Diving into Unified Data-Model Sparsity for Class-Imbalanced Graph Representation Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Even pruned by the state-of-the-art network compression methods, recent research shows that deep learning model training still suffers from the demand of massive data usage. In particular, Graph Neural Networks (GNNs) training upon such non-Euclidean graph data often encounters relatively higher time costs, due to its irregular and nasty density properties, compared with data in the regular Euclidean space (e.g., image or text). Another natural property concomitantly with graph is class-imbalance which cannot be alleviated by the massive graph data while hindering GNNs' generalization. To fully tackle these unpleasant properties, *(i) theoretically*, we introduce a hypothesis about what extent a subset of the training data can approximate the full dataset's learning effectiveness. The effectiveness is further guaranteed and proved by the gradients' distance between the subset and the full set; *(ii) empirically*, we discover that during the learning process of a GNN, some samples in the training dataset are informative for providing gradients to update model parameters. Moreover, the informative subset is not fixed during training process. Samples that are informative in the current training epoch may not be so in the next one. We refer to this observation as dynamic data sparsity. We also notice that sparse subnets pruned from a well-trained GNN sometimes forget the information provided by the informative subset, reflected in their poor performances upon the subset. Based on these findings, we develop a unified data-model dynamic sparsity framework named ***Graph Dec**antation* (GraphDec) to address challenges brought by training upon a massive class-imbalanced graph data. The key idea of GraphDec is to identify the informative subset dynamically during the training process by adopting sparse graph contrastive learning. Extensive experiments on multiple benchmark datasets demonstrate that GraphDec outperforms state-of-the-art baselines for class-imbalanced graph classification and class-imbalanced node classification tasks, with respect to classification accuracy and data usage efficiency.

1 Introduction

Graph representation learning (GRL) [23] has shown remarkable power in dealing with non-Euclidean structure data (e.g., social networks, biochemical molecules, knowledge graphs). Graph neural networks (GNNs) [23, 11, 38], as the current state-of-the-art of GRL, have become essential in various graph mining applications. To learn the representation of each node reflecting its local structure pattern, GNNs gather features of the neighbor nodes and apply message passing along edges. This topology-aware mechanism enables GNNs to achieve superior performances over different tasks.

However, in many real-world scenarios, graph data often preserves two properties: massiveness [36, 18] and class-imbalance [30]. Firstly, message-passing over nodes with high degrees brings about heavy computation burdens. Some of the calculations are even redundant, in that not all neighbors are informative for learning task-related embeddings. Unlike regular data such as images or texts, the connectivity of irregular graph data causes random memory access, which further slows down the

Submitted to NeurIPS 2022 GLFrontiers Workshop. Do not distribute.

efficiency of data readout. Secondly, class imbalance naturally exists in datasets from diverse practical domains, such as bioinformatics and social networks. GNNs are sensitive to this imbalance and can be biased toward the dominant classes. This bias may mislead GNNs' learning process, therefore making the model underfit on samples that are of real importance with respect to the downstream tasks, and as a result yielding poor performance on the test data.

Accordingly, recent studies [3, 43, 30] arise to address the issues of massiveness or class-imbalanced in graph data. To tackle the massiveness issue, [7, 2] explore efficient data sampling policies to reduce the computational cost from the data perspective. From the model improvement perspective, some approaches design the quantization-aware training and low-precision inference method to reduce GNNs' operating costs. For example, GLT [3] applies the lottery ticket technique [9] to simplify graph data and GNN model simultaneously. To deal with the imbalance issue in node classification on graphs, GraphSMOTE [43] tries to generate new nodes for the minority classes to balance the training data. Improved upon GraphSMOTE, GraphENS [30] further proposes a new augmentation method by constructing an ego network to learn the representations of the minority classes. Despite progress made so far, existing methods fail to tackle the two issues altogether. Furthermore, while one of the issues is being handled, extra computation costs are introduced at the same time. For example, the rewind steps in GLT [3] which search for lottery subnets and subsets heavily increase the computation cost, although the final lotteries are lightweight. The newly synthetic nodes in GraphSMOTE [43, 1] and GraphENS [30], although help alleviate the data imbalance, bring extra computational burdens for the next-coming training process. Regarding the issues above, we make several observations. Firstly, we notice that a sparse pruned GNN easily "forgets" the minority samples when trained with class-imbalanced graph data, as it yields worse performance over the minorities compared with the original GNN [17]. To investigate the cause of the above observation, we study how each graph sample affects the model parameters update process by taking a closer look at the gradients it brings to the parameters. Specifically, at early training stages, we found a small subset of the samples providing the most informative supervisory signals reflected by the gradient norms. One hypothesis we make is that the training effectiveness of the full training set can be approximated, to some extent, by that of the subset. We further hypothesize that this effective approximation is guaranteed by the distance between the gradients of the subset and the full dataset. Based on the above observations and the hypothesises, we propose a novel method called **Graph Dec**antation (GraphDec) to explore dynamic sparsity training from both model and data aspects. The principle behind our method is illustrated in Figure 1. Given that informative samples bring about higher gradient values/scores when trained with a sparse GNN, our method is inspired by contrastive self-supervised learning because it can be modified to dynamically prune one branch of contrastive backbone for improving its capability of identifying minority samples in class-imbalanced dataset. In particular, we design a new contrastive backbone with a sparse GNN and enable the model to identify informative samples in a self-supervised manner. To the best of our knowledge, other learning processes (e.g., graph auto-encoder, supervised learning) are either unable to identify informative samples or incapable of learning in a self-supervised manner. Accordingly, the proposed framework can score samples in the current training set and keep only k most informative samples as training set for the next epoch. Considering that a currently unimportant sample does not imply that it will always be unimportant, we further design a data recycling process to randomly recycle prior discarded data samples (samples that are considered unimportant in previous training epochs), and add them back to current informative sparse subsets for reuse. The dynamically updated informative subset supports the sparse GNN to learn more balanced representations. To summarize, our contributions in this work are:

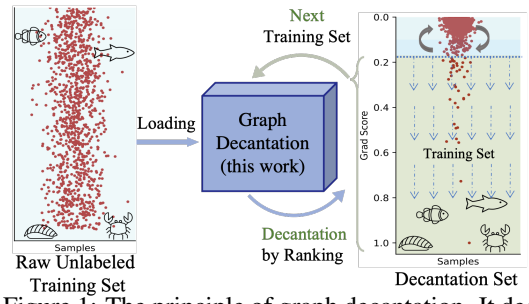

Figure 1: The principle of graph decantation. It decants data samples based on rankings of their gradient scores, and then uses them as the training set in the next epoch.

- We develop a novel framework, Graph Decantation, which leverages dynamical sparse graph contrastive learning on class-imbalanced graph data with efficient data usage. To our best knowledge, this is the first study to explore the dynamic sparsity property for class-imbalanced graphs.
- We introduce cosine annealing to dynamically control the sizes of the sparse GNN model and graph data subset to smooth the training process. Meanwhile, we introduce data recycling to refresh the current data subset to avoid overfitting.

- Comprehensive experiments on multiple benchmark datasets demonstrate that GraphDec outperforms state-of-the-art methods for both class-imbalanced graph classification and class-imbalanced node classification tasks. Additional results show that GraphDec dynamically finds an informative subset across different training epochs effectively.

2 Related Work

Training deep model with sparsity. Despite the fact that deep neural networks work generally well in practice, they are usually over-parameterized. Over-parameterized models, although usually achieve good performance when trained properly, are usually associated with enormous computational cost. Therefore, parameter pruning aiming at decreasing computational cost has been a popular topic and many parameter-pruning strategies are proposed to balance the trade-off between model performance and learning efficiency [5, 24]. Among all of the existing parameter pruning methods, most of them belong to the static pruning category and deep neural networks are pruned either by neurons [14, 13] or architectures (layer and filter) [15, 6]. Parameters deleted by these methods will not be recovered later. In contrast to static pruning methods, more recent works propose dynamic pruning strategies where different compact subnets will be dynamically activated at each training iteration [26, 28, 32]. The other line of computation cost reduction lies in the dataset sparsity [21, 25, 31]. The core idea is to prune the original dataset and filter out the most salient subset so that an over-parameterized deep model could be trained upon (e.g., data diet subset [31]). Recently, the property of sparsity is also used to improve model robustness [4, 10]. In this work, we attempt to accomplish dynamic sparsity from both the GNN model and the graph dataset simultaneously.

Class-imbalanced learning on graphs. In real-world scenarios, imbalanced class distribution is one of the natural properties in many datasets, including graph data. Except for conventional re-balanced methods, like reweighting samples [43, 30] and oversampling [43, 30], different methods have been proposed to solve the class imbalance issue in graph data given a specific task. For the node classification task, an early work [44] tries to accurately characterize the rare categories through a curriculum self-paced strategy while some other previous works [34, 43, 30] solve the class-imbalanced issue by proposing different methods to generate synthetic samples to rebalance the dataset. Compared to the node-level task, the re-balanced methods specific to graph-level task are relatively unexplored. A recent work [39] proposes to utilize additional supervisory signals from neighboring graphs to alleviate the class-imbalanced problem for a graph-level task. To the best of our knowledge, our proposed GraphDec is the first work to solve the class-imbalanced for both node-level and graph-level tasks.

3 Preliminary

In this work, we denote graph as $G = (V, E, X)$, where V is the set of nodes, E is the set of edges, and $X \in \mathbb{R}^d$ represents the node (and edge) attributes of dimension d. In addition, we represent the neighbor set of node $v \in V$ as N_v.

Graph Neural Networks. GNNs [40] learn node representations from the graph structure and node attributes. This process can be formulated as:

$$h_v^{(l)} = \mathrm{COMBINE}^{(l)}\left(h_v^{(l-1)}, \mathrm{AGGREGATE}^{(l)}\left(\left\{h_u^{(l-1)}, \forall u \in N_v\right\}\right)\right), \tag{1}$$

where $h_v^{(l)}$ denotes feature of node v at l-th GNN layer; $\mathrm{AGGREGATE}(\cdot)$ and $\mathrm{COMBINE}(\cdot)$ are neighbor aggregation and combination functions, respectively; $h_v^{(0)}$ is initialized with node attribute X_v. We obtain the output representation of each node after repeating the process in Equation (1) for L rounds. The representation of the whole graph, denoted as $h_G \in \mathbb{R}^d$, can be obtained by using a READOUT function to combine the final node representations learned above:

$$h_G = \mathrm{READOUT}\left\{h_v^{(L)} \mid \forall v \in V\right\}, \tag{2}$$

where the $\mathrm{READOUT}$ function can be any permutation invariant, like summation, averaging, etc.

Graph Contrastive Learning. Given a graph dataset $\mathcal{D} = \{G_i\}_{i=1}^N$, Graph Contrastive Learning (GCL) methods firstly implement proper transformations on each graph G_i to generate two views G_i' and G_i''. The goal of GCL is to map samples within positive pairs closer in the hidden space, while those of the negative pairs are further. GCL methods are usually optimized by contrastive loss. Taking the most popular InfoNCE loss [29] as an example, the contrastive loss is defined as:

$$\mathcal{L}_{CL}(G_i', G_i'') = -\log \frac{\exp\left(\mathrm{sim}\left(\mathbf{z}_{i,1}, \mathbf{z}_{i,2}\right)\right)}{\sum_{j=1, j\neq i}^{N} \exp\left(\mathrm{sim}\left(\mathbf{z}_{i,1}, \mathbf{z}_{j,2}\right)\right)}, \tag{3}$$

where $\mathbf{z}_{i,1} = f_\theta(G_i')$, $\mathbf{z}_{i,2} = f_\theta(G_i'')$, and sim denotes the similarity function.

Network Pruning. Given an over-parameterized deep neural network $f_\theta(\cdot)$ with weights θ, the network pruning is usually performed layer-by-layer. The pruning process of the l_{th} layer in $f_\theta(\cdot)$ can be formulated as follows:

$$\theta_{pruned}^{l_{th}} = \mathrm{TopK}(\theta^{l_{th}}, k), k = \alpha \times |\theta^{l_{th}}|, \tag{4}$$

where $\theta^{l_{th}}$ is the parameters in the l_{th} layer of $f_\theta(\cdot)$ and $\mathrm{TopK}(\cdot, k)$ refers to the operation to choose the top-k largest elements of $\theta^{l_{th}}$. We use a pre-defined sparse rate α to control the fraction of parameters kept in the pruned network $\theta_{pruned}^{l_{th}}$. Finally, only the top $k = \alpha \times |\theta^{l_{th}}|$ largest weights will be kept in the pruned layer. The pruning process will be implemented iteratively to prune the parameters in each layer of deep neural network [12].

4 Methodology

In this section, we first illustrate our sparse subset approximation hypothesis supported by the theorem, which means that if the gradients of a data subset approximate well to the gradients of the full data set, the model trained on subset performs closely to the model trained with full set. Guided by this hypothesis, we develop GraphDec to continually refine a compact training subset with the dynamic graph contrastive learning methodology. In detail, we describe procedures about how to rank the importance of each sample, smooth the refining procedure, and avoid overfitting.

4.1 Sparse Subset Approximation Hypothesis

Firstly, we propose the sparse subset approximation hypothesis to show how a model trained with a subset data $\mathcal{D}_S$ can approximate the effect of a model trained with full data $\mathcal{D}$. This hypothesis explains why the performance of the model trained with a subset data selected by specific methods (e.g., data diet [31]) achieves performance close to the one trained on the full dataset.

Theorem 1 *For a data selection algorithm, we assume the model is optimized with full gradient descent. At epoch t where $t \in [1, T]$, denote the model's parameters as $\theta^{(t)}$ where $\left\|\theta^{(t)}\right\|^2 \leqslant d^2$ and d is constant, the optimal model's parameters as θ^*, subset data as $\mathcal{D}_S^{(t)}$, and learning rate as α. Define gradient error $Err(\mathcal{D}_S^{(t)}, \mathcal{L}, \mathcal{L}_{train}, \theta^{(t)}) = \left\|\sum_{i \in \mathcal{D}_S^{(t)}} \nabla_\theta \mathcal{L}_{train}^i(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|$, where $\mathcal{L}$ denotes training loss $\mathcal{L}_{train}$ over full training data or validation loss $\mathcal{L}_{val}$ over full validation data. $\mathcal{L}$ is a convex function. Then we have the following guarantee:*

If $\mathcal{L}_{train}$ is Lipschitz continuous with parameter σ_T and $\alpha = \frac{d}{\sigma_T \sqrt{T}}$, then $\min_{t=1:T} \mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*) \leqslant \frac{d\sigma_T}{\sqrt{T}} + \frac{d}{T} \sum_{t=1}^{T-1} Err(\mathcal{D}_S^{(t)}, \mathcal{L}, \mathcal{L}_{train}, \theta^{(t)})$.

The detailed proof is provided in the Section A of Appendix. According to the above hypothesis, one intuitive illumination is that reducing the distance between gradients of the subset and the full set, formulated as $\left\|\sum_{i \in \mathcal{D}_S^{(t)}} \nabla_\theta \mathcal{L}_{train}^i(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|$, is the key to minimize the gap between the performance of the model trained with the subset and the optimal model, denoted as $\mathcal{L}(\theta) - \mathcal{L}(\theta^*)$. From the perspective of minimizing $\left\|\sum_{i \in \mathcal{D}_S^{(t)}} \nabla_\theta \mathcal{L}_{train}^i(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|$, the success of data diet [31] (a prior coreset algorithm) is understandable: data diet computes each sample's error/gradient norm based on a slight-trained model, then deletes a portion of the full set with smaller values, which can be represented as $\bar{\mathcal{D}_S}^{(t)} = D - \mathcal{D}_S{}^{(t)}$. The gradient $\sum_{j \in \bar{\mathcal{D}_S}^{(t)}} \nabla_\theta \mathcal{L}_{train}^j(\theta^{(t)})$ of the removed data samples is much smaller than that of the remaining data samples $\sum_{i \in \mathcal{D}_S^{(t)}} \nabla_\theta \mathcal{L}_{train}^i(\theta^{(t)})$. As we will show in the experiments (Section 5.5), the static data diet cannot always capture the most important samples across all epochs during training [31]. Although rankings of all elements in $\mathcal{D}_S$ seemly keep static and unchangeable, the ranking order of elements in full training dataset $\mathcal{D}$ changes much more actively than the diet subset $\mathcal{D}_S$, which implies one-shot subset $\mathcal{D}_S$ can not provide gradient $\sum_{i \in \mathcal{D}_S^{(t)}} \nabla_\theta \mathcal{L}_{train}^i(\theta^{(t)})$ to approximate the full set $\mathcal{D}$'s gradient $\nabla_\theta \mathcal{L}(\theta^{(t)})$.

4.2 Graph Decantation

Then, we follow the above Theorem 1 to develop GraphDec for achieving competitive performance and efficient data usage simultaneously by filtering out the most influential data subset. The overall framework of GraphDec is illustrated in Figure 2. The training processes are summarized into four steps: (i) First, compute gradients of all $M^{(t)}$ graph/node samples in t-th epoch from contrastive

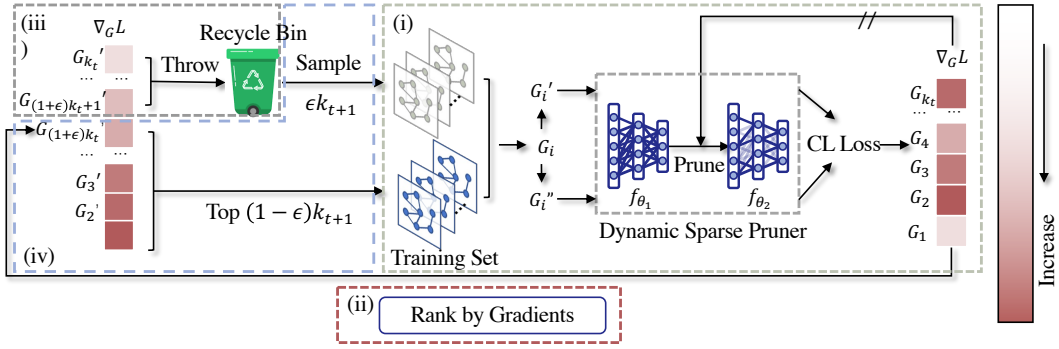

Figure 2: The overall framework of GraphDec: (i) The dynamic sparse graph contrastive learning model computes gradients for graph/node samples; (ii) The input samples are sorted according to their gradients; (iii) Part of samples with the smallest gradients are thrown into the recycling bin; (iv) Part of samples with the largest gradients in the current epoch and some sampled randomly from the recycling bin are jointly used as training input in the next epoch.

learning loss; (ii) The gradients are then normalized and the corresponding graph/node samples are ranked in a descending order by their magnitudes; (iii) We then decay the number of samples from $M^{(t)}$ to $M^{(t+1)}$ with cosine annealing and only keep the top $M^{(t+1)} \times (1-\epsilon)$ samples (ϵ is the exploration rate which controls the ratio of samples randomly resampled from recycle bin). The rest samples will be throwed into the recycle bin temporarily; (iv) Finally, randomly resample $M^{(t+1)} \times \epsilon$ samples from the recycled bin and these samples union the ones selected in step (iii) will be used for model training in the $t+1$ epoch. In the following, we describe each of these four steps in details.

Compute gradients by dynamic sparse graph contrastive learning model. In the first step, given a graph training set $\mathcal{D} = \{G_i\}_{i=1}^N$ as input, our dynamic sparse graph contrastive learning model (DS-GCL) takes two augmented views G' and G'' of an original graph $G \in \mathcal{D}$ as inputs. In detail, for each graph sample, DS-GCL has two GNN branches $f_{\theta_1}(\cdot)$ and $f_{\theta_2}(\cdot)$, which are pruned on-the-fly from an original GCN $f_\theta(\cdot)$ by a dynamic sparse pruner. For example, at l_{th} graph convolutional layer of $f_\theta(\cdot)$, a fraction of connections with the largest weight magnitudes are kept, which are chosen by the following formulation:

$$\theta_{pruned}^{l_{th}} = \mathrm{TopK}(\theta^{l_{th}}, k), k = \alpha^{(t)} \times |\theta^{l_{th}}|, \tag{5}$$

where $\alpha^{(t)}$ is the fraction of remaining neural connections, which is controlled under cosine annealing:

$$\alpha^{(t)} = \frac{\alpha^{(0)}}{2}\left\{1 + \cos\left(\frac{\pi t}{T}\right)\right\}, t \in [1, T], \tag{6}$$

where $\alpha^{(0)}$ is initialized as 1. In addition, some new connections are activated using the current gradient information. Every few epochs, the pruned neural connections are all re-involved in loss backward by following formulation:

$$\mathbb{I}_{\theta^{l_{th}}} = \mathrm{ArgTopK}(\nabla_{\theta^{l_{th}}}\mathcal{L}, k), k = \alpha^{(t)} \times |\theta^{l_{th}}|, \tag{7}$$

where ArgTopK returns indices of top-k elements and $\mathbb{I}_{\theta_{pruned}^{l_{th}}}$ denotes elements' indices in l_{th} layer weights $\theta^{l_{th}}$. These reactivated weights are then combined with other remaining connections for model pruning in the next iteration. We save the gradient values for all samples and use them in the next step. The benefits brought from DS-GCL reflects in two perspectives: (a) it scores the graph samples without any labeling effort from humans, compared with graph active learning; (b) it is more sensitive in selecting informative samples, empirically verified in the Section C of appendix.

Rank graph samples according to their gradients' L_2 norms. In the second step, since gradients of all graph samples in $\mathcal{D}_S^{(t)}$ ($\mathcal{D}_S^{(t)} = \mathcal{D}$ when $t = 0$) at t-th epoch are already saved, we can calculate their gradients' L_2 norms. For example, a graph input $G_i \in \mathcal{D}_S^{(t)}$ will be scored by its gradient norm:

$$g(x^{(t)}) = \left\|\nabla_{f_{\theta_{pruned}}}\mathcal{L}(f_{P\theta_{pruned}(G')}, f_{\theta_{pruned}}(G''))\right\|_2. \tag{8}$$

In this work, we use the popular InfoNCE contrastive loss [37] and the gradient of G is computed as:

$$\nabla_{f_{\theta_{pruned}}}\mathcal{L}(f_{\theta_{pruned}}(G'), f_{\theta_{pruned}(G'')}) = p(\theta_{pruned}, G') - p(\theta_{pruned}, G''), \tag{9}$$

where the $p(\theta_{pruned}, G')$ and $p(\theta_{pruned}, G'')$ denote model's predictions of G' and G'' with pruned parameters θ_{pruned}. All graph samples in $\mathcal{D}_S^{(t)}$ are ranked according to their scores. The ranked $\mathcal{D}_S^{(t)}$ will be provided to latter use.

Decay the size of $\mathcal{D}_S$ by cosine annealing. In the third step, we aim to prune the size of the subset for the next $t+1$ training epoch. To smooth this pruning procedure, we apply cosine annealing to control the decay rate. Specifically, the size $M^{(t+1)}$ is computed as follows:

$$M^{(t+1)} = \frac{M^{(0)}}{2}\left\{1 + \cos\left(\frac{\pi(t+1)}{T}\right)\right\}, t \in [1, T]\,. \tag{10}$$

It smoothly refines $\mathcal{D}_S$ and avoids manually choosing which training epoch for one-shot selection like data diet [31]. $M^{(t+1)}$ sets the number of graph samples in $\mathcal{D}_S^{(t+1)}$ for the next $t+1$ epoch.

As we will show in Figure 3 in the experiments, at early training, some graph samples only have low scores/importance. However, in the later training epochs, these graph samples yield much higher scores once given more patience in training. Upon this observation, we believe that it is worthwhile to not permanently remove samples with low scores at the current training epoch, since some samples in removal set $\bar{\mathcal{D}_S} = \mathcal{D} - \mathcal{D}_S^{(t)}$ might be re-identified as high-scored samples if they can be re-involved into the training process. In the opposite direction, if a model is only trained with a subset of graph samples that are highly scored in the early training stage, the training effect of such a model cannot approximate the full training set's gradient effects well. Based on this analysis, this step specializes in this dilemma: we build the above cosine annealing to control the removal rate of $\mathcal{D}_S^{(t)}$ during training instead of hastily scoring out a subset in one-shot mode like data diet and then use it to re-train the neural network model.

Recycle removed graph samples for next training epoch. In the last step, we already have the ranked $\mathcal{D}_S^{(t)}$ and the subset size $M^{(t+1)}$ for $t+1$ epoch. Our next aim is to update the elements in $\mathcal{D}_S^{(t+1)}$ for the next epoch. When updating elements in $\mathcal{D}_S^{(t+1)}$, since we think currently low-scored samples may still have the potential to be high-scored, removed samples are randomly recovered. We use an exploration rate ϵ to remove $\epsilon M^{(t+1)}$ lowest-scores graph samples in $\mathcal{D}_S^{(t)}$ and recycles $\epsilon M^{(t+1)}$ samples from $\bar{\mathcal{D}_S}^{(t-1)}$. At the same time, we keep $(1-\epsilon)M^{(t+1)}$ graph samples with highest scores from $\mathcal{D}_S^{(t)}$ to $\mathcal{D}_S^{(t+1)}$. The overall $\mathcal{D}_S^{(t+1)}$'s update is worked as follows:

$$\mathcal{D}_S^{(t+1)} = \mathrm{TopK}(\mathcal{D}_S^{(t)}, (1-\epsilon)M^{(t+1)}) \bigcup \mathrm{SampleK}(\bar{\mathcal{D}_S}^{(t-1)}, \epsilon M^{(t+1)}), \tag{11}$$

where $\mathrm{SampleK}(\bar{\mathcal{D}_S}^{(t-1)}, \epsilon M^{(t+1)})$ returns randomly sampled $\epsilon M^{(t+1)}$ samples from $\bar{\mathcal{D}_S}^{(t-1)}$. Given the compact sparse subset $\mathcal{D}_S^{(t+1)}$, we use it for model training in the next epoch and repeat to execute this pipeline until T epoch.

5 Experiments

In this section, we conduct extensive experiments to validate the effectiveness of the proposed model for both class-imbalanced graph classification and class-imbalanced node classification tasks. We also conduct ablation study and informative subset evolution analysis to better understand the effectiveness of the proposed model. Due to space limit, more analyses about GraphDec property are provided in Section C of Appendix.

5.1 Experimental Setup

Datasets. We use various graph benchmark datasets to evaluate our model for two tasks: graph classification and node classification in class-imbalanced data scenario. For the class-imbalanced graph classification task, we consider all seven datasets used in G^2GNN paper [39], i.e., MUTAG, PROTEINS, D&D, NCI1, PTC-MR, DHFR, and REDDIT-B in [27]. For the class-imbalanced node classification task, we use all five datasets used in the GraphENS paper [30], i.e., Cora-LT, CiteSeer-LT, PubMed-LT [33], Amazon-Photo, and Amazon-Computers. Detailed descriptions of these datasets are provided in the Section B of Appendix.

Baseline Methods. We compare our model with a variety of baseline methods using different rebalance methods. For class-imbalanced graph classification, we consider three rebalance methods, i.e., vanilla (without re-balancing when training), up-sampling [39], and re-weight [39]. For each rebalance method, we run three baseline methods including GIN [41], InfoGraph [35], and GraphCL

Table 1: Class-imbalanced graph classification results. Numbers after each dataset name indicate imbalance ratios of minority to majority categories. Best/second-best results are in bold/underline.

Rebalance Method	Basis	MUTAG (5:45)		PROTEINS (30:270)		D&D (30:270)		NCI1 (100:900)		Sparsity (%)	
		F1-ma.	F1-mi.	F1-ma.	F1-mi.	F1-ma.	F1-mi.	F1-ma.	F1-mi.	data	model
vanilla	GIN [41]	52.50	56.77	25.33	28.50	9.99	11.88	18.24	18.94	100	100
	InfoGraph [35]	69.11	69.68	35.91	36.81	21.41	27.68	33.09	34.03	100	100
	GraphCL [42]	66.82	67.77	40.86	41.24	21.02	26.80	31.02	31.62	100	100
up-sampling	GIN [41]	78.03	78.77	65.64	71.55	41.15	70.56	59.19	71.80	>100	100
	InfoGraph [35]	78.62	79.09	62.68	66.02	41.55	71.34	53.38	62.20	>100	100
	GraphCL [42]	80.06	80.45	64.21	65.76	38.96	64.23	49.92	58.29	>100	100
re-weight	GIN [41]	77.00	77.68	54.54	55.77	28.49	40.79	36.84	39.19	100	100
	InfoGraph [35]	80.85	81.68	65.73	69.60	41.92	72.43	53.05	62.45	100	100
	GraphCL [42]	80.20	80.84	63.46	64.97	40.29	67.96	50.05	58.18	100	100
G^2GNN [39]	remove edge	80.37	81.25	67.70	73.10	43.25	77.03	63.60	72.97	100	100
	mask node	83.01	83.59	67.39	73.30	43.93	**79.03**	64.78	74.91	100	100
GraphDec	dynamic sparsity	**85.71**	**85.71**	**76.92**	**76.89**	**77.97**	77.02	**76.30**	**76.29**	50	50

Rebalance Method	Basis	PTC-MR (9:81)		DHFR (12:108)		REDDIT-B (50:450)		Avg. Rank		Sparsity (%)	
		F1-ma.	F1-mi.	F1-ma.	F1-mi.	F1-ma.	F1-mi.	F1-ma.	F1-mi.	data	model
vanilla	GIN [41]	17.74	20.30	35.96	49.46	33.19	36.02	12.00	12.00	100	100
	InfoGraph [35]	25.85	26.71	50.62	56.28	57.67	67.10	11.00	11.14	100	100
	GraphCL [42]	24.22	25.16	50.55	56.31	53.40	62.19	10.71	10.57	100	100
up-sampling	GIN [41]	44.78	55.43	55.96	59.39	66.71	83.00	6.00	5.43	>100	100
	InfoGraph [35]	44.29	48.91	59.49	61.62	67.01	78.68	6.00	6.00	>100	100
	GraphCL [42]	45.12	53.50	60.29	61.71	62.01	75.84	6.29	6.43	>100	100
re-weight	GIN [41]	36.96	43.09	55.16	57.78	45.17	51.92	9.86	9.86	100	100
	InfoGraph [35]	44.09	49.17	58.67	60.24	65.79	77.35	5.43	5.29	100	100
	GraphCL [42]	44.75	52.22	60.87	61.93	62.79	76.15	6.00	6.29	100	100
G^2GNN [39]	remove edge	46.40	56.61	61.63	63.61	68.39	86.35	2.71	2.86	100	100
	mask node	46.61	56.70	59.72	61.27	67.52	85.43	2.71	2.71	100	100
GraphDec	dynamic sparsity	**54.03**	**61.17**	**64.25**	**67.91**	**69.70**	**87.00**	1.00	1.14	50	50

[42]. In addition, we adopt two versions of G^2GNN (i.e., remove-edge and mask-node) [39] for in-depth comparison. For class-imbalanced node classification, we consider nine baseline methods including vanilla, re-weight [19], oversampling [30], cRT [20], PC Softmax [16], DR-GCN [34], GraphSMOTE [43], and GraphENS [30]. We use Graph Convolutional Network (GCN) [23] as the default architecture for all rebalance methods. Further details about the baselines are illustrated in the Section B of Appendix.

Evaluation Metrics. To fully evaluate the model performance, we adopt F1-micro (F1-mi.) and F1-macro (F1-ma.) scores for the class-imbalanced graph classification, as well as accuracy (Acc.), balanced accuracy (bAcc.), and F1-macro (F1-ma.) score for the class-imbalanced node classification.

Experimental Settings. We adopt GCN [23] as the GNN backbone of GraphDec for both tasks. In particular, we use a two-layers GCN and a one-layer fully-connected layer for node classification, and add one extra average pooling operator as the readout layer for graph classification. We follow [39] and [30] to set the imbalance ratios for graph classification and node classification tasks, respectively. In addition, we use GraphCL [42] as the graph contrastive learning framework, and use cosine annealing to dynamically control the sparsity rate in the GNN model and the dataset. We set the initial sparsity rate the rate $\alpha^{(0)}$ for model to 0.8 and $\beta^{(0)}$ for dataset to 1.0. After the contrastive pre-training, we use the GCN output logits as the input to the Support Vector Machine for fine-tuning. GraphDec is implemented in PyTorch and trained on NVIDIA V100 GPU. Code is in the appendix.

5.2 Class-imbalanced Graph Classification Performance

We start by comparing GraphDec with the aforementioned baselines on class-imbalanced graph classification task. The results are reported in Table 1. The best and second-best values are highlighted by bold and underline. From the table, we find that GraphDec outperforms baseline methods on both metrics across different datasets, while only uses an average of 50% data and 50% model weights per round. Although a slight F1-micro difference has been detected on D&D when comparing GraphDec to the best baseline G^2GNN, this is understandable due to the fact that the graphs in D&D are significantly larger than those in other datasets, necessitating specialized designs for graph augmentations (e.g., the average graph size in terms of node number is 284.32 for D&D, but 39.02 and 17.93 for PROTEINS and MUTAG, respectively). However, in the same dataset, G^2GNN can only achieve 43.93 on F1-macro while GraphDec achieves 77.97, which complements the 2% difference on F1-micro and further demonstrates GraphDec's ability to learn effectively even on the dataset with large graphs. Specifically, models trained with vanilla setting perform the worst due to the ignorance of class imbalance. Up-sampling strategy improves the performance, but it introduces additional unnecessary data usage by sampling the minorities multiple times. Similarly, re-weight strategy tries

Table 2: Class-imbalanced node classification results. Best/second-best results are in bold/underline.

Method	Cora-LT			CiteSeer-LT			PubMed-LT			A.P. (ρ =82)		A.C. (ρ =244)		Sparsity (%)	
	Acc.	bAcc.	F1-ma.	Acc.	bAcc.	F1-ma.	Acc.	bAcc.	F1-ma.	(b)Acc.	F1-ma.	(b)Acc.	F1-ma.	data	model
vanilla	73.66	62.72	63.70	53.90	47.32	43.00	70.76	57.56	51.88	82.86	78.72	68.47	64.01	100	100
Re-Weight [30]	75.20	68.79	69.27	62.56	55.80	53.74	77.44	72.80	73.66	92.94	92.95	90.04	90.11	100	100
Oversampling [30]	77.44	70.73	72.40	62.78	56.01	53.99	76.70	68.49	69.50	92.46	92.47	89.79	89.85	>100	100
cRT [20]	76.54	69.26	70.95	60.60	54.05	52.36	75.10	67.52	68.08	91.24	91.17	86.02	86.00	100	100
PC Softmax [16]	76.42	71.30	71.24	65.70	**61.54**	61.49	76.92	75.82	74.19	93.32	93.32	86.59	86.62	100	100
DR-GCN [34]	73.90	64.30	63.10	56.18	49.57	44.98	72.38	58.86	53.05	N/A	N/A	N/A	N/A	100	100
GraphSmote [43]	76.76	69.31	70.21	62.58	55.94	54.09	75.98	70.96	71.85	92.65	92.61	89.31	89.39	>100	100
GraphENS [30]	77.76	72.94	73.13	**66.92**	60.19	58.67	78.12	74.13	74.58	93.82	93.81	91.94	91.94	>100	100
GraphDec	**78.29**	**73.94**	**74.25**	**66.90**	**61.56**	**61.85**	**78.20**	**76.05**	**76.32**	**93.85**	**94.02**	**92.19**	**92.16**	50	50

Table 3: Ablation study results for both tasks. Four rows of red represent removing four individual components from data sparsity perspective. Four rows of blue represent removing four individual components from model sparsity perspective. Best results are in bold.

Variant	Class-imbalanced Graph Classification (F1-ma.)							Class-imbalanced Node Classification (Acc.)				
	MUTAG	PROTEINS	D&D	NCI1	PTC-MR	DHFR	REDDIT-B	Cora-LT	CiteSeer-LT	PubMed-LT	A. Photos	A. Computer
GraphDec	**85.71**	**76.92**	**77.97**	**76.30**	**54.03**	64.25	**69.70**	**78.29**	**66.90**	**78.20**	**93.85**	**92.19**
w/o GS	80.10	72.49	63.16	72.83	48.48	48.57	61.40	68.96	60.33	56.22	73.22	67.84
w/o SS	80.95	72.44	72.26	73.85	52.96	63.99	70.61	77.15	64.67	76.15	79.09	91.33
w/o CAD	78.41	65.79	68.33	71.05	52.13	50.00	67.15	74.87	62.62	75.35	90.71	83.23
w/o RS	83.21	67.21	70.75	71.08	39.29	60.99	67.61	73.27	61.32	72.02	87.11	90.38
w/o RM	44.37	38.41	65.30	34.39	32.14	43.75	64.82	70.97	54.58	70.16	79.01	65.38
w/o SG	82.63	74.54	75.75	70.13	39.29	62.44	69.16	77.54	67.43	72.43	91.25	90.05
w/o CAG	83.50	61.02	69.23	72.83	41.18	62.41	64.14	75.78	63.43	73.07	92.77	87.40
w/o RW	79.25	65.54	67.37	72.99	45.90	61.53	63.16	76.46	65.36	75.54	90.54	89.10
w/o S.S.	80.07	71.90	73.79	69.72	45.58	**64.56**	65.67	74.82	65.28	74.00	86.14	86.40

to address the class-imbalanced issue by assigning different weights to different samples. However, it still requires the label data to calculate the weight and thus may not perform well when labels are missing. G^2GNN, as the best baseline, obtains decent performance by considering the usage of rich supervisory signals from both globally and locally neighboring graphs. Finally, the proposed model, GraphDec, achieves the best performance with the ability to capture dynamic sparsity on both GNN model and graph datasets. In addition, we rank the performance of GraphDec with regard to baseline methods on each dataset. GraphDec ranks 1.00 and 1.14 on average, which further demonstrates the superiority of GraphDec. Noticed that all existing methods utilize the entire datasets and the model weights. However, GraphDec uses only half of the data and weights to achieve superior performance.

5.3 Class-imbalanced Node Classification Performance

To demonstrate the effectiveness of GraphDec in handling class-imbalanced node data, we further evaluate GraphDec in the task of class-imbalanced node classification. We first evaluate GraphDec on three long-tailed citation networks (i.e., Cora-LT, CiteSeer-LT, PubMed-LT) and report the results on Table 2. We find that GraphDec obtains the best performance compared to baseline methods with different metrics. Specifically, GraphSmote and GraphENS achieve satisfactory performance by generating virtual nodes to enrich the information involved in the representations of minority category. However, GraphDec does not rely on synthetic virtual nodes to learn balanced representations, thereby avoiding the unnecessary learning costs on additional data. Similarly to the class-imbalanced graph classification task in Section 5.2, GraphDec leverages only half of the data and model weights, but achieves state-of-the-art performance, whereas all baselines require the full dataset and model weights but perform worse. To validate the efficacy of the proposed model on the real-world data, we also evaluate GraphDec on naturally class-imbalanced benchmark datasets (i.e., Amazon-Photo and Amazon-Computers). We can see that GraphDec yields the best performance on both datasets, which further demonstrates the effectiveness of our model in handling node imbalance.

5.4 Ablation Study

Since GraphDec is a unified learning framework composed of multiple components (steps) and explores dynamic sparsity training from both model and dataset perspectives, we conduct ablation study to evaluate the performance of different model variants. Specifically, GraphDec contains four components to address data sparsity and imbalance, including pruning samples by ranking gradients (GS), training with sparse dataset (SS), using cosine annealing to reduce dataset size (CAD), and recycling removed samples (RS), and another four components to address model sparsity and data imbalance, including pruning weights by ranking magnitudes (RM), using sparse GNN (SG), using cosine annealing to progressively reduce sparse GNN's size (CAG), and reactivate removed weights (RW). In addition, GraphDec employs self-supervision to calculate the gradient score. The details of model variants are provided in the Section B of Appendix. We analyze the contributions of different components by removing each of them independently. We conduct experiments for both tasks to comprehensively inspect each component. The results are shown in Table 3.

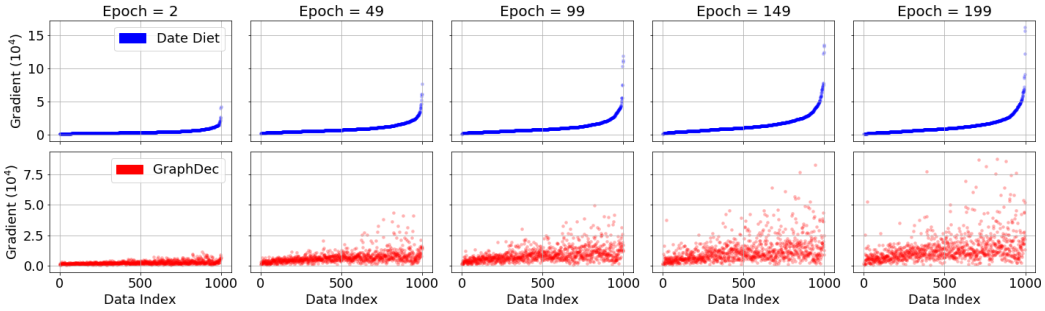

Figure 3: Evolution of data samples' gradients computed by data diet [31] (upper figures) and our GraphDec (lower figures) on NCI1 data.

From the table, we find that the performance drops after removing any component, which demonstrates the effectiveness of each component in enhancing the model performance. In general, both mechanisms for addressing data and model sparsity contribute significantly to the overall performance, demonstrating the necessity of these two mechanisms in solving sparsity problem. Self-supervision is also essential, contributing similarly to dynamic sparsity mechanisms. Besides, it enables us to identify informative data samples without human labels and capture graph knowledge in a self-supervised manner. In the dataset dynamic sparsity mechanism, GS and CAD contribute the most as sparse GNN's discriminability identifies hidden dynamic sparse subsets from the entire dataset accurately and efficiently. Regarding the model dynamic sparsity mechanism, removing RM and SG lead to a significant performance drop, which demonstrates that they are the key components in training the dynamic sparse GNN from the full GNN model. In particular, CAG enables the performance stability after the model pruning and helps capture information samples during decantation by assigning greater gradient norm. Among these variants, the full model GraphDec achieves the best result in most cases. This demonstrates the effectiveness of dataset dynamic sparsity mechanism, model dynamic sparsity mechanism, and self-supervision strategy in our model.

5.5 Analyzing Evolution of Sparse Subset by Scoring All Samples

To show GraphDec's capability in dynamically identifying informative samples, we show the visualization of sparse subset evolution of data diet and GraphDec on class-imbalanced NCI1 dataset in Figure 3. Specifically, we compute 1000 graph samples with their importance scores. These samples are then ranked according to their scores and marked with sample indexes. From the upper figures in Figure 3, we find that data diet is unable to accurately identify the dynamic informative nodes. Once a data sample has been removed from the training list due to the low score, the model forever disregards it as unimportant. However, the fact that a sample is currently unimportant does not imply that it will remain unimportant indefinitely. Especially when the model cannot detect the true importance of each sample in the early stage, it may lead to the premature elimination of vital nodes. Similarly, if a data sample is considered as important at early epochs (i.e., marked with higher sample index), it cannot be removed during subsequent epochs. Therefore, we observe that data diet can only increase the scores of samples within the high index range (i.e., 500–1000), while ignoring samples within the low index range (i.e., <500). However, GraphDec (Figure 3 (bottom)) can capture the dynamic importance of each sample regardless of the initial importance score. We see that samples with different indexes all have the opportunity to be considered important and therefore be included in the training list. Correspondingly, GraphDec takes into account a broader range of data samples when shrinking the training list, meanwhile maintaining flexibility towards the previous importance scores.

6 Conclusion

In this paper, to address the graph data imbalance challenge, we propose an efficient and effective method named **Graph Dec**antation (GraphDec). GraphDec leverages a dynamic sparse graph contrastive learning model to dynamically identified a sparse-but-informative subset for model training , in which the sparse GNN encoder is dynamically sampled from a dense GNN, and its capability of identifying informative samples is used to rank and update the training data in each epoch. Extensive experiments demonstrate that GraphDec outperforms state-of-the-art baseline methods for both node classification and graph classification tasks in the class-imbalanced scenario. The analysis of the sparse informative samples' evolution further explains the superiority of GraphDec catching the informative subset in different training periods effectively.

Limitations: Our method has not been validated on very large scale data due to the lack of very large-scale (e.g., OGB level) class-imbalanced graph datasets. In the future, we plan to create new larger-scale class-imbalanced graph benchmarks and extend our work to them.

References

[1] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *JAIR*, 2002.

[2] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018.

[3] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *ICML*, 2021.

[4] Tianlong Chen, Zhenyu Zhang, pengjun wang, Santosh Balachandra, Haoyu Ma, Zehao Wang, and Zhangyang Wang. Sparsity winning twice: Better robust generalization from more efficient training. In *ICLR*, 2022.

[5] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 2020.

[6] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. *NeurIPS*, 30, 2017.

[7] Talya Eden, Shweta Jain, Ali Pinar, Dana Ron, and C. Seshadhri. Provable and practical approximations for the degree distribution using sublinear graph samples. In *WWW*, 2018.

[8] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[9] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[10] Yonggan Fu, Qixuan Yu, Meng Li, Vikas Chandra, and Yingyan Lin. Double-win quant: Aggressively winning robustness of quantized deep neural networks via random precision training and inference. In *ICML*, 2021.

[11] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR*, 2016.

[14] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

[15] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[16] Youngkyu Hong, Seungju Han, Kwanghee Choi, Seokjun Seo, Beomsu Kim, and Buru Chang. Disentangling label distribution for long-tailed visual recognition. In *CVPR*, 2021.

[17] Sara Hooker, Aaron Courville, Gregory Clark, Yann Dauphin, and Andrea Frome. What do compressed deep neural networks forget? *arXiv preprint arXiv:1911.05248*, 2019.

[18] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[19] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 2002.

[20] Bingyi Kang, Saining Xie, Marcus Rohrbach, Zhicheng Yan, Albert Gordo, Jiashi Feng, and Yannis Kalantidis. Decoupling representation and classifier for long-tailed recognition. In *ICLR*, 2020.

[21] Zohar Karnin and Edo Liberty. Discrepancy, coresets, and sketches in machine learning. In *COLT*, 2019.

[22] Krishnateja Killamsetty, S Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *ICML*, 2021.

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

[24] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *ICLR*, 2019.

[25] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *ICML*, 2020.

[26] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 2018.

[27] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[28] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *ICML*, 2019.

[29] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[30] Joonhyung Park, Jaeyun Song, and Eunho Yang. GraphENS: Neighbor-aware ego network synthesis for class-imbalanced node classification. In *ICLR*, 2022.

[31] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. In *NeurIPS*, 2021.

[32] Md Aamir Raihan and Tor Aamodt. Sparse weight activation training. *NeurIPS*, 2020.

[33] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 2008.

[34] Min Shi, Yufei Tang, Xingquan Zhu, David Wilson, and Jianxun Liu. Multi-class imbalanced graph convolutional network learning. In *IJCAI*, 2020.

[35] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR*, 2019.

[36] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L Dyer, Remi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping. In *ICLR*, 2021.

[37] Aaron Van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[38] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[39] Yu Wang, Yuying Zhao, Neil Shah, and Tyler Derr. Imbalanced graph classification via graph-of-graph neural networks. *arXiv preprint arXiv:2112.00238*, 2021.

[40] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *TNNLS*, 2020.

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

[42] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *NeurIPS*, 2020.

[43] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *WSDM*, 2021.

[44] Dawei Zhou, Jingrui He, Hongxia Yang, and Wei Fan. Sparc: Self-paced network representation for few-shot rare category characterization. In *KDD*, 2018.

Checklist

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We believe there is no potential negative societal impact of our work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 4.1.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Section A of Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code is provided in the attachment.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 5.1.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section B of Appendix.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.1
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] They are public datasets/code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code is provided in the attachment.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] They are public datasets/code.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] They are public datasets/code.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Proof of Theorem 2

Theorem 2 *For a data selection algorithm [22, 31], we assume model training is optimized with full gradient descent. At $t \in [1, T]$ epoch, we denote the model's parameter as $\theta^{(t)}$ (satisfying $\left\|\theta^{(t)}\right\|^2 \leqslant d^2$, d is constant), the optimal model's parameter as θ^*, subset data as $\mathcal{D}_S^{(t)}$, learning rate as α. We also introduce the gradient error term as $Err(\mathcal{D}_S^{(t)}, \mathcal{L}, \mathcal{L}_{train}, \theta^{(t)}) = \left\|\sum_{i \in \mathcal{D}_S^{(t)}} \nabla_\theta \mathcal{L}_{train}^i(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|$, where $\mathcal{L}$ denotes training loss $\mathcal{L}_{train}$ over full training data or validation loss $\mathcal{L}_{val}$ over full validation data and $\mathcal{L}$ is a convex function. Then we have following guarantee:*

If $\mathcal{L}_{train}$ is Lipschitz continuous with parameter σ_T and $\alpha = \frac{d}{\sigma_T \sqrt{T}}$, then $\min_{t=1:T} \mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*) \leqslant \frac{d\sigma_T}{\sqrt{T}} + \frac{d}{T} \sum_{t=1}^{T-1} Err(\mathcal{D}_S^{(t)}, \mathcal{L}, \mathcal{L}_{train}, \theta^{(t)})$.

Proof 1 *The gradients of $\mathcal{L}_{val}$ and $\mathcal{L}_{train}$ are supposed to be σ-bounded by σ_V and σ_T respectively. According to gradient descent, we have:*

$$\nabla_\theta \mathcal{L}_{train}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) = \frac{1}{\alpha^{(t)}}(\theta^{(t)} - \theta^{(t+1)})^{\mathrm{T}}(\theta^{(t)} - \theta^*), \tag{12}$$

$$\nabla_\theta \mathcal{L}_{train}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) = \frac{1}{2\alpha^{(t)}}\left(\left\|\theta^{(t)} - \theta^{(t+1)}\right\|^2 + \|\theta_t - \theta^*\|^2 - \left\|\theta^{(t+1)} - \theta^*\right\|^2\right). \tag{13}$$

Since one update step $\theta^{(t)} - \theta^{(t+1)}$ can be optimized by gradient multiplying with learning rate $\alpha^{(t)} \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})$, we have:

$$\nabla_\theta \mathcal{L}_{train}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) = \frac{1}{2\alpha^{(t)}}\left(\left\|\alpha^{(t)} \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})\right\|^2 + \|\theta_t - \theta^*\|^2 - \left\|\theta^{(t+1)} - \theta^*\right\|^2\right). \tag{14}$$

Since $\nabla_\theta \mathcal{L}_{train}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*)$ can be represented as follows:

$$\begin{aligned} \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) &= \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) \\ &- \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) + \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*), \end{aligned} \tag{15}$$

then based on the combination of the Equation (14) *and Equation* (15)*, we have:*

$$\begin{aligned} &\nabla_\theta \mathcal{L}_{train}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) - \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) + \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) = \\ &\frac{1}{2\alpha^{(t)}}\left(\left\|\alpha^{(t)} \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})\right\|^2 + \|\theta_t - \theta^*\|^2 - \left\|\theta^{(t+1)} - \theta^*\right\|^2\right) \end{aligned} \tag{16}$$

$$\begin{aligned} \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) = &\frac{1}{2\alpha^{(t)}}\left(\left\|\alpha^{(t)} \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})\right\|^2 + \|\theta_t - \theta^*\|^2 - \left\|\theta^{(t+1)} - \theta^*\right\|^2\right) \\ &- \left(\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right)^{\mathrm{T}}(\theta^{(t)} - \theta^*). \end{aligned} \tag{17}$$

We assume learning rate $\alpha^{(t)}, t \in [0, T-1]$ is a constant value, then we have:

$$\begin{aligned} \sum_{t=0}^{T-1} \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) = &\frac{1}{2\alpha}\|\theta_0 - \theta^*\|^2 - \|\theta_T - \theta^*\|^2 + \sum_{t=0}^{T-1}\left(\frac{1}{2\alpha}\left\|\alpha \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})\right\|^2\right) \\ &+ \sum_{t=0}^{T-1}\left(\left(\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right)^{\mathrm{T}}(\theta^{(t)} - \theta^*)\right). \end{aligned}$$

Since we assume $\|\theta_T - \theta^*\|^2 \geqslant 0$*, then we have:*

$$\sum_{t=0}^{T-1} \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*) \leqslant \frac{1}{2\alpha}\|\theta_0 - \theta^*\|^2 + \sum_{t=0}^{T-1}(\frac{1}{2\alpha}\left\|\alpha \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})\right\|^2) + \sum_{t=0}^{T-1}\left(\left(\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right)^{\mathrm{T}}(\theta^{(t)} - \theta^*)\right). \tag{18}$$

We assume $\mathcal{L}$ *is convex and* $\mathcal{L}_{train}$ *is lipschitz continuous with parameter* σ_T*. Then for convex function* $\mathcal{L}(\theta)$*, we have* $\mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*) \leqslant \nabla_\theta \mathcal{L}(\theta^{(t)})^{\mathrm{T}}(\theta^{(t)} - \theta^*)$*. By combining this result with Equation 18, we get:*

$$\sum_{t=0}^{T-1} \mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*) \leqslant \frac{1}{2\alpha}\|\theta_0 - \theta^*\|^2 + \sum_{t=0}^{T-1}(\frac{1}{2\alpha}\left\|\alpha \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})\right\|^2) + \sum_{t=0}^{T-1}\left(\left(\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right)^{\mathrm{T}}(\theta^{(t)} - \theta^*)\right). \tag{19}$$

Since $\|L_T(\theta)\| \leqslant \sigma_T$*,* $\left\|\alpha \nabla_\theta \mathcal{L}_{train}(\theta^{(t)})\right\| \leqslant \sigma_T$*, and we assume* $\|\theta - \theta^*\| \leqslant d$*, then we have:*

$$\sum_{t=0}^{T-1} \mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*) \leqslant \frac{d^2}{2\alpha} + \frac{T\alpha\sigma_T^2}{2} + \sum_{t=0}^{T-1} d\left(\left\|\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|\right), \tag{20}$$

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*) \leqslant \frac{d^2}{2\alpha T} + \frac{\alpha\sigma_T^2}{2} + \sum_{t=0}^{T-1} \frac{d}{T}\left(\left\|\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|\right). \tag{21}$$

Since $\min\left(\mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*)\right) \leqslant \frac{1}{T}\sum_{t=0}^{T-1} \mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*)$*, based on Equation 21, we have:*

$$\min\left(\mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*)\right) \leqslant \frac{d^2}{2\alpha T} + \frac{\alpha\sigma_T^2}{2} + \sum_{t=0}^{T-1} \frac{d}{T}\left(\left\|\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|\right). \tag{22}$$

We set learning rate $\alpha = \frac{d}{\sigma_T\sqrt{T}}$ *and then have:*

$$\min\left(\mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^*)\right) \leqslant \frac{d\sigma_T}{\sqrt{T}} + \sum_{t=0}^{T-1} \frac{d}{T}\left(\left\|\nabla_\theta \mathcal{L}_{train}(\theta^{(t)}) - \nabla_\theta \mathcal{L}(\theta^{(t)})\right\|\right). \tag{23}$$

B Experimental Details

B.1 Datasets Details

In this work, seven graph classification datasets and five node classification datasets are used to evaluate the effectiveness of our proposed model, we provided their detailed statistics in Table 4. For graph classification datasets, we follow the imbalance setting of [39] to set the train-validation split as 25%/25% and change the imbalance ratio from 5:5 (balanced) to 1:9 (imbalanced). The rest of the dataset is used as the test set. The specified imbalance ratio of each dataset is clarified after its name in Table 5. For node classification datasets, we follow [33] to set the imbalance ratio of Cora, CiteSeer and PubMed as 10. Besides, the setting of Amazon-Photo and Amazon-Computers are borrowed from [30], where the imbalance ratio ρ is set as 82 and 244, respectively.

B.2 Baseline Details

We compare our model with a variety of baseline methods using different rebalance methods:

Table 4: Original dataset details for imbalanced graph classification and imbalanced node classification tasks.

Task	Dataset	# Graphs	# Nodes	# Edges	# Features	# Classes
Graph	MUTAG	188	∼17.93	∼19.79	-	2
	PROTEINS	1,113	∼39.06	∼72.82	-	2
	D&D	1,178	∼284.32	∼715.66	-	2
	NCI1	4,110	∼29.87	∼32.30	-	2
	PTC-MR	344	∼14.29	∼14.69	-	2
	DHFR	756	∼42.43	∼44.54	-	2
	REDDIT-B	2,000	∼429.63	∼497.75	-	2
Node	Cora	-	2,485	5,069	1,433	7
	Citeseer	-	2,110	3,668	3,703	6
	Pubmed	-	19,717	44,324	500	3
	A-photo	-	7,650	238,162	745	8
	A-computers	-	13,381	245,778	767	10

I. For **imbalanced graph classification** [39], four models are included as baselines in our work, we list these baselines as follow:

(1) **GIN** [41], a popular supervised GNN backbone for graph tasks due to its powerful expressiveness on graph structure;

(2) **InfoGraph** [35], an unsupervised graph learning framework by maximizing the mutual information between the whole graph and its local topology of different levels;

(3) **GraphCL** [42], learning unsupervised graph representations via maximizing the mutual information between the original graph and corresponding augmented views;

(4) **G^2GNN** [39], a re-balanced GNN proposed to utilize additional supervisory signals from both neighboring graphs and graphs themselves to alleviate the imbalance issue of graph.

II. For **imbalanced node classification**, we consider nine baseline methods in our work, including

(1) **vanilla**, denoting that we train GCN normally without any extra rebalancing tricks;

(2) **re-weight** [19], denoting we use cost-sensitive loss and re-weight the penalty of nodes in different classes;

(3) **oversampling** [30], denoting that we sample nodes of each class to make the data's number of each class reach the maximum number of corresponding class's data;

(4) **cRT** [20], a post-hoc correction method for decoupling output representations;

(5) **PC Softmax** [16], a post-hoc correction method for decoupling output representations, too;

(6) **DR-GCN** [34], building virtual minority nodes and forces their features to be close to the neighbors of a source minority node;

(7) **GraphSMOTE** [43], a pre-processing method that focuses on the input data and investigates the possibility of re-creating new nodes with minority features to balance the training data.

(8) **GraphENS** [30], proposing a new augmentation method to construct an ego network from all nodes for learning minority representation.

We use Graph Convolutional Network (GCN) [23] as the default architecture for all rebalance methods.

B.3 Details of GraphDec Variants

The details of model variants are provided as follows (code is provided in anonymous repository[1]):

I. Specifically, GraphDec contains four components to address data sparsity and imbalance: (1) **GS** is sampling informative subset data according to ranking gradients; (2) **SS** is training model with the sparse dataset, correspondingly; (3) **CAD** is using cosine annealing to reduce dataset size; (4) **RS** is

[1]https://www.dropbox.com/sh/ylk3y0uwwohd8pc/AABwmmKO4NfY4gURq7ztyjnZa?dl=0

Table 5: Imbalanced graph classification results. The numbers after each dataset name indicate the imbalance ratios of minority to majority categories. We report the macro F1-score and micro F1-score with the standard errors as Results are reported as $mean \pm std$ for 3 repetitions on each dataset. We bold the best performance.

Rebalance Method	Basis	MUTAG (5:45)		PROTEINS (30:270)		D&D (30:270)		NCI1 (100:900)	
		F1-ma.	F1-mi.	F1-ma.	F1-mi.	F1-ma.	F1-mi.	F1-ma.	F1-mi.
vanilla	GIN [41]	52.50 ± 18.70	56.77 ± 14.14	25.33 ± 7.53	28.50 ± 5.82	9.99 ± 7.44	11.88 ± 9.49	18.24 ± 7.58	18.94 ± 7.12
	InfoGraph [35]	69.11 ± 9.03	69.68 ± 7.77	35.91 ± 7.58	36.81 ± 6.51	21.41 ± 4.51	27.68 ± 7.52	33.09 ± 3.30	34.03 ± 3.68
	GraphCL [42]	66.82 ± 11.56	67.77 ± 9.78	40.86 ± 6.94	41.24 ± 6.38	21.02 ± 3.05	26.80 ± 4.95	31.02 ± 2.69	31.62 ± 3.05
up-sampling	GIN [41]	78.03 ± 7.62	78.77 ± 7.67	65.64 ± 2.67	71.55 ± 3.19	41.15 ± 3.74	70.56 ± 10.28	59.19 ± 4.39	71.80 ± 7.02
	InfoGraph [35]	78.62 ± 6.84	79.09 ± 6.86	62.68 ± 2.70	66.02 ± 3.18	41.55 ± 2.32	71.34 ± 6.76	53.38 ± 1.88	62.20 ± 2.63
	GraphCL [42]	80.06 ± 7.79	80.45 ± 7.86	64.21 ± 2.53	65.76 ± 2.61	38.96 ± 3.01	64.23 ± 8.10	49.92 ± 2.15	58.29 ± 3.30
re-weight	GIN [41]	77.00 ± 9.59	77.68 ± 9.30	54.54 ± 6.29	55.77 ± 7.11	28.49 ± 5.92	40.79 ± 11.84	36.84 ± 8.46	39.19 ± 10.05
	InfoGraph [35]	80.85 ± 7.75	81.68 ± 7.83	65.73 ± 3.10	69.60 ± 3.68	41.92 ± 2.28	72.43 ± 6.63	53.05 ± 1.12	62.45 ± 1.89
	GraphCL [42]	80.20 ± 7.27	80.84 ± 7.43	63.46 ± 2.42	64.97 ± 2.41	40.29 ± 3.31	67.96 ± 8.98	50.05 ± 2.09	58.18 ± 3.08
G^2GNN [39]	remove edge	80.37 ± 6.73	81.25 ± 6.87	67.70 ± 2.96	73.10 ± 4.05	43.25 ± 3.91	77.03 ± 9.98	63.60 ± 1.57	72.97 ± 1.81
	mask node	83.01 ± 7.01	83.59 ± 7.14	67.39 ± 2.99	73.30 ± 4.19	43.93 ± 3.46	79.03 ± 10.78	64.78 ± 2.86	74.91 ± 2.14
GraphDec	dynamic sparsity	**85.71±10.20**	**85.71±11.10**	**76.92±6.15**	**76.89±6.80**	**77.97±6.75**	77.02±6.26	**76.30±5.12**	**76.29±6.27**

Rebalance Method	Basis	PTC-MR (9:81)		DHFR (12:108)		REDDIT-B (50:450)	
		F1-ma.	F1-mi.	F1-ma.	F1-mi.	F1-ma.	F1-mi.
vanilla	GIN [41]	17.74 ± 6.49	20.30 ± 6.06	35.96 ± 8.87	49.46 ± 4.90	33.19 ± 14.26	36.02 ± 17.38
	InfoGraph [35]	25.85 ± 6.14	26.71 ± 6.50	50.62 ± 8.33	56.28 ± 4.58	57.67 ± 3.80	67.10 ± 4.91
	GraphCL [42]	24.22 ± 6.21	25.16 ± 5.25	50.55 ± 10.01	56.31 ± 6.12	53.40 ± 4.06	62.19 ± 5.68
up-sampling	GIN [41]	44.78 ± 8.01	55.43 ± 14.25	55.96 ± 10.06	59.39 ± 6.52	66.71 ± 3.92	83.00 ± 5.18
	InfoGraph [35]	44.29 ± 4.69	48.91 ± 7.49	59.49 ± 5.20	61.62 ± 4.18	67.01 ± 3.34	78.68 ± 3.71
	GraphCL [42]	45.12 ± 7.33	53.50 ± 13.31	60.29 ± 9.04	61.71 ± 6.75	62.01 ± 3.97	75.84 ± 3.98
re-weight	GIN [41]	36.96 ± 14.08	43.09 ± 20.01	55.16 ± 9.47	57.78 ± 6.69	45.17 ± 8.46	51.92 ± 12.29
	InfoGraph [35]	44.09 ± 5.62	49.17 ± 8.78	58.67 ± 5.82	60.24 ± 4.80	65.79 ± 3.38	77.35 ± 3.96
	GraphCL [42]	44.75 ± 7.62	52.22 ± 13.24	60.87 ± 6.33	61.93 ± 5.15	62.79 ± 6.93	76.15 ± 9.15
G^2GNN [39]	remove edge	46.40 ± 7.73	56.61 ± 13.72	61.63 ± 10.02	63.61 ± 6.05	68.39 ± 2.97	86.35 ± 2.27
	mask node	46.61 ± 8.27	56.70 ± 14.81	59.72 ± 6.83	61.27 ± 5.40	67.52 ± 2.60	85.43 ± 1.80
GraphDec	dynamic sparsity	**54.03±8.22**	**61.17±10.24**	**64.25±9.54**	**67.91±7.10**	**69.70±7.20**	**87.00±9.36**

Table 6: Imbalanced node classification results. We report the accuracy, balanced accuracy and macro F1-score with the standard errors as $mean \pm std$ for 3 repetitions on each dataset. We bold the best performance.

Method	Cora-LT			CiteSeer-LT			PubMed-LT			A.P. (ρ =82)		A.C. (ρ =244)	
	Acc.	bAcc.	F1-ma.	Acc.	bAcc.	F1-ma.	Acc.	bAcc.	F1-ma.	(b)Acc.	F1-ma.	(b)Acc.	F1-ma.
vanilla	73.66±0.28	62.72±0.39	63.70±0.43	53.90±0.70	47.32±0.61	43.00±0.70	70.76±0.74	57.56±0.59	51.88±0.53	82.86±0.30	78.72±0.52	68.47±2.19	64.01±3.18
Re-Weight [30]	75.20±0.19	68.79±0.18	69.27±0.26	62.56±0.32	55.80±0.28	53.74±0.28	77.44±0.21	72.80±0.38	73.66±0.27	92.94±0.13	92.95±0.13	90.04±0.29	90.11±0.28
Oversampling [30]	77.44±0.09	70.73±0.10	72.40±0.11	62.78±0.37	56.01±0.35	53.99±0.37	76.70±0.48	68.49±0.28	69.50±0.38	92.46±0.47	92.47±0.48	89.79±0.16	89.85±0.17
cRT [20]	76.54±0.22	69.26±0.48	70.95±0.50	60.60±0.25	54.05±0.22	52.36±0.22	75.10±0.23	67.52±0.72	68.08±0.85	91.24±0.28	91.17±0.29	86.02±0.55	86.00±0.56
PC Softmax [16]	76.42±0.34	71.30±0.45	71.24±0.52	65.70±0.42	61.54±0.45	61.49±0.49	76.92±0.26	75.82±0.25	74.19±0.25	93.32±0.25	93.32±0.25	86.59±0.92	86.62±0.91
DR-GCN [34]	73.90±0.29	64.30±0.39	63.10±0.57	56.18±1.10	49.57±1.08	44.98±1.29	72.38±0.19	58.86±0.15	53.05±0.13	N/A	N/A	N/A	N/A
GraphSmote [43]	76.76±0.31	69.31±0.37	70.21±0.64	62.58±0.30	55.94±0.34	54.09±0.37	75.98±0.22	70.96±0.36	71.85±0.32	92.65±0.31	92.61±0.32	89.31±0.34	89.39±0.35
GraphENS [30]	77.76±0.09	72.94±0.15	73.13±0.11	66.92±0.21	60.19±0.21	58.67±0.25	78.12±0.06	74.13±0.22	74.58±0.13	93.82±0.13	93.81±0.12	91.94±0.17	91.94±0.17
GraphDec	**78.29±0.40**	**73.94±0.67**	**74.25±0.83**	**66.90±0.65**	**61.56±0.72**	**61.85±0.96**	**78.20±0.45**	**76.05±0.66**	**76.32±0.66**	**93.85±0.72**	**94.02±0.67**	**92.19±0.73**	**92.16±0.75**

recycling removed samples, correspondingly. To investigate their corresponding effectiveness, we remove them correspondingly as:

(1) **w/o** GS is that we randomly sample subset from the full set;

(2) **w/o** SS is that we train GNN with the full set;

(3) **w/o** CAD is that we directly reduce dataset size to target dataset size and it is same as data diet;

(4) **w/o** RS is not recycling any removed samples.

II. Another four components to address model sparsity and data imbalance: (1) **RM** samples model weights according to ranking magnitudes; (2) **SG** is using sparse GNN, correspondingly; (3) **CAG** is using cosine annealing to progressively reduce sparse GNN's size; (4) **RW** is reactivating removed weights. To investigate their effectiveness, we remove them correspondingly as:

(1) **w/o RM** is that we randomly sample activated weights from full GNN model;

(2) **w/o SG** is that we train full GNN during forward and backward;

(3) **w/o CAG** is that we directly reduce the model size to target sparsity rate;

(4) **w/o RW** is not reactivating any removed weights during sparse training.

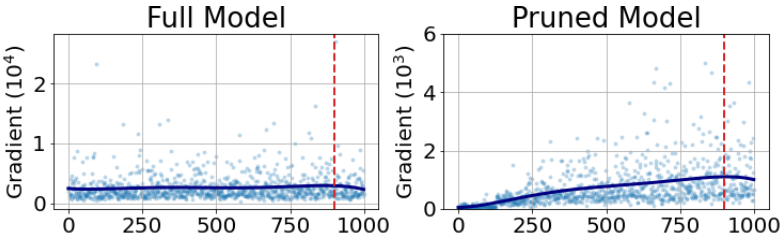

Figure 4: Results of data samples' gradients computed by full GNN model and our dynamic sparse GNN model on NCI1 data. Red dashed line: on the left side, points on the x-axis [0, 900] are majority class; on the right side, points on the x-axis [900, 1000] are minority class.

Table 7: Computational time (second) comparisons.

Model	Method	PubMed-LT	Cora-LT	CiteSeer-LT	PROTEINS	PTC_MR	MUTAG
GCN	vanilla	2.436	2.154	2.129	12.798	4.295	2.989
	re-weight	2.330	2.282	2.150	12.903	4.410	3.125
	re(/over)-sample	3.241	2.860	2.794	15.996	5.734	4.022
	GraphCL	3.747	3.412	3.399	14.981	5.049	3.215
	GraphDec	2.243	1.995	1.952	10.614	4.212	2.090

B.4 Full Results with Error Bars

We provide the F1-macro and F1-micro scores along with their standard deviation for our model and other baselines across both graph classification and node classification tasks in Table 5 and Table 6. We report their results as $mean \pm std$ for 3 repetitions on each metric for each dataset.

C Finding Informative Samples by Sparse GNN

Compared with the full GNN model, our dynamic sparse GNN model is more sensitive to recognizing informative data samples which can be empirically verified by Figure 4. As we can see in the figure, our dynamical pruned model can assign larger gradients for minority-class samples than majority-class samples during contrastive training, while the full model generally assigns relatively uniform gradients for both minority-class and majority-class samples. Thus, the proposed dynamically pruned model demonstrates its discriminatory ability on minority-class and can thereby sample more minority-class data according to computed unsupervised gradients.

D Computational Cost

To evaluate the proposed GraphDec's computational cost on a wide range of datasets, results in Table 7 that include three different class-imbalanced node classification datasets (PubMed-LT, Cora-LT, CiteSeer-LT), three different class-imbalanced graph classification datasets (MUTAG, PROTEINS, PTC_MR), and four baselines (vanilla GCN, re-weight, re(/over)-sample, GraphCL). We run 200 epochs for each method to measure their computational time (second) for training. On NVIDIA GeForce RTX 3090 GPU device, we get the running time, as reported in Table 7. All models are implemented in PyTorch Geometric [8]. According to the results, our GraphDec has less computation cost than prior methods. The following explains why augmentation doubles the input graph without increasing overall computation costs: (i) The augmentations we use (e.g, node dropping and edge dropping) reduces the size of input graphs (i.e., node number decreases 25%, edge number decreases 25-35%); (ii) During each epoch, our GraphDec prunes datasets so that only approximately 50% of the training data is used. (iii) our GraphDec prunes GNN model weight, resulting in a lighter model during training. (iv) Despite the fact that augmentation doubles the number of input graphs, the additional new views only consume forward computational resources without requiring a backward step or weight update step, thereby only marginally increasing computation.