Towards Expressive Graph Representations for Graph Neural Networks

Chengsheng Mao
Department of Preventive Medicine
Northwestern University
Chicago, USA
chengsheng.mao@northwestern.edu

Liang Yao
Tencent Inc.
Shenzhen, China
dryao@tencent.com

Yuan Luo
Department of Preventive Medicine
Northwestern University
Chicago, USA
yuan.luo@northwestern.edu

Abstract—Graph Neural Network (GNN) aggregates the neighborhood information into the node embedding and shows its powerful capability for graph representation learning in various application areas. However, most existing GNN variants aggregate the neighborhood information in a fixed non-injective fashion, which may map different graphs or nodes to the same embedding, detrimental to the model expressiveness. In this paper, we present a theoretical framework to improve the expressive power of GNN by taking both injectivity and continuity into account. Based on the framework, we develop injective and continuous expressive Graph Neural Network (icGNN) that learns the graph and node representations in an injective and continuous fashion, so that it can map similar nodes or graphs to similar embeddings, and non-equivalent nodes or non-isomorphic graphs to different embeddings. We validate the proposed icGNN model for graph classification and node classification on multiple benchmark datasets. The experimental results demonstrate that our model achieves state-of-the-art performances on most of the benchmarks.

Index Terms—graph neural network, graph representation, expressive power, injective mapping, continuous function, set representation.

I. INTRODUCTION

Graph representation learning that maps graphs or their components to vector representations has attracted growing attentions for graph analysis. Recently, graph neural networks (GNN) that can learn a distributed representation for a graph or a node in a graph are widely applied to a variety of areas, such as social network analysis [1], molecular structure inference [2], text mining [3], [4], clinical decision making [5], [6] and image processing [7]. GNN recursively updates the representation of a node in a graph by aggregating the feature vectors of its neighbors and itself [8], [9]. The graph-level representation can then be obtained through aggregating the final representations of all nodes in the graph.

Many GNN variants with different aggregation rules are proposed to achieve good performance for different tasks and problems [10]–[14]. However, most of the existing GNN aggregation rules are designed based on a fixed non-injective pooling function, (e.g., max pooling and mean pooling) or on non-continuous node types (e.g., graph isomorphism test). The non-injective aggregation may map different (non-isomorphic) graphs or (non-equivalent) nodes to the same embedding; and the non-continuous aggregation may map similar graphs or nodes to quite different embeddings, both detrimental to the expressive power of GNN. For example, for the graph with attributed nodes in Figure 1(a), mean pooling or sum aggregation in the neighborhoods generates the same neighborhood representation for all the nodes (Figure 1(d)), thus cannot capture any meaningful structure information. Xu et al. [8] showed that a powerful GNN can at most achieve discriminative power as the Weisfeiler-Lehman graph isomorphism test (WL test) which can discriminate a broad class graphs [15], and proposed the powerful graph isomorphism network (GIN). However, the theoretical framework of GIN is under the assumption that the input feature space is countable, which makes GIN less expressive when applied to graphs with continuous attributes, i.e., attributed graphs.

We argue that the expressive power of a graph mapping should imply two aspects, injectivity and continuity: the injectivity ensures different graphs are mapped to different representations and the continuity ensures that similar graphs are mapped to similar representations. Most previous works only took either one into account for GNN design; few considered both injectivity and continuity. Here, we present a theoretical framework that can guide us to design high expressive GNNs with both injectivity and continuity for general graphs with continuous attributes. We also present a necessary condition related to the representation dimension for a fully injective and continuous graph mapping. The general idea of our framework is illustrated in Figure 1, first, each node with attribute in a graph (Figure 1(a)) is converted to a tuple representation containing its attribute and a set of its neighborhood attributes through one WL test iteration (Figure 1(b)), then we design an injective and continuous set function to map each neighborhood to a vector (Figure 1(c)). After certain learnable transformations, the graph with node embeddings can go the next iteration. After $k$ iterations, a node representation can capture the structural information and attribute information within the node’s $k$-hop neighborhood. Due to the injective and continuous set mapping, equivalent nodes in the graph have the same representation.

Our main contributions are summarized as follows. (1)
We present a theoretical framework to guide the design of expressive GNNs by ensuring the injectivity and continuity in the neighborhood aggregation process. (2) We present a limitation about the representation dimension for a fully injective and continuous graph mapping. (3) Based on the framework, we implement two injective and continuous expressive GNN (iceGNN) models with a fixed and learnable aggregation function, respectively. (4) We validate our models on multiple benchmark datasets including simple graphs and attributed graphs for graph classification and node classification, the experimental results demonstrate that our models can achieve state-of-the-art performances on most of the benchmarks. Our code will be available at https://github.com/mocherson/Exp_GNN.

II. PRELIMINARIES

A. Multiset

A multiset can be considered as a generalized concept of a set where the order of objects does not matter and the same object can repeat multiple times. In a finite graph, the neighborhood of a node can be considered as a finite multiset. For a graph with continuous node features, the representation vectors are from an uncountable feature space \( \mathbb{R}^d \). In this paper, unless otherwise stated, the word "set" means a finite multiset.

B. Graph neural networks

Most modern GNNs fall into the category of message passing neural networks [2] that follow a neighborhood aggregation strategy that recursively updates a node representation by aggregating representations of its neighbors and the node itself. The graph-level representation is obtained through aggregating the final representations of all the nodes in the graph. Formally, the propagation rule of a GNN layer can be represented as

\[
H^{(k+1)}(v) = f^{(k)}_C \left( H^{(k)}(v), H^{(k)}_{N(v)} \right)
\]

where \( H^{(k)}(v) \) is the representation vector of node \( v \) in the \( k \)th layer, and \( H^{(0)}(v) \) is initialized with \( X(v) \), the original attributes of node \( v \). \( N(v) \) is the neighborhood of \( v \). \( f^{(k)}_A(\cdot) \) aggregates over the neighborhood \( N(v) \) to generate a neighborhood representation \( H^{(k)}_{N(v)} \), and \( f^{(k)}_C(\cdot) \) combines the node’s current representation \( H^{(k)}(v) \) and its neighborhood’s representation \( H^{(k)}_{N(v)} \) in the \( k \)th layer.

For node embedding, the node representation in the final layer \( H^{(K)}(v) \) (suppose a total of \( K \) layers) is considered as an informative representation. For graph or subgraph embedding, another aggregation function \( f_R(\cdot) \) is employed to obtain the graph-level representation \( h_G \) by aggregating the final representations of all nodes in the graph or subgraph \( G \), i.e.,

\[
H_G = f_R \left( \{ H^{(K)}(v) | v \in G \} \right)
\]

\( f_A(\cdot), f_C(\cdot) \) and \( f_R(\cdot) \) are all crucial for the expressive power of a GNN. \( f_A(\cdot) \) and \( f_R(\cdot) \) are set functions that map a set to a vector, they can be simple summations or sophisticated graph-level pooling functions [12, 14, 16]. \( f_C(\cdot) \) operates on two vectors, it can be usually modeled by a multi-layer perceptron (MLP) or linear function on the concatenated vector.

C. The expressive power of GNN

Recently, theoretical analysis showed that the expressive power of a GNN is associated with the WL test [8, 17]. Xu et al. [8] proved that the expressive power of GNN is bounded by the one-dimensional Weisfeiler-Lehman test. The following Lemma and Theorem from [8] describe the relation between GNNs and WL test in expressive power on discriminating graphs, refer to [8] for the proofs.

Lemma 1. If the WL test decides two graphs \( G_1 \) and \( G_2 \) are isomorphic, any GNNs defined by Eq. 2 and 3 will map \( G_1 \) and \( G_2 \) to the same embedding.

Theorem 1. If WL test decides two graphs \( G_1 \) and \( G_2 \) are not isomorphic, a GNN with sufficiently many GNN layers defined by Eq. 2 and 3 can also map \( G_1 \) and \( G_2 \) to different embeddings if the functions \( f_A(\cdot), f_C(\cdot) \) and \( f_R(\cdot) \) are all injective.

From the above Lemma and Theorem, The key is to design injective functions for \( f_A(\cdot), f_C(\cdot) \) and \( f_R(\cdot) \). An injective function for \( f_C(\cdot) \) that operates on two vectors can be easily obtained by concatenating the two vectors. Designing an injective function for \( f_R(\cdot) \) or \( f_R(\cdot) \) that operates on a set is not trivial, because a set can have a variable number of elements, and the operation on the set elements must be permutation-invariant. Some popular aggregations by pooling, e.g., mean pooling and max pooling, are continuous but not injective. Few works took both injectivity and continuity. In this paper, we try to design the continuous injective aggregation function on a set and show a necessary condition related to the dimension for injective and continuous aggregation.

III. METHODS

A. Set representation

A set function is a function defined in a domain that is a collection of sets. In a finite graph, the neighborhood of each
node is considered as a finite set (specifically a multiset). Thus, in this paper, we only consider set functions of finite sets. A continuous set function is of real importance in practice [18]. The continuity of a function ensures that the change in output is very small if the input is altered slightly by any reason such as truncating to machine precision. In this paper, we consider the ordinary continuity where the continuity of function $f(x)$ at point $c$ is defined by the limit as $\lim_{x \to c} f(x) = f(c)$.

For $M \in \mathbb{N}$, a set function $f(X)$ defined in domain $X_M = \{X | X \subset \mathbb{R}^d, |X| \leq M\}$ can be represented as a sequence of permutation-invariant functions $f_i$ for different set sizes, i.e.,

$$f(X) = f_1(x_1, \ldots, x_i) \quad \text{if} \quad |X| = i \leq M,$$

where $x_1, \ldots, x_i \in X$.

**Definition 1** (Continuous set function). For $M \in \mathbb{N}$ and a set function $f(X)$ defined in domain $X_M = \{X | X \subset \mathbb{R}^d, |X| \leq M\}$, $f(X)$ is represented as Eq. 4, if $f_j(x_1, \ldots, x_i)$ is continuous in the Euclidean space for every $i \leq M$, we call $f(X)$ a continuous set function.

Obviously, a continuous set function can also have the property that sufficiently small changes in the input (a sufficiently small change will not change the set size) result in arbitrarily small changes in the output. Thus, by a continuous set function, graphs with very similar structures and attributes could be mapped to similar embeddings. An injective set function can map distinct sets to distinct values. Thus, an injective and continuous set representation function can encode a set to a representation such that different sets have different representations and similar sets have similar representations.

**Theorem 2.** Assume $X_M$ is a set of finite subsets of $\mathbb{R}^d$ with size less than or equal to $M$, i.e., for $M \in \mathbb{N}$ and $X_M = \{X | \exists X \subset \mathbb{R}^d, |X| \leq M\}$, there exists an infinite number of continuous functions $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$ such that the set function $f : X_M \rightarrow \mathbb{R}^D$, $f(X) = \sum_{x \in X} \Phi(x)$ is continuous and injective.

In the proof, we also provide a way to construct such a function by defining a continuous injective function $g : \mathbb{R}^d \rightarrow \mathbb{R}^d$, $\Phi(g(x))$ can also satisfy the condition if the function $\Phi(x)$ satisfies the condition. We call $\Phi(x)$ the transformation function.

The following theorem tells a necessary condition of constructing an injective and continuous set function by sum aggregation.

**Theorem 3.** Let $M \in \mathbb{N}$ and $X_M = \{X | X \subset \mathbb{R}^d, |X| = M\}$, then for any continuous function $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^N$, if $N < dM$, the set function $f : X \rightarrow \mathbb{R}^N$, $f(X) = \sum_{x \in X} \Phi(x)$ is not injective.

Theorem 3 tells that, to construct a continuous injective set function for sets with $M$ $d$-dimensional vectors by sum aggregation with continuous transformation $\Phi(x)$, $\Phi(x)$ must have at least $dM$ dimensions. We are restricting $\Phi$ as a continuous function so that it can be modeled by a neural network, because a neural network can approximate any continuous function rather than any function by the universal approximation theorem [20].

**B. Injective and continuous graph neural networks**

Since Theorem 2 tells that a set can be uniquely represented by a sum aggregation of its elements through a continuous transformation function, we can use the unique set representation to model neighborhood of each node in a graph, and thus improve the expressiveness of graph representation. From Theorem 1, to design an expressive GNN, we need to design injective functions for $f_A(\cdot)$, $f_C(\cdot)$ and $f_R(\cdot)$. Since $f_C(\cdot)$ is easy to get continuous and injective, and $f_A(\cdot)$ and $f_R(\cdot)$ both operate on a set of vectors in $\mathbb{R}^d$, thus, we need Theorem 2 to guide us to construct a continuous and injective set function for $f_A(\cdot)$ and $f_R(\cdot)$ by sum aggregation after a certain continuous transformation.

1) **COMBINE function**: In the COMBINE function, we just concatenate the two input vectors to get an injective and continuous function, and apply a MLP to adaptively reduce the output dimension, Eq. 6.

$$f_C^{(k)}(x_1, x_2) = MLP^{(k)}([x_1, x_2]) \quad (6)$$

Note that an MLP mapping high-dimensional vectors to low-dimensional vectors cannot be continuous and injective. Here, MLP is used for task-driven feature reduction.

2) **AGGREGATE function**: $f_A(\cdot)$ operates on a set of node embeddings in the neighborhood of a node. We have two choices of the transformation function of $f_A(\cdot)$, i.e., fixed transformation and learnable transformation.

**Fixed transformation.** We find the function $\Phi'_M(x)$ defined in Eq. 5 can be used as a continuous transformation function to make the sum aggregation continuous and injective in most cases. Let $M_n$ be the maximum neighborhood size for all nodes in the graphs. We set the transformation function as $\Phi'_M(x)$.
for each layer $k$ to maintain the expressive power. Then $f_A(\cdot)$ for layer $k$ can be represented as
\[
 f_A^{(k)}\left(\{H^{(k)}(w)|w \in N(v)\}\right) = \sum_{w \in N(v)} \Phi_{M_w}^{(k)}(H^{(k)}(w))
\]

(7)

Though the function $\Phi_M(\cdot)$ defined in Eq. 5 can make the sum aggregation continuous and injective, it may result in numerical stability issue since the item $x[i]^M$ will make the number become very large or very close to 0 if $M$ is very large. To address this issue, we use a continuous and injective function $g(\cdot)$ to normalize the power, since in the proof of Theorem 2 we know $\Phi_M(g(\cdot))$ is also a qualified transformation function to make the sum aggregation continuous and injective, if $g(\cdot)$ is continuous and injective. In this paper, we set
\[
 g(x)[i] = \begin{cases} x[i]^{1/M}, & x[i] \geq 0 \\ (-x[i])^{1/M}, & x[i] < 0 \end{cases}
\]

(8)

Learnable transformation. Due to the continuity of the transformation function, we can also set a learnable MLP to approximate the transformation function for $f_A(\cdot)$ by the universal approximation theorem [20], i.e.,
\[
 f_A^{(k)}\left(\{H^{(k)}(w)|w \in N(v)\}\right) = \sum_{w \in N(v)} MLP\left(H^{(k)}(w)\right)
\]

(9)

By this way, we get all the node embeddings for all graphs. For graph or subgraph embedding, we need another aggregation function $f_R(\cdot)$ to aggregate all the node embeddings in a graph.

3) READOUT function: $f_R(\cdot)$ operates on a set of all node embeddings in a graph. For large graphs with many nodes, a fully injective and continuous set function will generate a high-dimensional embeddings. We also use a learnable MLP as the transformation function to reduce the output dimension.
\[
 H_G = f_R\left(\{H^{(K)}(v)|v \in G\}\right) = \sum_{v \in G} MLP_G(H^{(K)}(v))
\]

(10)

IV. EXPERIMENTS

A. An example on a synthetic dataset

To demonstrate the expressive power of iceGNN for graphs of nodes with continuous node attributes, we designed a synthetic dataset consisting of 1000 graphs and ran several comparable GNN models on it. In our dataset, each graph has the number of nodes ranging from 10 to 50, and each node in a graph has a random degree. 500 graphs with node features from $N(0, 1)$ are labeled as class 1, the other 500 graphs are labeled as class 2 and had node features drawn from $N(0, 2)$.

We implement 2 iceGNN variants: (1) iceGNN-fixed, where fixed transformation functions in all layers are set as $\Phi_M(\cdot)$ or $\Phi_M^*(\cdot)$ in Eq. 5; (2) iceGNN-MLP, where the transformation functions in all layers are set as a learnable MLP. We also implemented GIN-final [8] and GCN [8] by adjusting the corresponding official code. Since the output of the final layer is already considered to contain information from preceding layers, GIN-final did not use Jumping Knowledge structure.

| MUTAG | PTC | NCI1 | PROTEINS |
|-------|-----|------|----------|
| iceGNN-fixed | 91.1 ± 6.7 | 67.9 ± 7.3 | 82.9 ± 1.6 | 77.5 ± 6.2 |
| iceGNN-MLP | 90.6 ± 7.9 | 68.8 ± 7.2 | 83.6 ± 1.9 | 76.5 ± 5.5 |
| GIN [8] | 89.4 ± 5.6 | 64.6 ± 7.0 | 82.7 ± 1.6 | 76.2 ± 2.8 |
| GCN [10] | 87.8 ± 6.0 | 62.7 ± 8.0 | 73.5 ± 1.4 | 71.0 ± 5.0 |
| GraphsSAGE [11] | 85.1 ± 7.6 | 63.9 ± 7.7 | 77.7 ± 1.5 | 75.9 ± 3.2 |
| PSCGN [22] | 92.6 ± 4.2 | 60.0 ± 4.8 | 78.6 ± 1.9 | 75.9 ± 2.8 |
| CapsGNN [13] | 95.7 ± 4.3 | 78.4 ± 1.6 | 76.3 ± 1.6 |
| IEGN [23] | 84.6 ± 10 | 59.5 ± 7.3 | 73.7 ± 2.6 | 75.2 ± 4.3 |
| HaarPool [14] | 90.0 ± 3.6 | - | 78.6 ± 0.5 | 80.4 ± 0.8 |
| 3WLN [24] | 90.5 ± 8.7 | 66.2 ± 6.5 | 83.2 ± 1.1 | 77.2 ± 4.7 |
| InfoGraph [25] | 89.0 ± 1.1 | 61.6 ± 1.4 | - | - |
| CMV-GR [26] | 89.7 ± 1.1 | 62.5 ± 1.7 | - | - |
| WL subtree [27] | 90.4 ± 5.7 | 59.9 ± 4.3 | 86.0 ± 1.8 | 75.0 ± 3.1 |
| DGL [28] | 87.4 ± 2.7 | 60.1 ± 2.6 | 80.3 ± 0.5 | 75.7 ± 0.5 |
| WL-QA [29] | 84.5 ± 1.7 | 63.6 ± 1.5 | 86.1 ± 0.2 | 76.4 ± 0.4 |
| WWL [30] | 87.3 ± 1.5 | 66.3 ± 1.2 | 85.7 ± 0.3 | 74.2 ± 0.5 |
| MLG [31] | 87.9 ± 1.6 | 63.3 ± 1.5 | 81.8 ± 0.2 | 76.3 ± 0.7 |

but only used the output of the final layer as node embeddings to sum to graph embedding.

The training accuracy varied with training epochs is shown in Fig. 2a, where the training accuracy of both iceGNN-fixed and iceGNN-MLP increases significantly after a few training epochs, reaching almost 1, much higher than that of GCN (0.5) and GIN-final (0.7), indicating the high expressive power of iceGNN compared to GIN-final and GCN. We also recorded the output of the READOUT function (Eq. 10) as the graph embeddings. Both iceGNN-fixed and iceGNN-MLP are able to produce 1000 unique embeddings for the 1000 graphs, indicating that iceGNN models have a high level of injectivity. On the other hand, the GCN model is only able to produce 313 unique graph embeddings, while the GIN-final model produces 926. This suggests that the iceGNN models have a higher level of injectivity than GIN-final and GCN.

B. Graph classification

1) Datasets: We evaluate our models on 4 simple graph benchmarks (MUTAG, PTC, NCI1, PROTEINS) and 5 attributed graph benchmarks (ENZYMES, FRANKENSTEIN, PROTEINSatt, SYNTHETICNEW, Synthie) for graph classification. For simple graphs, the categorical node labels are encoded as one-hot input features. All the datasets are available from [21].

2) Performance on test set: Table I and II list the classification accuracies on test set for simple graph classification and attributed graph classification, respectively. We highlight the top 3 accuracies for each dataset in boldface. From Table I, iceGNN can achieve top 3 on all the datasets except for NCI dataset where we achieve the best results among the deep learning models though. From Table II, for all the attributed graph datasets, iceGNN can achieve top 3 in these 10 models, especially, iceGNN-MLP places first on 3 datasets. Comparing

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
iceGNN and GIN-final, iceGNN can consistently outperform GIN-final except for iceGNN-fixed on ENZYMES dataset.

3) Performance on training set: To evaluated the expressive power, Figure 2b-2c illustrates accuracies in training sets in the training process on 3 datasets for graph classification. We can see that iceGNN-MLP on different datasets is able to fit the training sets perfectly and is better than GIN-final and iceGNN-fixed, specifically from the figures, iceGNN-MLP>iceGNN-fixed>GIN-final>GCN, in terms of the expressive power. For MUTAG and PTC datasets, GIN in [8] can fit the training set, while GIN-final cannot, since GIN concatenates all the middle layer outputs as the graph embedding, it may be the reason that the final layer outputs of GIN may lose some information from middle layers.

To identify whether the expressive power is just due to more parameters, we enlarge GIN with 5 hidden layers in each MLP, so that the number of parameters achieve the same scale with iceGNN, denoting the GIN architecture as GIN-mlp5. We found that GIN-mlp5 still cannot achieve the training accuracy as iceGNN or even worse than GIN-final, as shown in Figure 2b-2c. The results demonstrate that, iceGNN is more expressive than GIN on most of the datasets, and the expressive power of iceGNN comes from the injective and continuous aggregation scheme rather than the number of parameters.

C. Node classification

We use three popular citation network datasets Cora, Citeseer, and Pubmed [35] for semi-supervised node classification. We compared our performance with a recent state-of-the-art, GCNII [36]. Since the expressive power describes the ability of a model to discriminate different graphs in training set, a larger training set can reflect the expressive power better. Because the official splits have only a few nodes in training, we split the nodes into training, validation and test sets by 8:1:1. The node classification results on training set and test set are listed in Table III, where we found that iceGNNs and can outperform all other baselines on training set, suggesting iceGNNs have strong expressive ability. For test set, iceGNN-MLP performs better than all other models on Pubmed dataset, but does not perform that well on Cora and Citeseer datasets comparing to GIN and GIN-final, suggesting that iceGNN-MLP also has better generalization ability on Pubmed dataset than Cora and Citeseer datasets. We also plot the training curves for node classification in Figure 2d, which once again shows the expressive power of iceGNN.

D. Expressive power analysis

The expressive power of a model refers to its ability to accurately distinguish between different samples and capture complex patterns and relationships in the data. This should contains two aspects: injectivity and continuity. An injective model will map different samples to different embeddings and a continuous model will map similar samples to similar embeddings by ensuring that slight changes in the input should only result in slight changes in the output. For classification problems, an expressive model should make samples in the same class compact together and samples in different classes highly discriminative. Here, we fetch the output embeddings of GNNs before feeding to the classifier, and visualize them to see how a GNN model can discriminate samples from different classes. Figure 3 shows the t-SNE visualization of graph embeddings of different GNN models on NCI1 dataset. We can see that the output embeddings of iceGNN-MLP and
iceGNN-fixed are discriminative on most datasets, validating that iceGNN models are more expressive than GIN-final. The iceGNN models demonstrate higher injectivity compared to GIN-final and GCN, as they have fewer overlaps between different classes. This suggests that the iceGNN models are able to more accurately distinguish between different samples. The iceGNN models also show higher continuity, as they produce more compacted same-class embeddings than GIN-final and GCN. This indicates that similar samples have more similar embeddings in the iceGNN models, as samples within the same class are expected similar.

V. CONCLUSION

In this paper, we present a theoretical framework to design highly expressive GNNS for general graphs. Based on the framework, we propose two iceGNN variants with fixed transformation function and learnable transformation function, respectively. Moreover, the proposed iceGNN can naturally learn expressive representations for graphs with continuous node attributes. We validate the proposed GNN for graph classification and node classification on multiple benchmark datasets including simple graphs and attributed graphs. The experimental results demonstrate that our model achieves state-of-the-art performances on most of the benchmarks. Future directions include extending the framework to graph with continuous edge attributes.

REFERENCES

[1] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, “Graph convolutional neural networks for web-scale recommender systems,” in ACM SIGKDD. ACM, 2018, pp. 974–983.
[2] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in ICML. JMLR. org, 2017, pp. 1263–1272.
[3] L. Yao, C. Mao, and Y. Luo, “Graph convolutional networks for text classification,” in AAAI, 2019.
[4] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, “Large-scale hierarchical text classification with recursively regularized deep graph-cnn,” in WWW, 2018, pp. 1063–1072.
[5] C. Mao, L. Yao, and Y. Luo, “Medgcn: Medication recommendation and lab test imputation via graph convolutional networks,” Journal of Biomedical Informatics, p. 104000, 2022.
[6] Y. Li, R. Jin, and Y. Luo, “Classifying relations in clinical narratives using segment graph convolutional and recurrent neural networks (seg-gcns),” JAMIA, vol. 26, no. 3, pp. 262–268, 2018.
[7] C. Mao, L. Yao, and Y. Luo, “Imagegcn: Multi-relational image graph convolutional networks for disease identification with chest x-rays,” IEEE Transactions on Medical Imaging, 2022.
[8] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in ICLR, 2019.
[9] C. Mao and Y. Luo, “Improving graph representation learning with distribution preserving,” in 2022 IEEE International Conference on Data Mining (ICDM). IEEE, 2022, pp. 1095–1100.
[10] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in ICLR, 2017.
[11] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in NIPS, 2017, pp. 1024–1034.
[12] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, “An end-to-end deep learning architecture for graph classification,” in AAAI, 2018.
[13] Z. Xinyi and L. Chen, “Capsule graph neural network,” in ICLR, 2019.
[14] Y. G. Wang, M. Li, Z. Ma, G. Montufar, X. Zhuang, and Y. Fan, “Haar graph pooling,” in ICML. PMLR, 2020, pp. 9952–9962.
[15] B. Weisfeiler and A. A. Lehman, “A reduction of a graph to a canonical form and an algebra arising during this reduction,” Nauchno-Technicheskaya Informatsia, vol. 2, no. 9, pp. 12–18, 1968.
[16] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, “Hierarchical graph representation learning with differentiable pooling,” in NeurIPS, 2018, pp. 4800–4810.
[17] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, “Weisfeiler and leman go neural: Higher-order graph neural networks,” in AAAI, 2019.
[18] E. Wagstaff, F. B. Fuchs, M. Engelke, I. Posner, and M. Osborne, “On the limitations of representing functions on sets,” in ICML, 2019.
[19] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, “Deep sets,” in NeurIPS, 2017, pp. 3391–3401.
[20] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” Mathematics of control, signals and systems, vol. 2, no. 4, pp. 303–314, 1989.
[21] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, “Benchmark data sets for graph kernels,” 2016, https://ssl11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets.
[22] M. Niepert, M. Ahmed, and K. Kutzkov, “Learning convolutional neural networks for graphs,” in ICML, 2016, pp. 2014–2023.
[23] H. Maron, H. B. Hamu, N. Shamir, and Y. Lipman, “Invariance and equivariance graph convolutional networks,” in ICLR, 2019.
[24] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, “Provably powerful graph networks,” in NeurIPS, 2019, pp. 2156–2167.
[25] F.-Y. Sun, J. Hoffmann, V. Vermu, and J. Tang, “Infograp: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization,” in ICLR, 2019.
[26] K. Hassani and A. H. Khasahmadi, “Contrastive multi-view representation learning on graphs,” in International Conference on Machine Learning. PMLR, 2020, pp. 4116–4126.
[27] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, “Weisfeiler-lehman graph kernels,” JMLR, vol. 12, no. Sep, pp. 2539–2561, 2011.
[28] P. Yanardag and S. Vishwanathan, “Deep graph kernels,” in ACM SIGKDD. ACM, 2015, pp. 1365–1374.
[29] N. M. Kriege, P.-L. Giscard, and R. C. Wilson, “On valid optimal assignment kernels and applications to graph classification,” in Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, pp. 1623–1631.
[30] M. Togninalli, E. Ghisu, F. Llinares-López, B. A. Rieck, and K. Borgwardt, “Multiscale graph kernels and applications to graph classification,” in Advances in Neural Information Processing Systems 32, pp. 6439–6449, 2019.
[31] R. Kondor and H. Pan, “The multiscale laplacian graph kernel,” in Proceedings of the 30th International Conference on Neural Information Processing Systems, 2016, pp. 2990–2998.
[32] C. Morris, N. M. Kriege, K. Kersting, and P. Mutzel, “Faster kernels for graphs with continuous attributes via hashing,” in 2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 2016, pp. 1095–1100.
[33] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt, “Scalable kernels for graphs with continuous attributes,” in NeurIPS, 2013, pp. 216–224.
[34] F. Orsini, P. Frasconi, and L. De Raedt, “Graph invariant kernels,” in Information Processing Systems 32, 2016, pp. 93–93, 2008.
[35] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, “Simple and deep graph convolutional networks,” in International Conference on Machine Learning. PMLR, 2020, pp. 1725–1735.

Fig. 3: t-SNE visualization of the output embeddings on training data of NCI1 dataset.