On Dropping Clusters to Regularize Graph Convolutional Neural Networks

Xikun Zhang, Chang Xu, and Dacheng Tao

UBTECH Sydney AI Centre, School of Computer Science, Faculty of Engineering,
The University of Sydney, Darlington, NSW 2008, Australia
{xzha0505@uni.,c.xu@,dacheng.tao@}sydney.edu.au

Abstract. Dropout has been widely adopted to regularize graph convolutional networks (GCNs) by randomly zeroing entries of the node feature vectors and obtains promising performance on various tasks. However, the information of individually zeroed entries could still present in other correlated entries by propagating (1) spatially between entries of different node feature vectors and (2) depth-wisely between different entries of each node feature vector, which essentially weakens the effectiveness of dropout. This is mainly because in a GCN, neighboring node feature vectors after linear transformations are aggregated to produce new node feature vectors in the subsequent layer. To effectively regularize GCNs, we devise DropCluster which first randomly zeros some seed entries and then zeros entries that are spatially or depth-wisely correlated to those seed entries. In this way, the information of the seed entries is thoroughly removed and cannot flow to subsequent layers via the correlated entries. We validate the effectiveness of the proposed DropCluster by comprehensively comparing it with dropout and its representative variants, such as SpatialDropout, Gaussian dropout and DropEdge, on skeleton-based action recognition.

Keywords: Regularization, Dropout, Graph convolutional networks

1 Introduction

GCNs have gained state-of-the-art results on various tasks including node classification [14, 27], graph generation [33, 35], skeleton-based action recognition [32, 22], tracking [8, 31] and so on. GCNs are designed for extracting features from graph structured data [14], and its general working mechanism is iteratively applying linear transformation to each node feature vector and aggregation on neighboring node feature vectors as the node features in the subsequent layer. Despite the successes, regularization of GCNs is not yet sufficiently studied and only started to receive attention recently [20].

Dropout [11] is currently a most widely adopted regularization in GCNs. However, as dropout is not originally designed for GCNs, it lacks an in-depth investigation on the information flow in GCNs, and thus its performance could

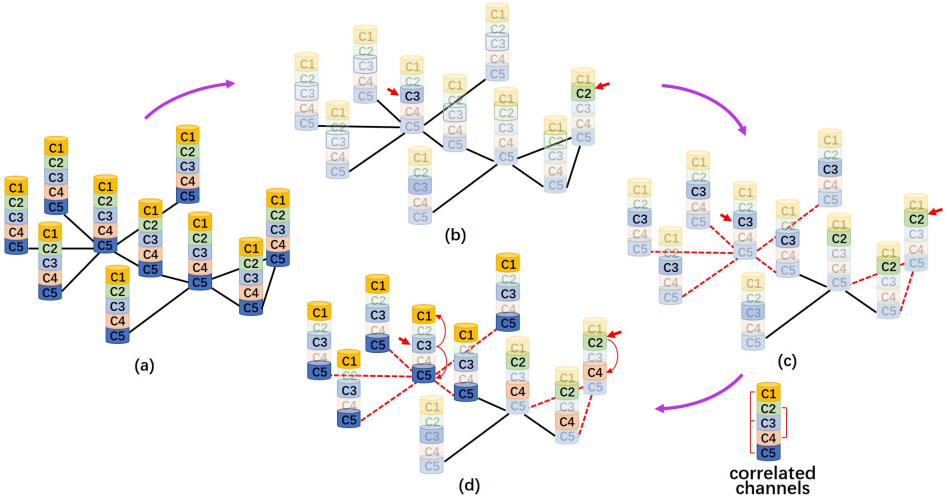

Fig. 1. (a) is a feature map of 10 nodes and 5 channels. In (b), two random seed entries are selected and marked by red arrows. (c) selects the entries spatially correlated to the seed entries. (d) further selects the entries depth-wisely correlated to the selected entries in (c). We assume that channel 2 and 4 are correlated, channel 1,3,5 are correlated, and only show 1-hop spatial correlation

be limited in practice. Specifically, dropout regularizes a network by randomly zeroing entries in a given feature map to remove part of the information. But in GCNs, as linear transformation on each node feature vector and local aggregation on neighboring node feature vectors are iteratively applied, the information is propagated both depth-wisely between different entries of each node feature vector and spatially between different node feature vectors. Thus the information of the individually zeroed entries by dropout could still present in other correlated entries. Although different variants of dropout were proposed to improve the regularization effectiveness, they mostly failed to consider the feature correlation in GCNs. The improvements are made in other aspects like using adaptive dropping rate [2], generalizing the random noise distribution from Bernoulli to Gaussian [23], randomly deleting graph edges to regularize GCNs [20], and so on. The only methods that consider feature correlation are several modified dropout techniques for CNNs, but due to the tremendous difference in data structure, these methods on Euclidean data are incapable of handling graph data. For example, DropBlock [9] considered the spatial correlation in the CNN feature maps and proposed to drop blocks of contiguous entries to remove information more effectively. However, its defined 'block' cannot be straightforwardly applied to graph data. Above all, there are no existing methods that could effectively regularize GCNs by taking the feature correlation into consideration.

In this paper, we propose DropCluster to better regularize GCNs by considering the feature correlations. We first randomly sample seed entries from the

given node feature vectors, and then concurrently drop the seed entries as well as other entries with spatial or depth-wise correlation. As the information of a node is propagated to its l-hop neighbors after the l-th GCN layer, we consider nodes within l-hop neighborhood of a node as spatially correlated. The depth-wise correlation between different feature channels is measured by the linear correlation coefficient. By concurrently removing the spatially and depth-wisely correlated entries, the dropped information is eliminated more effectively, and the network gets more properly regularized. Also, as we select to drop entries within l-hop neighborhood of the seed entries, the selected entries form clusters around the seed entries, which is the reason our method is named DropCluster (Fig. 1). To demonstrate the effectiveness of DropCluster, we comprehensively compare it with dropout and other representative variants in experiments. Besides, our method is implemented into different GCNs to demonstrate its generalization capability. Moreover, we also implement it into networks with extended depths to further show its effectiveness in regularizing deep GCNs. The experiments are conducted on skeleton-based action recognition task on Northwestern-UCLA and NTU-RGB+D datasets.

2 Related Work

GCNs GCNs are designed for extracting features from graph data. Two main streams of GCNs include the spectral- and the spatial-based GCNs. The spectral-based GCNs [3, 6, 14, 16] apply convolutional filters to graph spectrum with good theoretical foundations, but the spatial-based GCNs are more preferred due to efficiency, flexibility and generalization issues. Thus, we only focus on the spatial GCNs in our work. The first spatial GCN was proposed in [17] and the main operation is aggregating the neighboring information of each node to obtain gradually refined node representations. Later, different variants began to emerge [1, 27, 18, 10]. These variants mainly focus on two aspects, i.e. selecting which nodes to include in convolution and how to aggregate the selected nodes. Original GCN [14] selected 1-hop neighbors for the convolution and the receptive field of each convolution is restricted to 1-hop neighborhood. [25] utilized polynomials of functions of adjacency matrix as convolutional kernel thus the multi-hop neighborhood is captured by high order polynomials. Besides selecting distant nodes to enlarge receptive field, [37, 10, 4] applied different sampling strategies to neighboring nodes to reduce the computation burden. Works on designing aggregation mainly focus on different ways to decide the aggregation weights. [27] proposed to compute the weight between two nodes by inputting the node features into a feed forward network. [24] designed a graph agreement model to predict the probability of each edge correctly connecting two nodes, which then helps aggregating nodes more properly.

Dropout & **Variants** Dropout was proposed in [11] to regularize fully connected networks by randomly zeroing entries in the feature maps. Later on, different variants began to emerge, including Gaussian dropout [23] that generalized dropout by extending the Bernoulli distribution to Gaussian, DropEdge

[20] that regularizes GCNs by randomly deleting edges, and so on [2, 34, 12, 13, 28]. However, although these methods successfully improved dropout, they do not consider feature correlations thus their effectiveness is limited in GCNs, which is shown in our experiments. Besides the techniques dropping information randomly, there are also variants considering the spatial feature correlation, which were specially adapted to regularize CNNs. [26] proposed SpatialDropout to drop entire channels of a feature map, so that spatially correlated entries in the dropped channels are removed together. [7] proposed Cutout to randomly mask out square regions of the input. Inspired by Cutout, [9] proposed DropBlock to drop contiguous regions of a feature map and gained significant improvement. Among them, SpatialDropout [26] can be applied in GCNs and is shown in our experiments. While the other two cannot be applied to graph data as the 'block' on Euclidean data cannot be straightforwardly defined on graph. Moreover, depth-wise correlation is not considered in these methods. In this work, we propose a better regularization by considering feature correlations. Different from the methods above, we do not only consider spatial correlation, but also consider depth-wise correlation between channels. Moreover, our method is customized for GCNs on graph structured data.

3 DropCluster

Our proposed DropCluster aims to better regularize GCNs by considering spatial and depth-wise correlations when dropping entries. In this section, we first present the preliminaries, and then describe our model from two aspects, including addressing the spatial and depth-wise correlations. Finally, we discuss some related methods.

3.1 Preliminaries

We give a general formulation for directed graphs, and generalization to undirected graphs is straightforward with each undirected edge viewed as two directed edges. Each graph is represented as $G = (V, E)$, with $V = \{v_i | i = 1...N\}$ consisting of all the N nodes and $E = \{e_{ij}\}$ consisting of all the edges. Each node v_i has a feature vector $x_i \in \mathbb{R}^d$ with d channels (depth of d), and the feature map $X \in \mathbb{R}^{N \times d}$ is the concatenation of all feature vectors. Each edge e_{ij} denotes a directed edge pointing from v_i to v_j. The edges are also represented by an adjacency matrix $A \in \mathbb{R}^{N \times N}$, with $A_{ij} \in \{0, 1\}$ denoting whether the edge e_{ij} exists. Moreover, we d enote the adjacency matrix with added self-connections as $\tilde{A} = A + I_N$, where $I_N \in \mathbb{R}^{N \times N}$ is an identity matrix.

We denote a channel of a node feature vector as an entry. The j-th channel in x_i is then denoted as entry x_i^j. Given G with N nodes and d-dimensional feature vectors, its feature map $X \in \mathbb{R}^{N \times d}$ has $N \cdot d$ entries. In the following, depth-wise correlation denotes the correlation between different channels of the feature vectors. The neighboring nodes of a node v_i is defined as $\mathcal{N}_G = \{v_k | e_{ik} \in E\}$. The l-hop neighbors of v_i consist of the nodes with a distance of l from v_i, and are

denoted as $\mathcal{N}_G^l(v_i)$. The neighboring entries of an entry x_i^j are entries in the same channels of v_i's neighboring nodes, i.e. $\mathcal{N}_e(x_i^j) = \{x_k^j | v_k \in \mathcal{N}_G(v_i)\}$. And the l-hop neighboring entries are similarly defined as $\mathcal{N}_e^l(x_i^j) = \{x_k^j | v_k \in \mathcal{N}_G^l(v_i)\}$.

DropCluster regularizes a network by multiplying the input feature map by a dropping mask M element-wisely. A graph convolution operation with a dropping mask M could be formulated as:

$$X^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}(M \odot X^{(l)})W^{(l)}). \tag{1}$$

Here, $X^{(l)}$ is the output feature map from the l-th graph convolutional layer. $\tilde{D} \in \mathbb{R}^{N\times N}$ is a diagonal matrix denoting the degree of each node with $D_{ii} = \sum_j \tilde{A}_{ij}$. The term $W^{(l)}$ is a trainable matrix that conduct linear transformation on the node features at the l-th layer. σ is the activation function.

To generate the dropping mask M, the first step of DropCluster is selecting random seed entries. Specifically, we draw a matrix $M_{seed} \in \mathbb{R}^{N\times d}$ from a Bernoulli distribution parameterized by p_{seed}, i.e. $M_{ij} \sim \text{Bernoulli}(p_{seed})$, where the p_{seed} will be explained in Sec. 3.4. Then we will find other entries that are spatially or depth-wisely correlated to the seed entries to drop them concurrently.

3.2 Spatial Correlation

Spatial correlation comes from graph convolution. The information of each node is propagated to its l-hop neighbors after the l-th convolutional layer, therefore dropping individual entries cannot stop information from flowing to the subsequent layer. Thus, after the l-th layer, we propose to drop the seed entries together with the entries within their l-hop neighborhood. These entries form clusters centered at the seed entries with a radius of l, which is the reason our method is called DropCluster. The initial mask M_{seed} denotes seed entries with 1 and the others with 0, and we will update M_{seed} into mask M_s so that both the seed entries and their neighbors within l-hop are denoted with 1. As multiplying M_{seed} with $\tilde{A}$ propagates the value of seed entries to their 1-hop neighbors, multiplying M_{seed} with $\tilde{A}^l$ propagates the values to all entries within l-hop neighborhood. Thus we construct M_s as:

$$M_s = H(\tilde{A}^l \cdot M_{seed}), \tag{2}$$

where the Heaviside step function $H(\cdot)$ returns 1 for positive input and 0 otherwise, and is applied to binarize the obtained mask.

3.3 Depth-wise Correlation

In GCNs, a linear transformation ($W^{(l)}$ in Eq. 1) on node feature vectors always follows the convolution operation. Thus, there also exists depth-wise correlation between entries of different channels. In this part, we introduce how to drop the depth-wisely correlated entries. We adopt the linear correlation coefficient as the measurement of depth-wise correlation. Given the input feature map $X \in \mathbb{R}^{N\times d}$

with d channels, the i-th column $X^i \in \mathbb{R}^N$ corresponds to the i-th channel of the feature map. We first generate a correlation matrix $M_{corr} \in \mathbb{R}^{d \times d}$ [1] to store the correlation coefficients between each pair of channels:

$$M_{corr}^{ij} = corr(X^i, X^j), \tag{3}$$

where M_{corr}^{ij} is at row i and column j of M_{corr}, denoting the correlation coefficient between X^i and X^j. The correlation coefficient $corr(a, b)$ between two variables is a normalized version of their covariance, which is defined as:

$$corr(a, b) = \frac{Cov(a, b)}{\sigma(a) \cdot \sigma(b)}, \tag{4}$$

where $Cov(a, b)$ is the covariance of two variables:

$$Cov(a, b) = E((a - \bar{a}) \cdot (b - \bar{b})), \tag{5}$$

and $\sigma(\cdot)$ is the standard variance of a variable:

$$\sigma(a) = \sqrt{E[(a - \bar{a})^2]}. \tag{6}$$

M_{corr} stores the correlation coefficients ranging from 0 to 1. For the following usage, if the absolute value of the correlation coefficient between two channels exceeds a threshold t_c, we regard the two channels as correlated. Formally, we derive another binary matrix M_c:

$$M_c = H(|M_{corr}| - t_c). \tag{7}$$

The mask M_s stores the already selected entries to drop. With M_c, we update M_s to further include entries depth-wisely correlated to the already selected ones. The i-th row of M_s ($M_{s,i}$) indicates the selected channels of x_i, and the j-th column of M_c (M_c^j) indicates all the channels correlated to channel j. Thus the inner product between $M_{s,i}$ and M_c^j takes a positive value if entry x_i^j is depth-wisely correlated to the already selected entries of x_i and takes zero otherwise. And the matrix multiplication between M_s and M_c is a parallelization of this computation, which turns all the other entries depth-wisely correlated to the already selected ones in M_s into positive. Above all, the update of mask M_s with binarization is formulated as:

$$M_s^c = H(M_s \cdot M_c). \tag{8}$$

M_s^c denotes all the entries to drop with 1, thus the final mask M mentioned in Eq. 1 would be $M = 1 - M_s^c$. The rescaling rate for training is omitted for simplicity and is included in Algo. 1.

Linear correlation coefficient measures the linear correlation strength between channels. With strong linear correlation, two channels are mutually predictable, i.e. the presence of one feature indicates with high confidence the presence of the other one. Thus, individually dropping an entry in one channel cannot efficiently remove the semantic information, but concurrently dropping entries in other strongly correlated channels can remove the information more completely.

[1] Algorithm for parallelized computation of the correlation matrix is included in the supplementary material

Algorithm 1: Dropluster

Input: $X^{(l)} \in \mathbb{R}^{N\times d}$, $A \in \mathbb{R}^{N\times N}$, r_d, t_c, layer index l

Compute correlation matrix $M_{corr} \in \mathbb{R}^{d\times d}$;
Apply threshold to M_{corr}: $M_c = H(|M_{corr}|-t_c)$;
Average number of correlated channels: $n_c = \frac{SUM(M_c)}{d}$;
Average number of edges: $n_e = \frac{SUM(A)}{N}$;
Number of seed entries:
if $l = 1$ **then**
  $n_{seed} = \frac{N\times d\times r_d}{(1+n_e)\times n_c}$;
else
  $n_{seed} = \frac{N\times d\times r_d}{(1+n_e+\sum_{i=1}^{l-1} n_e\times(n_e-1)^{i-1})\times n_c}$;
end
$p_{seed} = \frac{n_{seed}}{N\times d}$;
Initialize mask $M_{seed} : M_{seed}^{ij} \sim \text{Bernoulli}(p_{seed})$;
$\tilde{A} = A + I_N$;
Spatial correlation: $M_s = H(\tilde{A}^l \cdot M_{seed})$;
Channel correlation: $M_s^c = H(M_s \cdot M_c)$;
Actual dropping rate: $r_d^a = \frac{SUM(M_s^c)}{N\times d}$;
Rescaling rate: $r_c = \frac{1}{1-r_d^a}$;
return $(1 - M_s^c) \cdot X^{(l)} \cdot r_c$

3.4 Number of Seed Entries

When handling the spatial and depth-wise correlation, we propagate the values of seed entries to neighboring and depth-wisely correlated entries. As the number of neighboring and depth-wisely correlated entries varies from node to node, the final number of chosen entries also varies. Thus a challenge is to choose a proper number of seed entries (p_{seed}) so that the final actual dropping rate is close to what we set. Our solution is as follows: Given the dropping rate r_d, the average number of edges denoted as n_e, the average number of correlated channels of each channel is n_c, the number of seed entries in the first layer should be:

$$n_{seed} = \frac{N \times d \times r_d}{(1 + n_e) \times n_c}, \tag{9}$$

while in the l-th ($l > 1$) layer they should be:

$$n_{seed} = \frac{N \times d \times r_d}{(1 + n_e + \sum_{i=2}^{l} n_e \times (n_e - 1)^{i-1}) \times n_c}, \tag{10}$$

and the p_{seed} is derived as:

$$p_{seed} = \frac{n_{seed}}{N \times d}. \tag{11}$$

Due to the page limit, detailed computations of Eqs. 9 and 10 are included in the supplementary materials.

3.5 Discussions

This part analyzes the difference between DropCluster and several related methods including SpatialDropout [26], DropBlock [9], and DropEdge [20].

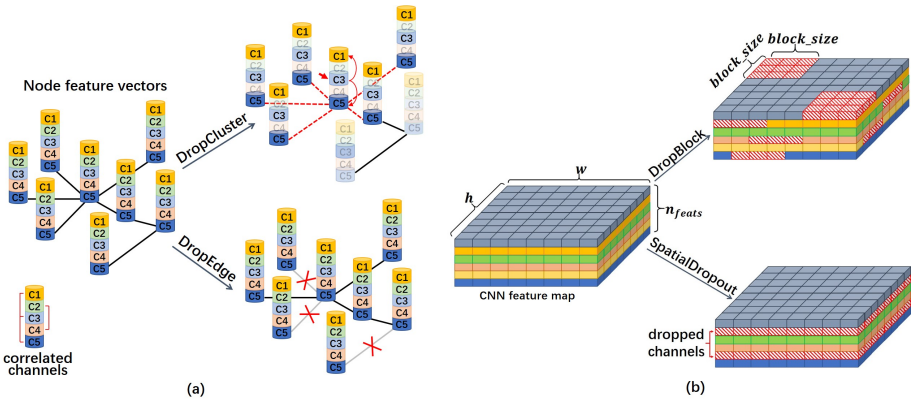

Fig. 2. (a): DropCluster and DropEdge on the given node features. For DropCluster, we highlight the seed entry (marked by red arrow) and other spatially or depth-wisely correlated entries. We assume the feature map is in the first GCN layer, thus only 1-hop neighboring entries are spatially correlated. For DropEdge, randomly deleted edges are marked with red crosses. (b): DropBlock and SpatialDropout on the given CNN feature map. The dropped entries are marked by red slashes

DropCluster vs. SpatialDropout Due to the convolution operation and spatial correlation in natural images, entries in CNN feature maps are also spatially correlated, which hampers the effectiveness of dropout. Thus, SpatialDropout [26] proposed to randomly drop entire feature channels to concurrently remove spatially correlated entries (Fig. 2 (b)). Given a feature map $F \in \mathbb{R}^{n_{feats} \times w \times h}$, SpatialDropout samples n_{feats} binary values, indicating whether to drop each channel. Differently, we consider the l-hop neighboring entries as spatially correlated in the l-th layer. Then the l-hop neighboring entries are removed together with the seed entries (Fig. 2 (a)). Moreover, considering depth-wise correlation is another great difference. Instead of randomly selecting channels, we drop entries in channels depth-wisely correlated to the selected entries.

DropCluster vs. DropBlock DropBlock [9] focused on the spatial correlation in CNN feature maps and drops blocks of contiguous entries. Given a feature map, DropBlock drops entries in several randomly selected square regions of size $block\_size \times block\_size$ (Fig. 2 (b)). DropBlock cannot regularize GCNs as the square region cannot be defined on graphs. DropCluster deals with spatial correlation by concurrently removing entries within l-hop neighborhood of the random seed entries in the l-th layer. Compared to DropBlock, this not only differs in that it could be used in GCNs, but is also more delicate in that the

size of dropped region is increased with the number of layer, corresponding to the increased receptive field in deeper layers. Moreover, DropCluster also removes depth-wisely correlated entries, which is not considered in DropBlock.

DropCluster vs. DropEdge Similar to our work, DropEdge [20] is also specially for GCNs, but the approach is largely different. We aim to regularize GCNs by removing part of the information contained in the node features, while DropEdge proposed to randomly delete edges to sparsify the graph (Fig. 2 (a)). Given an adjacency matrix A denoting $|E|$ edges and the dropping rate as p, the adjacency matrix after dropping is $A_{drop} = A - A'$, where A' contains $|E| \cdot p$ randomly picked edges. In our experiments, DropEdge is also a baseline.

4 Experiments

In this section, we first test different hyperparameters to investigate their influence. Then we conduct ablation studies to verify the effectiveness of different parts of our model. After that, we compare our performance with other state-of-the-art techniques. To further demonstrate the effectiveness of our method in deep GCNs, we implement it to networks with increasing depths and compare the performance with dropout [11]. Finally, we implement our method to different network structures to show its generalization capability.

4.1 Datasets

Northwestern-UCLA [29] The Multiview 3D event dataset contains RGB, depth and human skeleton data captured simultaneously by three Kinect cameras. This dataset include 10 action categories: pick up with one hand, pick up with two hands, drop trash, walk around, sit down, stand up, donning, doffing, throw, carry. Each action is performed by 10 actors. This dataset contains data taken from a variety of viewpoints. Following the setting in [15], samples obtained from the first two cameras are used for training and samples from the third camera are used for testing.

NTU-RGB+D [21] NTU-RGB+D contains 56,880 samples from 60 classes, with modality of RGB videos, depth map sequences, 3D skeletal data, and infrared videos. Each human skeleton graph has 25 joints denoted by 3D coordinates (*X, Y, Z*). Following the recommendation of [21], we split the dataset by cross-subject (x-sub) and cross-view (x-view). In x-sub setting, 40,320 samples generated by one group of people serve as training set, and the other 16,560 samples by the other group of people are used for testing. In x-view setting, 37,920 samples captured by one set of cameras are used for training, and 18,960 samples captured by the other set of cameras are used for testing.

4.2 Implementation Details

On Northwestern-UCLA, we adopt ST-GCN [32] with modification as backbone and denote it as ST-GCN-U. ST-GCN-U has 9 layers. The first 3 layers have 32 channels for output, and the following 2 layers have 64 channels for output. The 6-th and 7-th have 128 channels for output, and the final 2 layers have 256 channels for output. DropCluster is implemented after each convolutional layer. The optimizer is SGD, with starting learning rate of 0.01. The model is trained for 100 epoch, and we decay the learning rate by 0.1 at epoch 20, 50, and 80.

On NTU-RGB+D, we adopt the original ST-GCN [32] with 9 layers as backbone. The first 3 layers have 64 channels for output, the following 3 layers have 128 channels for output, and the final 3 layers have 256 channels for output. DropCluster follows every convolutional layer. The optimizer is SGD with starting learning rate of 0.1. We train the model for 80 epochs, and decay the learning rate by 0.1 after epoch 10 and 50.

In Sec. 4.6, the 2-layer net has 64 input and 128 output channels for the 1-st layer, and the 2-nd layer has 128 channels for input and 256 for output. The 3-layer net has same first two layer structure as the 2-layer net, and the 3-rd layer has 256 channels for both input and output. The 4-layer net is built by inserting a layer with 128 channels for both input and output between the first two layers of the 3-layer net. For the 6-layer net, two identical layers with 64 channels for both input and output are inserted before the 1-st layer of the 4-layer net. Finally, the 11-layer net is based on the 9-layer net. The additional 10-th layer has 256 channels and 512 channels for input and output, and the final layer has 512 channels for both input and output.

4.3 Hyperparameter Analysis

In this part, we implement DropCluster to ST-GCN-U on Northwestern-UCLA with different dropping rate r_d and correlation threshold t_c. First we fix $t_c = 0.5$ and implement different dropping rates with the results shown in Fig. 3. The red curve in Fig. 3 shows DropCluster performance with different dropping rates. We see that DropCluster works well with smaller rates and degrades when the rate increases. To further investigate this phenomenon, we simultaneously show performance of dropout. The behaviour of DropCluster and dropout are similar in that the performance first increases with dropping rate and degrades after reaching a peak with an optimal rate. As both DropCluster and dropout regularize a network by eliminating part of the information, for any dropping method, it is reasonable to expect an optimal dropping rate corresponding to a proper amount of information eliminated. Thus, as DropCluster eliminates semantic information more effectively, enough information is removed with a small dropping rate. On the contrast, by zeroing entries in randomly, dropout needs a higher rate to remove enough information. Thus we observe a smaller optimal dropping rate for DropCluster and a higher one for dropout. Moreover, even if a higher dropping rate could remove enough information, it may hurt the data quality as there are too many hollows everywhere in the feature map.

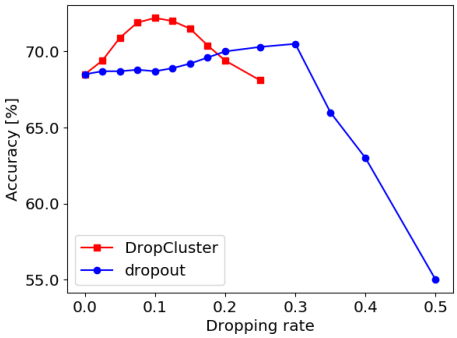

Fig. 3. Performance of DropCluster and dropout with different dropping rates on Northwestern-UCLA dataset

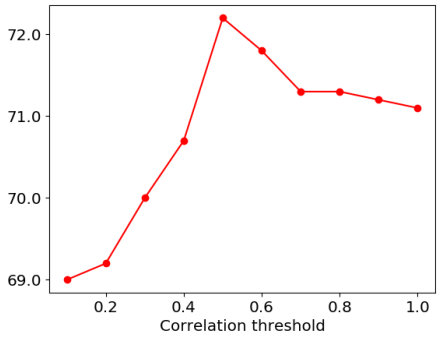

Fig. 4. Performance of DropCluster with different correlation threshold on Northwestern-UCLA dataset

Overall, DropCluster can eliminate enough information by zeroing a small part of the data and thus protect the data quality elsewhere. While in contrast, dropout needs higher dropping rate to achieve the same amount of information removal but also hurt data quality, resulting in a lower peak accuracy even with optimal dropping rate. Above, we set 0.1 as dropping rate for all experiments.

Then we vary t_c from 0 to 1, as shown in Fig. 4. According to Fig. 4, neither too small or too large threshold is promising, and 0.5 is approximately optimal. The correlation threshold determines the strength of co-dropping entries depth-wisely. Too high threshold causes some correlated channels unable to be also concurrently dropped, while too small ones render weakly correlated channels dropped concurrently. Above all, we set the threshold as 0.5 for all experiments.

4.4 Ablation Study

Spatial and depth-wise correlations are major concerns of our method. In this part, we implement two models that consider only spatial or depth-wise correlation to separately study them. The experiments are conducted on Northwestern-UCLA with ST-GCN-U as backbone.

Table 1. Ablation study on Northwestern-UCLA dataset

Method	Top-1	Top-5
Without dropping	68.5%	97.2%
Dropout	70.5%	97.3%
Spatial Correlation	71.1%	99.1%
Depth-wise Correlation	70.8%	97.5%
Full DropCluster	**72.2%**	**99.6%**

From Table 1, either considering sole spatial or depth-wise correlation improves the performance, but does not show strong superiority over dropout. The full DropCluster considering both correlation yields much better results, implying that both spatial and depth-wise correlations are significant in the GCN feature map and should be concurrently considered.

4.5 Comparisons with Other State-of-the-art Methods

In addition to dropout, in this part, we comprehensively compare DropCluster with various other state-of-the-art dropout variants as stated below. Gaussian dropout [23] generalized dropout by replacing the Bernoulli noise with Gaussian noise, and achieved equal or better performance than dropout. SpatialDropout [26] aimed at the spatial correlation in convolutional feature maps and randomly dropped entire channels in the feature map. Attention-based dropout [5] alternatively removes or highlights the most semantic areas with respect to a self-attention map denoting the distribution of semantic information. Jumpout [30] proposed modifications to dropout including monotone dropout rate, adapting dropout rate to number of activated neurons, and rescaling the output to work with batch normalization. None of these methods are specially for GCNs. DropEdge [20] randomly removes edges at each layer, aiming to alleviate the over-fitting and over-smoothing problems by sparsifying the graph. We carefully adjusted the hyperparameters to get the most out of them, and the results are listed in Table 2 and Table 3.

Table 2. Comparisons with state-of-the-art regularization methods on Northwestern-UCLA

Methods	Top-1	Top-5
Without dropping	68.5%	97.2%
Jumpout [30]	69.2%	97.0%
Attention-based dropout [5]	69.5%	97.1%
Gaussian dropout [23]	70.0%	97.0%
DropEdge [20]	70.3%	97.2%
Dropout [11]	70.5%	97.3%
SpatialDropout [26]	70.8%	97.4%
DropCluster	**72.2%**	**99.6%**

Table 3. Comparisons with state-of-the-art regularization methods on NTU-RGB+D dataset

Methods	X-Sub	X-View
Without dropping	80.6%	88.0%
Jumpout [30]	80.7%	87.9%
Attention-based dropout [5]	81.0%	88.1%
SpatialDropout [26]	81.2%	88.1%
DropEdge [20]	81.3%	88.4%
Dropout [11]	81.5%	88.3%
Gaussian dropout [23]	82.2%	88.4%
DropCluster	**83.3%**	**88.9%**

From Table 2 and 3, although most baselines are not specially designed for GCNs, they show promising performance. But DropCluster still significantly outperforms them. On Northwestern-UCLA, the performance is boosted by 3.7% with DropCluster, while the highest improvement obtained by comparison methods is 2.3% by SpatialDropout. On NTU-RGB+D, our method gets 2.3% and 0.9% improvement compared to the model without any dropping on two protocols, and the second highest improvement is 1.6% and 0.4% obtained by Gaussian dropout. The performance of DropCluster on NTU-RGB+D is lower than that on Northwestern-UCLA, which is mainly caused by volume difference of the datasets. NTU-RGB+D is 30 times larger than Northwestern-UCLA, thus the model is more prone to over-fitting on Northwestern-UCLA and would benefit more from regularization. Among all baselines, SpatialDropout is related to our method as it also considers spatial relationship between entries. It is similar to

DropCluster without considering depth-wise correlation, corresponding to 'Spatial Correlation' shown in Table 1. The performances of SpatialDropout and 'Spatial Correlation' are also similar with only 0.3% difference. The reason that SpatialDropout performs slightly worse could be that it drops the entire channels of the feature map, which causes severe information loss. On the contrast, for each channel, we drop clusters of nodes within the l-hop neighborhood of seed entries in the l-th layer, which flexibly removes only part of the information.

4.6 Implementation on Networks with Extended Depths

To further show DropCluster performs well in deep GCNs, we apply it to networks with increasing depths. We conduct experiments on NTU-RGB+D dataset with cross-view protocol. The results are shown in Tab. 4.

Table 4. Implementations of our DropCluster to networks with increasing depths

#layer	#para	w/o dropping	Dropout	Ours
2	0.37×10^6	81.1%	81.3%	81.4%
3	0.96×10^6	83.2%	83.3%	84.0%
4	1.11×10^6	86.4%	86.9%	87.7%
6	1.19×10^6	88.1%	88.4%	88.7%
9	1.96×10^6	88.1%	88.3%	88.9%
11	5.50×10^6	88.0%	88.6%	**90.0%**

Table 5. Implementation to GECNN and SLHM

Model	Regularizer	X-Sub	X-View
GECNN	-	83.6%	89.1%
GECNN	dropout	84.0%	89.4%
GECNN	DropCluster	**84.7%**	**90.4%**
SLHM	-	84.3%	89.2%
SLHM	dropout	84.7%	89.7%
SLHM	DropCluster	**85.3%**	**90.7%**

According to Tab. 4, the performance of the model without dropping first increases rapidly with depth then slows down, and stops at depth of 6, Moreover, the performance even gets lower when depth further increases. This phenomenon is reasonable. At the first stage wherein the depth increases from 2 to 6, the performance is boosted by the higher capacity of deeper networks. However, higher capacity also renders the network more prone to over-fitting. From Tab. 4, we see the parameters of 11-layer net is more than twice of the amount of 9-layer net, causeing much higher probability of over-fitting. With dropout, this is alleviated but a downwards trend still presents when depth increases from 6 to 9. With DropCluster, the improvement is amazing. It not only better regularizes the shallower networks, but also outperforms dropout by 1.3% when the depth is increased to 11. Above all, it is obvious that DropCluster is more effective at regularizing deep networks to better exploit their expressive potential.

4.7 Further Implementations

In this part, we implement DropCluster to more networks to further demonstrate its generalization capability. Specifically, we apply DropCluster to GECNN, SLHM and GCN-NAS with two independent branch models and a full model.

The experiments above are on GCNs that apply convolution on graph nodes. In this part, we first apply our method to GECNN and SLHM [36], which are different from the node-based GCN models in that the graph edges also participate in convolution. In GECNN, only edges are included in convolution, while in SLHM, both edges and nodes participate. Despite the difference in the convolution computation, the correlation between features are similar. Thus our method could be directly implemented, and the results are in Tab. 5.

From Tab. 5, DropCluster significantly improves both models. As we directly adopt the hyperparameters of DropCluster from the previous sections, the generalization ability of DropCluster is strongly demonstrated.

GCN-NAS [19] is a recent model that adopted neural architecture search to design GCN for skeleton-based action recognition. The obtained model has a joint- and a bone-stream. We implement DropCluster to both streams as well as the full model, and the results are shown in Tab. 6.

Table 6. Implementation to GCN-NAS on NTU-RGB+D dataset with cross-subject protocol

Backbone model	Stream	Regularization	Top-1	Top-5
GCN-NAS	Joint	-	87.5%	97.7%
GCN-NAS	Joint	DropCluster	88.0%	97.7%
GCN-NAS	Bone	-	87.5%	97.8%
GCN-NAS	Bone	DropCluster	88.1%	97.7%
GCN-NAS	Joint & Bone	-	89.4%	98.0%
GCN-NAS	Joint & Bone	DropCluster	**89.9%**	**98.2%**

From Tab. 6, we could see that the performance of GCN-NAS is significantly improved with the regularization of our proposed DropCluster.

Above all, DropCluster demonstrates strong generalization capability and could be easily implemented in different GCNs as an effective regularization without hyperparameter adjustments.

5 Conclusion

In this paper, we propose DropCluster, an effective method to regularize GCNs by considering spatial and depth-wise correlation between features. We implement it to deep GCNs with different structures on different datasets with comparisons to dropout and other variants. Moreover, we apply it to networks with increasing depths to further demonstrate its capability to regularize deep GCNs.

6 Acknowledgement

This work was supported by Australian Research Council Projects FL-170100117, DP-180103424, and DE-180101438

References

1. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 1993–2001 (2016)
2. Ba, J., Frey, B.: Adaptive dropout for training deep neural networks. In: Advances in Neural Information Processing Systems. pp. 3084–3092 (2013)
3. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs. arXiv preprint arXiv:1312.6203 (2013)
4. Chen, J., Ma, T., Xiao, C.: Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv preprint arXiv:1801.10247 (2018)
5. Choe, J., Shim, H.: Attention-based dropout layer for weakly supervised object localization. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2219–2228 (2019)
6. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems. pp. 3844–3852 (2016)
7. DeVries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552 (2017)
8. Gao, J., Zhang, T., Xu, C.: Graph convolutional tracking. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4649–4659 (2019)
9. Ghiasi, G., Lin, T.Y., Le, Q.V.: Dropblock: A regularization method for convolutional networks. In: Advances in Neural Information Processing Systems. pp. 10727–10737 (2018)
10. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in neural information processing systems. pp. 1024–1034 (2017)
11. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
12. Kang, G., Li, J., Tao, D.: Shakeout: A new regularized deep neural network training scheme. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
13. Kingma, D.P., Salimans, T., Welling, M.: Variational dropout and the local reparameterization trick. In: Advances in Neural Information Processing Systems. pp. 2575–2583 (2015)
14. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
15. Lee, I., Kim, D., Kang, S., Lee, S.: Ensemble deep learning for skeleton-based action recognition using temporal sliding lstm networks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1012–1020 (2017)
16. Levie, R., Monti, F., Bresson, X., Bronstein, M.M.: Cayleynets: Graph convolutional neural networks with complex rational spectral filters. IEEE Transactions on Signal Processing **67**(1), 97–109 (2018)
17. Micheli, A.: Neural network for graphs: A contextual constructive approach. IEEE Transactions on Neural Networks **20**(3), 498–511 (2009)
18. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: International Conference on Machine Learning. pp. 2014–2023 (2016)
19. Peng, W., Hong, X., Chen, H., Zhao, G.: Learning graph convolutional network for skeleton-based human action recognition by neural searching. arXiv preprint arXiv:1911.04131 (2019)

20. Rong, Y., Huang, W., Xu, T., Huang, J.: Dropedge: Towards deep graph convolutional networks on node classification. In: International Conference on Learning Representations (2019)
21. Shahroudy, A., Liu, J., Ng, T.T., Wang, G.: Ntu rgb+ d: A large scale dataset for 3d human activity analysis. arXiv preprint arXiv:1604.02808 (2016)
22. Shi, L., Zhang, Y., Cheng, J., Lu, H.: Skeleton-based action recognition with directed graph neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7912–7921 (2019)
23. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)
24. Stretcu, O., Viswanathan, K., Movshovitz-Attias, D., Platanios, E., Ravi, S., Tomkins, A.: Graph agreement models for semi-supervised learning. In: Advances in Neural Information Processing Systems. pp. 8710–8720 (2019)
25. Such, F.P., Sah, S., Dominguez, M.A., Pillai, S., Zhang, C., Michael, A., Cahill, N.D., Ptucha, R.: Robust spatial filtering with graph convolutional neural networks. IEEE Journal of Selected Topics in Signal Processing **11**(6), 884–896 (2017)
26. Tompson, J., Goroshin, R., Jain, A., LeCun, Y., Bregler, C.: Efficient object localization using convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 648–656 (2015)
27. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
28. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using dropconnect. In: International conference on machine learning. pp. 1058–1066 (2013)
29. Wang, J., Nie, X., Xia, Y., Wu, Y., Zhu, S.C.: Cross-view action modeling, learning and recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2649–2656 (2014)
30. Wang, S., Zhou, T., Bilmes, J.: Jumpout: Improved dropout for deep neural networks with relus. In: International Conference on Machine Learning. pp. 6668–6676 (2019)
31. Wang, X., Türetken, E., Fleuret, F., Fua, P.: Tracking interacting objects optimally using integer programming. In: European Conference on Computer Vision. pp. 17–32. Springer (2014)
32. Yan, S., Xiong, Y., Lin, D.: Spatial temporal graph convolutional networks for skeleton-based action recognition. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
33. You, J., Liu, B., Ying, Z., Pande, V., Leskovec, J.: Graph convolutional policy network for goal-directed molecular graph generation. In: Advances in neural information processing systems. pp. 6410–6421 (2018)
34. Zhai, K., Wang, H.: Adaptive dropout with rademacher complexity regularization (2018)
35. Zhang, M., Jiang, S., Cui, Z., Garnett, R., Chen, Y.: D-vae: A variational autoencoder for directed acyclic graphs. In: Advances in Neural Information Processing Systems. pp. 1586–1598 (2019)
36. Zhang, X., Xu, C., Tian, X., Tao, D.: Graph edge convolutional neural networks for skeleton-based action recognition. IEEE Transactions on Neural Networks and Learning Systems (2019)
37. Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., Gu, Q.: Layer-dependent importance sampling for training deep and large graph convolutional networks. In: Advances in Neural Information Processing Systems. pp. 11247–11256 (2019)