GSSNN: Graph Smoothing Splines Neural Networks

Shichao Zhu,[1, 3*] **Lewei Zhou,**[2, 4*] **Shirui Pan,**[5] **Chuan Zhou,**[2, 3] **Guiying Yan,**[2, 4] **Bin Wang**[6]
[1]Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2]Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China
[3]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[4]University of Chinese Academy of Sciences, Beijing, China
[5]Faculty of Information Technology, Monash University, Melbourne, Australia, [6]Xiaomi AI Lab, Beijing, China
zhushichao@iie.ac.cn, {zhoulevi, zhouchuan, yangy}@amss.ac.cn, shirui.pan@monash.edu, wangbin11@xiaomi.com

Abstract

Graph Neural Networks (GNNs) have achieved state-of-the-art performance in many graph data analysis tasks. However, they still suffer from two limitations for graph representation learning. First, they exploit non-smoothing node features which may result in suboptimal embedding and degenerated performance for graph classification. Second, they only exploit neighbor information but ignore global topological knowledge. Aiming to overcome these limitations simultaneously, in this paper, we propose a novel, flexible, and end-to-end framework, Graph Smoothing Splines Neural Networks (GSSNN), for graph classification. By exploiting the smoothing splines, which are widely used to learn smoothing fitting function in regression, we develop an effective feature smoothing and enhancement module Scaled Smoothing Splines (S^3) to learn graph embedding. To integrate global topological information, we design a novel scoring module, which exploits closeness, degree, as well as self-attention values, to select important node features as knots for smoothing splines. These knots can be potentially used for interpreting classification results. In extensive experiments on biological and social datasets, we demonstrate that our model achieves state-of-the-arts and GSSNN is superior in learning more robust graph representations. Furthermore, we show that S^3 module is easily plugged into existing GNNs to improve their performance.

Introduction

Deep learning on graphs is an important and ubiquitous task with a broad set of domains ranging from social networks to biological networks (Zhou et al. 2018; Qiu et al. 2018; Duvenaud et al. 2015; Shi et al. 2019). Graph Neural Networks (GNNs) is a general type of deep-learning architectures that can be directly applied to structured data (Battaglia et al. 2018). The success of GNNs in node representation learning has inspired many deep-learning-based approaches to leverage on node embeddings extracted from GNNs to generate graph embeddings for graph-based applications (Zhang et al. 2018; Xu et al. 2019).

The key to many graph neural networks (Kipf and Welling 2017; Klicpera, Bojchevski, and Günnemann 2019) is to define an `aggregation` operation, which propagates the neighbor information to a target node in an iterative manner. This aggregation operation can be further interpreted as two sequential steps, 1) applying a feature fitting function $g(X) = XW$ on the node feature matrix X with a learnable parameter matrix W, then 2) propagating the new representation on the graph adjacency matrix A by using $A \cdot g(X)$ before fitting it into a nonlinear activation function. While being mathematically simple and easy to implement, these GNNs may result in degenerated node embedding due to the non-smooth feature fitting function $g(X)$ employed in the first step, particularly in graphs with noisy features. What is more, when the resulted node embedding is further used to perform downstream graph classification via summarization or pooling operation, the over-fitting of nodes features and spread of noise features will render the entire graph vulnerable, leading to suboptimal results. This problem, unfortunately, has not been studied in all existing GNNs.

The second limitation of current GNN-based approaches in the graph classification task is that they only exploit local information via convolution or neighbor aggregation, but have largely ignored the global topological information. They either summarize the embedding for all the nodes via a simple readout function, or apply a pooling operation sequentially in the GNNs frameworks. The former approaches such as GIN (Xu et al. 2019) can not distinguish the importance of different nodes, while the latter methods, such as DiffPool (Ying et al. 2018) and SAGPool (Lee, Lee, and Kang 2019), coarsen the graph structure progressively and thus will lose important information for nodes. How to incorporate the global information in GNNs to improve the graph representation learning is less studied in the literature.

Motivated by the above observations, in this paper, we propose a novel framework, Graph Smoothing Splines Neural Networks (GSSNN), to address *both* limitations in an end-to-end framework. To address the first problem, we develop a feature smoothing component which applies smoothing splines to enhance the node features. Specifically, we construct a natural cubic splines function with two continuous derivatives to minimize the penalized residual sum of squares. Fig. 1 illustrates the effectiveness of smoothing splines. This feature smoothing operation will dramatically

*Equal Contributions

reduce the noisy effect that may exist in graph data, enabling a high-quality and more robust graph representation.

To capture the global topological information for the graph classification (problem 2), we develop a node feature scoring function, which selects important node features as *knots*, based on which a set of basis functions are defined to expand each node feature for graph representation learning. Specifically, we combine the global information, closeness centrality, as well as the degree of each node with a self-attention module, to score the importance of each node, so that both local and global information are maximumly exploited in our unified framework.

Our novel design enjoys a number of benefits. First, the node feature scoring function provides interpretability for the classification results. This is because when we utilize *knot* features to enhance each node, the amount of information generated by the enhancement process is personalized by different nodes. In other words, each node has a unique description of the characteristics and sub-structure of important nodes. Second, the feature smoothing module, called Scaled Smoothing Splines (S^3), can be easily inserted into existing graph neural networks like GCN (Kipf and Welling 2017) and GAT (Veličković et al. 2018) to improve their performance. Third, the node feature scoring, node feature smoothing, and graph convolution components can be integrated in an end-to-end framework, enabling effective graph classification.

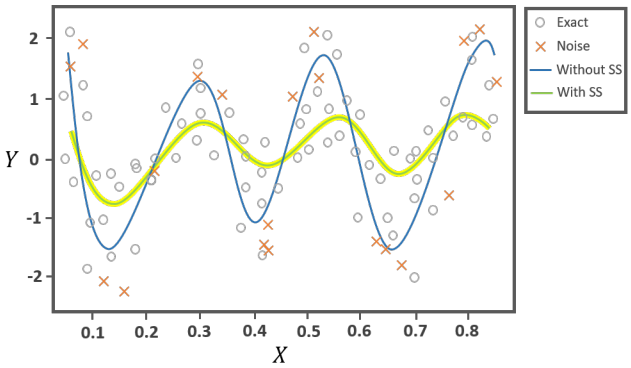


Figure 1: An example of the smoothing splines in two-dimensional space, where the fitted curve without Smoothing Splines (SS) in blue and the fitted curve with SS in green. The blue curve try to fit all the data including exact and noise data, while the green curve with SS is smoother and generalized to fit the trend of exact data.

Our extensive evaluations demonstrate that GSSNN outperforms the SOTA methods on six biological and social datasets on graph classification. The major contributions of this paper are summarized as follows.

- We propose a novel Scaled Smoothing Splines (S^3) module to smooth each propagation layer for graph neural networks, which can be inserted into existing graph neural networks to improve their performance.
- We propose an end-to-end model Graph Smoothing Splines Neural Networks (GSSNN) that flexibly smooth and enhance node features, jointly exploit local and global information, and easily provide interpretability for graph classification.
- Extensive experimental results have verified the effectiveness of our model for graph classification on benchmark datasets.

Related Works

In this section, we briefly summarize two main kinds of methods of graph classification: kernel-based methods and GNN-based methods.

Kernel-based methods

Typically, kernel-based algorithms first decompose graphs into sub-components based on the kernel definition, then build graph embeddings in a feature-based manner. Lastly, some machine learning algorithms (i.e., SVM) are applied to perform graph classification. The representative methods include Weisfeiler-Lehman subtree kernel (WL) (Shervashidze et al. 2011), the graphlet count kernel (GK) (Shervashidze et al. 2009) and the Random Walk (RW) (Vishwanathan et al. 2010).

GNN-based methods

With the powerful capabilities of GNN in non-Euclidean data, the GNN-based approach for graph representation learning has also spawned some representative works, including GCAPS-CNN (Verma and Zhang 2019), CapsGNN (Xinyi and Chen 2019) and GIN (Xu et al. 2019). The intuition of these methods is to collect all nodes features through GNNs to generate graph representation. Graph pooling methods based on GNNs also have achieved good results, including global pooling methods Set2Set (Vinyals, Bengio, and Kudlur 2015), SortPool (Zhang et al. 2018), and hierarchical pooling methods DiffPool (Ying et al. 2018), SAGPool (Lee, Lee, and Kang 2019). The key of graph pooling methods is to reduce the size of nodes through learning topology-based node assignments. For instance, SAGPool method evaluates the importance of each node, then retains the top K important nodes and discards the remaining nodes to generate graph representation. In contrast, we will use all node embedding for graph classification.

Our work is focused on GNN-based method. Different from existing methods, we propose a novel model GSSNN that not only keeps the overall structure but highlights important nodes features through S^3 module.

Notation and Preliminaries

In this section, we introduce the notation and problem definition, as well as backgrounds on smoothing splines.

Notation and Problem Definition

We introduce some basic concepts and formalize the problem of graph embedding. For easy retrieval, we summarize the commonly used notations in Table 1.

Definition 1. Node Importance Estimation. Given a graph $G = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ are the set of nodes, $\mathcal{E}$ are edges

Table 1: Commonly used notations.

Notations	Descriptions
$\mathcal{G} = \{G_i\}_{i=1}^t$	a set of graphs
$G = (\mathcal{V}, \mathcal{E})$	A graph
$\mathcal{V}$	The set of nodes in graph G
$\mathcal{E}$	The set of edges in graph G
v, w	The nodes $v, w \in \mathcal{V}$
N	The number of nodes in graph G, $N = \lvert\mathcal{V}\rvert$
$D(v)$	The degree centrality of node v
$C(v)$	The closeness centrality of node v
S_v	The self-attention importance score of node v
$P(v)$	The final importance score of node v
ξ_k^j	The kth knot for the jth feature
K	The number of knots for one feature dimension

between them. The goal of node importance estimation is to learn a mapping function: $\mathcal{V} \to \mathbb{R}^{N\times 1}$ that estimates the importance score of each node v in graph.

Definition 2. Graph-level Representation Learning. Given a set of graphs $\mathcal{G} = \{G_i\}_{i=1}^t$ with adjacency matrix $A_i \in \mathbb{R}^{N_i\times N_i}$ and node features $X_i \in \mathbb{R}^{N_i\times d}$, where t denotes the number of graphs, $N_i := |G_i|$ denotes the number of nodes in graph G_i, d denotes the dimension of nodes features, each graph G_i belongs to one category $label_i$. The goal of graph-level representation learning is to learn a mapping function : $\mathcal{G} \to \mathbb{R}^n$ that project each graph G_i into low dimensional vectors in spaces $\mathbb{R}^n$, where $n \ll t$.

Smoothing Splines

Smoothing Splines (SS) is a kind of regression skill, similar to regularization methods like ridge and lasso regression. SS aims to solve the following problem: among all functions $f(x)$ with two continuous derivatives, find one that minimizes the penalized residual sum of squares

$$\mathrm{RSS}(f,\lambda) = \sum_{i=1}^{N}\left\{y_i - f\left(x_i\right)\right\}^2 + \lambda\int_a^b \left\{f''(t)\right\}^2 dt, \quad (1)$$

where λ is a fixed smoothing parameter, x_i is a scalar and $a < x_1 < \cdots < x_N < b$. It has been shown that (1) has an explicit, finite-dimensional, unique minimizer $\hat{f}(x)$ which is a natural cubic spline with knots at the unique values of the x_i and $\hat{f}(x_i) = y_i, i = 1, ..., N$. The natural cubic spline with K knots $\xi_k \in \mathbb{R}, k = 1, ..., K$ can be represented by:

$$\alpha_1(x) = 1, \quad (2)$$

$$\alpha_2(x) = x, \quad (3)$$

$$\alpha_{k+2}(x) = d_k(x) - d_{K-1}(x), \quad (4)$$

where

$$d_k(x) = \frac{(x-\xi_k)_+^3 - (x-\xi_K)_+^3}{\xi_K - \xi_k}, \quad (5)$$

$$\xi_1 < \xi_2 < ... < \xi_K. \quad (6)$$

Each of these basis functions can be proved to have zero second and third derivative for $X \geq \xi_K$, which means the minimizer is linear beyond the boundary. The $\hat{f}(x)$ that minimizes (1) with two continuous derivatives has the form

$$\hat{f}(x) = \sum_{i=1}^{N}\alpha_i(x)\theta_i, \quad (7)$$

where θ_i is the learnable parameter. In this paper, the superiority of smoothing splines is exploited for graph embedding.

Methodology

In this section, we outline the unified model GSSNN and show how it is used to generate high-quality graph embeddings which then can be applied to graph classification. Figure 2 shows the schematic of GSSNN model.

Scaled Smoothing Splines

Inspired by smoothing splines, we designed Scaled Smoothing Splines (S^3) in graph neural networks. Graph Convolutional Networks (GCN) can be represented as:

$$f(X,\hat{A}) = \sigma\left(\hat{A}\,\sigma\left(\hat{A}XW^{(0)}\right)W^{(1)}\right), \quad (8)$$

where $\hat{A}$ is the symmetric normalized laplacian of graph G; $X \in \mathbb{R}^{N\times d}$ is the node features; $W^{(0)} \in \mathbb{R}^{d\times d_1}$ and $W^{(1)} \in \mathbb{R}^{d_1\times d_2}$ are learnable parameters. Considering the first layer in GCN:

$$f_1(X,\hat{A}) = \sigma\left(\hat{A}XW^{(0)}\right), \quad (9)$$

we use $h_{j,k}^1$ to denote the input of the jth node's kth feature in the second layer. Then we have:

$$g_k(X_i) = X_i^T W_k^0, \quad (10)$$

$$h_{j,k}^1 = \sum_{i=1}^{N}\hat{A}_{j,i}g_k(X_i), \quad (11)$$

with $i = 1, 2, ..., N$ and $k = 1, 2, ..., d_1$. When finishing the training of GCN, $g_k(x)$ is sensitive to noisy data and the non smoothness potentially reduces the performance of GCN. Let $(x_i^1, x_i^2, ..., x_i^d)$ be the feature of node v_i. Among all the GCN models, we choose the one with the best performance and use y_i to denote the value of $g_k\left(x_i^1, x_i^2, ..., x_i^d\right)$ for v_i in that GCN model. To make g_k smooth and insensitive to noisy data, we hope to minimize the following penalized residual sum of squares:

$$\mathrm{RSS}(g_k,\lambda) = \sum_{i=1}^{N}\left\{y_i - g_k\left(x_i^1, x_i^2, ..., x_i^d\right)\right\}^2 + \lambda\int_B \sum_{j=1}^{d}\left(\frac{\partial^2 g_k}{\partial {x^j}^2}\right)^2 dx, \quad (12)$$

where $B = (a_1, b_1) \times (a_2, b_2) \times ... \times (a_d, b_d)$ and $x_i^j \in (a_j, b_j), \forall i = 1, ..., N$.

Theorem 1 *If $g_k\left(x^1, x^2, ..., x^d\right)$ that minimizes Eq.(12) with two continuous derivatives has the form*

$$g_k\left(x^1, x^2, ..., x^d\right) = \sum_{j=1}^{d} u_j(x^j), \quad (13)$$

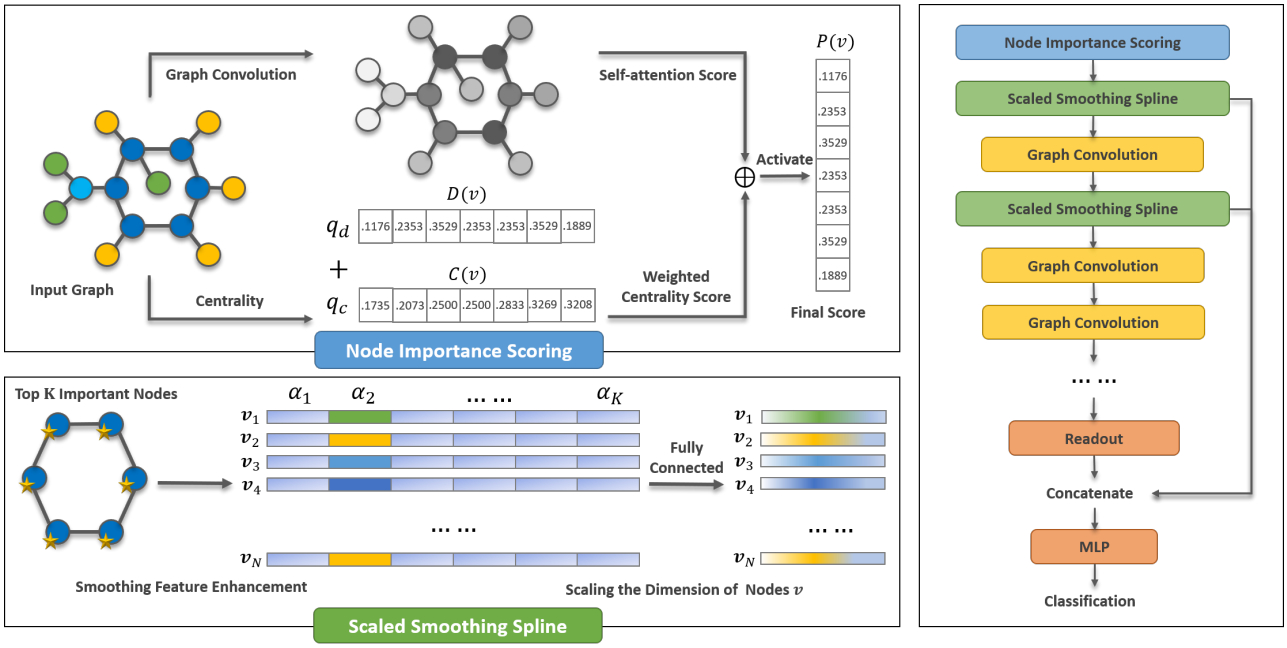


Figure 2: Schematic of GSSNN architecture. (i) Node Importance Scoring: estimates the importance of nodes based on graph features and topology through the integration of self-attention score and weighted centrality score; (ii) Scaled Smoothing Splines: uses top K important nodes features as knots based on node importance scores, to enhance each node via a unique structural perspective for identified features, and scales the dimension through a fully connected layer; (iii) The simplified architecture of GSSNN (right): consists of a node importance scoring layer, two S^3 layers, additional graph convolution layers, and a readout layer. The model is trained end-to-end.

and $u_j(x^j)$ has two continuous derivatives, then Eq.(12) has an explicit, finite-dimensional, unique minimizer:

$$g_k\left(x^1, x^2, ..., x^d\right) = \sum_{j=1}^{d}\sum_{i=1}^{N} \alpha_i^j(x^j)\theta_{ij}, \tag{14}$$

where $\alpha_i^j(x^j)$ can be represented by:

$$\alpha_1^j(x^j) = 1, \tag{15}$$

$$\alpha_2^j(x^j) = x^j, \tag{16}$$

$$\alpha_{k+2}^j(x^j) = d_k^j(x^j) - d_{N-1}^j(x^j), \tag{17}$$

where

$$d_k^j(x^j) = \frac{\left(x^j - \xi_k^j\right)_+^3 - \left(x^j - \xi_N^j\right)_+^3}{\xi_N^j - \xi_k^j}, \tag{18}$$

$$\xi_k^j = x_{l_{j,k}}^j \in \mathbb{R} \text{ and } a_j < x_{l_{j,1}}^j < ... < x_{l_{j,N}}^j < b_j. \tag{19}$$

(a_j, b_j) is the range of the jth feature values of nodes $\mathcal{V}$; x_i^j is the value of the jth feature of node v_i; $l_{j,k}$ are the node indexes that make $x_{l_{j,1}}^j < ... < x_{l_{j,N}}^j$; θ_{ij} is the learnable parameter.

Smoothing Feature Enhancement. According to $Theorem1$, we design a natural cubic splines function on $(x^1, x^2, ..., x^d)$ to make $g_k(x)$ minimize Eq.(12).

$$F_s(X) = \sigma([\beta(X_1^T), \beta(X_2^T), ..., \beta(X_N^T)]^T W_s + b_s) \tag{20}$$

$$\beta(x^1, x^2, ..., x^d) = (\gamma(x^1), \gamma(x^2), ..., \gamma(x^d)) \tag{21}$$

$$\gamma(x^j) = (\alpha_1^j(x^j), \alpha_2^j(x^j), ..., \alpha_K^j(x^j)), \tag{22}$$

where K is the number of knots for one feature dimension, σ is the activation function, W_s and b_s are learnable parameters for scaling the expanded nodes dimension. We can apply them on every single layer of the graph neural network as follows.

$$f_1(X, \hat{A}) = \sigma\left(\hat{A}F_s(X)W^{(0)}\right). \tag{23}$$

Theoretically the natural cubic splines function can make every $g_k(x)$ minimize Eq.(12), if N knots are chosen for every feature dimension. Eq.(11) guarantees the input of every node in the second layer can be written as a linear combination of $g_k(x), k = 1, 2, ..., d_1$. Therefore the input of every node in the second layer can also minimize a penalized residual sum of squares like Eq.(12).

Due to the expansion in dimensions of the enhancement process may lead to overfitting, we scaled each node features into a fixed dimension through a fully connected layer in Eq.(20). After the enhancement process, each node will interact with different description through graph convolution and finally all the nodes can vote which label the graph belong to.

It would be very time-consuming and over smooth if we use all the nodes features as the knots. In practice, thinning strategy is used and we choose the top K most important nodes features as the knots. Let ξ_k^j denote the jth feature of the $l_{j,k}$th important node, then $\left(x^j - \xi_k^j\right)_+^3$ can be regarded as the information enhancement function. The difference between $\left(x^j - \xi_k^j\right)_+^3$ and $\left(x^j - \xi_K^j\right)_+^3$ can help characterize the relationship between the $l_{j,k}$th and the $l_{j,K}$th important nodes. When x^j equals to the node feature value x_i^j, $d_k^j(x^j)$ can be viewed as a description of the relationship between the $l_{j,k}$th and the $l_{j,K}$th important nodes for the node v_i.

Based on $Theorem1$, ξ_k^j should be sorted according to the top K important nodes feature values. In practice, we simplify the sorting operation for every feature dimension and fix the order of $\xi_k^j, j = 1, ..., d$, according to the importance scores of the important nodes. There are two reasons why we simplify the sorting operation. One is that we hope every dimension of the augmented features in Eq.(21) describe the information from the same subset of the important nodes. The other is that if the order of the importance scores is consistent with the order of the feature values in any dimension, S^3 can still promise the smoothness to some extent.

In practice, when using node features as knots in S^3, each dimension of node feature cannot guarantee the consistent greater-than relationship $a_j < x_{l_{j,1}}^j < ... < x_{l_{j,N}}^j < b_j$ in Eq.(18), and there may be cases where the features are equal, resulting in that the denominator of $d_k^j(x^j)$ could be zero. Therefore, we add a small value ϵ to the denominator to ensure it's not zero.

Node Importance Scoring Mechanism

In order to select important nodes features adaptively to serve as knots in S^3 module, we propose the node importance scoring mechanism to learn a mapping function: $\mathcal{V} \to \mathbb{R}^{N\times 1}$ to evaluate the node importance, according to both the topology and features of graph, which consists of two components: self-attention scoring and centrality scoring.

Self-attention Scoring. Without any information about node importance in a graph, we try to use the topology and feature of itself to learn an initial score. Self-attention mechanism have been widely used in recent deep learning studies. We adopt the self-attention scoring method of SAGPool (Lee, Lee, and Kang 2019) to get an initial importance score $S \in \mathbb{R}^{N\times 1}$ as follows.

$$S = \sigma\left(\hat{A}\,\sigma\left(\hat{A}XW^{(0)}\right)W^{(1)}\right), \tag{24}$$

where σ is the activation function, $\hat{A}$ is the symmetric normalized laplacian of graph G; $X \in \mathbb{R}^{N\times d}$ is the node features; $W^{(0)} \in \mathbb{R}^{d\times d}$ and $W^{(1)} \in \mathbb{R}^{d\times 1}$ are learnable parameters.

Centrality Scoring. Without any other information, it is reasonable to assume that highly central nodes are more important than less central ones. Thus, we adjust the initial scores based on node centrality to preserve more global characteristics, including degree and closeness centrality.

Degree centrality $D(v)$ is a simple centrality measure that determines the importance of nodes by counting the number of neighbors. Some works also utilize this information like (Wu, He, and Xu 2019; Park et al. 2019). We normalize it by dividing the maximum possible degree $N-1$ in a simple graph, where N is the number of nodes, so that it allows comparisons between nodes of graphs of different sizes. Closeness centrality $C(v)$ implies the proximity of a node v to other nodes w, calculated as the reciprocal of the average shortest path distance between the node v and all $N-1$ reachable nodes.

As prior knowledge in a graph, degree and closeness centrality provide useful information about the node importance in graph classification. We need to balance of degree centrality, closeness centrality and self-attention scores. Therefore, we utilize weighted sum with three learnable weights q_s, q_d and q_c to measure their contributions to the final scores. At last, we apply an activation function σ as follows.

$$P(v) = \sigma\left(q_s S_v + q_d D(v) + q_c C(v)\right), \tag{25}$$

where for node v, S_v is initial important score, and $P(v)$ is final importance score.

Model Architecture

We implement our model GSSNN with an end-to-end architecture. The right panel in Figure 2 shows a simplified version. As we can see, for each epoch, we first compute the important scores of nodes, and then with top K important nodes features as knots, node features are smoothed via S^3 module before each graph convolution layer. At last, following several graph convolution layers to make the expanded features deeply integrated based on graph topology. In addition, batch normalization is added after each propagation layer for improving the speed and stability of propagation.

Readout Layer. For keeping the permutation invariant and graph isomorphism properties, we apply summation pooling function that sums all node features after graph convolution to get a fixed size graph-level representation. In addition, the summarized output feature can be concatenated with the knots features used in S^3.

$$h_G = \text{CONCAT}\left\{\text{SUM}\left\{h_v | v \in G\right\}, p\left(\xi_i | i = 1, ..., K\right)\right\},$$

where p is a pooling function, such as mean operation.

Model Training

After readout layer, we take the graph embedding and feed it to a multilayer perceptron (MLP) for predicting graph labels. Our model is trained by minimizing the cross-entropy loss over all labeled training examples:

$$\mathcal{L} = -\sum_{l\in\mathcal{Y}_L}\sum_{f=1}^{F} Y_{lf}\ln X_{lf}, \tag{26}$$

where $\mathcal{Y}_L$ is the set of labeled graph, X_{lf} is the f-th entry of the network output for i-th labeled graph and Y_{lf} denotes its ground truth label.

Complexity Analysis

For the S^3 module, the time complexity is $O(NdK)$, where N is the number of nodes in the graph, d is the dimensions of the original node feature and K is the number of knots for one dimension. The computational complexity of the remaining structures is the same with GCN (Kipf and Welling 2017).

Experiments

In this section, we conduct extensive experiments to verify the performance of proposed model GSSNN on graph classification.

Experiments Setup

Datasets. We validate the performance of generated graph representations on classification task over 4 biological datasets and 3 social datasets (Kersting et al. 2016). The statistics of the datasets are summarized in Table 2.

Biological datasets. The MUTAG dataset consists of 188 chemical compounds divided into two classes according to their mutagenic effect on a bacterium. PROTEINS are sets of proteins that are classified into enzymes and non-enzymes. D&D consists of protein structures, with the nodes in each graph represent amino acids. NCI1 is a subset of balanced datasets of chemical compounds screened for ability to suppress or inhibit the growth of a panel of human tumor cell lines. Since nodes in the four datasets have discrete labels, the node input features are initialized as one-hot of labels.

Social datasets. IMDB-BINARY and IMDB-MULTI are movie collaboration datasets. Each graph corresponds to an ego-network for each actor/actress, and the task is to classify the genre graph it is derived from. COLLAB is a scientific collaboration dataset, derived from public collaboration datasets, and the task is to classify each graph to a field the corresponding researcher belongs to. Since nodes have no label in the three datasets, the node input features are initialized as the degree of each node.

Table 2: A summary of the benchmark Datasets. For each dataset, we report the number of graphs, the source of graphs, number of classes, number of averaged nodes, and number of averaged edges, respectively.

Dataset	Source	Graphs	Classes	Avg.N	Avg.E
MUTAG	Bio	188	2	17.93	19.79
PROTEINS	Bio	1113	2	39.06	72.81
D&D	Bio	1178	2	284.31	715.65
NCI1	Bio	4110	2	29.87	32.30
IMDB-B	Social	1000	2	19.77	193.06
IMDB-M	Social	1500	3	13	131.87
COLLAB	Social	5000	3	74.49	4914.99

Baselines. We compare our model to several state-of-the-art baselines, including 3 kernel-based methods and 6 GNN-based methods, to evaluate the effectiveness of our model. The details are given as follows.

- Kernel-based methods: Weisfeiler-Lehman Kernels (WL) (De Vries and de Rooij 2015), Graphlet Count Kernel (GK) (Shervashidze et al. 2009) and Deep Graph Kernel (DGK) (Yanardag and Vishwanathan 2015). These three algorithms are sub-components based and then machine learning algorithms are applied to perform graph classification.
- GNN-based methods: Three algorithms without pooling are compared: GCAPS-CNN (Verma and Zhang 2019), Capsule Graph Neural Network (CapsGNN) (Xinyi and Chen 2019) and Graph Isomorphism Network (GIN) (Xu et al. 2019); Another three GNNs with pooling are compared: SortPool (Zhang et al. 2018), DiffPool (Ying et al. 2018) and SAGPool (Lee, Lee, and Kang 2019).

Parameter Settings. In our experiments, we evaluated all the GNN-based methods over the same random seed using 10-fold cross validation. 10 percent of the data was used for testing and the rest were used for training. In addition, the same early stopping criterion, hyper-parameter selection strategy are used for all the baselines to ensure a fair comparison. The optimal hyper-parameters are obtained by grid search. The ranges of grid search are summarized in Table 3. We implemented GSSNN using the Adam optimizer (Kingma and Ba 2014) and the geometric deep learning extension library provided by (Fey and Lenssen 2019).

Table 3: The grid search space for the hyperparameters.

Hyperparameter	Range
Hidden dimension.	16, 32, 64
Weight decay	1e-4, 5e-4
S^3 layer	1, 2, 3
Convolutional layer	2, 3, 4, 5
ξ number	3, 4, 5

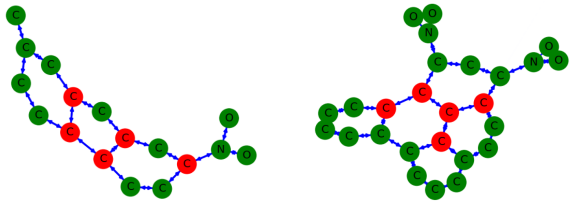
Experimental Results

Graph Classification. We report averaged accuracy and standard deviation in Table 4. By default, we use the results reported in the original work for kernel-based baselines (Xinyi and Chen 2019). Compared with kernel-based and GNN-based baselines, our proposed method GSSNN achieves the best graph classification performance on six biological and social datasets, demonstrating the capability of GSSNN on classifying graphs. At the same time, the smaller standard deviation compared with baselines can indicate that GSSNN can generate a more robust graph representation.

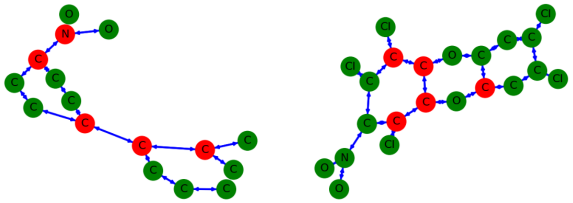
Interpretability. One attractive property of our algorithm is that the important nodes, which are used as knots for the Scaled Smoothing Splines, provides interpretability to the classification results. Here we visualize the important nodes selected by our algorithm in Figure 3. Specifically, We choose to depict the topology of MUTAG graphs, which consists of chemical compounds divided into two classes according to their mutagenic effect on a bacterium, where mutagenic effect mainly depends on the structure of the chemical compound.

Table 4: Graph classification results of biological and social datasets in accuracy.

Method	MUTAG	NCI1	PROTEINS	DD	COLLAB	IMDB-B	IMDB-M
WL	82.05±0.36	**82.19±0.18**	74.68±0.49	79.78±0.36	79.02±1.77	73.40±4.63	49.33±4.75
GK	81.58±2.11	62.49±0.27	71.67±0.55	78.45±0.26	72.84±0.28	65.87±0.98	43.89±0.38
DGK	87.44±2.72	80.31±0.46	75.68±0.54	73.50±1.01	73.09±0.25	66.96±0.56	44.55±0.52
GCAPS-CNN	89.62±5.38	81.35±2.37	75.70±3.86	78.82±3.17	77.32±1.98	72.02±4.10	49.31±5.30
GapsGNN	87.78±6.68	78.25±2.22	75.68±3.22	75.88±3.41	79.67±1.24	74.68±3.10	52.17±4.25
GIN	93.50±6.49	80.85±2.34	76.81±3.78	77.76±2.27	80.50±1.43	78.60±3.37	54.33±4.49
SortPool	86.62±4.72	70.36±4.36	76.72±3.77	75.27±2.60	78.70±1.52	74.40±5.29	53.07±5.20
DiffPool	89.79±8.15	78.29±3.33	77.02±3.23	70.95±2.41	79.70±1.84	78.08±4.24	53.13±4.70
SAGPool	90.42±7.78	77.62±2.37	76.55±3.50	76.91±2.12	79.88±1.02	78.10±4.20	53.80±4.08
GSSNN	**96.77±4.68**	80.75±4.07	**79.73±3.31**	**80.26±2.50**	**81.60±1.26**	**80.10±3.25**	**59.00±3.80**



(a) Class 1: graph 1 (b) Class 1: graph 2



(c) Class 2: graph 1 (d) Class 2: graph 2

Figure 3: Visualization of important nodes in MUTAG dataset. We extract 4 graphs from two different classes and draw the topology of each graph, where the red nodes represent important nodes selected by the trained GSSNN.

As we can see from Figure 3, important nodes are mainly focused on heavy atoms with large degree, which determine the structure and properties of the compound to a large extent. Therefore, the important nodes features have a great influence on the mutagenic effect. Understanding these important nodes or substructures which affect the graph classification results is very important to many applications such as drug discovery in biological domain.

Global Information. Our scoring component exploits both local and global information from three aspects (as shown in Eq.(25)), self-attention score S_v, node degree $D(v)$, and closeness $C(v)$. We report the results using different strategy in Table 5. It is clear that exploiting global information $C(v)$ leads to significant improvement over other strategies which only consider local information such as self-attention score and degree information. This demonstrates the effectiveness of our design in exploiting global topological information.

Table 5: Graph classification accuracy with different scoring strategies.

Method	S_v	$S_v + D(v)$	$S_v + D(v) + C(v)$
MUTAG	88.89	94.44	96.77
DD	74.62	77.78	80.26
IMDB-B	77.30	79.30	80.10

S^3 as a Plugin. Our S^3 module can also serve as an individual layer to plug into existing graph neural networks. To demonstrate the effectiveness of the smoothing module, we integrate S^3 with two most important GNNs, GCN and GAT. Specifically, S^3 module is added after the propagation layers of GCN and GAT respectively, and following the same readout layer to generate graph representation. The parameter setting is consistent across all methods. As shown in Table 6, the performance on graph classification of models plugged with S^3 outperform the model itself.

Table 6: Graph classification results of existing GNNs plugged with S^3 in accuracy.

Method	MUTAG	PROTEINS	DD	IMDB-M
GCN	93.50	76.81	77.76	54.33
GCN+S^3	96.77	79.73	80.26	59.00
GAT	95.33	77.48	77.78	55.33
GAT+S^3	96.89	80.18	81.20	56.67

Conclusion

GNNs are effective algorithms for graph classification. However, existing GNNs fail to explore the smoothing node feature and ignore global topological information. To overcome these limitations, we present an end-to-end learning algorithm, Graph Smoothing Splines Neural Networks (GSSNN), for graph classification. By employing the smoothing splines to smooth nodes features, our algorithm enables a high-quality and more robust graph representation.

With the integration of global information closeness centrality, as well as degree information into a self-attention module, our algorithm provides important score to each node. The important nodes features are served as knots in Scaled Smoothing Splines (S^3) that can be potentially used for interpreting classification results. Our smoothing component S^3 can be easily fit into existing GNN models, achieving improvement for the graph classification task. Extensive experimental results on graph classification demonstrate the state-of-the-art performance on six biological and social datasets.

Acknowledgments

This work was supported by the NSFC (No.11688101, 61872360 and 11631014), the Youth Innovation Promotion Association CAS (No. 2017210), and by the US National Science Foundation (NSF) through Grants IIS-1763452 and CNS-1828181. Shirui Pan is the corresponding author.

References

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

De Vries, G. K. D., and de Rooij, S. 2015. Substructure counting graph kernels for machine learning from rdf data. *Web Semantics: Science, Services and Agents on the World Wide Web* 35:71–84.

Duvenaud, D. K.; Maclaurin, D.; Iparraguirre, J.; Bombarell, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, 2224–2232.

Fey, M., and Lenssen, J. E. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark data sets for graph kernels.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *Proceedings of the 7th International Conference on Learning Representations*.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning*.

Park, N.; Kan, A.; Dong, X. L.; Zhao, T.; and Faloutsos, C. 2019. Estimating node importance in knowledge graphs using graph neural networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Qiu, J.; Tang, J.; Ma, H.; Dong, Y.; Wang, K.; and Tang, J. 2018. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2110–2119. ACM.

Shervashidze, N.; Vishwanathan, S.; Petri, T.; Mehlhorn, K.; and Borgwardt, K. 2009. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, 488–495.

Shervashidze, N.; Schweitzer, P.; Leeuwen, E. J. v.; Mehlhorn, K.; and Borgwardt, K. M. 2011. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research* 12(Sep):2539–2561.

Shi, C.; Hu, B.; Zhao, W. X.; and Philip, S. Y. 2019. Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 31(2):357–370.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations*.

Verma, S., and Zhang, Z.-L. 2019. Graph capsule convolutional neural networks. In *Proceedings of the 7th International Conference on Learning Representations*.

Vinyals, O.; Bengio, S.; and Kudlur, M. 2015. Order matters: Sequence to sequence for sets.

Vishwanathan, S. V. N.; Schraudolph, N. N.; Kondor, R.; and Borgwardt, K. M. 2010. Graph kernels. *Journal of Machine Learning Research* 11(Apr):1201–1242.

Wu, J.; He, J.; and Xu, J. 2019. Demo-net: Degree-specific graph neural networks for node and graph classification. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM.

Xinyi, Z., and Chen, L. 2019. Capsule graph neural network. In *Proceedings of the 7th International Conference on Learning Representations*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations*.

Yanardag, P., and Vishwanathan, S. 2015. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1365–1374. ACM.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems*, 4800–4810.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*.

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; and Sun, M. 2018. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.