Graph Deformer Network

Wenting Zhao , **Yuan Fang** , **Zhen Cui*** , **Tong Zhang** and **Jian Yang**
Key Lab of Intelligent Perception and Systems for
High-Dimensional Information of Ministry of Education,
Jiangsu Key Lab of Image and Video Understanding for Social Security,
School of Computer Science and Engineering, Nanjing University of Science and Technology
{wtingzhao, fangyuan, zhen.cui, tong.zhang, csjyang}@njust.edu.cn

Abstract

Convolution learning on graphs draws increasing attention recently due to its potential applications to a large amount of irregular data. Most graph convolution methods leverage the plain summation/average aggregation to avoid the discrepancy of responses from isomorphic graphs. However, such an extreme collapsing way would result in a structural loss and signal entanglement of nodes, which further cause the degradation of the learning ability. In this paper, we propose a simple yet effective Graph Deformer Network (GDN) to fulfill anisotropic convolution filtering on graphs, analogous to the standard convolution operation on images. Local neighborhood subgraphs (acting like receptive fields) with different structures are deformed into a unified virtual space, coordinated by several anchor nodes. In the deformation process, we transfer components of nodes therein into affinitive anchors by learning their correlations, and build a multi-granularity feature space calibrated with anchors. Anisotropic convolutional kernels can be further performed over the anchor-coordinated space to well encode local variations of receptive fields. By parameterizing anchors and stacking coarsening layers, we build a graph deformer network in an end-to-end fashion. Theoretical analysis indicates its connection to previous work and shows the promising property of graph isomorphism testing. Extensive experiments on widely-used datasets validate the effectiveness of GDN in graph and node classifications.

1 Introduction

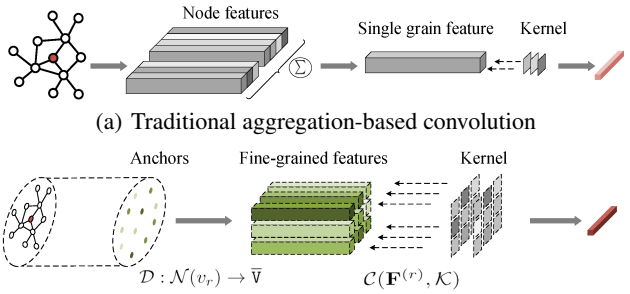
Graph is a flexible and universal data structure consisting of a set of nodes and edges, where nodes can represent any kind of object and edges indicate some relationship between a pair of nodes. Research on graphs is not only important in theory, but also beneficial to wide backgrounds of applications. Recently, advanced by the powerful representation capability of convolutional neural networks (CNNs) on grid-shaped data, the study of convolution on graphs is drawing increasing attention in the fields of artificial intelligence and data mining. So far, many graph convolution methods [Kipf and Welling, 2017; Hamilton *et al.*, 2017; Jiang *et al.*, 2019; Gao and Ji, 2019; Zhao *et al.*, 2019; Gao *et al.*, 2020; Xu *et al.*, 2020; Zhang, 2020; Hong *et al.*, 2021] have been proposed, and raise a promising direction.

The main challenge is the irregularity and complexity of graph topology, causing difficulty in constructing convolutional kernels. Most existing works take the plain summation or average aggregation scheme, and share a kernel for all nodes as shown in Fig. 1(a). However, there exist two non-ignorable weaknesses for them: i) losing the structure information of nodes in the local neighborhood, and ii) causing signal entanglements of nodes due to collapsing to one central node. Thereby, an accompanying problem is that the discriminative ability of node representation would be impaired, and further non-isomorphic graphs/subgraphs may produce the same responses.

Contrastively, in the standard convolutional kernel used for images, it is important to encode the variations of local receptive fields. For example, a 3×3 kernel on images can well encode local variations of 3×3 patches. An important reason is that the kernel is anisotropic to spacial positions, where each pixel position is assigned to a different mapping. However, due to the irregularity of graphs, defining and operating such an anisotropic kernel on graphs are intractable. To deal with this problem, Niepert et al. [Niepert *et al.*, 2016] attempted to sort and prune neighboring nodes, and then run different kernels on the ranked size-fixed nodes. However, this deterministic method is sensitive to node ranking and more prone to being affected by graph noises. Furthermore, some graph convolution methods [Veličković *et al.*, 2018; Wang *et al.*, 2019] introduce an attention mechanism to learn the importance of nodes. Such methods emphasize mining those significant structures/features rather than designing anisotropic convolution kernels, so they cannot well represent local variations of structures in essence.

In this work, we propose a novel yet effective graph deformer network (GDN) to implement anisotropic convolutional filtering on graphs as shown in Fig. 1(b). Inspired by image-based convolution, we deform local neighborhoods of different sizes into a virtual coordinate space, implicitly

*Contact Author



(a) Traditional aggregation-based convolution

(b) Graph deformer convolution

Figure 1: An illustration of difference between the previous convolution and our proposed graph deformer convolution. The red node is a reference node. (a) In traditional graph convolution, the convolution kernel is shared for all nodes due to the plain aggregation over all nodes in the neighborhood. (b) In our method, the irregular neighborhood is deformed into a unified anchor space, which is a pseudo-grid shape, and then the anisotropic fully-connected convolution kernel can be used to encode the variations of deformable features.

spanned by several anchor nodes, where each space granularity corresponds to one anchor node. In order to perform space transformation, we define the correlations between neighbors and anchor nodes, and project neighboring nodes into the regular anchor space. Thereby, irregular neighborhoods are deformed into the anchor-coordinated space. Then, the image-like anisotropic convolution kernels can be imposed on the anchor-coordinated plane, and local variations of neighborhoods can be perceived effectively. Due to the importance of anchors, we also deform anchor nodes with adaptive parameters to match the feature space of nodes. As anisotropic convolution kernels are endowed with the fine-grained encoding ability, our method can better perceive subtle variations of local neighborhood regions as well as reduce signal confusion. We also show its connection to previous work, and theoretically analyze the stronger expressive power and the satisfactory property of the isomorphism test. Extensive experiments on graph/node classification further demonstrate the effectiveness of the proposed GDN.

2 Our Approach

In this section, we elaborate on the proposed graph deformer method. Below we first give an abstract formulation for our method and then elaborate on the details.

Denote $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as an undirected graph , where $\mathcal{V}$ represents a set of nodes with $|\mathcal{V}| = n$ and $\mathcal{E}$ is a set of edges with $|\mathcal{E}| = e$. According to the link relations in $\mathcal{E}$, the corresponding adjacency matrix can be defined as $\mathbf{A} \in \mathbb{R}^{n\times n}$. And $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the feature matrix. To state conveniently, we use $\mathbf{X}_{i\cdot}$ or $\mathbf{x}_i$ to denote the feature of the i-th node. Besides, for a node v_i, the first-order neighborhood consists of nodes directly connected to v_i, which is denoted as $\mathcal{N}^1_{v_i} = \{v_j | (v_j, v_i) \in \mathcal{E}\}$. Accordingly, we can define s-order neighborhood $\mathcal{N}^s_{v_i}$ as the set of s-hop reachable nodes.

2.1 A Basic Formulation

Given a reference node v_r in graph $\mathcal{G}$, we need to learn its representation based on the node itself as well as its contextual neighborhood $\mathcal{N}_{v_r}$. However, the irregularity causes difficulty in designing anisotropic spatial convolution. To address this problem, we introduce anchor nodes to deform the neighborhood. All neighboring nodes are calibrated into a pseudo space spanned by anchors. We denote the set of anchor nodes by $\overline{\mathbb{V}} = \{\overline{\mathbb{v}}_0, \overline{\mathbb{v}}_1, ..., \overline{\mathbb{v}}_{m-1}\}$. The convolution on $\mathcal{N}_{v_r}$ is formulated as:

$$\widetilde{\mathbf{x}}_r = (\mathcal{G} * f)(v_r) = \mathcal{C}(\mathbf{F}^{(r)}, \mathcal{K}), \tag{1}$$

$$\mathbf{F}^{(r)}_i = \sum_{v_t \in \mathcal{N}_{v_r}} \mathcal{D}_{v_t \to \overline{\mathbb{v}}_i}(\overline{\mathbb{x}}_i, \mathbf{x}_t, \Theta), \tag{2}$$

where

- $\overline{\mathbb{v}}_\cdot, \overline{\mathbb{x}}_\cdot$: an anchor node and a pseudo coordinate vector (a.k.a. feature vector). Please see Section 2.2 for anchor generation.
- $\mathcal{D}$: the deformer function. It transforms node v_t into a virtual coordinate space spanned by anchors. Θ is the deformer parameter to be learned. Please see Section 2.3 for details.
- $\mathbf{F}^{(r)} \in \mathbb{R}^{m\times d}$: the deformed multi-granularity feature from the neighborhood of node v_r. Each granularity $\mathbf{F}^{(r)}_i$ corresponds to an anchor node $\overline{\mathbb{v}}_i$.
- $\mathcal{C}, \mathcal{K}$: the anisotropic convolution operation on anchor space and convolution kernel. $\mathcal{G} * f$ represents filter f acting on graph $\mathcal{G}$. The relationship between anchor nodes can be built by some metrics such as Cosine distance, and anchor nodes may be formatted as a pseudo 2-D grid just like the patch in images. Please see the details in Section 2.3.

2.2 Anchor Generation

Anchor nodes are crucial to the graph convolution process because neighborhood regions are unitedly calibrated with them. Rigid anchors will not adapt to the variations of the feature space during convolution learning. Thus we choose to optimize anchor nodes as one part of the entire network learning. In the beginning, we cluster nodes randomly sampled from the graph as initial anchors. When enough anchors cover the space of neighborhood nodes, the anchors can be endowed with a strong expressive ability to encode neighborhoods like a code dictionary. Formally, we use the K-means clustering to generate initial anchors,

$$\overline{\mathcal{V}} \leftarrow \text{Clustering}\ \{(v_i, \mathbf{x}_i) | v_i \in \mathcal{V}_{\text{sampling}}\}, \tag{3}$$

where $\mathcal{V}_{\text{sampling}}$ is the sampled node set, in which each node is randomly sampled from the graph, $\overline{\mathcal{V}} = \{(\overline{\mathbb{v}}_k, \overline{\mathbb{x}}_k)\}|^{m-1}_{k=0}$ is the initial anchor set generated by clustering, in which $\overline{\mathbb{v}}_k$ represents k-th anchor node and $\overline{\mathbb{x}}_k$ represents its feature vector, m is the number of anchor nodes. Note when given anchor nodes, the response of our method will be invariant to permuted nodes of one graph during the training as well as testing stage. The clustering algorithm might affect the final anchors due to random sampling for initialization, but it cannot affect the property of permutation invariance, which just like random initialization on the network parameters.

Due to the sparsity of graphs, in practice, several anchors are sufficient to encode each neighborhood. To better collaborate with node/feature variations during graph convolution learning, we transform the initial anchors into a proper space by parameterizing them:

$$\overline{\mathbf{a}}_k = \mathrm{ReLU}\left(\mathbf{W}_A\overline{\mathbf{x}}_k + \mathbf{b}_A\right), \quad k = 0, 1, \cdots, m-1, \quad (4)$$

where $\mathbf{W}_A, \mathbf{b}_A$ are the learnable parameters, and ReLU is the classic activation function. Besides, other flexible multi-layer networks may be selected to learn deformable anchors.

2.3 Deformer Convolution

Space Transformation

Now we define the deformer function $\mathcal{D}$ in Eqn. (2), transforming neighbor nodes to the anchor space. For each node $v_j \in \mathcal{N}_{v_r}$ ($n_r = |\mathcal{N}_{v_r}|$), we derive the anchor-related feature (also query feature) and value feature vectors as

$$\mathbf{q}_j = \mathrm{ReLU}\left(\mathbf{W}_Q\mathbf{x}_j + \mathbf{b}_Q\right), \quad j = 0, 1, \cdots, n_r - 1, \quad (5)$$

$$\mathbf{u}_j = \mathrm{ReLU}\left(\mathbf{W}_U\mathbf{x}_j + \mathbf{b}_U\right), \quad j = 0, 1, \cdots, n_r - 1, \quad (6)$$

where $\mathbf{W}_Q, \mathbf{W}_U$ are the learnable weight matrices, and $\mathbf{b}_Q, \mathbf{b}_U$ are the biases. The query feature $\mathbf{q}_j$ indicates how to transform v_j to the anchor space by interacting with anchors, and the value vector $\mathbf{u}_j$ is the transformable component to the anchor space.

For the neighborhood $\mathcal{N}_{v_r}$ of node v_r, the correlation to anchors defines a set of weights $\alpha = \{\alpha_{0,0}, \cdots, \alpha_{0,m-1}, \cdots, \alpha_{n_r-1,0}, \cdots, \alpha_{n_r-1,m-1}\}$, which measures the scores of all nodes within the neighborhood projected onto the directions of anchor nodes. Formally,

$$\alpha_{j,k} = \frac{\exp(\langle \mathbf{q}_j, \overline{\mathbf{a}}_k\rangle/\sqrt{d})}{\sum_{k'}\exp(\langle \mathbf{q}_j, \overline{\mathbf{a}}_{k'}\rangle/\sqrt{d})}, \quad k' = 0, 1, \cdots, m-1, \quad (7)$$

where $\langle\cdot,\cdot\rangle$ denotes the inner production, $\frac{1}{\sqrt{d}}$ is the scaling factor as [Vaswani *et al.*, 2017], then normalization is done by softmax function. $\alpha_{j,k}$ may be viewed as the attention score of the node v_j w.r.t. the anchor $\overline{\mathrm{v}}_k$. After obtaining the attention score, the irregular neighborhood can be transformed into the anchor-coordinated space,

$$\widetilde{\mathbf{u}}_k = \sum_j \alpha_{j,k}\mathbf{u}_j, \quad j = 0, 1, \cdots, n_r - 1. \quad (8)$$

The deformed components are accumulated on each anchor, and form the final deformed features. Thus, any neighborhood with different sizes can be deformed into the virtual normalized space coordinated by anchors. In experiments, for simplicity, the query feature and value feature are shared with the same parameters in Eqns. (5) and (6).

Anisotropic Convolution in the Anchor Space

Afterward, the s-hop neighborhood of node v_r is deformed into the size-fixed anchor space, i.e., $\mathcal{N}^s_{v_r} \rightarrow \{\widetilde{\mathbf{u}}_0, \widetilde{\mathbf{u}}_1, \cdots, \widetilde{\mathbf{u}}_{m-1}\}$. The anisotropic graph convolution can be implemented by imposing different mapping on each anchor as

$$\widetilde{\mathbf{x}}^{(s)}_r = \mathrm{ReLU}\left(\sum_i \mathcal{K}_i^\intercal\widetilde{\mathbf{u}}_i + \mathbf{b}\right), \quad i = 0, 1, \cdots, m-1, \quad (9)$$

where $\widetilde{\mathbf{x}}^{(s)}_r \in \mathbb{R}^{d'}$, the matrix $\mathcal{K}_i$ is a $d \times d'$ weight imposed on the features w.r.t. an anchor, and $\mathbf{b}$ is the bias vector.

In the convolution process, different filter weights are imposed on different features of anchor nodes, which is an anisotropic filtering operation. In contrast to the traditional aggregation method, the deformer convolution has two aspects of advantages: i) well preserving structure information and reducing signal entanglement; ii) transforming different-sized neighborhoods into the size-fixed anchor space to well advocate anisotropic convolution like the standard convolution on images.

Multi-Scale Extension

Intuitively, the first-order neighborhood is necessary to be used for node aggregation, because it indicates that two nodes linked by an edge are always similar. However, real-world graphs are often so sparse, and there exist many nodes that are similar to each other but not linked by direct edges. The first-order neighborhood alone is not sufficient for extracting useful features and preserving the structural information. It is natural to incorporate higher-order proximity to capture more information. Generally, second-order information is sufficient as most works [Abu-El-Haija *et al.*, 2019; Wang *et al.*, 2016; Tang *et al.*, 2015]. Higher-order information can also be considered, but the computational complexity will increase. It can be understood as a trade-off of expressive ability and computational complexity. In this paper, we consider both first-order and second-order neighborhoods. Specifically, we deform both first-order and second-order neighborhoods into feature space represented by anchor nodes, and convolve over them respectively. Then the learned different neighborhood representations and the original node feature are concatenated as the final filtering response,

$$\widetilde{\mathbf{x}}_r \leftarrow [\mathbf{x}_r; \widetilde{\mathbf{x}}^{(1)}_r; \widetilde{\mathbf{x}}^{(2)}_r], \quad (10)$$

where $\widetilde{\mathbf{x}}_r$ denotes the convolution response on the s-order neighborhood $\mathcal{N}^s_{v_r}$ of node v_r. Further, we can stack multiply layers to extract robust features on larger receptive fields.

2.4 Graph Coarsening

Similar to the pooling in standard CNN, the coarsening on graphs could enlarge the receptive field of nodes as well as reduce the computation cost. Below we simply introduce graph coarsening used here.

Graph classification. We employ the graph cut used in [Jiang *et al.*, 2019] to partition an entire graph into several subgraphs. During graph coarsening, a binary cluster matrix $\mathbf{Z} \in \mathbb{R}^{n\times c}$ is obtained, where only one element in each row is non-zero, i.e., $Z_{ic} = 1$ when the vertex v_i falls into the cluster c. Then the adjacency matrix and feature matrix of the input graph are transformed into

$$\mathbf{A}' \leftarrow \mathbf{Z}^\intercal\mathbf{A}\mathbf{Z}, \quad \mathbf{X}' \leftarrow \mathbf{Z}^\intercal \otimes \widetilde{\mathbf{X}}, \quad (11)$$

where $\otimes$ represents a max operation. The output can be used as the input of the next convolutional layer. Then the graph convolution and coarsening can be alternatingly stacked into a deep network.

Node classification. We do not need to remove nodes for node classification task. The pooling is node-wise diffusion on a local receptive field, and can be performed over multi-scale neighborhoods. The pooling over S scale neighborhoods w.r.t. the reference node v_r is

$$\mathcal{P}(\mathcal{G}(v_r)) = \mathcal{P}(\{\mathbf{x}_j | v_j \in \mathcal{N}_{v_r}^s, s = 1, \cdots, S\}), \quad (12)$$

where the pooling $\mathcal{P}$ is usually defined as "max" or "mean". In practice, their performance has little difference, so we choose the mean operation in experiments.

2.5 Neural Network Training

For graph classification, we adopt the cross-entropy as cost function to learn graph representation. However, in the scenario of semi-supervised node classification, a main limitation is that a small portion of nodes is annotated as the training set. Our aim is to use labeled nodes as well as graph structure to train a model with a good generalization ability. A straightforward way is to add a regularization term to avoid overfitting. To this end, we employ a global consistency constraint through positive pointwise mutual information as used in [Zhuang and Ma, 2018] to regularize the loss function. Finally, backpropagation algorithm is used to compute the gradient and the stochastic gradient descent algorithm updates neural network parameters.

2.6 Computational Complexity

For the computational complexity, we analyze the main module of graph convolution. In one-layer convolution, GCN [Kipf and Welling, 2017] is about $\mathcal{O}(edS + ndd'S)$, where n, e are the numbers of nodes and edges, S is the scale of the neighborhood, and d, d' are the dimensions of the input/hidden layers. For our GDN model, the computational complexity is mainly from two parts, i.e., "Space Transformation" and "Convolution in Anchor Space", which are about $\mathcal{O}(emd^2S)$ and $\mathcal{O}(nmdd'S)$, respectively, where m is the number of anchor nodes. Thus, the total computational complexity is $\mathcal{O}(emd^2S + nmdd'S)$, which is linearly proportional to GCN with the factor m when d and d' have the same order number.

3 Theoretical Analysis

We present a theoretical analysis about the expressive power of several aggregation methods including the mean, sum, and graph deformer operation. Inspired by [Xu *et al.*, 2019], we evaluate them by verifying whether graphs are isomorphic. Then, we give the following propositions.

Proposition 1 *There exists a set of network parameters that make two non-isomorphic graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ can be distinguished by graph deformer process, but cannot be distinguished by mean/sum aggregation.*

Proposition 2 *The proposed anisotropic graph deformer convolution can be as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test.*

The proofs of the above two propositions can be found in the supplementary material[1]. We can conclude that the expressive power of the proposed graph deformer network is provably stronger than mean/sum aggregation, which can accomplish an injective mapping as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test.

[1]https://github.com/wtzhao1631/gdn

4 Experiments

In this section, we carry out extensive experiments to assess the proposed GDN model on both graph and node classification tasks. For node classification, three citation graphs are used: Cora, Citeseer, and Pubmed. We adopt the data preprocessed in the work [Yang *et al.*, 2016], and follow its data partitioning rules. For graph classification, we adopt eight datasets [Jiang *et al.*, 2019] to assess our GDN method: MUTAG, PTC, NCI1, PROTEINS, ENZYMES, IMDB-BINARY, IMDB-MULTI, and REDDIT-MULTI-12K.

4.1 Experimental Setups

Graph classification. A three-layer GDN model is applied for the learning of overall graphs. Each convolutional layer is followed by a pooling/coarsening layer with a downsampling rate of 0.5. The channels of the three convolutional layers are set to $\{64, 128, 256\}$, respectively. Finally, a fully-connected layer with the softmax function directly predicts the label. The network structure can be simply represented as $\text{Input} - \mathcal{C}(64) - \mathcal{P}(0.5) - \mathcal{C}(128) - \mathcal{P}(0.5) - \mathcal{C}(256) - \mathcal{P}(all) - \mathcal{FC}(\text{softmax}) - \text{Output}$, where $\mathcal{C}, \mathcal{P}$ and $\mathcal{FC}$ denote convolution, pooling/coarsening, and fully connected layer, respectively. $\mathcal{P}(all)$ means that only a supernode is retained at the last pooling layer. The number of anchor nodes is set to 16. The scale of the neighborhood is set to 2, i.e, including the features of the node itself, first-order and second-order neighborhoods. We randomly divide the dataset with the proportion of 9:1, where 9 folds are as the training set and the remaining 1 fold is as the testing set. The accuracies are reported in terms of "mean $\pm$ standard deviation" of 10-fold cross-validation. We adopt Momentum optimizer to train the network for 500 epochs, where its batch size, initial learning rate, decay rate and momentum are set to 128, 0.05, 0.95 and 0.9, respectively. The dropout rate is set to 0.5, and the ReLU unit is leveraged as a nonlinear activation function.

Node classification. Two convolutional layers are used to form the network architecture, each of which is followed by a pooling layer. Then a fully connected layer and an output layer with the softmax function generate the final predictions. The network structure can be simply represented as $\text{Input} - \mathcal{C} - \mathcal{P}(mean) - \mathcal{C} - \mathcal{P}(mean) - \mathcal{FC}(\text{softmax}) - \text{Output}$, where $\mathcal{P}(mean)$ indicates the "$mean$" operation in the pooling layer. Also, the number of anchor nodes is set to 16 and the scale of the neighborhood is set to 2. We adopt Adam optimizer to train the model for 500 epochs with an initial learning rate of 0.05, decay rate of 0.95. The dropout rate is set to 0.5 and the ReLU unit is nonlinear activation function. We run 10 experiments to take the averaged accuracy "mean$\pm$standard deviation" as the metric to measure the performance.

4.2 Comparison with State-of-the-arts

Graph classification. Table 1 shows the results on graph classification. Overall, our GDN approach achieves state-of-the-art or comparable performance and obtains remarkable

Method	MUTAG	PTC	NCI1	ENZYMES	PROTEINS	IMDB-BINARY	IMDB-MULTI	REDDIT-MULTI-12K
WL	80.72 ± 3.00	56.97 ± 2.01	80.13 ± 0.50	53.15 ± 1.14	72.92 ± 0.56	72.86 ± 0.76	50.55 ± 0.55	-
GK	81.66 ± 2.11	57.26 ± 1.41	62.28 ± 0.29	26.61 ± 0.99	71.67 ± 0.55	65.87 ± 0.98	43.89 ± 0.38	31.82 ± 0.08
DGK	82.66 ± 1.45	57.32 ± 1.13	62.48 ± 0.25	27.08 ± 0.79	71.68 ± 0.50	66.96 ± 0.56	44.55 ± 0.52	32.22 ± 0.10
PSCN	92.63 ± 4.21	60.00 ± 4.82	78.59 ± 1.89	-	75.89 ± 2.76	71.00 ± 2.29	45.23 ± 2.84	41.32 ± 0.42
NgramCNN	94.99 ± 5.63	68.57 ± 1.72	-	-	75.96 ± 2.98	71.66 ± 2.71	50.66 ± 4.10	-
IGN	84.61 ± 10	59.47 ± 7.3	73.71 ± 2.6	-	75.19 ± 4.3	71.27 ± 4.5	48.55 ± 3.9	-
PPGN	90.55 ± 8.7	66.17 ± 6.54	83.19 ± 1.11	-	77.2 ± 4.73	72.6 ± 4.9	50 ± 3.15	-
GNTK	90.0 ± 8.5	67.9 ± 6.9	84.2 ± 1.5	-	75.6 ± 4.2	76.9 ± 3.6	52.8 ± 4.6	-
CapsGNN	86.67 ± 6.88	-	78.35 ± 1.55	54.67 ± 5.67	76.28 ± 3.63	73.10 ± 4.83	50.27 ± 2.65	$\mathbf{46.62 \pm 1.9}$
GIC	94.44 ± 4.43	$\mathbf{77.64 \pm 6.98}$	84.08 ± 1.77	62.50 ± 5.12	77.65 ± 3.21	76.70 ± 3.25	51.66 ± 3.40	42.98 ± 0.87
GIN	89.4 ± 5.6	64.6 ± 7.0	82.7 ± 1.7	-	76.2 ± 2.8	75.1 ± 5.1	52.3 ± 2.8	-
GDN (3L)	$\mathbf{97.39 \pm 2.65}$	75.57 ± 7.56	$\mathbf{86.03 \pm 1.23}$	$\mathbf{67.5 \pm 6.96}$	$\mathbf{81.32 \pm 3.09}$	$\mathbf{79.3 \pm 3.26}$	$\mathbf{55.2 \pm 4.34}$	42.0 ± 1.9

Table 1: Comparison with state-of-the-arts on graph classification. The number in parentheses $(*)$ denotes the number of convolutional layers in the neural network.

Method	Cora	Citeseer	Pubmed
GCN	81.5	70.3	79.0
GAT	83.0	72.5	79.0
DGCN	83.5	72.6	80.0
JK-Net	79.71 ± 0.62	69.03 ± 0.55	78.17 ± 0.27
GIN	79.49 ± 0.65	67.78 ± 0.89	78.37 ± 0.29
GraphNAS	83.7	73.5	80.5
g-U-Nets	84.4 ± 0.6	73.2 ± 0.5	79.6 ± 0.2
GRAPH-BERT	84.3	71.2	79.3
DiffNet	85.1	72.7	78.3
GDN(1L)	$\mathbf{85.16 \pm 0.47}$	73.13 ± 0.83	79.8 ± 0.30
GDN(2L)	84.76 ± 0.59	$\mathbf{73.77 \pm 0.45}$	$\mathbf{80.77 \pm 0.24}$

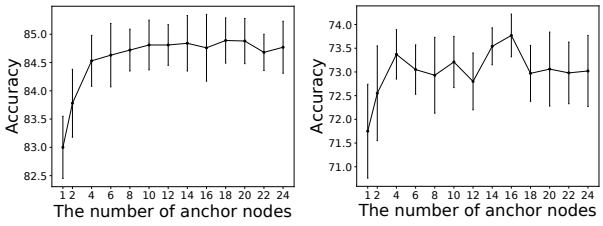
Table 2: Comparison with state-of-the-arts on node classification.

improvement on most datasets. For graph kernel-based methods (WL [Shervashidze *et al.*, 2011], GK [Shervashidze *et al.*, 2009] and DGK [Yanardag and Vishwanathan, 2015]), we can observe the WL kernel can obtain better results on most datasets than GK and DGK. In contrast to WL, the proposed GDN is able to improve by a large margin of 5.9% on NCI1, 14.35% on ENZYMES, 6.44% on IMDB-BINARY, etc. Recently the neural network-based works (PSCN [Niepert *et al.*, 2016], NgramCNN [Luo *et al.*, 2017], IGN [Maron *et al.*, 2018], PPGN [Maron *et al.*, 2019], GNTK [Du *et al.*, 2019], CapsGNN [Xinyi and Chen, 2018], GIC [Jiang *et al.*, 2019], GIN [Xu *et al.*, 2019]) are superior to traditional machine learning methods. Compared to GIC, the GDN model still achieves superior performances, about 3 percentages on average, although a relatively lower result is gotten on the PTC dataset. This may be attributed to differences in the dataset or less appropriate parameter settings. Compared with these baseline methods, GDN can render impressive performance.

Node classification. We compare the performance of GDN against several baseline works: GCN [Kipf and Welling, 2017], GAT [Veličković *et al.*, 2018], DGCN [Zhuang and Ma, 2018], JK-Net [Xu *et al.*, 2018], GIN [Xu *et al.*, 2019], GraphNAS [Gao *et al.*, 2020], g-U-Nets [Gao and Ji, 2019], GRAPH-BERT [Zhang *et al.*, 2020] and DiffNet [Zhang, 2020]. The accuracies are reported in Table 2, indicating that our GDN obtains a remarkable improvement. Compared to GCN, GAT, JK-Net, and GIN, our GDN achieves a relatively large gain. We attribute this improvement to the graph deformer convolution. Though GDN utilizes global consistency constraint, it still obtains better results than DGCN. Compared to recent methods GraphNAS, g-U-Nets, GRAPH-BERT, and DiffNet, GDN still achieves superior performance on these three datasets. These demonstrate that the proposed GDN method performs well on various graph datasets by building the graph deformer process, where structure variations can be well captured and fine-grained node features can be extracted to enhance the discriminability between nodes. In summary, the remarkable gains indicate that the proposed GDN is effective to deal with graph classification.

Method	Cora	Citeseer	Pubmed
GDN-$\mathcal{N}^{(0)}$	54.95 ± 2.00	55.7 ± 1.26	72.94 ± 1.58
GDN-$\mathcal{N}^{(0,1)}$	82.62 ± 0.67	72.56 ± 0.95	80.24 ± 0.22
GDN-$\mathcal{N}^{(0,1,2)}$	$\mathbf{84.76 \pm 0.59}$	$\mathbf{73.77 \pm 0.45}$	$\mathbf{80.77 \pm 0.24}$

Table 3: Comparison on the scales of neighborhood regions.



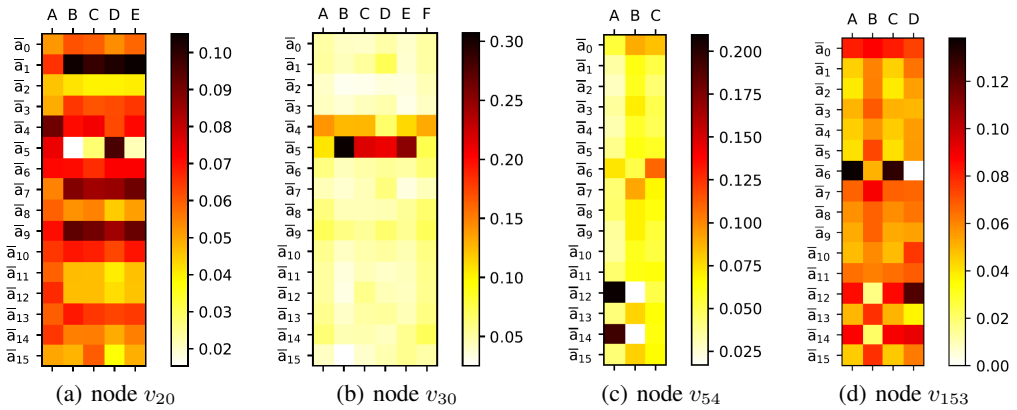
(a) Cora (b) Citeseer

Figure 2: Comparison on the number m of anchor nodes.

4.3 Ablation Study

The scale s of neighborhood region. The influences of neighborhood scales are reported in Table 3. GDN-$\mathcal{N}^{(0)}$ denotes that only the feature of the node itself is used, GDN-$\mathcal{N}^{(0,1)}$ includes the features of the node itself and first-order neighborhood, and so on. Due to the lack of structural information, we find that the performance of GDN-$\mathcal{N}^{(0)}$ is obviously lower. As more information is considered, the accuracy of GDN-$\mathcal{N}^{(0,1,2)}$ is generally superior to GDN-$\mathcal{N}^{(0,1)}$. This validates the importance of local neighborhood information, which is also a crucial property of traditional CNNs.

The number m of anchor nodes. We select the value m in the range $[1, 25]$ to observe the changes of performance. As shown in Fig. 2, the performance of GDN indeed varies with the number of anchor nodes, but the accuracy trend is generally similar on different datasets (increasing first, then turning to degrade or be stable). Specifically, when $m = 1$, the accuracies are significantly lower, because only one anchor node is used, which is similar to sum aggregation. When $m = 2$, the performance is also relatively lower, 2 anchor nodes are insufficient to capture more variations. Then, the performance is relatively stable with m increasing. As real-world graph data is rather sparse, e.g., on average about 2



(a) node v_{20} (b) node v_{30} (c) node v_{54} (d) node v_{153}

Figure 3: Visualization of attention scores in the first-order neighborhood of nodes v_{20} and v_{30} in the Cora, and v_{54} and v_{153} in the Citeseer dataset. (a) Node v_{20} has 5 neighbors while (b) node v_{30} has 6 neighbors. (c) Node v_{54} has 3 neighbors while (d) node v_{153} has 4 neighbors.

edges for each node in Cora and Citeseer datasets, so a few anchors matching the neighborhood size should be saturated to represent the variations without the information loss.

Graph convolution and graph coarsening. We further explore the effectiveness of graph convolution by removing the pooling layer from the GDN model, named as "GDN w/o $\mathcal{P}$". Similarly, "GDN w/o $\mathcal{C}$" means that graph convolution is removed. Table 4 shows the performance on citation datasets. Compared to GDN, both "GDN w/o $\mathcal{P}$" and "GDN w/o $\mathcal{C}$" obtain lower performances. But "GDN w/o $\mathcal{P}$" is better than "GDN w/o $\mathcal{C}$", while they are comparable on the Pubmed dataset. It indicates that the deformer convolution indeed improves the discriminability of nodes. Note that "GDN w/o $\mathcal{C}$" has no convolutional layer, but average aggregation still is performed in the pooling layer for node classification.

Attention scores α. We visualize correlation scores α between neighborhood nodes and anchors. We respectively select some nodes from Cora and Citeseer datasets as center nodes and compute the scores of their first-order neighbors to anchor nodes. The neighbors have different emphasis on anchor nodes. As shown in Fig. 3, for node v_{20} in Cora, the attention score of the neighbor "A" on anchor node $\overline{\mathbf{a}}_4$ is largest while the other four neighbors is close to anchor node $\overline{\mathbf{a}}_1$. For node v_{153} in Citeseer, the neighbors "A" and "C" are more inclined to anchor node $\overline{\mathbf{a}}_6$, while neighbor "D" prefers $\overline{\mathbf{a}}_{12}$, and neighbor "B" is similar in most directions. This shows that neighbors of center nodes indeed place different emphasis on anchor nodes, so different proportions of features are assigned to the directions of these anchors, and then anisotropic convolution can be used to extract more fined-grained representation, which is superior to the sum/mean aggregation.

5 Discussion and Conclusion

How to set depth and width of the neural network?

The depth of network (i.e., layer number) and the width of filtering (i.e., order number) has some relation to the diameter of the graph. Usually, the total receptive field size of the top layer in the convolution network may be upper-bounded by the diameter of the graph. For a fixed receptive field size, we may employ a deeper network (stacking multiple layers) with small order (small width), or a shallow network with large order number (large width). Like numerous standard CNN, deeper networks (e.g., ResNet) with a small width (small kernel size) usually have better performance than shallow network with a large width.

Method	Cora	Citeseer	Pubmed
GDN w/o $\mathcal{P}$	83.97 ± 0.39	72.53 ± 0.59	78.86 ± 0.90
GDN w/o $\mathcal{C}$	82.82 ± 0.72	71.57 ± 0.83	78.75 ± 0.62
GDN	$\mathbf{84.76 \pm 0.59}$	$\mathbf{73.77 \pm 0.45}$	$\mathbf{80.77 \pm 0.24}$

Table 4: The effect of convolutional layer $\mathcal{C}$ and pooling layer $\mathcal{P}$ in our GDN method.

How the problem of over-smoothness is handled?

Specifically, we introduce anchor nodes generated by imposing a non-linear transformation on cluster centers to match the updated features. Then, we project local neighborhoods with different structures onto multiple directions (w.r.t. anchors). Anisotropic convolution kernels can thus be performed over the anchor-coordinated space to well encode subtle variations of local neighborhoods. The anisotropic convolution realizes weight sharing for different local neighborhoods, which not only maintains computational efficiency, but also able to capture some common property between local neighborhoods just like the standard convolution kernel on images, and helps improve the expressive power of model.

Summarily, analogous to the standard convolution on images, we present a novel yet effective graph deformer network (GDN) to fulfill anisotropic convolution filtering on graphs. Further, we build a graph deformer network in an end-to-end learning fashion by stacking the deformable convolutional layers as well as the coarsening layers. Our proposed GDN archives significantly better performances on graph and node classifications. In the future, we will extend the GDN to more applications in the real world, such as link prediction, heterogeneous graph analysis, etc.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grants Nos., 62072244 and 61906094), the Natural Science Foundation of Jiangsu Province (Grant BK20190019), the Natural Science Foundation of Shandong Province (Grant No. ZR2020LZH008), and in part by State Key Laboratory of High-end Server & Storage Technology.

References

[Abu-El-Haija *et al.*, 2019] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *ICML*, pages 21–29, 2019.

[Du *et al.*, 2019] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *NeurIPS*, 2019.

[Gao and Ji, 2019] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *ICML*, pages 2083–2092, 2019.

[Gao *et al.*, 2020] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *IJCAI*, volume 20, pages 1403–1409, 2020.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pages 1024–1034, 2017.

[Hong *et al.*, 2021] Xiaobin Hong, Tong Zhang, Zhen Cui, Yuge Huang, Pengcheng Shen, Shaoxin Li, and Jian Yang. Graph game embedding. In *AAAI*, 2021.

[Jiang *et al.*, 2019] Jiatao Jiang, Zhen Cui, Chunyan Xu, and Jian Yang. Gaussian-induced convolution for graphs. In *AAAI*, 2019.

[Kipf and Welling, 2017] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[Luo *et al.*, 2017] Zhiling Luo, Ling Liu, Jianwei Yin, Ying Li, and Zhaohui Wu. Deep learning of graphs with ngram convolutional neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2125–2139, 2017.

[Maron *et al.*, 2018] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[Maron *et al.*, 2019] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *NeurIPS*, pages 2156–2167, 2019.

[Niepert *et al.*, 2016] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.

[Shervashidze *et al.*, 2009] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

[Shervashidze *et al.*, 2011] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(23):2539–2561, 2011.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, pages 1067–1077, 2015.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *ICLR*, 2018.

[Wang *et al.*, 2016] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *SIGKDD*, pages 1225–1234, 2016.

[Wang *et al.*, 2019] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *WWW*, pages 2022–2032, 2019.

[Xinyi and Chen, 2018] Zhang Xinyi and Lihui Chen. Capsule graph neural network. In *ICLR*, 2018.

[Xu *et al.*, 2018] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pages 5453–5462, 2018.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

[Xu *et al.*, 2020] Chunyan Xu, Zhen Cui, Xiaobin Hong, Tong Zhang, Jian Yang, and Wei Liu. Graph inference learning for semi-supervised classification. *ICLR*, 2020.

[Yanardag and Vishwanathan, 2015] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *SIGKDD*, pages 1365–1374, 2015.

[Yang *et al.*, 2016] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, volume 48, pages 40–48, 2016.

[Zhang *et al.*, 2020] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

[Zhang, 2020] Jiawei Zhang. Get rid of suspended animation problem: Deep diffusive neural network on graph semi-supervised classification. *CoRR*, abs/2001.07922, 2020.

[Zhao *et al.*, 2019] Wenting Zhao, Zhen Cui, Chunyan Xu, Chengzheng Li, Tong Zhang, and Jian Yang. Hashing graph convolution for node classification. In *CIKM*, pages 519–528, 2019.

[Zhuang and Ma, 2018] Chenyi Zhuang and Qiang Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *WWW*, pages 499–508, 2018.