CurGraph: Curriculum Learning for Graph Classification

Yiwei Wang
National University of Singapore
Singapore
wangyw_seu@foxmail.com

Wei Wang
National University of Singapore
Singapore
wangwei@comp.nus.edu.sg

Yuxuan Liang
National University of Singapore
Singapore
yuxliang@outlook.com

Yujun Cai
Nanyang Technological University
Singapore
yujun001@e.ntu.edu.sg

Bryan Hooi
National University of Singapore
Singapore
bhooi@comp.nus.edu.sg

ABSTRACT

Graph neural networks (GNNs) have achieved state-of-the-art performance on graph classification tasks. Existing work usually feeds graphs to GNNs in random order for training. However, graphs can vary greatly in their difficulty for classification, and we argue that GNNs can benefit from an easy-to-difficult curriculum, similar to the learning process of humans. Evaluating the difficulty of graphs is challenging due to the high irregularity of graph data. To address this issue, we present the **CurGraph** (Curriculum Learning for Graph Classification) framework, that analyzes the graph difficulty in the high-level semantic feature space. Specifically, we use the infomax method to obtain graph-level embeddings and a neural density estimator to model the embedding distributions. Then we calculate the difficulty scores of graphs based on the intra-class and inter-class distributions of their embeddings. Given the difficulty scores, CurGraph first exposes a GNN to easy graphs, before gradually moving on to hard ones. To provide a soft transition from easy to hard, we propose a smooth-step method, which utilizes a time-variant smooth function to filter out hard graphs. Thanks to CurGraph, a GNN learns from the graphs at the border of its capability, neither too easy or too hard, to gradually expand its border at each training step. Empirically, CurGraph yields significant gains for popular GNN models on graph classification and enables them to achieve superior performance on miscellaneous graphs.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; *Learning from implicit feedback*; *Batch learning*; *Learning latent representations*; *Neural networks.*

KEYWORDS

graph classification, curriculum learning, graph neural networks

ACM Reference Format:
Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. CurGraph: Curriculum Learning for Graph Classification. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3442381.3450025

WWW '21, April 19–23, 2021, Ljubljana, Slovenia

ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3450025

1 INTRODUCTION

Graph classification is a fundamental task on graph data, which aims to predict the class labels of entire graphs. The modern tools of choice for this task are graph neural networks (GNNs). Typically, GNNs build node representations from node features and graph topology via the 'message passing' mechanism and then make graph-level predictions by summarizing the node representations through a readout function [63], [49].

Although a lot of attention has been paid to developing new GNN architectures of higher representational capacity [30], [63], [65], it is also valuable to explore how to design advanced training methods to improve GNNs. Most existing work performs training of GNNs in a straightforward manner, i.e., all graphs are treated equally and presented in random order during training. However, even in the same dataset, graphs can vary significantly in their difficulty levels. For example, some graphs are easy to discriminate by their significant and popular substructures, while others require sophisticated reasoning due to their complicated topology and indistinct patterns (see Fig. 2). Extensive research discovers that feeding the training samples in a meaningful order, starting from easy ones and gradually taking more difficult ones, can benefit machine learning algorithms [4], [51], [62]. This strategy is known as Curriculum Learning.

Curriculum Learning was first formally proposed in [4], inspired by humans' learning process: an infant starts with a simple initial state, and then builds on that to handle more and more sophisticated concepts gradually. Recent years have witnessed successful applications of Curriculum Learning in the fields of Computer Vision [24], [21], [42] and Natural Language Processing [41], [38]. In terms of optimization, Curriculum Learning excludes the negative impacts from difficult or even noisy samples in the early training stages, and guides the model towards better local minima in the parameter space. Motivated by this, we argue that GNNs can benefit from Curriculum Learning on graph classification, which, however, remains under-explored.

A key challenge of designing a Curriculum Learning method for graph classification lies in how to evaluate the difficulty of graphs. The evaluation is non-trivial because the graph data is highly irregular and noisy. Each graph exhibits complicated relationships between nodes, and the number of nodes (#Nodes) and edges (#Edges) vary by orders of magnitudes for different graphs. Prior curriculum learning approaches evaluate difficulty by defining the heuristic metrics by observing the characteristics of the particular target task and data [4], [38]. In [48], for instance, authors take shorter sentences as easier samples for grammar induction. We can follow them to take the #Nodes and #Edges to evaluate graphs because higher #Nodes or #Edges implies more complicated topology. However, these heuristics, relying on low-level features, may not reflect the difficulty perceived by GNNs, which learn high-level semantic features via a multi-layer nonlinear function, and thus cannot generalize to different GNNs and datasets.

To address this challenge, the central idea of this paper is to encode the graphs into high-level semantic embeddings using GNNs and to calculate the difficulty scores based on the intra-class and inter-class distributions of embeddings (see Fig. 1). We encapsulate this idea in a new GNN framework, called **CurGraph** (Curriculum Learning for Graph Classification), that uses the infomax method to obtain graph embeddings and a neural density estimator to model embedding distributions. Based on the difficulty scores, we further propose a smooth-step method to provide a soft transition from easy to hard graphs for GNNs. Then at each training step, a GNN focuses on 'interesting' examples, that are near its border of capability, neither too easy nor too hard, to expand the border gradually. CurGraph can be incorporated into popular GNN architectures for graph classification. It enhances GNNs without extra inference cost by feeding the graphs in an easy-to-difficult fashion for training.

We evaluate CurGraph on the graph classification task using the standard chemical [15] and social [64] datasets. Qualitatively, CurGraph conducts interpretable difficulty evaluation on graphs based on the statistical analysis in the high-level embedding space. Quantitatively, we observe the improvements in test accuracy of graph classification for GNN models. The improvements are higher than that given by the heuristic curriculum and the advanced Curriculum Learning methods from other fields [21], [42]. Overall, CurGraph improves the popular GIN [63] and EigenPool [33] by a significant margin, and enhances them to outperform the baseline methods.

2 RELATED WORK

Graph Classification and Graph Neural Networks. Early solutions to graph classification include graph kernels. The pioneering work [23] decomposes graphs into small substructures and computes kernel functions based on their pair-wise similarities. Subsequent work proposes various substructures, such as subgraphs [27], paths [7], and subtrees [47], [35]. We refer readers to [37], [28] for a general overview. More recently, many efforts have been made to design graph neural networks (GNNs) for graph classification [43], [31], [36], [20], [65], [68], [63].

Due to the long history of Graph Neural Networks, we refer readers to [61] and [69] for a comprehensive review. The first work that proposes the convolution operation on graph data is [9]. More recently, [26] and [13] speed up the graph convolution operations by introducing localized filters based on Chebyshev expansion. Specifically, [26] has made breakthrough advancements in the representation learning on graphs. As a result, the model proposed in [26] is generally denoted as the vanilla GCN, or GCN (Graph Convolutional Network). After [26], numerous methods are proposed for better performance on the graph learning [52], [60], [14], [59], [58], [5], [66].

To improve the model capacity, [54], [67], and [22] use the attention mechanism to better capture neighbor features by dynamically adjusting edge weights. Mixture Model Network (MoNet) [34] adopts a different approach to assign edge weights. It introduces node pseudo-coordinates to determine the relative position between a node and its neighbors, then defines a weight function to map the relative positions to edge weights. [55] alternatively drives local network embeddings to capture global structural information by maximizing local mutual information. [10] proposes a non-uniform graph convolutional strategy, which learns different convolutional kernel weights for different neighboring nodes according to their semantic meanings. LGCN [19] ranks a node's neighbors based on node features. It assembles a feature matrix that consists of its neighborhood and sorts this feature matrix along each column. [57] proposes the low-pass 'message passing' for robust graph neural networks, inhibiting the transmission of the adversarial information propagated through edges.

The above-mentioned work focuses on developing GNN architectures. In contrast, our framework is orthogonal to them in the sense that we propose a new training method that enhances a GNN model by feeding the training samples to it in an easy-to-difficult curriculum. As far as we know, we are the first to develop a curriculum learning approach for graph classification.

Curriculum Learning. Early research [17], [39], [29] at the intersection between cognitive science and machine learning proposes the idea of training machine learning models in an easy-to-difficult fashion. Based on these work, [4] proposes Curriculum Learning, that organizes training samples in a meaningful order to learn more complex concepts gradually. Recent research successfully applies Curriculum Learning into Computer Vision [50], [11], [24], [21], [42] and Natural Language Processing [41], [51], [62], [38], which generally follows two steps: evaluating the difficulty first, and then ordering the training samples accordingly. To the best of our knowledge, no work has discussed curriculum learning in the context of graph classification. On the idea of designing the curriculum learning method, our approach is most closely related to [21], which distinguishes the noisy images in the feature space. Compared with it, we propose three main advancements in CurGraph. First, we use the state-of-the-art infomax method to extract graph embeddings and the advanced neural density estimator to model embedding distributions instead of the simple clustering in CurriculumNet. Second, we analyze the effects on the difficulty of both intra-class and inter-class embeddings using statistical metrics, while CurriculumNet only considers the outliers in each class separately. Third, we propose the smooth-step curriculum learning strategy to provide soft transitions across training stages.

3 METHODOLOGY

In this section, we describe our CurGraph (Curriculum Learning for Graph Classification) framework for graph classification. CurGraph accepts a GNN model f and a graph set $\mathcal{G}$ as the inputs: it strengthens the inference performance of f by feeding the graphs

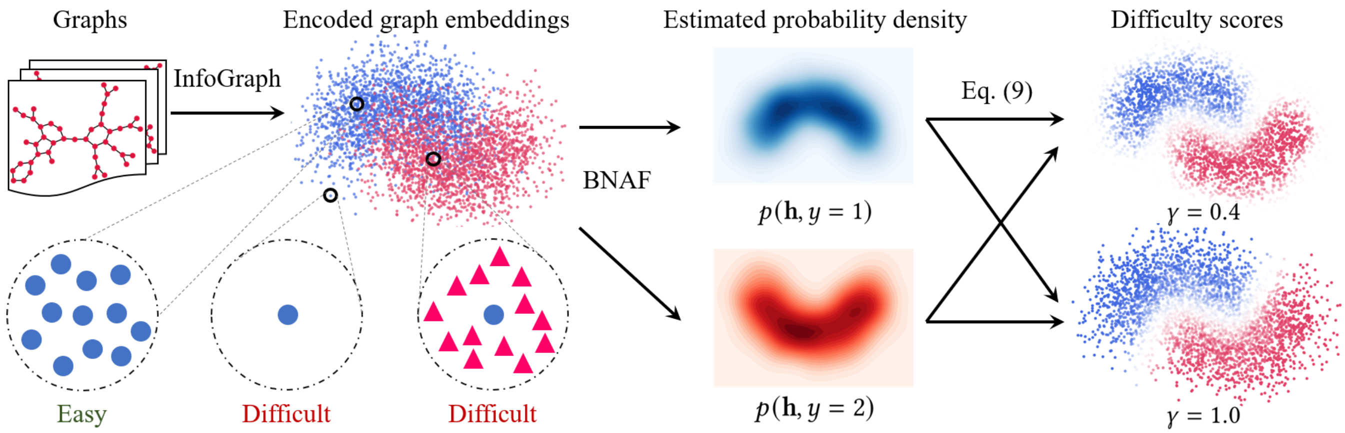


Figure 1: *Infomax Curriculum Design.* We use InfoGraph [49] to obtain graph representations, and BNAF [12] for density estimation. We calculate difficulty scores from intra-class and inter-class densities of Graph Embeddings (by Eq. (9)). Levels of transparency are positively related to difficulty scores. $\gamma = 0.4$ assigns higher difficulty values to outliers than $\gamma = 1.0$.

in $\mathcal{G}$ to f in an easy-to-difficult fashion during training. CurGraph consists of two main components: (i) An *infomax curriculum design module* evaluates the difficulty of graphs from the inter-class and intra-class distributions of graph-level embeddings. (ii) A *smooth-step curriculum learning strategy* provides soft transitions when exposing the GNN model to graphs across different difficulty levels. Details are introduced next.

3.1 Infomax Curriculum Design

We define a set of graphs as $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$, where N is the number of graphs. A graph G_i consists of a set of nodes $\mathcal{V}_i$ and a set of edges $\mathcal{E}_i$. Graph classification aims to learn a mapping function $f : G_i \rightarrow \hat{y}_i$ that maps every graph to a predicted class label $\hat{y}_i$.

Graph neural networks (short as GNNs) are known as the state-of-the-art solution for graph classification. Typically, GNNs obtain the nodes' representations $\mathbf{h}_{v_i}^{(l)}$ through the 'message passing' mechanism, and summarizes nodes' representations into a single graph embedding through a 'readout' function:

$$\mathbf{h}_i = \text{READOUT}\left(\left\{\mathbf{h}_{v_j}^{(L)} \mid v_j \in \mathcal{V}_i\right\}\right), \tag{1}$$

where L is the number of GNN layers. READOUT can be a simple permutation invariant function such as summation or a more sophisticated graph-level pooling function [65], [68].

At the core of our idea is to develop difficulty measurements on graphs that are justified for GNN models. One straightforward way is to use some heuristic metrics observed from the low-level data characteristics. For example, we can simply take the number of nodes (#Nodes) or the number of edges (#Edges) to measure the difficulty. This intuitively makes sense in humans' views because larger #Nodes/#Edges implies more complicated graph structures. However, GNNs encode graphs through the advanced non-linear transformations. The low-level heuristic metrics, which depend only on the data rather than a specific model, may not meet the difficulty perceived by GNNs and cannot generalize to different GNNs.

In this work, we analyze the graph embeddings $\mathbf{h}_i$ retrieved by the GNN model f, and calculate the difficulty scores from the intra-class and inter-class distributions of $\{\mathbf{h}_i \mid G_i \in \mathcal{G}\}$, as presented in Fig. 1. The graph embedding $\mathbf{h}_i$ reflects the high-level semantic information that f obtains from G_i. Thus, our analysis on $\mathbf{h}_i$ infers the specific difficulty perceived by f and can generalize to different GNN models adaptively. We use the state-of-the-art unsupervised GNN scheme, InfoGraph [49], to obtain the graph-level embeddings. InfoGraph obtains graph representations by maximizing the mutual information between graph-level representations ($\mathbf{h}_i$) and node-level ones $\left(\left\{\mathbf{h}_{v_j}^{(L)} \mid v_j \in \mathcal{V}_i\right\}\right)$, so that the graph representations can learn to encode aspects of the data that are shared across all substructures. Recent research shows that the graph embeddings provided by InfoGraph are powerful on multiple downstream tasks [53], [2]. Here, we extend its application to curriculum learning. InfoGraph follows the infomax optimization principle [32], [3], so we call our method 'infomax curriculum design'.

In the embedding space of $\{\mathbf{h}_i \mid G_i \in \mathcal{G}\}$, for graph G_i and its embedding $\mathbf{h}_i \in \mathbb{R}^d$, we build a d-dimensional ball with radius R centered at $\mathbf{h}_i$:

$$B(\mathbf{h}_i, R) = \left\{\mathbf{x} \mid \|\mathbf{x} - \mathbf{h}_i\| \leq R\right\}, \tag{2}$$

where R is a small value to include the graphs semantically similar to G_i. For the ease of expression, we term the graphs that fall in $B(\mathbf{h}_i, R)$ as the 'neighbors' of G_i. Suppose the ground-truth label of graph G_i is y_i. We count the ratio of the graphs belonging to class c and being the neighbors of G_i as:

$$\frac{1}{N} \sum_{j=1,\ldots,N, y_j = c} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right), \tag{3}$$

where $I(\text{Statement})$ is an indicator function which outputs 1 if Statement is true, and 0 otherwise. Intuitively, the higher the fraction of neighbors of G_i belongs to the same class as G_i, the lower G_i's difficulty should be. Hence, we propose the following function

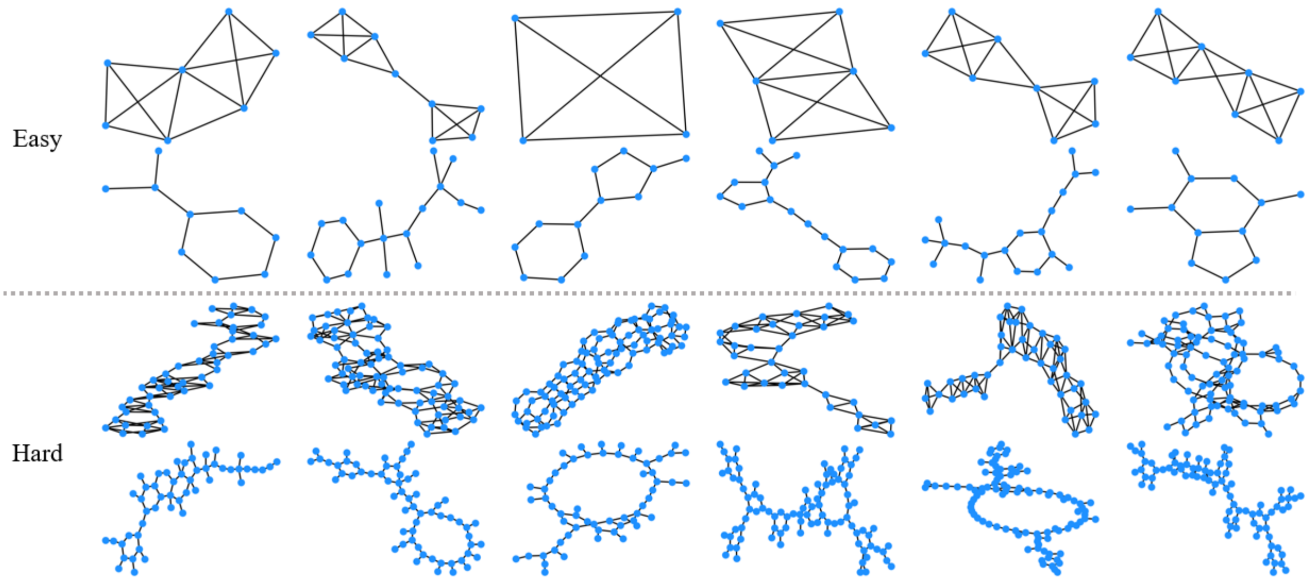


Figure 2: *Example Graphs from* PROTEINS *[8] (first and third row) and* NCI1 *[56] (second and fourth row) datasets.* Difficulty is determined by our CurGraph implemented on GIN [63].

to evaluate the difficulty of G_i:

$$D_R(G_i) = 1 - \frac{\frac{1}{N}\sum_{j=1,\dots,N,y_j=y_i} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)}{\frac{1}{N}\sum_{j=1,\dots,N} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right) + \epsilon_R}. \tag{4}$$

$\sum_{j=1,\dots,N,y_j=y_i} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)$ counts the neighbors of G_i that belong to the same class as G_i. $\sum_{j=1,\dots,N} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)$ counts all the neighbors of G_i. If $\epsilon_R = 0$ holds, (4) is negatively related to the ratio of G_i's neighbors belonging to class y_i. However, for different graphs G_i, the number of neighbors can vary by orders of magnitudes, which this simple ratio does not capture appropriately. For example, if G_i has massive neighbors, then G_i is a normal graph that has many peers similar to itself. If these neighbors belong to the same class as G_i, then it is easy for f to classify, because G_i holds the popular and discriminative feature for its ground-truth class y_i. In this case, $D_R(G_i)$ returns a high value with $\epsilon_R = 0$. On the other hand, if G_i has few neighbors, G_i is an outlier in the dataset. Then G_i is difficult to classify no matter whether its neighbors belong to class y_i or not. Based on the analysis above, we use the hyper-parameter ϵ_R to adjust difficulty values for different graphs. When $\frac{1}{N}\sum_{j=1,\dots,N} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right) \gg \epsilon_R$, we have

$$D_R(G_i) \approx 1 - \frac{\sum_{j=1,\dots,N,y_j=y_i} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)}{\sum_{j=1,\dots,N} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)}. \tag{5}$$

Otherwise, ϵ_R penalizes the difficulty of G_i, i.e., $D_R(G_i)$ returns a higher value. The strength of the penalty grows as the number of neighbors decreases (becoming an outlier). Therefore, we can view ϵ_R as a penalty element for outliers. For convenience, we set ϵ_R as:

$$\begin{aligned}\epsilon_R = &\left(\frac{1}{N}\max_i \sum_{j=1,\dots,N} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)\right)^{(1-\gamma)} \times \\ &\left(\frac{1}{N}\min_i \sum_{j=1,\dots,N} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)\right)^{\gamma},\end{aligned} \tag{6}$$

where γ is an hyper-parameter between 0 and 1 to replace ϵ_R. We set ϵ_R as the weighted geometric mean of the maximum and minimum values of $\sum_{j=1,\dots,N} I\left(\mathbf{h}_j \in B(\mathbf{h}_i, R)\right)$ weighted by γ. Given the possibly large gap on the order of magnitudes between the maximum and minimum, this setting makes ϵ_R sensitive to both the minimum and maximum, instead of only the maximum.

To put Eq. (4) into practice, setting the value of R is an issue. Next, we show that R does not necessarily influence the difficulty evaluation. Eq. (3) is a consistent estimator of the quantity [44]:

$$P\left(\mathbf{h} \in B(\mathbf{h}_i, R), y = c\right) = \int_{B(\mathbf{h}_i,R)} p(\mathbf{h}, y = c)d\mathbf{h}, \tag{7}$$

where $p(\mathbf{h}, y = c)$ is the probability density function that a graph G belongs to class c while its embedding is located at $\mathbf{h}$. As explained above, R has a small value. Thus, the density $p(\mathbf{h}, y = c)$ within the region $B(\mathbf{h}_i, R)$ does not change too much. Namely, $p(\mathbf{h}) \approx p(\mathbf{h}_i)$ for every $\mathbf{h} \in B(\mathbf{h}_i, R)$. Hence, we have:

$$\begin{aligned}\int_{B(\mathbf{h}_i,R)} p(\mathbf{h}, y = c)d\mathbf{h} &\approx p(\mathbf{h}_i, y = c)\int_{B(\mathbf{h}_i,R)} d\mathbf{h} \\ &= p(\mathbf{h}_i, y = c) \times V_d \times R^d,\end{aligned} \tag{8}$$

where $V_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$ is the volume a unit d-dimensional ball and $\Gamma(\cdot)$ is the Gamma function [1]. Bringing Eq. (8) and (6) into Eq. (4)

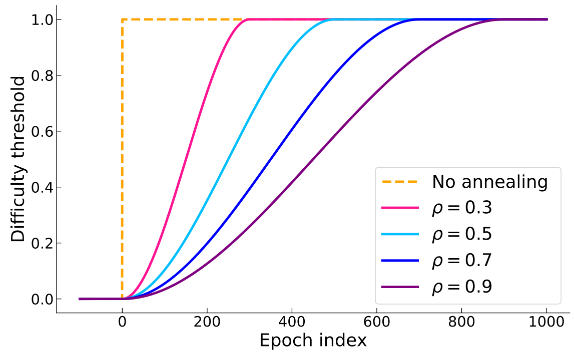


Figure 3: *We design smooth-step threshold function for curriculum learning.* We plot our smooth-step function $D_s(t)$ in Eq. (14) with $T = 1000$, $D_{s-1} = 0$, and $D_s = 1$. The hyper-parameter ρ changes the speed at which $D_s(t)$ increases.

leads to our final difficulty function:

$$D(G_i) = 1 - \frac{p(\mathbf{h}_i, y = y_i)}{\sum_c p(\mathbf{h}_i, y = c) + \epsilon}, \tag{9}$$

where

$$\epsilon = \left(\max_i \sum_c p(\mathbf{h}_i, y = c)\right)^{1-\gamma} \left(\min_i \sum_c p(\mathbf{h}_i, y = c)\right)^{\gamma}. \tag{10}$$

Another interpretation of Eq. (9) can be obtained with the posterior distribution $p\left(y = y_i | \mathbf{h}_i\right)$:

$$\begin{aligned} D(G_i) =& 1 - \frac{p(\mathbf{h}_i, y = y_i)}{\sum_c p(\mathbf{h}_i, y = c) + \epsilon} \\ =& 1 - \frac{p(\mathbf{h}_i, y = y_i)}{p(\mathbf{h}_i)} \frac{p(\mathbf{h}_i)}{p(\mathbf{h}_i) + \epsilon} \\ =& 1 - p\left(y = y_i | \mathbf{h}_i\right) \frac{p(\mathbf{h}_i)}{p(\mathbf{h}_i) + \epsilon}. \end{aligned} \tag{11}$$

Given $\mathbf{h}_i$, a higher $p\left(y = y_i | \mathbf{h}_i\right)$ indicates that the graph is more likely to be classified with its ground-truth class. In other words, G_i is similar to other graphs belonging to the class y_i and dissimilar to the graphs in other classes. The element $\frac{p(\mathbf{h}_i)}{p(\mathbf{h}_i)+\epsilon}$ is used as a penalty term for outlier graphs, as explained above.

Given Eq. (9), the remaining task is to estimate the probability densities $p(\mathbf{h}_i, y = y_i)$ given the embedding set: $\{\mathbf{h}_i, G_i \in \mathcal{G}\}$. This is a classical machine learning task known as Density Estimation, in which the state-of-the-art approach is neural density estimation. In our work, we use Block Neural Autoregressive Flow (BNAF) [12], a popular recent neural density estimator due to its flexibility and efficiency.

We visualize the effects of γ in Fig. 1. When $\gamma = 1.0$, i.e., ϵ reaches the minimum value, the outlier graphs are assigned to low difficulty values, which is not what we expect. But when $\gamma = 0.4$, the outlier graphs have high difficulty values. This demonstrates that introducing γ is effective in giving appropriate difficulty values for different graphs. Overall, $D(G_i)$ returns a difficulty value between 0 and 1. And a larger value of $D(G_i)$ infers a higher difficulty of G_i.

We show the example graphs with the evaluated difficulty from the `PROTEINS` [8] and `NCI1` [56] datasets in Fig. 2. The easy graphs hold simple topology and share popular and significant substructures, such as the four-node complete subgraphs in the first row, and the six-node ring subgraphs in the second row. In contrast, harder graphs generally hold more complex structures and their dominant pattern is not obvious.

3.2 Smooth-Step Curriculum Learning

In this section, we describe our method to train GNNs based on our difficulty scores. CurGraph divides the training of GNNs into S stages. Accordingly, we sort the graphs by their difficulty scores in the ascending order, and split them into S buckets, so the graphs are allocated into S levels of difficulty. More precisely, we set $S-1$ threshold values $\{D_s \mid s = 1, \dots, S-1\}$, and put the graphs:

$$\mathcal{G}_s = \{G_i, D_{s-1} < D(G_i) \le D_s\} \tag{12}$$

into the sth bucket, where $D_0 = 0$ and $D_S = 1$ holds for consistency.

At the sth training stage, we add the graph set $\mathcal{G}_s$ to the existing graph subset $\mathcal{G}_{cur}$:

$$\mathcal{G}_{cur} \leftarrow \mathcal{G}_{cur} \cup \mathcal{G}_s, \tag{13}$$

where $\mathcal{G}_{cur}$ is initialized as an empty set before the first training stage. Then, the GNN model is trained to converge on G_{cur} before the next training stage starts.

Existing work on curriculum learning generally adds the whole $\mathcal{G}_s$ to $\mathcal{G}_{cur}$ in one shot. Denote an auxiliary time-variant threshold on the difficulty values as $D_s(t)$, where t is the epoch index reset to 0 at the beginning of each training stage. The graphs of difficulty values lower than $D_s(t)$ are used for training GNNs at epoch t. Then, we can see that, in the existing curriculum learning methods, $D_s(t)$ is a step function jumping from D_{s-1} to D_s when $t = 0$. This is a hard transition because massive graph samples of diverse difficulty values (from D_{s-1} to D_s) are added at the same time.

We propose a novel *smooth-step threshold function*, which gradually increases from D_{s-1} to D_s in a smooth style. Our smooth-step function is S-shaped and continuously differentiable, similar to the logistic function [6]. Let ρ be a hyper-parameter between 0 and 1. The smooth-step function is a cubic polynomial in the interval $[0, \rho T]$, D_{s-1} to the left of the interval, and D_s to the right, where T is the maximum epoch number. More formally, we assume that the function takes the parametric from $D_s(t) = at^3 + bt^2 + ct + d$ for $t \in [0, \rho T]$, where a, b, c, d are scalar parameters depending on ρ. We then solve for the parameters under the following continuity and differentiability constraints: (i) $D_s(0) = D_{s-1}$, (ii) $D(\rho T) = D_s$, (iii) $\frac{\partial D_s(t)}{\partial t}|_{t=0} = \frac{\partial D_s(t)}{\partial t}|_{t=\rho T} = 0$. This leads to:

$$D_s(t) = \begin{cases} D_{s-1} & \text{if } t \le 0 \\ \frac{2(D_{s-1}-D_s)}{\rho^3 T^3} t^3 + \frac{3(D_s - D_{s-1})}{\rho^2 T^2} t^2 + D_{s-1} & \text{if } 0 < t \le \rho T \\ D_s & \text{if } t \ge \rho T \end{cases} \tag{14}$$

We visualize $D_s(t)$ in Fig. 3. We observe that our smooth-step function $D_s(t)$ is both continuous and continuously differentiable for any $t \in R$ (including $t = 0$ and $t = \rho T$) by our careful construction.

For $t \ge \rho T$, $D_s(t)$ keeps at D_s, i.e., all the graphs in the bucket $\mathcal{G}_s$ are included for training. The choice of ρ controls the speed at which the difficult graphs are added to $\mathcal{G}_{cur}$. A very small ρ can put most graphs in the bucket $\mathcal{G}_s$ used for training at the beginning, i.e., degrading our method to the original hard transition.

Algorithm 1 Smooth-Step Curriculum Learning.

Input: A graph set $\mathcal{G} = \{G_1, G_2, \ldots, G_N\}$. the difficulty scores $\mathcal{G}$: $\{D(G_i) \mid G_i \in \mathcal{G}\}$, a GNN model: $f : G_i \to \hat{y}_i$, the ground-truth labels $\{y_i \mid i = 1, \ldots, N\}$, the number of training stages S, the difficulty thresholds: $\{D_s \mid s = 1, \ldots, S-1\}$, the maximum epoch number per stage T, the hyper-parameter ρ in Eq. (14).
Output: The predicted classes $\{\hat{y}_i \mid i = 1, \ldots, N\}$, the trained parameters of the GNN model f.

Intilize all parameters of f.
Intilize the temporary graph set for training $\mathcal{G}_{cur} \leftarrow \emptyset$.
for $s \leftarrow 1$ to S **do**
  $D_s(0) \leftarrow 0$
  for t $\leftarrow 1$ to T **do**
    if $t \leq \rho T$ **then**
      $D_s(t) \leftarrow \frac{2(D_{s-1}-D_s)}{\rho^3 T^3} t^3 + \frac{3(D_s - D_{s-1})}{\rho^2 T^2} t^2 + D_{s-1}$
    else
      $D_s(t) \leftarrow D_s$
    end if
    $\mathcal{G}_{cur} \leftarrow \mathcal{G}_{cur} \cup \{G_i \mid D_s(t-1) < D(G_i) \leq D_s(t)\}$
    Predict the classes $\{\hat{y}_i \mid G_i \in \mathcal{G}_{cur}\}$ by f.
    Calculate cross-entropy loss $\mathcal{L}$ on $\{\hat{y}_i, y_i, \mid G_i \in \mathcal{G}_{cur}\}$.
    if $\mathcal{L}$ converges **then**
      break
    else
      Back-propagation on f for minimizing $\mathcal{L}$.
    end if
  end for
end for

We note that variants of the smooth-step functions are popular in computer graphics [16], [40]. However, to the best of our knowledge, the smooth-step function has not been used in GNNs or curriculum learning. It is also worth noting that the cubic polynomial used for interpolation in Eq. (14) can be substituted with high-order polynomials (e.g., polynomial of degree 5, where the first and second derivatives vanish at $t = 0$ and $t = \rho T$). Our smooth-step curriculum learning approach directly applies to the case of higher-order polynomials. We show the pseudo-code of our smooth-step curriculum learning in Alg. 1.

Easy graphs can be seen as clean samples, which have fewer redundant graph structures and less noisy labels, while the difficult graphs are noisy. At early stages, CurGraph feeds the GNN model only clean samples, helping GNN to learn fundamental features while protecting them from being perturbed by noisy samples. After that, CurGraph feeds noisy samples to GNN gradually, allowing it to learn more meaningful and discriminative features. The data added later improves the generalization capability of the model and allows the model to avoid over-fitting over the easy graphs, by providing a manner of regularization. Within the sth training stage, the difficulty gap $D_s - D_{s-1}$ exists between newly added graphs. Our smooth-step method makes the samples be added smoothly following the order of their difficulty, where the hyper-parameter ρ controls the speed of adding difficult graphs. As a result, at each training step, CurGraph feeds a GNN 'interesting' examples, which would be standing near the border of the GNN's capability, neither too easy nor too hard, so that the GNN can expand the border gradually. Overall, CurGraph guides the optimization of GNNs, which is non-convex, towards better local minima.

Table 1: *Statistics of the utilized datasets.* #Nodes denotes the average number of nodes per graph, while #Edges denotes the average number of edges per graph.

Dataset	#Graphs	#Nodes	#Edges	#Classes
D&D	1,178	284.32	715.66	2
ENZYMES	600	32.63	62.14	6
NCI1	4,110	29.87	32.30	2
NCI109	4,127	29.68	32.13	2
PROTEINS	1,113	39.06	72.82	2
Mutagenicity	4,337	30.32	30.77	2
COLLAB	5,000	74.49	2457.78	3
IMDB-B	1,000	19.77	96.53	2
IMDB-M	1,500	13.00	65.94	3
REDDIT-B	2,000	429.63	497.75	2
REDDIT-5K	4,999	508.52	594.87	5

4 EXPERIMENTS

In this section, we present the graph classification performance of GCN models trained by CurGraph. We compare our method with baselines without curriculum learning, the strong curriculum learning methods from other fields, as well as heuristic difficulty measures based on the graph structure. Besides, we conduct ablation studies to show the influence of different components of CurGraph, as well as the sensitivity of the performance with respect to the hyper-parameters of CurGraph.

We use the standard benchmark datasets: D&D [15], ENZYMES [45], NCI1, NCI109 [56], PROTEINS [8], Mutagenicity [25], COLLAB, IMDB-B, IMDB-M, REDDIT-B, and REDDIT-5K [64] for evaluation. The former six are chemical datasets, where the nodes have categorical input features. The latter five are social datasets that do not have node features. We follow [63], [68] to use node degrees as features. The statistics of these datasets are summarized in Table 1.

We use popular graph classification models as the baselines: GRAPHLET [47] and Weisfeiler-Lehman Kernel (WL) are classical graph kernel methods, while DGCNN [68], DiffPool [65], EigenPool [33], and GIN [63] are the GNNs designed for graph classification, which hold the state-of-the-art performance. In addition, we take the recently proposed curriculum learning frameworks designed for convolution neural networks on image classification, CurriculumNet [21] and DCL [42], for comparison.

For the hyper-parameters of InfoGraph, BNAF, and baselines, e.g., the number of layers, the optimizer, the learning rate, we set them as suggested by their authors. When implementing our CurGraph with the GNN models, we use the GNN model under consideration as the backbone of InfoGraph for consistency. For the hyper-parameters of our CurGraph, we set the number of training stages $S = 4$, the difficulty thresholds $\{D_s = s/S \mid s = 1, \ldots, S-1\}$, $\rho = 0.4$ for the smooth-step, $\gamma = 0.4$, and $T = T_{max}/S$ by default,

Table 2: *Test Accuracy (%) of graph classification on chemical datasets.* We perform 10-fold cross-validation to evaluate model performance, and report the mean and standard derivations over 10 folds. We highlight best performances in bold.

Method	D&D	ENZYMES	NCI1	NCI109	PROTEINS	Mutagencity
GRAPHLET [47]	72.1 ± 3.7	41.4 ± 5.2	64.3 ± 2.2	62.5 ± 2.8	70.1 ± 4.1	62.3 ± 1.9
WL [46]	73.2 ± 1.8	53.7 ± 6.0	76.3 ± 1.9	75.8 ± 2.3	72.3 ± 3.4	79.5 ± 1.8
DGCNN [68]	76.7 ± 4.1	39.3 ± 5.9	76.5 ± 1.9	75.9 ± 1.7	72.9 ± 3.5	79.5 ± 1.7
DiffPool [65]	75.2 ± 3.8	59.7 ± 5.3	76.8 ± 2.0	75.5 ± 1.9	73.6 ± 3.6	79.8 ± 1.8
EigenPool [33]	75.9 ± 3.9	62.4 ± 3.8	78.7 ± 1.9	77.4 ± 2.5	74.1 ± 3.1	80.2 ± 1.7
GIN [63]	75.4 ± 2.6	60.3 ± 4.2	79.7 ± 1.8	78.2 ± 2.1	73.5 ± 3.8	79.9 ± 1.4
CurriculumNet [21] + GIN	75.7 ± 2.8	60.7 ± 4.4	80.2 ± 1.8	78.5 ± 2.0	73.7 ± 3.7	80.2 ± 1.6
DCL [42] + GIN	76.0 ± 3.2	61.1 ± 4.9	79.8 ± 2.1	78.9 ± 2.2	73.8 ± 3.9	80.5 ± 1.9
CurGraph + EigenPool	**78.6 ± 3.0**	**64.8 ± 3.3**	80.6 ± 1.9	79.2 ± 2.2	**75.4 ± 3.1**	**81.7 ± 1.7**
CurGraph + GIN	77.1 ± 2.4	62.5 ± 3.9	**81.3 ± 1.7**	**80.1 ± 2.0**	74.7 ± 3.7	81.6 ± 1.4

Table 3: *Test Accuracy (%) of graph classification on social datasets.* We perform 10-fold cross-validation to evaluate model performance, and report the mean and standard derivations over 10 folds. We highlight best performances in bold.

Method	COLLAB	IMDB-B	IMDB-M	REDDIT-B	REDDIT-5K
GRAPHLET [47]	61.7 ± 2.2	54.8 ± 4.1	42.6 ± 2.7	62.1 ± 1.6	36.2 ± 1.8
WL [46]	70.4 ± 1.8	69.1 ± 3.5	45.4 ± 2.9	81.7 ± 1.7	49.4 ± 2.1
DGCNN [68]	71.1 ± 1.7	69.2 ± 2.8	45.6 ± 3.4	87.6 ± 2.1	49.8 ± 1.9
DiffPool [65]	68.9 ± 2.2	68.6 ± 3.1	45.7 ± 3.4	89.2 ± 1.8	53.6 ± 1.4
EigenPool [33]	70.8 ± 1.9	70.4 ± 3.3	47.2 ± 3.0	89.9 ± 1.9	54.5 ± 1.7
GIN [63]	75.5 ± 2.3	71.2 ± 3.9	48.5 ± 3.3	89.8 ± 1.9	56.1 ± 1.6
CurriculumNet [21] + GIN	75.8 ± 2.2	71.8 ± 3.7	49.4 ± 3.0	90.0 ± 2.0	56.6 ± 1.6
DCL [42] + GIN	76.2 ± 2.4	72.1 ± 4.0	49.8 ± 3.3	90.1 ± 2.1	57.2 ± 1.9
CurGraph + EigenPool	73.2 ± 1.8	72.4 ± 3.0	49.9 ± 3.1	**91.4 ± 1.9**	56.2 ± 1.7
CurGraph + GIN	**77.8 ± 1.9**	**73.4 ± 3.2**	**51.8 ± 2.7**	91.2 ± 1.9	**59.7 ± 1.8**

Table 4: *Test Accuracy (%) of graph classification with heuristic curriculum designs.* We perform 10-fold cross-validation to evaluate model performance, and report the mean and standard derivations over 10 folds.

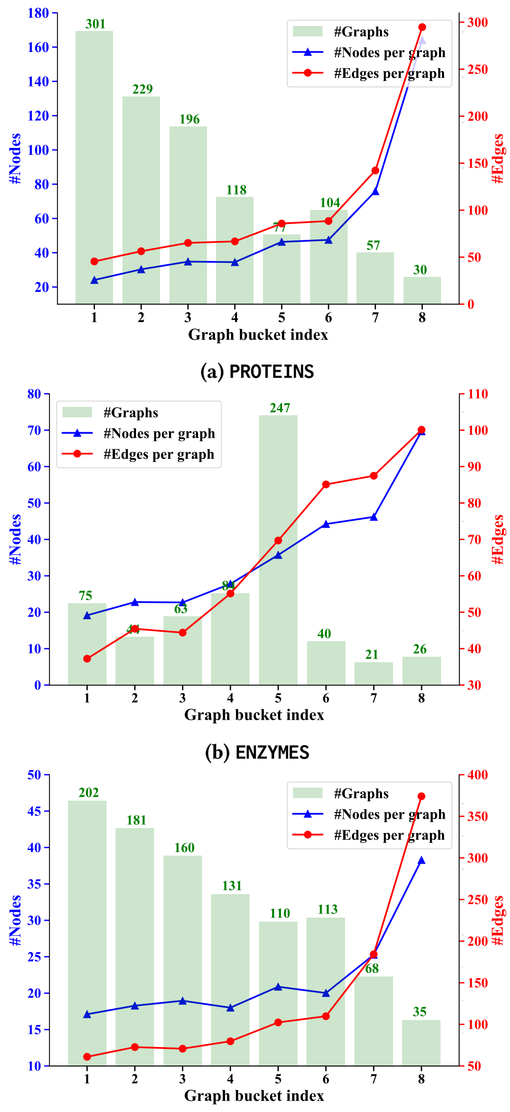
Method	D&D	NCI1	NCI109	Mutagencity	COLLAB	IMDB-M	REDDIT-5K
GIN [63]	75.4 ± 2.6	79.7 ± 1.8	78.2 ± 2.1	79.9 ± 1.4	75.5 ± 2.3	48.5 ± 3.3	56.1 ± 1.6
CurGraph (w. #Nodes) + GIN	75.6 ± 2.6	79.7 ± 1.8	78.4 ± 2.0	80.0 ± 1.5	75.5 ± 2.3	48.6 ± 3.1	56.2 ± 1.7
CurGraph (w. #Edges) + GIN	75.5 ± 2.5	79.9 ± 1.9	78.3 ± 2.2	80.0 ± 1.4	75.6 ± 2.1	48.5 ± 3.2	56.3 ± 1.8
CurGraph (Ours) + GIN	**77.1 ± 2.4**	**81.3 ± 1.7**	**80.1 ± 2.0**	**81.6 ± 1.4**	**77.8 ± 1.9**	**51.8 ± 2.7**	**59.7 ± 1.8**

where T_{max} is the maximum epoch number specified by the GNN model in use.

4.1 Graph Classification

We follow [63], [18] to use the 10-fold cross-validation scheme to calculate the classification performance for a fair comparison. For each training fold, as suggested by [18], we conduct an inner holdout technique with a 90%/10% training/validation split. In detail, we train fifty times on a training fold holding out a random fraction (10%) of the data to perform early stopping. These fifty separate trials are needed to smooth the effect of unfavorable random weight initialization on test performances. The final test fold score is obtained as the mean of these fifty runs.

We report the average and standard deviation of test accuracy across the 10 folds within the cross-validation on the chemical and social datasets in Table 2 and 3 respectively. On the chemical datasets, we observe that CurGraph improves the test accuracy of EigenPool by 3.6% on D&D, 3.8% on ENZYMES, 2.4% on NCI1, 2.3% on NCI109, 1.8% on PROTEINS, and 1.9% on Mutagencity respectively. In addition, CurGraph improves GIN by 2.3% on D&D, 3.6% on ENZYMES, 2.0% on NCI1, 2.6% on NCI109, 1.6% on PROTEINS, and 2.1% on Mutagencity. On the social datasets, CurGraph improves EigenPool by more than 3% on COLLAB, IMDB-M, REDDIT-5K, and more



(a) PROTEINS

(b) ENZYMES

(c) IMDB-B

Figure 4: *The heuristic difficulty metrics #Nodes and #Edges averaged for the graphs in each bucket.* The graph buckets are given by CurGraph with $S = 8$. The number of graphs in each bucket are shown by the bars' lengths and the numbers over the bars.

than 2% on IMDB-B and REDDIT-B. Besides, CurGraph improves GIN by more than 3% on COLLAB, IMDB-B, IMDB-M, REDDIT-5K, and 1.6% on REDDIT-B. As a result, CurGraph enhances EigenPool and GIN to outperform all the benchmark methods.

Taking a closer look, we observe that the graph kernel methods, GRAPHLET and WL, generally present worse performance than the GNN methods. This demonstrates the stronger fitting capacity of the advanced neural network models. Both CurriculumNet and DCL achieve improvements over GIN, which validates the idea of curriculum learning on the task of graph classification. Our CurGraph gains higher improvements on the test accuracy than CurriculumNet and DCL. The reasons are as follows. CurriculumNet evaluates the difficulty scores by the distribution of the examples in each category separately, but does not consider the inter-class distributions as we do. However, similar graphs belonging to different classes can mislead the GNN model to make wrong predictions and thus indicate higher difficulty. Besides, CurriculumNet makes hard transitions between different training stages without our smooth-step learning method. Last but not least, we utilize the state-of-the-art infomax method to extract graph embeddings and advanced neural density estimator to model embedding distributions instead of the simple clustering in CurriculumNet. DCL introduces the extra class-level and instance-level parameters to enable the neural networks to learn the difficulty values automatically during training. These redundant parameters increase the risks of over-fitting, which is validated by their higher std. values.

Table 5: Test Accuracy (%) of graph classification with and without our smooth-step learning method.

Method	NCI109	COLLAB
EigenPool [33]	77.4	70.8
+ CurGraph (w.o. smooth-step)	78.5 (+1.1)	72.4 (+1.6)
+ smooth-step	**79.2 (+1.8)**	**73.2 (+2.4)**
GIN [63]	78.2	75.5
+ CurGraph (w.o. smooth-step)	79.4 (+1.2)	76.9 (+1.4)
+ smooth-step	**80.1 (+1.9)**	**77.8 (+2.3)**

4.2 Comparison with Heuristic Curriculum Design

On the difficulty evaluation of different graphs, we compare our CurGraph with two heuristic curriculum designs. In principle, we use #Nodes and #Edges as the heuristic difficulty metrics. We define difficult graphs as those of greater numbers of nodes or edges, since more nodes and edges generally imply more complex graph structures, indicating higher recognition difficulty. We show #Nodes and #Edges averaged in graph buckets split by CurGraph in Fig. 4 We observe the difficult graphs provided by our CurGraph tend to hold more nodes and edges.

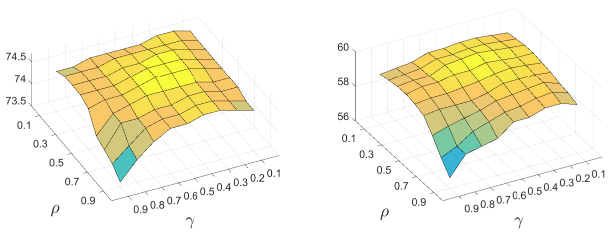
We implement CurGraph with #Nodes and #Edges as the heuristic difficulty metrics, of which the graph classification accuracy is presented in Table 4. The results show that even with the simple metrics #Nodes and #Edges, our CurGraph achieves improvements in effectiveness over GIN. Our infomax curriculum design makes CurGraph achieve much higher advancements than the heuristic metrics, since CurGraph evaluates the difficulty scores of miscellaneous graphs in the high-level semantic space and utilizes our principled statistical difficulty measurements. Compared with the heuristics, our CurGraph correlates with the difficulty 'preference' of GNNs and generalizes to GNNs better.

4.3 Ablation Study

We conduct a number of ablations to analyze CurGraph. First, we investigate the effects of the smooth-step curriculum learning. We

Table 6: Test Accuracy (%) of graph classification of different number of training stages S.

Method	S	D&D	NCI1	IMDB-M
CurGraph + GIN	2	76.9 ± 2.5	81.1 ± 1.7	51.5 ± 2.7
	4	**77.1 ± 2.4**	81.3 ± 1.7	**51.8 ± 2.7**
	6	77.0 ± 2.3	**81.4 ± 1.6**	51.7 ± 2.5
	8	76.9 ± 2.3	81.3 ± 1.6	51.6 ± 2.4



(a) PROTEINS **(b) REDDIT-5K**

Figure 5: The test accuracy (z-axis) of GIN with CurGraph under different values of the hyper-parameters ρ and γ.

compare the test accuracy of CurGraph implemented with and without our smooth-step method on the NCI109 and COLLAB datasets in Table 5. Without our smooth-step method, the classical curriculum learning strategy with our difficulty evaluation methods achieve significant improvements on EigenPool and GIN. This demonstrates the effectiveness of our infomax curriculum design module. With smooth-step, GNNs are enhanced further. This shows the soft transitions provided by our smooth-step method is advantageous and helps the GNNs to be exposed to the samples of appropriate difficulty levels.

In Table 6, we evaluate the sensitivity of CurGraph to the number of training stages S. As we can see, the performance of CurGraph implemented with GIN is generally smooth for S between 2 and 8. On all the three datasets, our default value of $S = 4$ achieves satisfactory performance, which implies that our setting on S is practical for graph classification.

Last but not least, we evaluate how sensitive our CurGraph is to the selection of hyper-parameter values: ρ to control the transition speed of our smooth-step learning and γ to adjust the penalty weight for outliers in difficulty evaluation. As we can see, the performance of GIN with CurGraph is relatively smooth when parameters are within certain ranges. However, extremely large values of ρ and γ result in low performances in all cases, which should be avoided in practice. Moreover, increasing ρ from 0.1 to 0.4 improves the test accuracy of GIN with CurGraph, demonstrating that the soft transitions provided by our smooth-step learning method play an important role in improving the performance of GNNs.

5 CONCLUSION

In this paper, we study the problem of exploring Curriculum Learning to strengthen the GNN models on graph classification. We propose a novel GNN framework that trains the GNNs in an easy-to-difficult curriculum to improve their inference performance without human heuristics or extra inference cost. CurGraph uses an infomax method to obtain graph embeddings with GNNs and estimate the embedding densities using a neural density estimator. CurGraph evaluates the graph difficulty from the intra-class and inter-class embedding distributions. To provide the soft transitions across different training stages corresponding to difficulty levels, we design the smooth-step curriculum learning method. With the smooth-step method, CurGraph enables a GNN to learn from the graphs, which stand near the border of its capability, neither too hard nor too easy, to gradually expand its border at each training step. Our experimental results show that CurGraph yields significant gains for graph classification evaluated on multiple benchmark datasets. Future work can explore applying Curriculum Learning to more general graph-related tasks beyond graph classification, to further increase the effectiveness and practical utility of graph neural networks.

ACKNOWLEDGMENTS

This paper is supported by NUS ODPRT Grant R252-000-A81-133 and Singapore Ministry of Education Academic Research Fund Tier 3 under MOEs official grant number MOE2017-T3-1-007.

REFERENCES

[1] Emil Artin. 2015. *The gamma function.* Courier Dover Publications.
[2] Sambaran Bandyopadhyay, Manasvi Aggarwal, and M Narasimha Murty. 2020. Robust Hierarchical Graph Classification with Subgraph Attention. *arXiv preprint arXiv:2007.10908* (2020).
[3] Anthony J Bell and Terrence J Sejnowski. 1995. An information-maximization approach to blind separation and blind deconvolution. *Neural computation* 7, 6 (1995), 1129–1159.
[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.
[5] Siddharth Bhatia, Bryan Hooi, Minji Yoon, Kijung Shin, and Christos Faloutsos. 2020. Midas: microcluster-based detector of anomalies in edge streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3242–3249.
[6] Christopher M Bishop. 2006. *Pattern recognition and machine learning.* springer.
[7] Karsten M Borgwardt and Hans-Peter Kriegel. 2005. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*. IEEE, 8–pp.
[8] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. 2005. Protein function prediction via graph kernels. *Bioinformatics* 21, suppl_1 (2005), i47–i56.
[9] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).
[10] Yujun Cai, Liuhao Ge, Jun Liu, Jianfei Cai, Tat-Jen Cham, Junsong Yuan, and Nadia Magnenat Thalmann. 2019. Exploiting spatial-temporal relationships for 3d pose estimation via graph convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 2272–2281.
[11] Xinlei Chen and Abhinav Gupta. 2015. Webly supervised learning of convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*. 1431–1439.
[12] Nicola De Cao, Ivan Titov, and Wilker Aziz. 2019. Block neural autoregressive flow. *arXiv preprint arXiv:1904.04676* (2019).
[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.
[14] Ailin Deng and Bryan Hooi. 2021. Graph Neural Network-Based Anomaly Detection in Multivariate Time Series. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[15] Paul D Dobson and Andrew J Doig. 2003. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology* 330, 4 (2003), 771–783.
[16] David S Ebert, F Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. 2003. *Texturing & modeling: a procedural approach.* Morgan Kaufmann.

[17] Jeffrey L Elman. 1993. Learning and development in neural networks: The importance of starting small. *Cognition* 48, 1 (1993), 71–99.

[18] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. 2019. A fair comparison of graph neural networks for graph classification. *arXiv preprint arXiv:1912.09893* (2019).

[19] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1416–1424.

[20] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212* (2017).

[21] Sheng Guo, Weilin Huang, Haozhi Zhang, Chenfan Zhuang, Dengke Dong, Matthew R Scott, and Dinglong Huang. 2018. Curriculumnet: Weakly supervised learning from large-scale web images. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 135–150.

[22] Lu Haonan, Seth H Huang, Tian Ye, and Guo Xiuyan. 2019. Graph Star Net for Generalized Multi-Task Learning. *arXiv preprint arXiv:1906.12330* (2019).

[23] David Haussler. 1999. *Convolution kernels on discrete structures*. Technical Report. Technical report, Department of Computer Science, University of California ....

[24] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2018. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*. 2304–2313.

[25] Jeroen Kazius, Ross McGuire, and Roberta Bursi. 2005. Derivation and validation of toxicophores for mutagenicity prediction. *Journal of medicinal chemistry* 48, 1 (2005), 312–320.

[26] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[27] Nils Kriege and Petra Mutzel. 2012. Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483* (2012).

[28] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. 2020. A survey on graph kernels. *Applied Network Science* 5, 1 (2020), 1–42.

[29] Kai A Krueger and Peter Dayan. 2009. Flexible shaping: How learning in small steps helps. *Cognition* 110, 3 (2009), 380–394.

[30] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. 2019. Self-attention graph pooling. In *36th International Conference on Machine Learning, ICML 2019*. International Machine Learning Society (IMLS), 6661–6670.

[31] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493* (2015).

[32] Ralph Linsker. 1988. Self-organization in a perceptual network. *Computer* 21, 3 (1988), 105–117.

[33] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 723–731.

[34] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5115–5124.

[35] Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning* 102, 2 (2016), 209–245.

[36] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *International conference on machine learning*. 2014–2023.

[37] Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. 2019. Graph kernels: A survey. *arXiv preprint arXiv:1904.12218* (2019).

[38] Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M Mitchell. 2019. Competence-based curriculum learning for neural machine translation. *arXiv preprint arXiv:1903.09848* (2019).

[39] Douglas LT Rohde and David C Plaut. 1999. Language acquisition in the absence of explicit negative evidence: How important is starting small? *Cognition* 72, 1 (1999), 67–109.

[40] Randi J Rost, Bill Licea-Kane, Dan Ginsburg, John Kessenich, Barthold Lichtenbelt, Hugh Malan, and Mike Weiblen. 2009. *OpenGL shading language*. Pearson Education.

[41] Mrinmaya Sachan and Eric Xing. 2016. Easy questions first? a case study on curriculum learning for question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 453–463.

[42] Shreyas Saxena, Oncel Tuzel, and Dennis DeCoste. 2019. Data parameters: A new family of parameters for learning a differentiable curriculum. In *Advances in Neural Information Processing Systems*. 11095–11105.

[43] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20, 1 (2008), 61–80.

[44] Fritz Scholz. 2013. Stat 425 Introduction to Nonparametric Statistics Comparison of More Than Two Treatments. (2013).

[45] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. 2004. BRENDA, the enzyme database: updates and major new developments. *Nucleic acids research* 32, suppl_1 (2004), D431–D433.

[46] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12, 9 (2011).

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

[48] Valentin I Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2009. Baby Steps: How "Less is More" in unsupervised dependency parsing. (2009).

[49] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. 2019. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000* (2019).

[50] James S Supancic and Deva Ramanan. 2013. Self-paced learning for long-term tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2379–2386.

[51] Yi Tay, Shuohang Wang, Luu Anh Tuan, Jie Fu, Minh C Phan, Xingdi Yuan, Jinfeng Rao, Siu Cheung Hui, and Aston Zhang. 2019. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. *arXiv preprint arXiv:1905.10847* (2019).

[52] Zekun Tong, Yuxuan Liang, Changsheng Sun, David S Rosenblum, and Andrew Lim. 2020. Directed graph convolutional network. *arXiv preprint arXiv:2004.13970* (2020).

[53] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. 2020. Graph Clustering with Graph Neural Networks. *arXiv preprint arXiv:2006.16904* (2020).

[54] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph Attention Networks. *arXiv preprint arXiv:1710.10903* (2017).

[55] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2018. Deep graph infomax. *arXiv preprint arXiv:1809.10341* (2018).

[56] Nikil Wale, Ian A Watson, and George Karypis. 2008. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems* 14, 3 (2008), 347–375.

[57] Yiwei Wang, Shenghua Liu, Minji Yoon, Hemank Lamba, Wei Wang, Christos Faloutsos, and Bryan Hooi. 2020. Provably Robust Node Classification via Low-Pass Message Passing. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 621–630.

[58] Yiwei Wang, Wei Wang, Yujun Ca, Bryan Hooi, and Beng Chin Ooi. 2020. Detecting Implementation Bugs in Graph Convolutional Network based Node Classifiers. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 313–324.

[59] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2020. Graphcrop: Subgraph cropping for graph classification. *arXiv preprint arXiv:2009.10564* (2020).

[60] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, Juncheng Liu, and Bryan Hooi. 2020. Nodeaug: Semi-supervised node classification with data augmentation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 207–217.

[61] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. 2019. A Comprehensive Survey on Graph Neural Networks. *arXiv preprint arXiv:1901.00596* (2019).

[62] Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang. 2020. Curriculum Learning for Natural Language Understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 6095–6104.

[63] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[64] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1365–1374.

[65] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in neural information processing systems*. 4800–4810.

[66] Yuxuan Liang Yujun Cai Yiwei Wang, Wei Wang and Bryan Hooi. 2020. Progressive Supervision for Node Classification. In *Proceedings of ECML-PKDD*, Vol. 2020. 2020.

[67] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. 2018. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294* (2018).

[68] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[69] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434* (2018).

A MORE DETAILS ABOUT EXPERIMENTS FOR REPRODUCIBILITY

To support the reproducibility of the results in this study, we introduce the details of our experimental settings.

A.1 Datasets and Software Versions

We download the D&D [15], ENZYMES [45], NCI1, NCI109 [56], PROTEINS [8], COLLAB, IMDB-B, IMDB-M, REDDIT-B, and REDDIT-5K [64] datasets from the website[1]. PROTEINS and D&D are two protein graph datasets, where nodes represent the amino acids and two nodes are connected by an edge if they are less than six Angstroms apart. The label indicates whether or not a protein is a non-enzyme. NCI1 and NCI109 are two biological datasets screened for activity against non-small cell lung cancer and ovarian cancer cell lines, where each graph is a chemical compound with nodes and edges representing atoms and chemical bonds, respectively. ENZYMES is a dataset of protein tertiary structures, and each enzyme belongs to one of the six EC top-level classes.

REDDIT-B is a balanced dataset where each graph corresponds to an online discussion thread where nodes correspond to users, and there is an edge between two nodes if at least one of them responded to another's comment. The collectors crawled top submissions from four popular subreddits, namely, IAmA, AskReddit, TrollXChromosomes, and atheism. IAmA and AskReddit are two question/answer based subreddits, and TrollXChromosomes and atheism are two discussion-based subreddits. The task is then to identify whether a given graph belongs to a question/answer-based community or a discussion-based community. REDDIT-5K is a balanced dataset from five different subreddits, namely, worldnews, videos, AdviceAnimals, aww and mildlyinteresting where the collectors simply label each graph with their correspondent subreddit.

COLLAB is a scientific collaboration dataset, derived from three public collaboration datasets, namely, High Energy Physics, Condensed Matter Physics and Astro Physics. The collectors generated ego-networks of different researchers from each field, and labeled each graph as the field of the researcher. The task is then to determine whether the ego-collaboration graph of a researcher belongs to the High Energy, Condensed Matter, or Astro Physics field.

IMDB-B is a movie collaboration dataset where we collected actor/actress and genre information of different movies on IMDB. For each graph, nodes represent actors/actresses and there is an edge between them if they appear in the same movie. The collectors generated collaboration graphs on Action and Romance genres and derived ego-networks for each actor/actress. Note that a movie can belong to both genres at the same time, therefore the collectors discarded movies from the Romance genre if the movie is already included in the Action genre. Similar to COLLAB dataset, we simply labeled each ego-network with the genre graph it belongs to. The task is then simply to identify which genre an ego-network graph belongs to. IMDB-M is the multi-class version of IMDB-B and contains a balanced set of ego-networks derived from Comedy, Romance, and Sci-Fi genres.

Regarding software versions, we install CUDA 10.0 and cuDNN 7.0. TensorFlow 1.12.0 and PyTorch 1.0.0 with Python 3.6.0 are used. Note that all the experiments are running a Linux Server with the Intel(R) Xeon(R) E5-1650 v4 @ 3.60GHz CPU, and the GeForce GTX 1080 Ti GPU.

[1] https://chrsmrrs.github.io/datasets/

A.2 Settings of the Baseline

When implementing graph neural networks as the benchmark and implementing it with CurGraph, we follow the the suggested settings and utilize the early stopping training strategy: stop optimization if the validation loss is larger than the mean of validation losses of the last 50 epochs. We utilize 5 GNN layers (including the input layer), and MLPs of 2 layers. Batch normalization is applied on every hidden layer. We use the Adam optimizer with an initial learning rate 0.01 and decay the learning rate by 0.5 every 50 epochs. The hyper-parameters we tune for each dataset are: (1) the number of hidden units belonging to 16, 32 for chemical graphs and 64 for social graphs; (2) the batch size belonging to 32, 128; (3) the dropout ratio belonging to 0, 0.5 after the dense layer; (4) the number of epochs, i.e., a single epoch with the best cross-validation accuracy averaged over the 10 folds was selected.

We refer to the following websites when implementing the above mentioned models:

(1) **GRAPHLET**: https://github.com/nkahmed/PGD
(2) **WL**: https://github.com/BorgwardtLab/P-WL
(3) **GCN**: https://github.com/tkipf/gcn
(4) **DGCNN**: https://github.com/muhanzhang/DGCNN
(5) **DiffPool**: https://github.com/RexYing/diffpool
(6) **EigenPool**: https://github.com/alge24/eigenpooling
(7) **GIN**: https://github.com/weihua916/powerful-gnns