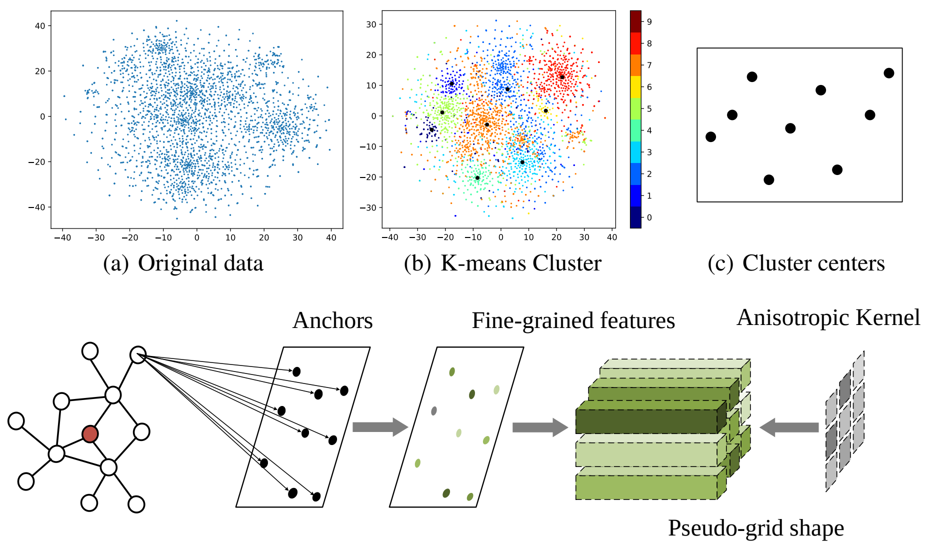
A Appendix



(a) Original data (b) K-means Cluster (c) Cluster centers

(d) The process of projection and convolution on one node

Figure 4: A toy example: visualization of (a) original data and (b) its K-means cluster for Cora dataset. T-SNE is used to project high-dimensional data into 2-D space. (c) 9 cluster centers can be regarded as a pseudo-grid in the 2-D space. Next, (d) a neighbor is transformed into the anchor space, and anisotropic convolution is implemented.

Fig. 4 shows the main idea of Graph Deformer Network, K-means Cluster on the original data produce 9 cluster centers, pseudo-grid shape, indicating several implicit directions like 3×3 receptive field (top left, top right, etc.) in images. Initially anchor nodes are generated by imposing an non-linear transformation on cluster centers in order to match the updated features. Then, the irregular neighbors can be deformed into the anchor space. Finally anisotropic convolution can be implemented by using different filters on the anchor node.

A.1 Related Work

In this section, we briefly retrospect the graph convolutional neural networks related to our work. Recently, numerous graph convolution methods accrue in the field of artificial intelligence. These works roughly fall into two categories: spectral based methods (Bruna et al., 2014; Susnjara et al., 2015; Defferrard et al., 2016; Levie et al., 2018) and spatial based methods (Gilmer et al., 2017; Such et al., 2017; Simonovsky & Komodakis, 2017; Gao et al., 2018; Huang et al., 2018; Liu et al., 2019). Based on the Spectral Graph Theory (Chung, 1997), the work (Shuman et al., 2013) presents a basic framework to process graph data via filtering. Bruna et al. firstly generalized convolutional neural networks to graphs through the decomposition of the graph Laplacian matrix (Bruna et al., 2014). The work (Henaff et al., 2015) seeks to express spatial localization of filters with smoothing coefficients. Then, ChebyNet (Defferrard et al., 2016) and GCN (Kipf & Welling, 2016) take advantage of recursive Chebyshev polynomials to approximate parameterized filters, where the computing efficiency is significantly improved. Ever since, increasing work (Li et al., 2018a; 2019; Chen et al., 2018; Zhuang & Ma, 2018) is dedicated to designing, improving, and optimizing convolution operators on graphs. On the other hand, the spatial based approaches perform convolution directly on graphs by aggregating node features over a spatial neighborhood. DCNN (Atwood & Towsley, 2016) proposes a diffusion-convolution method. Niepert et al. (Niepert et al., 2016) normalized a graph to a grid-shaped structure and performed traditional convolution operations. DGCNN (Wu et al., 2017) adds a disordered graph convolutional layer(DGCL) to avoid the loss of information. GraphSAGE (Hamilton et al., 2017) generates embeddings by sampling and aggregating node features in a local neighborhood. Velickovic et al. (Velickovic et al., 2017) adopted an attention mechanism into graph learning. MoNet (Monti et al., 2017) utilized a Gaussian mixture model to encode the weights between nodes and aggregate node features. JK-Net (Xu et al., 2018) aggregates features of different layers by max-pooling, concatenation, or LSTM-attention. GIN (Xu et al., 2019) presents a theoretical framework to analyze the expressive power of GNNs. GIC method (Jiang et al., 2019) attempts to discover the direction of variations by Gaussian mixture models.

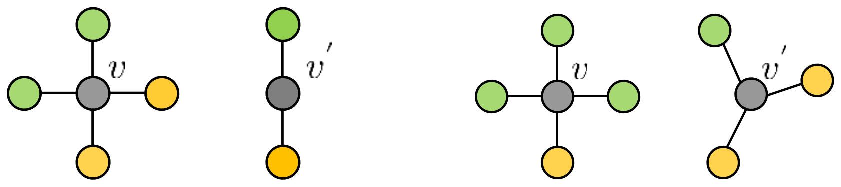
A.2 The Notations

We summarize the used notations of our paper as follows.

Table 5: The notations used in the paper.

Notation	Representation
$\mathcal{G}$	graph
$\mathcal{V}$	vertex set
$\mathcal{E}$	edge set
$\mathbf{A}, \mathbf{A}^{'}$	adjacency matrix
$\mathbf{X}, \mathbf{X}^{'}$	feature matrix
$\mathbf{x}_i, \mathbf{X}_i$	feature vector of node in graph
$\mathcal{N}_{v_i}^{s}$	set of s-hop neighbors for node v_i
$\overline{\mathbb{V}}$	anchor set
$\overline{\mathrm{v}}$	one anchor node
$\overline{\mathbf{x}}$	feature vector of anchor node
f	filters
$\mathcal{D}$	deformer function
$\mathcal{C}$	anisotropic convolution operator
$\mathcal{K}$	convolution kernel
$\mathcal{P}$	plooing / graph coarsening operator
$\mathbf{F}^{(r)}$	deformed multi-granularity feature matrix for node v_r
$\mathcal{V}_{\text{sampling}}$	sampled node set from graph
$\overline{\mathbf{a}}_k$	feature vector of anchor node k after updating
$\mathbf{q}_{\cdot}$	anchor-related feature vector (query feature vector)
$\mathbf{u}_{\cdot}$	value feature vector
$\widetilde{\mathbf{u}}_k$	deformed feature vector on anchor node k
α	weight matrix
$\widetilde{\mathbf{x}}_{v_r}$	output of graph deformer convolution for node v_r
$\mathbf{Z}$	binary cluster matrix for graph coarsening

A.3 Proof of Proposition 1



(a) Mean aggregator fails (b) Sum aggregator fails

Figure 5: Examples of graph structures on which mean and sum aggregators fail. The neighborhood regions of v and $v^{'}$ have the same representation even though their graph structures are different.

Proof. Let us first illustrate the cases that cannot be distinguished by mean/sum aggregation. There exist two non-isomorphic graphs whose response outputs are consistent on mean and sum aggregation. As shown in Fig. 5, the gray node is a reference node, denoting features of the green and yellow nodes as $\mathbf{x}_g, \mathbf{x}_y$. Then, (a) $\frac{1}{4}(2\mathbf{x}_g + 2\mathbf{x}_y) = \frac{1}{2}(\mathbf{x}_g + \mathbf{x}_y)$, and the mean aggregator cannot distinguish them. (b) $(3\mathbf{x}_g + \mathbf{x}_y)$=$(\mathbf{x}_g + 2\mathbf{x}_y)$ if $2\mathbf{x}_g = \mathbf{x}_y$, and the sum aggregator cannot distinguish them. In practice, the sum aggregator is relatively better than the mean aggregator because the sum operation reflects the accumulation number (i.e., node degree).

Next, we will prove the proposed graph deformer process (not including the anisotropic convolution part) can distinguish these two cases. Projecting the different neighborhoods into anchor space can keep the energy constant, i.e., the sum of neighbors in the local neighborhood is the same as the sum on each anchor node after deforming.

(1) Thus, for two non-isomorphic graphs, if their sum on a local neighborhood is different such as Fig. 5(a), then it must be different after deforming into anchor space.

(2) If two non-isomorphic graphs have the same sum on the local neighborhood as shown in Fig. 5(b), then the sum is still same after deforming. But their representations corresponding to the same anchor node are different with a high probability. If two local neighborhoods have the same representation on each anchor after being deformed, taking Fig. 5(b) as an example, setting $m = 3$, i.e., 3 anchor nodes, then we have

$$\alpha_{0,0}\mathbf{x}_0 + \alpha_{1,0}\mathbf{x}_1 + \alpha_{2,0}\mathbf{x}_2 + \alpha_{4,0}\mathbf{x}_4 + \alpha_{5,0}\mathbf{x}_5 = \alpha_{6,0}\mathbf{x}_6 + \alpha_{7,0}\mathbf{x}_7 + \alpha_{8,0}\mathbf{x}_8 + \alpha_{9,0}\mathbf{x}_9, \tag{14}$$

$$\alpha_{0,1}\mathbf{x}_0 + \alpha_{1,1}\mathbf{x}_1 + \alpha_{2,1}\mathbf{x}_2 + \alpha_{4,1}\mathbf{x}_4 + \alpha_{5,1}\mathbf{x}_5 = \alpha_{6,1}\mathbf{x}_6 + \alpha_{7,1}\mathbf{x}_7 + \alpha_{8,1}\mathbf{x}_8 + \alpha_{9,1}\mathbf{x}_9, \tag{15}$$

$$\alpha_{0,2}\mathbf{x}_0 + \alpha_{1,2}\mathbf{x}_1 + \alpha_{2,2}\mathbf{x}_2 + \alpha_{4,2}\mathbf{x}_4 + \alpha_{5,2}\mathbf{x}_5 = \alpha_{6,2}\mathbf{x}_6 + \alpha_{7,2}\mathbf{x}_7 + \alpha_{8,2}\mathbf{x}_8 + \alpha_{9,2}\mathbf{x}_9, \tag{16}$$

$$\text{s.t.} \begin{cases} \mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_4 + \mathbf{x}_5 = \mathbf{x}_6 + \mathbf{x}_7 + \mathbf{x}_8 + \mathbf{x}_9, \\ \alpha_{j,k} = \dfrac{\exp(\langle \mathbf{x}_j, \bar{\mathbf{a}}_k \rangle)}{\sum_{k'} \exp(\langle \mathbf{x}_j, \bar{\mathbf{a}}_{k'} \rangle)}, \quad k' = 0, 1, \cdots, m-1, \end{cases} \tag{17}$$

$\Longrightarrow$

$$\beta_1 = \alpha_{0,0} = \alpha_{1,0} = \alpha_{2,0} = \alpha_{4,0} = \alpha_{5,0} = \alpha_{6,0} = \alpha_{7,0} = \alpha_{8,0} = \alpha_{9,0}, \tag{18}$$

$$\beta_2 = \alpha_{0,1} = \alpha_{1,1} = \alpha_{2,1} = \alpha_{4,1} = \alpha_{5,1} = \alpha_{6,1} = \alpha_{7,1} = \alpha_{8,1} = \alpha_{9,1}, \tag{19}$$

$$\beta_3 = \alpha_{0,2} = \alpha_{1,2} = \alpha_{2,2} = \alpha_{4,2} = \alpha_{5,2} = \alpha_{6,2} = \alpha_{7,2} = \alpha_{8,2} = \alpha_{9,2}, \tag{20}$$

$$\beta_1 + \beta_2 + \beta_2 = 1, \tag{21}$$

$$\beta_i = \frac{\exp(\langle \mathbf{x}_j, \bar{\mathbf{a}}_k \rangle)}{\sum_{k'} \exp(\langle \mathbf{x}_j, \bar{\mathbf{a}}_{k'} \rangle)}, \quad k' = 0, 1, \cdots, m-1. \tag{22}$$

Because these two graphs are non-isomorphic but the sum is the same, then the features $\mathbf{x}_i$ is different. Moreover, the derived normalized attention score for all nodes should be the same. Thus, the inner product between these nodes and anchors should satisfy,

$$\langle \mathbf{x}_0, \bar{\mathbf{a}}_k \rangle = \langle \mathbf{x}_1, \bar{\mathbf{a}}_k \rangle + C_1 = \cdots = \langle \mathbf{x}_9, \bar{\mathbf{a}}_k \rangle + C_9, \quad k' = 0, 1, \cdots, m-1, \tag{23}$$

which means that the inner product between any two nodes and all anchor nodes differs by a same constant C_i. In this case, it is equivalent to a linear normalization of the inner product. Both $\mathbf{x}_i$ and $\bar{\mathbf{a}}_{k'}$ are optimized by the neural network. And the formula of Eqn. 23 difficultly holds. Thus, the proposed graph deformer process usually is an injective, stronger than sum/mean aggregation. □

A.4 Proof of Proposition 2

Before giving the Proof of Proposition2, we first simply elaborate on the Weisfeiler-Lehman (WL) graph isomorphism test.

WL test (Shervashidze et al., 2011) is a state-of-the-art graph kernel method, which concatenates each vertex of a labeled graph and its neighbors to create a sorted multiset label, and then assigns a new label/color to this multiset by hashing. Vertices with the same multiset are assigned the same label. This can be formulated as

$$c^{(t)}(v) = \text{HASH}\Big(\big(c^{(t-1)}(v), \{\!\{ c^{(t-1)}(u) | u \in \mathcal{N}_v^1 \}\!\}\big)\Big), \tag{24}$$

where the $c^{(t)}(v)$ means the color/label of node v in iteration t, $\mathcal{N}_v^1$ is the set including the 1-hop neighbors of node v, and $\{\!\{\cdot\}\!\}$ is a multiset.

Proof. The graph deformer process has been analyzed in A.3, which usually is injective. For a single-scale neighborhood, the anisotropic convolution described in Eqn. 9 is equal to the next form

$$\widetilde{\mathbf{x}}_{v_r} = \text{MLP}\left([\widetilde{\mathbf{u}}_1, \cdots, \widetilde{\mathbf{u}}_{m-1}]\right), \tag{25}$$

which is a combination of concatenation on anchor nodes and a multilayer perceptron.

For a multiset $\{\!\{\widetilde{\mathbf{u}}_i\}\!\}$, $i = 0, 1, \cdots, m-1$, where $\widetilde{\mathbf{u}}_i$ represents a d-dimensional vector. There exists some order making the concatenation on the $\{\!\{\widetilde{\mathbf{u}}_i\}\!\}$ unique. That means the concatenation can obtain different results for different local neighborhoods. Thus, the concatenation aggregation can satisfy an injective, which also is implemented in the WL test. Additionally, according to the universal approximation theorem (Hornik, 1991), multilayer feedforward networks are, under very general conditions on the hidden unit activation function, universal approximators provided that sufficiently many hidden units are available. Thus, we can reach the conclusion.

□

A.5 DATASETS

The global properties of datasets for node and graph classifications have been summarized in Table 6 and Table 7, and details are as follows:

Table 6: Summary of graph datasets for node classification.

Dataset	Nodes	Edges	Features	Classes	Label rate
Cora	2708	5429	1433	7	0.052
Citeseer	3327	4732	3703	6	0.036
Pubmed	19717	44338	500	3	0.003

Table 7: Summary of graph datasets for graph classification.

Dataset	Graphs	Classes	Node labels	Max nodes	Avg.nodes	Avg.edges
MUTAG	188	2	7	28	17.93	19.79
PTC	344	2	19	109	25.56	14.69
NCI1	4110	2	37	111	29.87	32.3
ENZYMES	600	6	3	126	32.63	62.14
PROTEINS	1113	2	3	620	39.06	72.82
IMDB-BINARY	1000	2	-	136	19.77	96.53
IMDB-MULTI	1500	3	-	89	13.0	65.94

- Citation graph. The Cora dataset is constructed by 2708 machine learning papers belonging to 7 classes. Each node represents a paper, and two papers are connected if one cites another paper. There are a total of 5429 edges. Node features are bag-of-words representation indicating the absence/presence of the corresponding word from the dictionary that consists of 1433 unique words. Similarly, Citeseer contains 3327 papers divided into 6 classes. Node features are also bag-of-words representation with 3037 unique words. There exist 4732 edges between the nodes. Pubmed is a larger dataset containing 19717 papers and 44338 edges. The node features are real-valued entries indicating Term Frequency-Inverse Document Frequency (TF-IDF) of the corresponding word from a dictionary.
- Bioinformatics datasets. MUTAG (Debnath et al., 1991) is a nitro compounds dataset including 188 samples and is divided into 2 classes. PTC (Toivonen et al., 2003) consists of compounds labeled according to carcinogenicity on rodents with 19 node labels. NCI1 (Wale et al., 2008) is a balanced dataset of chemical compounds collected by the National Cancer Institute (NCI), which contains 4110 chemical compounds with 37 discrete node labels about the human tumor cell. ENZYMES (Borgwardt et al., 2005) consists of 600 protein tertiary structures divided into 6 classes and obtained from the BRENDA enzyme database. PROTEINS (Borgwardt et al., 2005) consists of 1113 proteins in which nodes are secondary structure elements (SSEs). There exists an edge between two nodes if they are contiguous in the amino acid sequence.
- Social network datasets. IMDB-BINARY and IMDB-MULTI are movie collaboration datasets derived from the work (Yanardag & Vishwanathan, 2015), where every graph belongs to a kind of movie genre which is also to be predicted. The IMDB-BINARY dataset

contains two types: Action and Romance. While the IMDB-MULTI dataset is constructed from Comedy, Romance, and Sci-Fi genres. Each node represents an actor/actress. If two actors appear in the same movie, then they are connected by an edge.

A.6 Experimental Setups

Node classification. For node classification, we adopt the data preprocessed in the work (Yang et al., 2016), and follow its data partitioning rules. There are only 20 samples in each class for training. A total of 500 samples are used for validation and 1000 samples are used for testing for all datasets. The label rate has been summarized in the Appendix. The GDN model consists of two convolutional layers, each of which follows by a pooling layer. Then a fully connected layer and an output layer with the softmax function generate the prediction results. The network structure can be simply represented as $\mathrm{Input} - \mathcal{C} - \mathcal{P}(mean) - \mathcal{C} - \mathcal{P}(mean) - \mathcal{FC}(\mathrm{softmax}) - \mathrm{Output}$, where $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{FC}$ denote convolution, pooling, and fully connected layer, respectively. $\mathcal{P}(mean)$ indicates the "$mean$" operation in the pooling layer. The number of anchor nodes is set to 16. The scale of the neighborhood is set to 2 in both convolutional layers, i.e, including the features of the node itself, first-order and second-order neighborhoods. We train the model 600 epochs with the initial learning rate of 0.05. The dropout rate is set to 0.5 to alleviate over-fitting. We run 10 experiments to take the average as the accuracy"mean $\pm$ standard deviation".

Graph classification. A three-layer GDN model is applied for the learning of overall graph. Each convolutional layer is followed by a pooling layer with the 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 $\mathrm{Input} - \mathcal{C}(64) - \mathcal{P}(0.5) - \mathcal{C}(128) - \mathcal{P}(0.5) - \mathcal{C}(256) - \mathcal{P}(all) - \mathcal{FC}(\mathrm{softmax}) - \mathrm{Output}$, where $\mathcal{P}(all)$ means that only a supernode is retained at the last pooling layer. Also, the number of anchor nodes is set to 16 and the scale of the neighborhood is set to 2. 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. For each cross-validation, we train 500 epochs with the initial learning rate of 0.05.

A.7 Discussion

How to set depth and width of the network?

The network depth (i.e., layer number) and filtering width (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.

How the problem of over-smoothness is handled by the GDN?

(1) Simply speaking, we introduce anchor nodes and deform the irregular neighborhoods into regular anchor space, and use different filters on different anchors. It makes anisotropic convolution available. Theoretical analysis and experimental results validate its effectiveness.

(2) Making an analogy, the first principal component is the principal direction of the data distribution when Principal Component Analysis (PCA) is used to fit data. If only the first principal component is used, the information in other directions will be lost. If more principal components are retained, the data distribution can be better characterized. Similarly, traditional GCN integrates neighborhood nodes via a sum aggregator, only retaining the most important information of the receptive field. And our method tries to project the data in multiple directions (w.r.t. anchors), each of which is intuitively similar to the PCA projection direction. Therefore, our method retains more information.

A.8 GDN Algorithm

Our proposed graph deformer network (GDN) for node classification can be summarized as follows:

Algorithm 1: GDN algorithm for node classification

```
Input: node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$; adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$; the scale of neighborhoods
       $S$; the number of layers $L$; the number of key nodes $m$; the deformer function $\mathcal{D}$; the
       coarsening function $\mathcal{P}$; the deformer parameter $\Theta$; the convolution operation $\mathcal{C}$; the
       fully-connected function $g$; the convolution kernel $\mathcal{K}$.
Output: the predicted labels of nodes $\widehat{\mathbf{Y}} \in \mathbb{R}^{n \times C}$ .
1  $\overline{\mathcal{V}}^0 = \{(\overline{\mathrm{v}}_k^0, \overline{\mathbf{x}}_k^0)\}|_{k=0}^{m-1} \leftarrow$ Clustering $\{(v_i, \mathbf{x}_i)|v_i \in \mathcal{V}_{\text{sampling}}\}$;
2  $\widetilde{\mathbf{z}}_t^0 \leftarrow \mathbf{x}_t, \quad t = 1, \cdots, n$;
3  for $l = 1, 2, \cdots, L$ do
4      $\mathbf{x}_t^{l-1} \leftarrow \widetilde{\mathbf{z}}_t^{l-1}, t = 1, \cdots, n$;
5      $\overline{\mathbf{x}}_k^l \leftarrow \mathrm{ReLU}(\mathbf{W}_A^l \overline{\mathbf{x}}_k^{l-1} + \mathbf{b}_A^l), k = 1, \cdots, m$;
6      for $v_r \in \mathcal{V}$ do
7          for $s = 1, \cdots, S$ do
8              $\mathbf{F}_i^{(r,s)} \leftarrow \sum_{v_t \in \mathcal{N}_{v_r}^s} \mathcal{D}_{v_t \to \overline{\mathrm{v}}_i}(\overline{\mathbf{x}}_i^l, \mathbf{x}_t^{l-1}, \Theta^l)$;
9              $\widetilde{\mathbf{x}}_r^{(s)} \leftarrow \mathcal{C}(\mathbf{F}^{(r,s)}, \mathcal{K}^l)$;
10         end
11         $\widetilde{\mathbf{x}}_r \leftarrow [\mathbf{x}_r; \widetilde{\mathbf{x}}_r^{(1)}; \cdots; \widetilde{\mathbf{x}}_r^{(S)}]$;
12         $\mathbf{z}_r^l = g^l(\widetilde{\mathbf{x}}_r)$ ;
13         $\widetilde{\mathbf{z}}_r^l = \mathcal{P}\{\mathbf{z}_r^l, \mathbf{z}_t^l, \cdots\}, v_t \in \bigcup_s^S \mathcal{N}^s(v_r)$ ;
14     end
15 end
16 $\widehat{\mathbf{Y}}_r = \mathbf{argmax}\ \{\mathbf{softmax}\{f^L(\widetilde{\mathbf{z}}_r^L)\}\}$.
```