1) is shown using the normalized histogram of ten million sampled factors.

Fig. 4. Comparison of our SSFG method and ReLU. When used together with ReLU, our SSFG can be seen as a stochastic ReLU using random slopes in forward and backward propagations.

An illustration of our method for sampling factors (Eq. (2))

Pdfs of Eq.(2) and the Beta distribution

- Histgram (normalized) of the factors sampled from Eq.(2) with $a=0.5$ - Pdf of Beta($a=0.5, b=0.5$)
distribution. We wish the expectation of the cumulated factors that are applied to node features and gradients to be unchanged. To this end, we adopt the following trick to define the scaling factor:

\[
\lambda = \begin{cases} 
\bar{\lambda}, & \bar{\lambda} \leq 1 \\
1/(2-\bar{\lambda}), & \bar{\lambda} > 1,
\end{cases}
\]

where \(\bar{\lambda} \sim \text{Beta}(\alpha, \alpha) + 0.5\). (2)

Beta(\alpha, \alpha) is a probability distribution with the mean equal to 0.5, and \(\alpha\) is a hyperparameter. When \(\alpha\) is set to 1.0, Beta(\alpha, \alpha) is equivalent to the standard uniform distribution. With the above method, the scaling factor falls in the interval [0.5, 2]. By directly scaling features, the oversmoothing issue is mitigated. We detail our SSFG regularization method in Algorithm 1. Note that the scaling factors for forward propagation and those for backward propagation are sampled independently using (2). Our SSFG regularization method does not introduce additional trainable model parameters.

A schematic illustration of our SSFG regularization method and its comparison to dropout is shown in Fig. 2. Unlike the dropout method, which performs dropout at the neuron level, our SSFG regularization in forward propagation can be seen as a variant of dropout that is applied at the feature level. When the scaling factor \(\lambda\) is smaller than 1, a linear proportion of the node feature is dropped out; and when the scaling factor \(\lambda\) is larger than 1, a linear proportion of the node feature is added back to the node feature. Applying stochastic scaling to gradients further increases randomness in the optimization procedure. This can further help improve the overall performance. To the best of our knowledge, this is the first research on regularizing neural networks at both the feature level and the gradient level. We show through experiments that stochastically scaling gradients are complementary to stochastically scaling features to improve the overall performance.

For node \(v_i\), the accumulated factor applied to its hidden feature outputted at layer \(l\) can be approximated as follows:

\[
\Lambda = \prod_{i=1}^{l} \lambda_i = \lambda_1, \lambda_2, \ldots, \lambda_l
\]

where \(\lambda_1, \lambda_2, \ldots, \lambda_l\) are sampled independently using (2). As shown in Fig. 3, \(p(1-\delta) = p((1/1-\delta))\) for any \(\delta \in [0, 0.5]\); therefore, the expectation of \(\Lambda\) equals to 1, i.e., \(\mathbb{E}(\Lambda) = 1\). Based on this analysis, we apply a scaling factor of 1 to node features at test time for target tasks. In our experiments, we also compare the performance of using different scaling factors at test time.

ReLU has been commonly used as the nonlinear activation function in graph networks. When used together with ReLU, our SSFG can be seen as a stochastic ReLU activation function. An illustration of this explanation is shown in Fig. 4. By using stochastic slopes in both forward and backward propagations, the network model can be robust to different feature variations. This property makes our SSFG method not specific to graph networks. It could potentially be applied to other types of neural networks, such as (vision) transformers [31], [32] and convolutional neural networks, to improve the generalization performance.

D. Regularizing Graph Networks With SSFG

A typical GCN takes \(X\) and the graph structure \(\tilde{A}\) as input and updates node features layerwisely as follows:

\[
h^{l+1}_i = f\left(h^l_i, \{h^l_j\}_{j \in \mathcal{N}_i}\right), \quad l = 0, \ldots, L - 1
\]

where \(L\) is the number of graph convolutional layers and \(\mathcal{N}_i\) is the set of neighbor nodes of \(v_i\). The proposed SSFG regularization is a general method that can be applied to a wide variety of graph networks. In this work, we evaluate our SSFG on three types of commonly used GCNs, i.e., GraphSAGE, GAT, and GatedGCN, to demonstrate its effectiveness.

1) GraphSAGE: GraphSAGE is a general inductive framework that leverages node feature information to efficiently generate node features for previously unseen data. In a GraphSAGE layer, a fixed-size set of neighbors are randomly sampled, and the features of these sampled neighbors are aggregated using an aggregator function, such as the mean operator and LSTM [33] to update a node’s feature as follows:

\[
h^{l+1}_i = \sigma\left( W \cdot \text{Concat}(h^l_i, \text{Aggregator}\{h^l_j\}_{j \in \mathcal{N}_i}) \right)
\]

where \(W\) is the weight matrix of the shared linear transformation function and \(\sigma\) is a nonlinear activation function, e.g., the ReLU activation function. SSFG can be applied to input node features to the GraphSAGE layer or after the nonlinear activation function.

2) GATs: GATs are inspired by the self-attention mechanism [31] that is widely applied in NLP and computer vision tasks. In a GAT layer, each neighbor of a node is assigned with an attention weight computed by an attention function in feature aggregation as follows:

\[
h^{l+1}_i = \sigma\left( \sum_{j \in \mathcal{N}_i} \text{attn}(h^l_i, h^l_j) \cdot W \cdot h^l_j \right)
\]

Algorithm 1 Pseudocode of Our SSFG Regularization Method in a Pytorch-Like Style

Require: Node features \(h \in \mathbb{R}^{N \times d}\) (\(N\) nodes, each with a \(d\) dimensional feature), hyperparameter \(\alpha\) used in the Beta function for sampling.

1: function FORWARD(h)
2: \(\text{beta} = \text{torch.distributions.beta.Beta}(\alpha, \alpha)\)
3: \(\text{lamda} = \text{beta.sample}(h, \text{shape}[:1]) + 0.5\)
4: \(\text{lamda} = \text{lamda} \cdot (\text{lamda} > 1) = 1/(2-\text{lamda} \cdot (\text{lamda} > 1))\)
5: return \(\text{lamda} \ast h\)
6: end function

7: function BACKWARD(grad_output)
8: \(\text{beta} = \text{torch.distributions.beta.Beta}(\alpha, \alpha)\)
9: \(\text{lamda} = \text{beta.sample}(\text{grad_output}, \text{shape}[:1]) + 0.5\)
10: \(\text{lamda} = \text{lamda} \cdot (\text{lamda} > 1) = 1/(2-\text{lamda} \cdot (\text{lamda} > 1))\)
11: return \(\text{lamda} \ast \text{grad_output}\)
12: end function

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
where \( \text{attn} \) is the attention function. GATs usually employ multiple attention heads to improve the overall performance. By default, we apply our SSFG regularizaton to the output of each attention head.

3) **GatedGCNs**: GatedGCNs use the edge gating mechanism [34] and residual connection in aggregating features from a node’s local neighborhood as follows:

\[
\begin{align*}
    h_i^{l+1} &= \sigma \left( U \cdot h_i^l + \sum_{j \in N_i} e_{ij} \odot V \cdot h_j^l \right) + h_i^l \\
    e_{ij}^{l+1} &= \sigma \left( A \cdot h_i^l + B \cdot h_j^l \right) + e_{ij}^l
\end{align*}
\]

where \( U, V, A, \) and \( B \) are weight matrices of linear transformations, and \( \odot \) is the Hadamard product. GatedGCNs explicitly maintain edge feature \( e_{ij} \) at each layer. By default, we apply our SSFG regularization to both the node features and the edge features outputted by a GatedGCN layer.

**IV. EXPERIMENTS**

**A. Experimental Setup**

1) **Datasets**: Our experiments are conducted on seven recently released benchmark datasets [6], i.e., PATTERN, CLUSTER, MNIST, CIFAR10, TSP, COLLAB, and ZINC. These datasets are used for four graph-based tasks: node classification (PATTERN and CLUSTER), graph classification (MNIST and CIFAR10), link prediction (TSP and COLLAB), and graph regression (ZINC). The statistics of the seven datasets are given in Table I.

| Dataset   | Graphs | Nodes/graph | #Training | #Val. | #Test |
|-----------|--------|-------------|-----------|-------|-------|
| PATTERN   | 14K    | 44-188      | 10,000    | 2000  | 2000  |
| CLUSTER   | 12K    | 41-190      | 10,000    | 1000  | 1000  |
| MNIST     | 70K    | 40-75       | 55,000    | 5000  | 10,000|
| CIFAR10   | 60K    | 85-150      | 45,000    | 5000  | 10,000|
| TSP       | 12K    | 50-500      | 10,000    | 1000  | 1000  |
| COLLAB    | 12K    | 9-37        | 10,000    | 1000  | 1000  |
| ZINC      | 12K    | 235,868     | 1         |       |       |

2) **Training Details**: We closely follow the experimental setup used in the work of Dwivedi et al. [6]. The Adam method [35] is used to train all the models. The learning rate is initialized to \( 10^{-3} \) and reduced by a factor of 2 if the loss has not improved for a number of epochs (10, 20, or 30); note that we use a larger patience number for some experiments. The training procedure is terminated when the learning rate is reduced to smaller than \( 10^{-6} \). For the node classification task, we conduct experiments using different layers (four and 16). For the remaining tasks, the number of layers is set to 4. For each evaluation, we run the experiment four times using different random seeds and report the mean and standard deviation of the four results. Our method is implemented using Pytorch [36] and the DGL library [37].

3) **Evaluation Metrics**: Following [6], the following evaluation metrics are used for different tasks.

1) **Accuracy**: Weighted average node classification accuracy is used for the node classification task (PATTERN and CLUSTER), and classification accuracy is used for the graph classification task (MNIST and CIFAR10).

2) **F1 score** for the positive class is used for performance evaluation on the TSP dataset, due to high class imbalance, i.e., only the edges in the TSP tour are labeled as positive.

3) **Hits@K** [38] is used for the COLLAB dataset, aiming to measure the model’s ability to predict future collaboration relationships. This method ranks each true collaboration against 100000 randomly sampled negative collaborations and counts the ratio of positive edges that are ranked at the \( K \)th place or above.

4) The mean absolute error (MAE) is used to evaluate graph regression performance on ZINC.

**B. Experimental Results**

1) **Quantitative Results**: Table II reports the quantitative results of node classification on PATTERN and CLUSTER. It can be seen that applying our SSFG regularization effectively improves the test accuracies except for GraphSAGE on the PATTERN dataset. Applying our SSFG regularization on GATs with 16 layers yields 3.190% and 3.102% performance improvements on PATTERN and CLUSTER, respectively. These improvements are higher than those obtained for GATs with four layers. For GatedGCNs, applying our SSFG regularization yields more performance improvements on CLUSTER than those on PATTERN.

The graph classification results on MNIST and CIFAR10 are reported in Table III. We see that applying our SSFG regularization helps improve the overall performance for the three baseline graph networks on the two datasets. While vanilla GraphSAGE and GatedGCN perform well compared to GAT on MNIST, the three baseline graph networks with our SSFG regularization achieve comparable performances. On CIFAR10, applying our SSFG regularization to GatedGCN improves the accuracy from 67.312% to 71.938, yielding a 4.626% performance gain, which is higher than those obtained by applying SSFG to GraphSAGE and GAT.

The results for link prediction are reported in Table IV. Once again, applying SSFG regularization results in improved performance for the three baseline graph networks. On TSP, the use of SSFG regularization in GraphSAGE, GAT, and GatedGCN yields 0.049, 0.011, and 0.015 performance gains, respectively. On the COLLAB dataset, applying SSFG regularization to GraphSAGE and GAT yields 2.528 and 3.214 performance improvements, respectively. For GatedGCN, SSFG slightly improves the prediction performance.

Table V reports the experimental results on ZINC, demonstrating the effectiveness of our SSFG regularization to improve the graph regression performance for the three baseline graph networks. For GraphSAGE, GAT, and GatedGCN, the use of our SSFG regularization reduces 0.027, 0.009, and 0.098 MAEs, respectively.
We have shown that the proposed SSFG regularization method helps improve the overall performance of the three baseline graph networks for different graph-based tasks. For most experiments, the performance on the test data improves, while that on the training data reduces. This indicates that the improvements are obtained by reducing the overfitting issue. It is worth noting that, for some experiments, e.g., the experiments on TSP and the experiments of GatedGCN with four layers on PATTERN and CLUSTER, the performances on the test data and the training data improve simultaneously. This indicates that our SSFG regularization also helps to address the underfitting issue. While the implication of oversmoothing remains unsolved in previous studies, our results demonstrate that oversmoothing can lead to both the overfitting issue and the underfitting issue.

Besides, we observe that, for most experiments, our SSFG regularization method results in small standard deviations. On CLUSTER and COLLAB, the use of our SSFG regularization consistently results in small standard deviations compared to those obtained without using SSFG regularization. For the remaining datasets, our SSFG regularization also results in small standard deviations for most cases. This shows that the proposed SSFG regularization method can stabilize the learning algorithms.

For the three baseline GCNs, we show the impact of the value of $\alpha$ used for sampling scaling factors on the overall performance in the quantitative results. We see that the value of $\alpha$ has different impacts for different tasks. Even for the same task, a graph network with different layers may use different values of $\alpha$ to obtain the best task performance. This indicates that the proposed SSFG regularization method can stabilize the learning algorithms.

### TABLE II

**NODE CLASSIFICATION RESULTS ON PATTERN AND CLUSTER.** We experiment using four and 16 layers in the graph networks.

| Method | \(L\) | PATTERN Test (Acc.) | Train (Acc.) | Method | \(L\) | CLUSTER Test (Acc.) | Train (Acc.) |
|--------|------|---------------------|-------------|--------|------|---------------------|-------------|
| GraphSAGE w/o SSFG | 4 | 50.516 ± 0.001 | 50.473 ± 0.014 | GraphSAGE w/o SSFG | 4 | 50.454 ± 0.145 | 54.374 ± 0.203 |
| GraphSAGE + SSFG (\(\alpha=5.0\)) | 4 | 50.000 ± 0.000 | 50.000 ± 0.000 | GraphSAGE + SSFG (\(\alpha=5.0\)) | 4 | 50.505 ± 0.080 | 53.014 ± 0.029 |
| GraphSAGE + SSFG (\(\alpha=10.0\)) | 4 | 50.000 ± 0.000 | 50.000 ± 0.000 | GraphSAGE + SSFG (\(\alpha=5.0\)) | 4 | 50.562 ± 0.070 | 53.014 ± 0.025 |
| GraphSAGE + SSFG (\(\alpha=5.0\)) | 16 | 50.492 ± 0.001 | 50.478 ± 0.005 | GraphSAGE + SSFG (\(\alpha=5.0\)) | 4 | 50.332 ± 0.082 | 53.152 ± 0.207 |
| GAT w/o SSFG | 4 | 75.824 ± 1.823 | 77.883 ± 1.632 | GAT w/o SSFG | 4 | 57.732 ± 0.323 | 58.331 ± 0.342 |
| GAT + SSFG (\(\alpha=7.0\)) | 4 | 77.080 ± 0.640 | 77.381 ± 0.688 | GAT + SSFG (\(\alpha=7.0\)) | 4 | 58.799 ± 0.049 | 59.631 ± 0.032 |
| GAT + SSFG (\(\alpha=8.0\)) | 4 | 77.290 ± 0.460 | 77.938 ± 0.528 | GAT + SSFG (\(\alpha=8.0\)) | 4 | 58.888 ± 0.044 | 59.656 ± 0.025 |
| GAT + SSFG (\(\alpha=5.0\)) | 16 | 78.271 ± 0.186 | 80.212 ± 0.476 | GAT + SSFG (\(\alpha=7.0\)) | 16 | 73.512 ± 0.097 | 73.173 ± 0.402 |
| GAT + SSFG (\(\alpha=8.0\)) | 16 | 78.141 ± 0.123 | 82.724 ± 0.385 | GAT + SSFG (\(\alpha=8.0\)) | 16 | 78.691 ± 0.088 | 79.476 ± 0.302 |
| GAT + SSFG (\(\alpha=5.0\)) | 16 | 83.131 ± 0.529 | 83.330 ± 0.332 | GAT + SSFG (\(\alpha=8.0\)) | 16 | 78.267 ± 0.097 | 80.955 ± 0.383 |

### TABLE III

**SUPERPIXEL GRAPH CLASSIFICATION RESULTS ON MINIST AND CIFAR10.** The number of layers is set to 4.

| Method | MNIST Test (Acc.) | CIFAR10 Test (Acc.) |
|--------|-------------------|---------------------|
| GraphSAGE vanilla | 97.312 ± 0.097 | 65.767 ± 0.308 |
| GraphSAGE + SSFG (\(\alpha=3.0\)) | 97.925 ± 0.097 | 68.503 ± 0.373 |
| GraphSAGE + SSFG (\(\alpha=4.0\)) | 97.915 ± 0.098 | 68.803 ± 0.471 |
| GraphSAGE + SSFG (\(\alpha=5.0\)) | 97.943 ± 0.147 | 68.792 ± 0.292 |
| GAT vanilla | 95.353 ± 0.205 | 64.223 ± 0.455 |
| GAT + SSFG (\(\alpha=1.0\)) | 97.584 ± 0.075 | 65.158 ± 0.418 |
| GAT + SSFG (\(\alpha=2.0\)) | 97.938 ± 0.075 | 65.954 ± 0.116 |
| GAT + SSFG (\(\alpha=3.0\)) | 97.930 ± 0.082 | 66.065 ± 0.171 |
| GatedGCN vanilla | 97.340 ± 0.143 | 67.312 ± 0.311 |
| GatedGCN + SSFG (\(\alpha=1.0\)) | 97.848 ± 0.106 | 71.585 ± 0.361 |
| GatedGCN + SSFG (\(\alpha=1.5\)) | 97.730 ± 0.116 | 71.938 ± 0.190 |
| GatedGCN + SSFG (\(\alpha=2.0\)) | 97.985 ± 0.032 | 71.383 ± 0.427 |
| GatedGCN + SSFG (\(\alpha=2.5\)) | 97.703 ± 0.054 | 70.913 ± 0.306 |
that the value of $\alpha$ needs to be carefully tuned for the best task performance. Based on our results, it is recommended that the hyperparameter $\alpha$ is tuned by starting with 1.0 and gradually increasing to a larger value for node classification, graph classification, and edge prediction. For the graph regression task, it is recommended that $\alpha$ is tuned by starting with a smaller value, e.g., 10, and then gradually increasing the value.

Fig. 5 shows the training and test accuracy curves with respect to the training epoch. We see that our SSFG regularization method helps to address both the overfitting issue and the underfitting issue. Compared with the work of Dwivedi et al. [6], we use a large patience value for the optimizer when learning on some datasets. Therefore, it can take more epochs to complete the training procedure. We also observe that the training procedure takes comparable epochs on some datasets (e.g., MNIST) as that without using SSFG regularization. As aforementioned, our method can be seen as a stochastic ReLU activation function; the results show that our method outperforms standard ReLU in graph representation learning. As with the ReLU activation function, our method does not increase the number of learnable parameters.

2) Ablation Study: Our SSFG regularization method performs stochastic scaling at both the feature level and the gradient level. We conduct an ablation study to show the effect of scaling features and gradients on the overall performance. In our experiments, we use GAT on COLLAB and GatedGCN on the remaining datasets. According to the quantitative results, the value of $\alpha$ that achieves the best task performance is used in the experiments. The ablation study results are reported in Table VI. We see that scaling features and scaling gradients contribute differently to different datasets. On a whole, stochastically scaling features and gradients are complementary to each other to achieve the best task performance.

3) Comparison With Dropout: Our SSFG method can be seen as a variant of dropout that is applied at both the feature level and the gradient level. To demonstrate the advantage of our method for graph networks, we further compare the performance of our method and dropout. We apply dropout with a probability of 0.5 to node features. The comparison of results is shown in Table VII. Dropout results in slightly improved performance on MINST, CIFAR10, and COLLAB, and reduced performance on the remaining datasets. In contrast to dropout, our SSFG regularization consistently improves the overall performance on the seven datasets. This demonstrates the advantage of our method over dropout for graph networks.

We also tested using different probability values (0.8 and 0.9) in dropout; we found that this leads to similar or reduced performance. We also found that dropout requires more epochs to converge than our SSFG method. Dropout essentially uses the ensemble of numerous trained subnetworks to make a prediction, which does not effectively address the oversmoothing problem, whereas our SSFG regularization addresses the oversmoothing issue to improve the overall performance.

4) Comparison of Beta Distribution Against Other Distributions: In our SSFG regularization method, the scaling factors are sampled from a distribution transformed from the beta distribution [see Eq. (2)]. We validate the performance of our regularization method using the standard uniform distribution

| Method                  | TSP       | COLLAB   |
|-------------------------|-----------|----------|
|                          | Test (F1) | Train (F1)|
| GraphSAGE w/o SSFG      | 0.665±0.003 | 0.669±0.003          |
| GraphSAGE + SSFG (α=4.0) | 0.703±0.008 | 0.706±0.007          |
| GraphSAGE + SSFG (α=5.0) | 0.714±0.003 | 0.717±0.003          |
| GraphSAGE + SSFG (α=6.0) | 0.706±0.003 | 0.707±0.005          |
| GAT w/o SSFG            | 0.671±0.002 | 0.673±0.002          |
| GAT + SSFG (α=200)      | 0.679±0.003 | 0.680±0.004          |
| GAT + SSFG (α=300)      | 0.682±0.000 | 0.684±0.001          |
| GAT + SSFG (α=350)      | 0.681±0.003 | 0.684±0.004          |
| GatedGCN w/o SSFG       | 0.791±0.003 | 0.793±0.003          |
| GatedGCN + SSFG (α=4.0) | 0.802±0.001 | 0.804±0.001          |
| GatedGCN + SSFG (α=5.0) | 0.806±0.001 | 0.807±0.001          |
| GatedGCN + SSFG (α=6.0) | 0.805±0.001 | 0.808±0.001          |
| GatedGCN + SSFG (α=7.0) | 0.805±0.001 | 0.807±0.001          |

| Method                  | ZINC      | COLLAB   |
|-------------------------|-----------|----------|
|                          | Test (MAE) | Train (MAE)|
| GraphSAGE vanilla       | 0.468±0.003 | 0.251±0.004          |
| GraphSAGE + SSFG (α=10) | 0.441±0.006 | 0.191±0.005          |
| GraphSAGE + SSFG (α=20) | 0.461±0.008 | 0.221±0.003          |
| GraphSAGE + SSFG (α=30) | 0.459±0.007 | 0.252±0.010          |
| GAT vanilla             | 0.475±0.007 | 0.317±0.006          |
| GAT + SSFG (α=10)       | 0.479±0.002 | 0.331±0.016          |
| GAT + SSFG (α=20)       | 0.466±0.001 | 0.329±0.010          |
| GAT + SSFG (α=30)       | 0.470±0.004 | 0.312±0.009          |
| GatedGCN vanilla        | 0.435±0.011 | 0.287±0.014          |
| GatedGCN + SSFG (α=100) | 0.371±0.018 | 0.175±0.015          |
| GatedGCN + SSFG (α=200) | 0.352±0.023 | 0.164±0.022          |
| GatedGCN + SSFG (α=250) | 0.347±0.023 | 0.154±0.021          |
| GatedGCN + SSFG (α=300) | 0.357±0.021 | 0.157±0.010          |

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Fig. 5. Training and test accuracy/F1/hits@50/MAE curves with respective to the training epoch, showing that our SSFG regularization helps to address both the overfitting issue and the underfitting issue.

TABLE VI
ABLATION STUDY. WE SHOW THE EFFECT OF STOCHASTICALLY SCALING FEATURES AND GRADIENTS ON THE OVERALL PERFORMANCE

| Method          | GatedGCN             | GAT          |
|-----------------|----------------------|--------------|
|                 | Pattern (L=4, α=5.0) | Cluster (L=16, α=5.0) | MNIST (L=4, α=2.0) | CIFAR10 (L=4, α=1.5) | TSP (F1) (L=4, α=5.0) | ZINK (MAE ↓) (L=4, α=7.0) | COLLAB (L=4, α=250) |
| w/o SSFG        | 84.480               | 73.840       | 97.340          | 67.312               | 0.791                   | 0.435                   | 51.501               |
| +SSF (forward regul. only) | 85.122               | 75.318       | 97.733          | 71.107               | 0.802                   | 0.352                   | 54.394               |
| +SSG (backward regul. only) | 84.639               | 72.951       | 97.537          | 68.325               | 0.795                   | 0.348                   | 52.440               |
| +SSF (full)     | 85.334               | 75.960       | 97.985          | 71.983               | 0.806                   | 0.347                   | 54.715               |

TABLE VII
PERFORMANCE COMPARISON OF DROPOUT AND OUR SSFG REGULARIZATION METHOD. DROPOUT (p = 0.5) RESULTS IN ONLY SLIGHTLY IMPROVED OR REDUCED PERFORMANCE, WHEREAS OUR SSFG METHOD CONSISTENTLY IMPROVES THE OVERALL PERFORMANCE

| Method         | GatedGCN             | GAT          |
|----------------|----------------------|--------------|
|                 | Pattern (L=4, α=5.0) | Cluster (L=16, α=5.0) | MNIST (L=4, α=2.0) | CIFAR10 (L=4, α=1.5) | TSP (F1) (L=4, α=5.0) | ZINK (MAE ↓) (L=4, α=7.0) | COLLAB (L=4, α=250) |
| w/o SSFG/Dropout | 84.480               | 73.840       | 97.340          | 67.312               | 0.791                   | 0.435                   | 51.501               |
| +Dropout (p=0.5) | 50.000               | 24.988       | 97.494          | 67.345               | 0.551                   | 0.675                   | 51.589               |
| +SSF (full)     | 85.334               | 75.960       | 97.985          | 71.983               | 0.806                   | 0.347                   | 54.715               |

TABLE VIII
PERFORMANCE COMPARISON OF USING BETA (α, α) AGAINST THE STANDARD UNIFORM DISTRIBUTION U(0, 1) AND TRUNCATED NORMAL DISTRIBUTION N(μ = 0.5, σ, MIN = 0, MAX = 1) IN (2) FOR DEFINING THE SCALING FACTORS

| Distribution   | GatedGCN             | GAT          |
|----------------|----------------------|--------------|
| U(0,1)         | 85.187               | 72.320       | 97.848          | 71.585               | 0.792                   | 0.466                   | 52.521               |
| N(μ = 0.5, σ = 1) | 85.291               | 72.858       | 97.824          | 71.329               | 0.784                   | 0.450                   | 52.681               |
| N(μ = 0.5, σ = 0.5) | 85.180               | 73.692       | 97.796          | 71.411               | 0.794                   | 0.444                   | 52.548               |
| N(μ = 0.5, σ = 0.1) | 85.058               | 75.103       | 97.832          | 70.158               | 0.805                   | 0.355                   | 54.035               |
| Beta(α, α)     | 85.334               | 75.960       | 97.985          | 71.983               | 0.806                   | 0.347                   | 54.715               |

and the truncated normal distribution to define the scaling factors. We replace Beta(α, α) in Eq. (2) with the standard uniform distribution U(0, 1) or the truncated normal distribution N(μ = 0, σ, 0, 1) (bounded between 0 and 1). For the truncated normal distribution N(μ = 0, σ, 0, 1), we test the performance using three different deviation values, i.e., σ = 1, 1/2, 1/10. The results are reported in Table VIII. We see that using beta distribution achieves improved perfor-
TABLE IX
PERFORMANCE COMPARISON OF USING DIFFERENT SCALING FACTORS AT
TEST TIME FOR SUPERPIXEL GRAPH CLASSIFICATION ON MNIST AND
CIFAR10

| Scaling factor | GatedGCN (c=2.0) | CIFAR10 (c=1.5) |
|---------------|------------------|-----------------|
| 0.8           | 97.626±0.064     | 71.377±0.304    |
| 0.9           | 97.810±0.039     | 71.637±0.267    |
| 1.0           | 97.985±0.032     | 71.938±0.190    |
| 1.1           | 97.759±0.069     | 71.690±0.244    |
| 1.2           | 97.711±0.020     | 71.441±0.369    |

Fig. 6. Illustration of the probability density functions for beta distributions
with $\alpha = \beta$.

Fig. 7. Illustration of the probability density functions $\text{Beta}(\alpha, \alpha) + 0.5$.

C. Broader Impact

We have shown that our SSFG regularization method is effective in improving the overall performance of graph networks. The proposed SSFG regularization method helps to address both the overfitting issue and the underfitting issue without increasing the number of trainable parameters. When used together with ReLU, our SSFG method can be seen as a stochastic ReLU activation function that is applied at both the feature level and the gradient level. This explanation makes our SSFG method not specific for graph networks. Overfitting and underfitting are also issues with neural networks for other tasks, such as image recognition and natural language processing tasks. It could be potentially useful to replace the standard ReLU with our SSFG method in the network models to improve the generalization performance, especially when training data are small.

V. CONCLUSION

In this article, we presented a stochastic regularization method for GCNs. In our method, we SSFG by a factor sampled from a probability distribution in the training procedure. Our SSFG regularization method helps to address the oversmoothing issue caused by repeatedly applying graph convolutional layers. We showed that applying stochastic scaling at the feature level is complementary to that applied at the gradient level in improving the overall performance. When used together with ReLU, our method can also be seen as a stochastic ReLU activation function. We experimentally validated our SSFG regularization method on seven benchmark datasets for different graph-based tasks, including node classification, graph classification, link prediction, and graph regression. We conducted an ablation study to show the effects of applying scaling at the feature level and the gradient level on the overall performance. The experimental results demonstrated that our SSFG method helps to address both the overfitting issue and the underfitting issue. While the oversmoothing issue was identified several years ago, the implication of this issue remains unsolved. Our experimental results suggest that the oversmoothing issue can lead to both overfitting and underfitting.

APPENDIX

BETA DISTRIBUTION

The beta distribution is a continuous probability distribution defined in the interval $[0, 1]$ parameterized by two positive parameters denoted by $\alpha$ and $\beta$. The probability density function for $X \sim \text{Beta}(\alpha, \beta)$ is defined as follows:

$$f(X = x) = \begin{cases} \frac{1}{B(\alpha, \beta)} x^{\alpha - 1}(1-x)^{\beta - 1}, & 0 < x < 1 \\ 0, & \text{otherwise} \end{cases}$$

where $B(\alpha, \beta) = \int_0^1 x^{\alpha - 1}(1-x)^{\beta - 1} dx$, which is known as the beta function. The expectation of $X$ is $E(X) = (\alpha/\alpha + \beta)$, and the variance of $X$ is $\text{Var}(X) = (\alpha\beta/((\alpha + \beta)^2(\alpha + \beta + 1)))$.

By setting $\beta = \alpha$, the expectation of a random variable sampled the beta distribution equals to 0.5, i.e., $E(X \sim \text{Beta}(\alpha, \alpha)) = 0.5$. When $\alpha$ and $\beta$ are set to 0, the beta
distribution equals to the standard uniform distribution. Fig. 6 shows examples of the probability density functions of beta distributions with $\alpha = \beta$.

Fig. 7 shows examples of probability density distributions for Beta($\alpha, \alpha$) + 0.5. We can see that the probability density functions are symmetric about $x = 1.0$ and $f(1 - \delta) = f(1 + \delta)$ for $0 < \delta < 0.5$. In our SSFG regularization method, we use the reparameterization trick for sampling scaling factors [see (2)]; we first sample a value $\tilde{\lambda}$ from Beta($\alpha, \alpha$) + 0.5. If 0.5 $\leq \tilde{\lambda} \leq 1.0$, the sampled value is used for scaling features or gradients. If 1.0 $< \tilde{\lambda} \leq 1.5$, the scaling factor is defined as $\lambda = (1/2 - \tilde{\lambda}) = (1/1 - (\tilde{\lambda} - 1))$. Because Beta($\alpha, \alpha$) + 0.5 is symmetrical about $x = 1.0$, for 1.0 $< \tilde{\lambda} \leq 1.5$, we have $p(\tilde{\lambda}) = p(1 + (1 - \tilde{\lambda}) = p(1 - (\tilde{\lambda} - 1)) = p(2 - \tilde{\lambda})$. Therefore, for any $\delta \in [0, 0.5]$, $p(1 - \delta) = p((1/1 - \delta))$ [see Fig. 3(a)].

Our SSFG regularization method is applied to each layer of the GCN. For node $v_l$, the cumulative factor applied to hidden feature at layer $l$ can be approximated as $\Lambda = \prod_{i=1}^{l} \lambda_i = \lambda_1, \lambda_2, \ldots, \lambda_l$, where $\lambda_1, \lambda_2, \ldots, \lambda_l$ are sampled independently using (2). Because $p(1 + \delta) = (1/1 - \delta)$ for any $\delta \in [0, 0.5]$, the expectation of $\Lambda$ equals to 1, i.e., $E(\Lambda) = 1$. Therefore, we apply a scaling factor of 1 to node features at test time for target tasks.

ACKNOWLEDGMENT

The authors would like to gratefully thank the editor and reviewers for reviewing the manuscript.

REFERENCES

[1] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, “Session-based recommendation with graph neural networks,” in Proc. AAAI Conf. Artif. Intell., vol. 33, 2019, pp. 346–353.
[2] C. Zhang and Y. Chen, “Link prediction based on graph neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2018, pp. 5165–5175.
[3] I. Chami, A. Wolf, D.-C. Juan, F. Sala, S. Ravi, and C. Ré, “Low-dimensional hyperbolic knowledge graph embeddings,” in Proc. ACL, 2020, pp. 6901–6914.
[4] L. Nie, Y. Li, F. Feng, X. Song, M. Wang, and Y. Wang, “Large-scale question tagging via joint question-topic embedding learning,” ACM Trans. Inf. Syst., vol. 38, no. 2, pp. 1–23, Mar. 2020.
[5] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, “Self-normalizing neural networks,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 971–980.
[6] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, “Benchmarking graph neural networks,” 2020, arXiv:2003.00982.
[7] Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in Proc. AAAI Conf. Artif. Intell., 2018, pp. 3538–3545.
[8] G. Li, M. Müller, A. Thabet, and B. Ghanem, “DeepGCNs: Can GCNs go as deep as CNNs?” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 9267–9276.
[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” J. Mach. Learn. Res., vol. 15, pp. 1929–1958, Jun. 2014.
[10] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” in Proc. Int. Conf. Learn. Represent., 2018. [Online]. Available: https://openreview.net/forum?id=rJrMnplKZ.
[11] Y. Rong, W. Huang, T. Xu, and J. Huang, “DropEdge: Towards deep graph convolutional networks on node classification,” in Proc. Int. Conf. Learn. Represent., 2020.
Haimin Zhang received the bachelor’s degree from Zhejiang Sci-Tech University, Hangzhou, China, in 2010, the master’s degree from Nankai University, Tianjin, China, in 2015, and the Ph.D. degree from the University of Technology Sydney, Ultimo, NSW, Australia, in 2020.

He is currently a Post-Doctoral Research Fellow with the School of Electrical and Data Engineering, University of Technology Sydney. His research interests include pattern recognition, machine learning, computer vision, and multimedia analytics.

Dr. Zhang also serves as a Reviewer for a number of international journals, including Pattern Recognition and IEEE Transactions on Multimedia.

Min Xu (Member, IEEE) received the B.E. degree from the University of Science and Technology of China, Hefei, China, the M.S. degree from the National University of Singapore, Singapore, and the Ph.D. degree from The University of Newcastle, Callaghan NSW, Australia.

She is currently an Associate Professor with the University of Technology Sydney, Ultimo, NSW, Australia. She has published over 170 research papers in international journals and conferences. Her research interests include multimedia, computer vision, and machine learning.

Dr. Xu is an Editorial Board Member of Neurocomputing journal (Elsevier). She has served as a program chair/area chair for many major international conferences.

Guoqiang Zhang (Member, IEEE) received the B.Eng. from the University of Science and Technology of China (USTC), Hefei, China, in 2003, the M.Phil. degree from The University of Hong Kong, Hong Kong, in 2006, and the Ph.D. degree from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2010.

From the spring of 2011, he was a Post-Doctoral Researcher with the Delft University of Technology, Delft, The Netherlands. From the spring of 2015, he was a Senior Researcher with Ericsson AB, Stockholm. Since 2017, he has been a Senior Lecturer with the School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW, Australia. His current research interests include distributed optimization, large-scale optimization, deep learning, and multimedia signal processing.

Kenta Niwa (Senior Member, IEEE) received the B.E., M.E., and Ph.D. degrees in information science from Nagoya University, Nagoya, Japan, in 2006, 2008, and 2014, respectively.

Since joining the Nippon Telegraph and Telephone Corporation (NTT), Kyoto, Japan, in 2008, he has been engaged in research on microphone array signal processing. From 2017 to 2018, he was a Visiting Researcher with the Victoria University of Wellington, Wellington, New Zealand, where he was involved with research on distributed optimization.

He is currently a Distinguished Researcher with the NTT Communication Science Laboratories and the NTT Computer and Data Science Laboratories, NTT Corporation, Kyoto.

Dr. Niwa was a recipient of the Awaya Prize by the Acoustical Society of Japan (ASJ) in 2010.