Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Haishan Wang

Benchmark of Self-supervised Graph Neural Networks

Master's Thesis
Espoo, July 28, 2022

Supervisor: Prof. Arno Solin, Aalto University
Advisor: Dr. Vikas Verma, Aalto University

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author: Haishan Wang

Title:
Benchmark of Self-supervised Graph Neural Networks

Date:	July 28, 2022	**Pages:**	53
Major:	Computer Science	**Code:**	SCI3044
Supervisor:	Prof. Arno Solin		
Advisor:	Dr. Vikas Verma		

A graph is an abstract data structure with abundant applications, such as social networks, biochemical molecules, and traffic maps. Graph neural networks (GNNs), deep learning tools which adapt to irregular non-Euclidean space, are designed for such graph data with heavy reliance on manual labels. Learning generalizable and reliable representation for unlabeled graph-structured data has become an attractive and trending task in academia because of the promising application scenarios. Recently, numerous SSL-GNN algorithms have been proposed with success on this task. However, the proposed methods are often evaluated with different architecture and evaluation processes on different small-scale datasets, resulting in unreliable model comparisons. To address this problem, this thesis aims to build a benchmark with a unified framework, a standard evaluation process, and replaceable blocks.

In this thesis, a benchmark of SSL-GNNs algorithms is created with the implementation of 9 state-of-art algorithms. These algorithms are compared on this benchmark with consistent settings: shared structure of the GNN encoder, pre-training and fine-tuning scheme, and a unified evaluation protocol. Each model is pre-trained on large-scale datasets: ZINC-15 with two million molecular data and then fine-tuned on eight biophysical downstream datasets for the graph classification task. The experiment results support that two of the nine algorithms outperform others under the benchmark set. Furthermore, the comparison between algorithms also shows the correlation between the pre-training dataset and certain fine-tuning datasets, and the correlation is analyzed by the model mechanisms.

The implemented benchmark and discoveries in this thesis are expected to promote transfer learning on graph representation learning.

Keywords: machine learning, benchmark, graph neural networks, self-supervised learning, pre-training and fine-tuning

Language: English

Acknowledgements

In the acknowledgments, I want to express my gratitude from my heart to the world.

Thanks to my supervisor Arno and advisor Vikas. Arno taught me to write and think as a researcher in many details. Vikas guided me into the world of self-supervised graph representation learning. Both of them helped me with much patience and solid research ability. I cannot complete this thesis and learn so much without them.

Thanks to my parents. They gave birth to me, gave me love and care, and gave me the opportunity to be educated to this day. I love them.

Thanks to AaltoML group. All the group members were so friendly and enthusiastic about research and provided lots of help to me. I must have been influenced a lot positively by the environment here. I have really enjoyed the time with them, and I feel lucky to have the opportunity to spend more time with them.

I am so glad to have more time to get along with them.

Thanks to China and Finland. The social security and cultural edification of these two countries is the background color of my life.

Thanks to Aalto University. I cherish the study experience and campus life here.

Thanks to all my friends. I got so much help and cared from them. Especially during the time I am far away from my hometown, various friends help me to go through so many challenges in my life.

Thanks to myself. I am going to be better.

Espoo, July 28, 2022

Haishan Wang

Symbols and Abbreviations

Symbols

$G = \{V, E, \mathbf{X}\}$	Graph data
V	Set of vertice (nodes)
E	Set of edges (links)
$\mathbf{x}_v$	Node features of v
$\mathbf{e}_{uv}$	Edge features of e_{uv}, the edge connected nodes u and v
$\mathbf{X} = \{\mathbf{x}_v \| v \in V\}$	Matrix of node features
$\mathcal{N}(v)$	Set of neighbors of node v
$\mathbf{A}$	Adjacency matrix
L	Depth of network
$COMBINE^{(k)}$	Combination function of the k-th layer of MPNN, works for message propagation between layers
$AGG^{(k)}, \phi^{(k)}$	Aggregation functions of the k-th layer of MPNN, works for message collection
$READOUT$	Read-out function, maps nodes embeddings to graph embedding
$\mathbf{x}_v^{(k)}$	Node embeddings of node v after the k-th layer of network
$\mathbf{h}_v$	Node embedding of v mapped by GNN
$\mathbf{h}_G$	Graph embedding of G mapped by GNN
$\mathbf{L}^{norm}$	Normalized graph Laplacian
$\mathbf{I}_n$	Identity matrix with size of $n \times n$
Θ	Matrix of parameters
f_θ	GNN encoder of SSL-GNN algorithms
g_ϕ	Pretext decoder of SSL-GNN algorithms
h_ψ	Decoder of downstream task
y	Data label
$\mathcal{D}$	Data distribution

$\mathcal{L}_{ssl}$	Loss function of self-supervised pretext task
$\mathcal{L}_{sl}$	Loss function of supervised downstream task
$\mathcal{T}$	Agumentor function, maps graph to augmented graph
MI	Mutual information estimator
$U(a, b)$	Continuous uniform disribution on interval (a, b)
$\mathcal{D}_{sim}$	Discriminator function, measures the similarity between vectors
$\mathbb{P}$	Distribution
$dist(\cdot, \cdot)$	Distance function
$G_{K-hop}(v)$	Set of K-hop neighobrs of node v
$G_{context}(v, r_1, r_2)$	Context graph of node v between radius r_1 and r_2
$\mathcal{C}$	Hierarchical prototypes tree of GraphLoG algorithm
$\mathbf{z}_G^k$	Hidden state of graph G for k-th prototype
$sim(\cdot, \cdot)$	Similarity function
$\mathcal{L}_{local}, \mathcal{L}_{global}$	The local and global loss functions of GraphLoG algorithm
A/B	The relative complement of set A with respect to set B
$\sigma(\cdot)$	Sigmoid function
t_G	Target component of graph G in generative SSL-GNNs
$\tilde{t}_G$	Recostrcuted target component of graph G in generative SSL-GNNs
l	Label extractor
y_G	Pseudo-label of graph G in predictive SSL-GNNs
$\tilde{y}_G$	Predicted label of graph G in predictive SSL-GNNs
$\mathcal{A}$	SSL-GNN algorithm
S_{pt}	Graph dataset of pre-training
S_{ft}	Graph dataset of fine-tuning
$\theta^{\mathcal{A}}$	Parameter of GNN-encoder pre-trained by SSL-GNN algorithm $\mathcal{A}$
$\theta^{\mathcal{A},S_{ft}}$	Parameter of GNN-encoder pre-trained by SSL-GNN algorithm $\mathcal{A}$ and fine-tuned on dataset S_{ft}
$\Theta_{\mathcal{A}}$	Model hyper-parameter

Abbreviation

CNN	Convolutional Neural Network
GNN	Graph Neural Network

GAE	Graph Autoencoder
SSL	Self-supervised Learning
OGB	Open Graph Benchmark
PyG	Pytorch Geometric
WL	Weisfeiler–Lehman
MPNN	Message Passing Neural Network
GNNConv	Graph Convolution
BatchNorm	Batch Normalization
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network
GAT	Graph Attention Network
PT&FT	Pre-training and Fine-tuning
URL	Unsupervised Representation Learning
JL	Joint Learning
MI	Mutual Information
ML	Machine Learning
IG	InfoGraph
DACL	Domain-Agnostic Contrastive Learning
NT-Xent	Normalized Temperature-scaled Cross Entropy Loss
NCE	Noise-Contrastive Estimation
CP	Context Prediction
GraphCL	Graph Contrastive Learning
JOAO	Joint Augmentation Optimization
GraphLoG	Local-instance and Global-semantic Learning
AM	Attribute Masking
EP	Edge Prediction
GROVER	Graph Representation from Self-supervised Message Passing Transformer
G-Motif	GROVER Motif Prediction
dyMPNs	Dynamic Message Passing Networks
BBBP	Blood–brain Barrier Penetration
Tox21	Toxicology in the 21st Century
Sider	Side Effect Resource
Muv	Maximum Unbiased Validation
ROC	Receiver Operating Characteristic Curve
AUC-ROC	Area under the ROC Curve
ReLU	Rectified Linear Unit
LeakyReLU	Leaky Rectified Linear Unit

Contents

Symbols and Abbreviations **4**

1 Introduction **9**
1.1 Motivation and Contribution 10
1.2 Thesis Structure 11

2 Background **12**
2.1 Attributed Graph 12
2.2 Graph Neural Networks 13
2.2.1 Graph Convolutional Network 15
2.2.2 Graph Attention Network 16
2.2.3 Graph Isomorphism Network 17
2.3 Pretext and Downstream Tasks 17
2.4 Self-supervised Training Schemes 18

3 SSL-GNN Algorithms **20**
3.1 Contrastive Models 20
3.2 Generative Models 27
3.3 Predictive Models 29
3.4 Summary of Methodologies 31

4 Experiments **32**
4.1 Training Scheme and Workflow 32
4.2 Experiment Settings 33
4.3 Experiment Results 35

5 Discussion **39**

6 Conclusion **43**

Bibliography **45**

A Appendix **52**
A.1 Model hyper-parameters list 52
A.2 Datasets information 53

Chapter 1

Introduction

In recent years, neural network algorithms have boosted the development of data science in many areas. This fast development of neural networks is mainly ascribed to the access to huge datasets, advanced computation resources, and well-designed algorithms to extract hidden information from certain data types, such as convolutional neural networks (CNNs) [33] for image domains.

Graph-structured data has drawn increasing attention from the industry due to its representative ability, such as the molecules in biochemistry and citation network in academia. To capture meaningful features from graph data, a number of graph neural networks (GNNs) have been designed in the last decade, including recurrent GNNs [19, 44], convolutional GNNs [32, 56], and graph autoencoders (GAEs) [9, 35]. Motivated by traditional deep learning architectures, different GNNs are created by redefining essential operations on graph domains. For example, the definition of graph convolution in spectral manner [8, 24, 32] and spatial manner [1, 49, 56] allow researchers to build convolutional GNNs. From various GNNs, algorithms following a message passing (or neighborhood aggregation) scheme stand out, outperforming theoretically designed GNN [15]. Until now, GNN algorithms have been successfully applied in many practical tasks, including molecular property research [36], knowledge graph representation [30], and many other applications.

Most early studies of GNN algorithms focus on supervised training tasks with vast amounts of manual labels. Strong reliance on labels brings many problems to GNN models: high labor cost for label collection, poor generalization on domains out of the training set, and sensitivity to adversarial attacks. Aware of these, researchers have combined GNNs with the self-supervised learning (SSL) paradigm, in which models are trained by self-defined objectives without demand on manually annotated labels, to mit-

igate these shortcomings. The main tasks in SSL methods: pretext and downstream tasks, are introduced in Section 2.3.

However, despite the remarkable performance of SSL in many classical areas, including image representation [31] and natural language processing [39], transferring SSL methods directly from Euclidean space to graph domain remains difficult. The main challenges include, but are not limited to, the high dependence between node features, unique and irregular geometry inside the graph, and the gap between optimization objectives of pretext and downstream tasks. Thus, the key idea of self-supervised graph neural networks (SSL-GNNs), is to design pretext tasks consistent with downstream tasks carefully, aiming to capture valuable representations intrinsically from the topological structure and dependent node features of a graph.

1.1 Motivation and Contribution

Motivation Recently, researchers have proposed a number of state-of-art SSL-GNN algorithms [22, 28, 43, 48, 51, 59, 60], with different promising application scenarios. However, most of them only compare their algorithms with classical unsupervised methods, such as graph kernels, to show the advance of the algorithm. Even in comparing advanced SSL algorithms, different model hyper-parameters settings and various GNN structures lead to unconvincing comparison. To better analyze and compare different SSL-GNN models in a global view, a benchmark platform with unified settings is needed.

Benchmarking models is challenging because it requires abundant related knowledge to provide a fair comparison, and the tedious workload of merging different algorithms and datasets onto the same platform with unified sockets. However, benchmarking is beneficial to drive the progress of related research [52]. Indeed, when comparing SSL-GNN models, many factors influence their performance, including dataset size and domain, built-in GNN architectures, and model selection method. Motivated by the success of different benchmarking platforms, such as DomainBed [20] in domain generalization and ImageNet [12] in computer vision, this thesis aims to benchmark SSL-GNN algorithms.

Related work There are several related studies: A famous benchmark dataset platform, Open Graph Benchmark (OGB) [27], collects large-scale and abundant real-world datasets across different domains, and standardizes graph-related algorithms evaluation process with multiple downstream tasks. In 2020, another benchmark framework was built for GNNs on medium-scale

datasets with meaningful findings from many comparison experiments in comprehensive views [15]. Many surveys [37, 53, 55] on SSL-GNN algorithms present various models methodically, and investigate the structural similarities of algorithms with a unified framework. At the same time, different graph deep learning libraries have been built to support GPU acceleration. One of the most popular is Pytorch Geometric (PyG) [16]. The benchmark implementation of this thesis is based on the PyG library.

Main contribution The main contribution of this thesis is to build a shared benchmark of SSL-GNN algorithms for a fixed downstream task: graph classification. This benchmark provides sockets for customizing combinations of algorithms, datasets, and built-in GNN architectures. Therefore, other researchers could contribute their algorithms or datasets to this shared platform. This thesis aims to provide open source benchmarking tools, help researchers with fast model evaluation and allow for reproducible research. It will further bring potential developments in related fields.

In addition, nine latest SSL-GNN algorithms are tested on this benchmark with unified settings for fairness. The comparison experiment provides convincing evidence to show the advantages of specific algorithms. Furthermore, the discussion chapter provides a new view to analyze the correlation of datasets distribution based on the experiment: Successful transfer learning of SSL algorithms means the method mechanism captures the distribution correlation between datasets. For example, the fact that an augmentation-based contrastive algorithm achieves relatively high accuracy on molecular toxicity prediction under SSL manner, implies that certain chemical properties are easily captured by certain graph augmentation.

1.2 Thesis Structure

This thesis is organized in the following structure. Chapter 2 defines notations and concepts related to SSL-GNN algorithms. Chapter 3 introduces three types of SSL-GNN models and corresponding algorithm examples implemented in the benchmark. Chapter 4 presents the selected training scheme, training workflow, experiment settings, and results. Chapter 5 discusses the experiments from two views. Finally, Chapter 6 summarizes the whole thesis and discusses potential future work.

Chapter 2

Background

This chapter introduces the basic notations and concepts related to GNNs, a general framework of GNN inference: the message passing neural network (MPNN), and three popular MPNN-based architectures. After that, the pretext and downstream tasks and categorization of the SSL-GNN algorithms training scheme are described in the following.

2.1 Attributed Graph

In the data science area, meaningful representations of data are essential for researchers to learn valuable patterns, which benefits further applications. A graph, as an abstract data structure, stores the topological relationship between different inner parts of data. Thus, graph-structured data is so ubiquitous across different research areas and daily life. For example, biological molecules could be naturally viewed as graphs: each atom is a node, and the chemical bond between atoms is represented by the edge between corresponding nodes.

Before the introduction of GNNs, formal definitions of graph-related concepts are given as follows.

Definition 1 (Graph) *A graph is a pair $G = (V, E)$, where:*

- $V = \{v_1, \ldots, v_n\}$ *represents the set of nodes (or vertices).*
- $E = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in V\}$ *represents the set of edges (or links).*

A connection between node v_i and v_j in G is denoted by $e_{ij} \in E$. The neighborhood set of node $v_i \in V$ is denoted by $\mathcal{N}(v_i) = \{v_j \mid e_{ij} \in E\}$. In other words, $\mathcal{N}(v_i)$ is the set of nodes connected to v_i. And $|\mathcal{N}(v)|$ denotes degree of node v, which is the number of its neighbors.

Usually, graph data are featured by node or edge attributes. For example, the types of atoms and the energy inside the chemical bond are the node attributes and edge attributes for molecular graphs, respectively. In this thesis, only node features are considered.

Definition 2 (Attributed Graph) *A graph is defined as* $G = (V, E, \mathbf{X})$*, where* $\mathbf{X} \in \mathbb{R}^{n \times d}$ *denotes node feature matrix,* $n = |V|$ *denotes the number of nodes, and* d *denotes the dimension of node features. The topological structure of a graph can be concluded by an adjacency matrix* $\mathbf{A} \in \{0, 1\}^{n \times n}$*, where*

$$\mathbf{A}_{ij} = \begin{cases} 1, & \text{if } v_i \in \mathcal{N}(v_j) \\ 0, & \text{if } v_i \notin \mathcal{N}(v_j). \end{cases}$$

Under many conditions, an attributed graph can also be represented as $G = (\mathbf{A}, \mathbf{X})$*, since adjacency matrix* $\mathbf{A}$ *contains enough information inside the graph structure.*

Traditional deep learning tools, such as CNNs, are difficult to be applied to graph data directly, because the number of nodes in each graph data is not constant, and the number of neighbors for every node is also non-fixed. For example, in the molecules dataset, it is meaningless to require that every molecule havs the same number of nodes. To address this problem, special neural networks designed for graphs are needed.

2.2 Graph Neural Networks

Graph neural networks (GNNs) are neural networks specifically for processing graph-structured data. In theory, GNNs can be viewed as a generalization of convolutional neural networks on non-Euclidean space [7], and also a weak form [56] of the Weisfeiler–Lehman (WL) isomorphism test [14]. In applications, GNNs are popularized for its successful application on molecular structure research in quantum chemistry [17], and widely applied to many domains, for instance, recommender systems [6, 40, 58] and natural language processing [3, 4, 38].

In the development of GNNs, researchers realized the message passing (or neighborhood aggregation) scheme between layers is essential for extracting topological structure information. Message passing neural networks (MPNNs) [17] were proposed, providing a general scheme for GNN layer design. Thereafter, various GNN models [32, 50, 56] with different innovative message passing schemes gained higher computation efficiency on larger graphs.

Definition 3 (Message Passing Neural Network) *Message Passing Neural Network (MPNN) [17] is a general GNN architecture similar to convolutional neural networks, which generates node representations by iterative layer-wise message aggregation and update. In each layer, MPNN aggregates processed features of the neighboring node, then updates node representation with the previous embedding and aggregated information.*

Given a graph $G = (V, E, \mathbf{X})$, the general message passing scheme between k-th and $(k+1)$-th layer of MPNN on node $v \in V$ is formulated as following:

$$\mathbf{x}_v^{(k+1)} = COMBINE^{(k)}\left(\mathbf{x}_v^{(k)}, AGG_{u\in\mathcal{N}(v)}^{(k)}\left(\phi^{(k)}\left(\mathbf{x}_v^{(k)}, \mathbf{x}_u^{(k)}, \mathbf{e}_{uv}\right)\right)\right), \quad (2.1)$$

where $k \in \{1, \ldots, L-1\}$, L denotes the number of network layers, $\mathbf{x}_v^{(k)} \in \mathbb{R}^d$ denotes node features of v in the k-th layer , and $\mathbf{e}_{uv}$ denotes the edge features between node u and v, $COMBINE^{(k)}$ and $\phi^{(k)}$ are differentiable functions, which could be a small neural networks, $AGG^{(k)}$ denotes aggregation function, which is differentiable and invariant to permutation, e.g. mean function.

The aforementioned MPNN layer with message passing scheme is named as GNN Convolution (GNNConv) layer here, because this scheme could be viewed as graph convolution. More relationships between the scheme and convolution will be introduced in Section 2.2.1. In the message passing formula: Equation (2.1), the component functions $\phi^{(k)}, AGG^{(k)}$ define how the GNN architecture aggregates messages for each node from its neighbors, and $COMBINE^{(k)}$ guides how GNNConv layers combine the aggregated message and propagate to the next layer. They decide the model's capability to capture hidden information from graph topology and initial features. Most GNNs differ mainly in their component functions from others.

An example process of an L-layer GNN is listed in Algorithm 1. The node attributes are the initial embeddings for the first layer. In each layer of this GNN, there are a GNNConv layer and a BacthNorm layer. After the convolution of L layers, an extra Readout layer generates the graph representation as the final output.

Readout layer For node-level tasks, each node inside the graph is viewed as a data point, and the node representation $\mathbf{x}_v^{(L)}$ of the final layer can be utilized as the embeddings in the hidden space directly. However, for graph-level tasks, the embeddings of the graph are required. Then the permutation invariant function $READOUT$ serves as the readout layer, which is necessary to obtain the representation h_G of the entire graph:

$$\mathbf{h}_G = READOUT\left(\left\{\mathbf{x}_v^{(L)} | v \in V\right\}\right).$$

Algorithm 1: Graph neural network (example algorithm)

Input : Training graph $G = \{V, E, \mathbf{X} = \{\mathbf{x}_v | v \in V\}\}$, network depth L, message aggregation function $AGG(\phi(\cdot))$, propagation function $COMBINE(\cdot)$, read out function $READOUT(\cdot)$

Output: Graph emebddings $\mathbf{h}_G$

`/* Node embedding initialization */`

for $v \in V$ **do**

  $\mathbf{x}_v^{(0)} \leftarrow \mathbf{x}_v$;

for $k \leftarrow 1$ **to** L **do**

  `/* Layerwise message passing */`

  for $v \in V$ **do**

    $\mathbf{x}_{\mathcal{N}(v)}^{(k)} \leftarrow AGG_{u \in \mathcal{N}(v)}^{(k)} \left(\phi^{(k)}(\mathbf{x}_v^{(k-1)}, \mathbf{x}_u^{(k-1)}, \mathbf{e}_{uv}) \right)$;

    $\mathbf{x}_v^{(k)} \leftarrow COMBINE^{(k)} \left(\mathbf{x}_v^{(k-1)}, \mathbf{x}_{\sqsubseteq}^{(k)} \right)$;

    `/* Batch normalization */`

    $\mathbf{x}_v^{(k)} \leftarrow \mathbf{x}_v^{(k)} / \|\mathbf{x}_v^{(k)}\|$;

$\mathbf{h}_v = \mathbf{x}_v^{(L)}$;

`/* Graph emebedding generated by READOUT function */`

$\mathbf{h}_G = READOUT\left(\{\mathbf{h}_v | v \in V\}\right)$;

The simplest way to define $READOUT$ function is just to sum or average all node embeddings. This thesis concerns the graph-level task. Therefore it focuses on GNN with a readout layer. If there is no specific note, the readout layer is contained at the bottom of GNNs.

2.2.1 Graph Convolutional Network

The graph convolutional network (GCN) [32] is one of the best MPNN-based architectures, which migrates CNN structure to graph domain by Fourier Transform. It utilizes the first-order approximation of spectral graph convolution [23] to design its unique message passing scheme.

Spectral graph convolution For graph $G = (\mathbf{A}, \mathbf{X})$, the normalized graph Laplacian is defined as: $\mathbf{L}^{norm} = \mathbf{I_n} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\mathbf{D}$ is diagonal matrix of graph degrees: $D_{ii} = \sum_j^n A_{ij}$. The graph Fourier transform

and inverse Fourier transform of node feature x_v on graph G are defined as

$$\mathscr{F}(\mathbf{x_v}) = \mathbf{U}^\top \mathbf{x_v}, \quad \text{and} \quad \mathscr{F}^{-1}(\tilde{\mathbf{x}}_v) = \mathbf{U}\tilde{\mathbf{x}}_v,$$

where $\mathbf{U}$ is the eigenvector matrix of $\mathbf{L}^{norm}$ with standard eigenvalue decomposition $\mathbf{L}^{norm} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$. Thus, the graph convolution between parameterized filter α_θ and $\mathbf{x}_v$ is defined through the spectral domain:

$$\alpha_\theta * \mathbf{x}_v = \mathscr{F}^{-1}\left(\mathscr{F}(\alpha_\theta) \odot \mathscr{F}(\mathbf{x}_v)\right) = \mathbf{U}\alpha_\theta \mathbf{U}^{-1}\mathbf{x}_v. \tag{2.2}$$

After the convolution operation on the graph was defined in the spectral style, various spectral-based approaches were designed. To address high computational cost on large graphs and non-continuous filter in this operation, researchers proposed different improved approaches, such as approximation of graph Laplacian [11] and smooth constraint on frequency domain [24].

For practical reasons, GCN simplifies Equation (2.2) by the first-order polynomial approximation and parameter constraints, to reduce the computational cost and address the overfitting problem. The simplified spectral graph convolution between filter α_θ and $\mathbf{x_v}$ becomes

$$\alpha_\theta * \mathbf{x_v} = \Theta(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}})\mathbf{x}_v,$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I_n}$ is a self-loop inserted adjacency matrix, $\tilde{\mathbf{D}}$ is defined similary as before: $\tilde{\mathbf{D}}_{ii} = \sum_j^n \tilde{\mathbf{A}}_{ij}$, Θ denotes parameter matrix of filter.

GCN message passing scheme Given node representations $\mathbf{X}^{(k)}$ in the k-th layer, GCN updates the message of the $(k+1)$-th layer by the following scheme:

$$\mathbf{X}^{(k+1)} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^{(k)}\mathbf{\Theta}.$$

where $\mathbf{\Theta} \in \mathbb{R}^{d\times F}$ denotes filter parameters, F denotes spectral dimension.

2.2.2 Graph Attention Network

The graph attention network (GAT) [50] designs an efficient spatial convolution, by combining the structure of MPNNs and the idea of attention mechanism [2] from sequence modeling to improve generalization on inductive graph task.

GAT utilizes a shared attention mechanism $f_{a,\Theta} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ to calculate the importance e_{vu} of node u to node v in layer k:

$$e_{vu}^{(k)} = f_{a,\Theta}(\mathbf{x}_v^{(k)}, \mathbf{x}_u^{(k)}) = \text{LeakyReLU}\left(\mathbf{a}^\top \left[\mathbf{\Theta}\mathbf{x}_v^{(k)} || \mathbf{\Theta}\mathbf{x}_u^{(k)}\right]\right),$$

where attention $f_{a,\Theta}$ is a simple neural network with parameters $\mathbf{a} \in \mathbb{R}^{2d}, \Theta \in \mathbb{R}^{d' \times d}$ and LeakyReLU nonlinearity, d' is the dimension of output representation, and $||$ is the concatenation operation.

Masked attention is used in GAT for structural information: when assigning weight coefficient to graph convolution, only neighborhood nodes are considered. The normalized convolution weight is defined using softmax function

$$\alpha_{vu}^{(k)} = \frac{\exp(e_{vu}^{(k)})}{\sum_{w \in \mathcal{N}(v)} \exp(e_{vw}^{(k)})}. \tag{2.3}$$

GAT message passing scheme Given the representation $\mathbf{x_v}^{(k)}$ of node $\in V$ in the k-th layer, GAT updates the message of the $(k+1)$-th layer by the following scheme:

$$\mathbf{x}_v^{(k+1)} = \alpha_{vv}^{(k)} \mathbf{\Theta} \mathbf{x}_v^{(k)} + \sum_{u \in \mathcal{N}(v)} \alpha_{vu}^{(k)} \mathbf{\Theta} \mathbf{x}_u^{(k)},$$

where Θ is the model parameter matrix with weights defined in Equation (2.3).

2.2.3 Graph Isomorphism Network

The graph isomorphism network (GIN) is a simple spatial-convolution-based MPNN algorithm, which follows certain conditions. Under these conditions, GNN is proven to achieve similar strength as the WL test by theoretical analysis [56].

GIN message passing scheme Given node representations $\mathbf{X}^{(k)}$ in the k-th layer, GIN updates the message of $(k+1)$-th layer by the following scheme:

$$\mathbf{X}^{(k+1)} = h_{\mathbf{\Theta}} \left((\mathbf{A} + (1+\varepsilon)\mathbf{I}) \cdot \mathbf{X}^k \right)$$

Where ε could be fixed or trainable parameters, $h_{\mathbf{\Theta}}$ denotes a parameterized neural network, e.g. simple multilayer perceptron.

2.3 Pretext and Downstream Tasks

There are two stages in tasks of SSL-GNN: pretext tasks and downstream tasks [29]. Pretext tasks teach models to learn generalized representations in a self-supervised manner, which is beneficial to downstream tasks with processed information. Downstream tasks often refer to supervised graph tasks,

such as node classification or link prediction, which are used for performance evaluation of the whole model.

The main objective of the SSL-GNN algorithm is to design an ingenious pretext task, which could teach the model-related patterns contained in the downstream task as much as possible. To better evaluate different algorithms here, the downstream task is fixed to be graph classification, which is the most significant task in many realistic applications.

2.4 Self-supervised Training Schemes

To train models with unlabeled data, self-supervised learning is necessary. Most systematic research classifies SSL graph learning training schemes into three main types, based on the detailed training logic of parameters, which depends on the relationship between pretext and downstream tasks [37, 53, 55] in the Section 2.3. The three main schemes are pre-training and fine-tuning (PT&FT), unsupervised representation learning (URL), and joint learning (JL).

In this thesis, the SSL-GNN algorithm is standardized by the encoder-decoder framework [37]. The encoder f_θ maps graph data $G = (V, E, \mathbf{X})$ to a compact representation $\mathbf{h}_G$, the decoder g_ϕ utilizes the graph representation $\mathbf{h}_G$ to solve the pretext task. Usually, the f_θ could be MLPs or GNNs, such as the aforementioned GCNs or GINs, and the g_ϕ is the key factor of model performance on downstream tasks.

Pre-training and fine-tuning (PT&FT) In the PT&FT training scheme, the encoder f_θ is first pre-trained with pretext decoder g_ϕ in an SSL manner. Then trained f_θ is fine-tuned together with the downstream decoder h_ψ, in a supervised manner on other datasets. The mathematical formulation of this scheme is presented here:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}_{\text{ssl}}(f_\theta, g_\phi, \mathcal{D}), \tag{2.4}$$

$$\theta^{**}, \psi^* = \arg\min_{\theta,\psi} \mathcal{L}_{\text{sl}}(f_\theta, h_\psi, \mathcal{D}, y), \quad \theta_{initial} = \theta^*, \tag{2.5}$$

where $\mathcal{L}_{\text{ssl}}, \mathcal{L}_{\text{sl}}$ denote loss function of self-supervised pretext task and supervised downstream task, respectively, $\mathcal{D}$ denotes distribution of graph data and y denotes corresponding labels.

Unsupervised representation learning (URL) In the URL training scheme, after the encoder is trained to be f_{θ^*} by the SSL pretext task, the parameters of encoder f_{θ^*} are frozen, then the downstream decoder g_ϕ is trained with labels on the same dataset:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}_{\text{ssl}}(f_\theta, g_\phi, \mathcal{D}),$$
$$\psi^* = \arg\min_{\psi} \mathcal{L}_{\text{sl}}(f_{\theta^*}, h_\psi, \mathcal{D}, y).$$

Joint learning (JL) In the JL training scheme, the encoder and decoders of both tasks f_θ, g_ϕ, h_ψ are trained mutually at the same time:

$$\theta^*, \phi^*, \psi^* = \arg\min_{\theta,\phi} \alpha\mathcal{L}_{\text{ssl}}(f_\theta, g_\phi, \mathcal{D}) + \mathcal{L}_{\text{sl}}(f_{\theta^*}, h_\psi, \mathcal{D}, y),$$

where the parameter α balances the weights between two tasks.

Chapter 3

SSL-GNN Algorithms

The SSL-GNN algorithms aim to train a GNN-encoder with unlabeled data for downstream tasks. Depending on the types of pretext task, they are usually divided into three types: generative, predictive and contrastive [53], with visualization in Figures 3.1, 3.2, and 3.3 respectively. This chapter introduces these three types of models, and examples of state-of-art algorithms implemented in the benchmark.

In some sense, the latter two types could be viewed as contrastive models, if the reconstructed graph in the generative model and self-generated pseudo-labels are viewed as special augmentations of the graph. Thus, contrastive SSL-GNNs are more valued in our benchmark. Especially, a mixup [62] based contrastive SSL-GNN: DACL [51], will be highlighted in this chapter, because it could achieve the same performance even without domain-related expertise.

3.1 Contrastive Models

For the contrast-based method, the central idea is to create meaningful embedding of graphs for the specific downstream task, which could discriminate correlated data pairs from random pairs, by maximizing agreement between multiple augmented graphs. This agreement is often measured by maximizing the mutual information (MI) of augmentation pairs. The main process of contrastive SSL-GNN is visualized in Figure 3.1.

Model description The contrastive SSL-GNN comprises three main modules: data augmentation, contrastive objective, and contrast scale. Given graph data $G = (V, E, \mathbf{X})$, data augmentation transformations $\{\mathcal{T}_i\}_{i=1}^K$ generate multiple views of the graph $\{\mathcal{T}_i(G)\}_{i=1}^K$, then the pretext encoders $\{f_{\theta_i}\}_{i=1}^K$

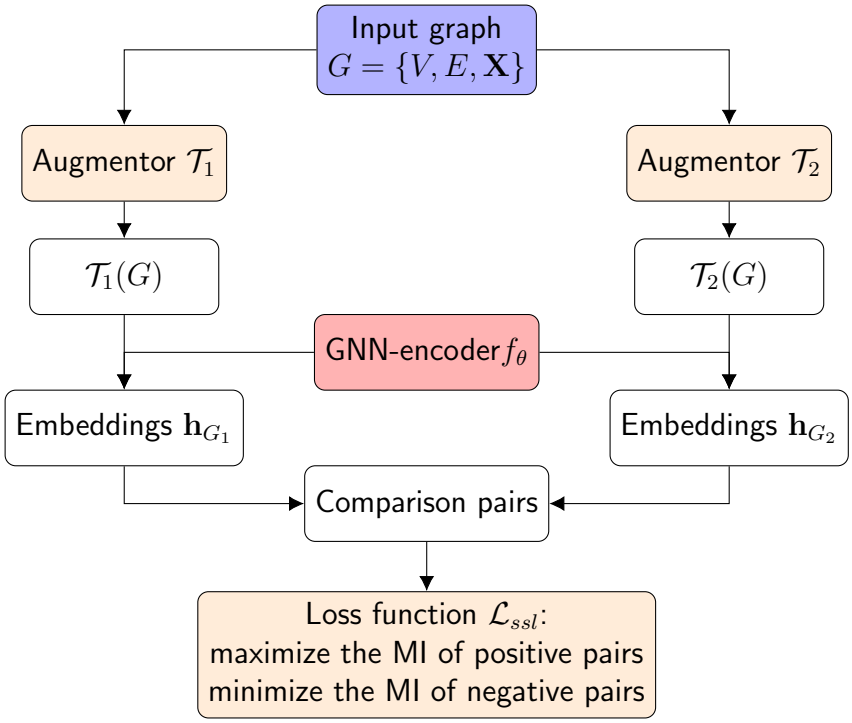


Figure 3.1: Visualization of contrastive SSL-GNNs. Given input graph G, graph augmentors $\mathcal{T}_1, \mathcal{T}_2$ create two views of it. All views $\mathcal{T}_1(G), \mathcal{T}_2(G)$ are mapped to embeddings for comparison pairs by shared GNN-encoder f_θ. Finally, the encoder parameter θ is optimized by an MI-related objective $\mathcal{L}_{ssl}$ based on these comparison pairs.

map these views to representations $\{\mathbf{h}_1, \ldots, \mathbf{h}_K\}$:

$$\mathbf{h}_i = f_{\theta_i}\left(\mathcal{T}_i(G)\right), \quad i = 1, \ldots, K,$$

where the encoders $\{f_{\theta_i}\}_{i=1}^K$ often share the parameters, and usually two augmentations are used: $K = 2$. The pretext task Equation (2.4) can then be reformulated as:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}_{\text{ssl}}(g_\phi\left(f_{\theta_1}\left(\mathcal{T}_1(G)\right), \ldots, f_{\theta_K}\left(\mathcal{T}_K(G)\right)\right)).$$

Here the $\mathcal{L}_{\text{ssl}}$ is the estimator which helps the model maximize the MI between positive pairs and minimize it between negative pairs. Augmentations pair of the same instance are considered to be positive, while the other pairs are negative.

Module: Data augmentation Different augmentation transformations in a model generate multiple views of the graph as an initial stage. The

augmentation processes the features or topological structure of the graph, such as node feature masking or shuffle, edge modification, graph diffusion, random-walk sampling, and some hybrid methods. Each augmented graph is a new graph that shares partial information with the original instance:

$$\tilde{G}_i = (\tilde{\mathbf{A}}_i, \tilde{\mathbf{X}}_i) = \mathcal{T}_i(\mathbf{A}, \mathbf{X}) = \mathcal{T}_i(G).$$

Module: Comparison scale With various pretext decoders g_ϕ, contrastive methods compare augmented views in the same scale (node-level, graph-level, context-level) or across scales. Comparisons on different scales help the model capture different semantic information from the data.

Module: Training objective The contrastive objective is to optimize agreement between different views of data. Given a pair of graph representations $\mathbf{h}_i, \mathbf{h}_j$, the agreement between them is measured by viewing these representations as random variables and calculating the mutual information:

$$MI(\mathbf{h}_i, \mathbf{h}_j) = \mathbb{E}_{p(\mathbf{h}_i)p(\mathbf{h}_j)} \left[\log \left(\frac{p(\mathbf{h}_i, \mathbf{h}_j)}{p(\mathbf{h}_i)p(\mathbf{h}_j)} \right) \right]. \tag{3.1}$$

For practical computation, various estimators based on Equation (3.1) are utilized in application-oriented algorithms. There are many popular estimators for contrastive SSL-GNN training: some of them provide lower-bound estimation of MI, including Jensen–Shannon estimator [41], InfoNCE [21], triplet marginal loss [45]. Some non-bound MI estimations are also used, such as Barlow twins loss [61].

Algorithm: InfoGraph (IG) [48] As a contrastive graph learning algorithm, IG sets the objective, which compares embeddings across the graph-node level. This local-global comparison idea is motivated by a similar idea from the image domain: Deep InfoMax[26]. As early SSL-GNN work, IG utilizes the identity map as augmentation. In other words, there is no special augmentation of data. Thus the effect of IG mainly relies on cross-scale comparison.

For k-th graph $G_k = \{V_k, E_k, \mathbf{X}_k\}$, IG creates embeddings for graph $f_\theta(G_k)$ and nodes $\{f^*_\theta(v_{k,i})\}_{i=1}^{N_k}$ with the same GNN-encoder f_θ, where $V_k = \{v_{k,i}\}_{i=1}^{N_k}$ is the node set of graph G_k, $N_k = |V_k|$ is the size of node-set, and f^*_θ denotes the encoder without the readout layer. Then IG compares data pairs in hidden space:

$$\{f_\theta(G_{k_1}), f^*_\theta(v_{k_2,i})\}, \quad \text{for} \quad i = 1, \ldots, N_{k_2}.$$

The pairs are considered to be positive when it compares the local-global information from the same graph, which means $k_1 = k_2$. Otherwise, IG minimizes their similarity in the objective. The MI objective $\mathcal{L}_{ssl}$ is measured by the Jensen-Shannon MI estimator[41]:

$$\mathcal{L}_{ssl} = \mathbb{E}_{(\mathbf{h},\tilde{\mathbf{h}})\in\mathbb{P}\times\tilde{\mathbb{P}}}\left[\log(1-\mathcal{D}_{sim}(\mathbf{h},\tilde{\mathbf{h}}))\right] - \mathbb{E}_{\mathbf{h},\mathbf{h}'\in\mathbb{P}}\left[\log(\mathcal{D}_{sim}(\mathbf{h},\mathbf{h}'))\right],$$

where $\tilde{\mathbb{P}}$ is the negative distribution. $\mathcal{D}_{sim}$ denotes the discriminator, usually defined as a normalized inner product. In IG, $\mathcal{D}_{sim}$ is the combination of a sigmoid function and another discriminator.

Algorithm: Domain-Agnostic Contrastive Learning (DACL) [51] The DACL is a mixup [62] based contrastive learning algorithm, without reliance on augmentation designed by domain knowledge. Thus it could be applied to many domains, such as images, tables, and graphs. The main trick in DACL is using embeddings of training data to create adaptive noise with linear combination, which is computationally effective.

In the graph augmentation stage, DACL maps data into hidden space by graph encoder and then creates semantically similar examples with mixup noise for comparison. For the k-th graph G_k of the training batch $S = \{G_k\}_{k=1}^N$, two views of it are generated by linear mixup:

$$\mathcal{T}_i(G_k) = \lambda_i f_\theta(G_k) + (1-\lambda_i) f_\theta(G_{anchor,i}), \quad k = 1,\ldots,N,\ i = 1,2 \tag{3.2}$$

where $G_{anchor,1}, G_{anchor,1}$ are randomly sampled from S/G_k, λ_1, λ_2 are sampled from uniform distribution $U(\alpha, 1)$, and $f_\theta(\cdot)$ is the GNN-encoder. Hyperparameter α controls similarity between two augmented views of the same graph and is set to be high for generating correlated pairs.

After augmentation, an extra projection head g_ϕ processes augmented views $\{\mathcal{T}_1(G_k), \mathcal{T}_2(G_k)\}_{k=1}^N$ as pretext decoder, which is proved to be beneficial to contrastive loss definition [10]. Then comparison pairs are created for objective optimization:

$$\{g_\phi(\mathcal{T}_1(G_i)), g_\phi(\mathcal{T}_2(G_j))\}, \quad i,j = 1,2,\ldots,N.$$

Here the pair is positive when the two views come from the same graph, which means $i = j$, otherwise negative.

In the comparison stage, DACL compares pairs at graph-graph level with NT-Xent loss [46] objective, by maximizing the mutual information of positive pairs created in the augmentation stage, then updates the encoder parameters. Here NT-Xent loss is a special version of InfoNCE [21], which

Algorithm 2: Domain-agnostic Contrastive Learning

Input : GNN-encoder f_θ, projection-head g_ϕ, training graph batch $\{G_1, \ldots, G_N\}$, hyperparameters α, τ

Output: Trained parameters θ^*, ϕ^*

for $k \leftarrow 1$ **to** N **do**

  `/* Sample coefficients, data indexes for mixup */`

  $\lambda_1 \sim U(\alpha, 1.0), \lambda_2 \sim U(\alpha, 1.0)$;

  $t_1 \sim \{1, \ldots, N\}/k$, $t_2 \sim \{1, \ldots, N\}/k$;

  `/* Apply encoder */`

  $h_k \leftarrow f_\theta(G_k)$, $h_{t_1} \leftarrow f_\theta(G_{t_1})$, $h_{t_2} \leftarrow f_\theta(G_{t_2})$;

  `/* Create positive pairs by lienar mixup */`

  $\tilde{h}_{k,1} \leftarrow \lambda_1 h_k + (1 - \lambda_1) h_{t_1}$;

  $\tilde{h}_{k,2} \leftarrow \lambda_1 h_k + (1 - \lambda_2) h_{t_2}$;

  `/* Apply projection-head */`

  $z_{2k-1} \leftarrow g_\phi(\tilde{h}_{k,1}), z_{2k} \leftarrow g_\phi(\tilde{h}_{k,2})$;

for $i \leftarrow 1$ **to** $2N$ **do**

  `/* Calculate pairwise discrimination */`

  for $j \leftarrow 1$ **to** $2N$ **do**

    $\mathcal{D}_{sim}(i, j) \leftarrow z_i^T z_j / (\tau \|z_i\| \|z_j\|)$;

  $D_i \leftarrow \sum_{j=1}^{2N} \exp(\mathcal{D}_{sim}(i, j))$;

Define $MI_{NCE}(\mathbf{z}_i, \mathbf{z}_j) \leftarrow -\log \frac{\exp(\mathcal{D}_{sim}(i,j))}{D_i - \exp(\mathcal{D}_{sim}(i,i))}$;

$Loss \leftarrow \frac{1}{2N} \sum_{k=1}^{N} (MI_{NCE}(\mathbf{z}_{2k-1}, \mathbf{z}_{2k}) + (MI_{NCE}(\mathbf{z}_{2k}, \mathbf{z}_{2k-1}))$;

`/* Parameters update by gradient descent */`

$\theta^*, \phi^* \leftarrow \arg\min_{\theta,\phi} Loss$;

estimates the lower bound of MI:

$$MI_{NCE}(\mathbf{x}, \mathbf{y}) = -\log \frac{\exp(\mathcal{D}_{sim}(\mathbf{x}, \mathbf{y}))}{\sum_{k=1, k \neq i}^{N} \exp(\mathcal{D}_{sim}(\mathbf{x}, \mathbf{y}))}.$$

where $\mathcal{D}_{sim}$ denotes discriminator, which measures the similarity between vectors

$$\mathcal{D}_{sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^\top \mathbf{y}}{\tau \|\mathbf{x}\| \|\mathbf{y}\|},$$

and τ controls the discriminator temperature. For a clearer presentation of the contrastive algorithm process and structure, the detailed training pseudo-code of DACL on batch data is displayed in Algorithm 2 as an example.

Algorithm: Context Prediction (CP) [28] The CP proposes a creative way to design a context scale for comparison, and trains the model to predict the structural context by local graph information. At first, K-hop neighborhood $G_{K-hop}(v)$ of node v in graph $G = (V, E, \mathbf{X})$ is defined to be all nodes which are accessible for v in K hops:

$$G_{K-hop}(v) = \{u \in V | dist_G(u, v) \leq K\}, \tag{3.3}$$

where $dist_G(u, v)$ is the shortest path in graph G. Then the context graph with radius r_1, r_2 is defined as

$$G_{context}(v, r_1, r_2) = G_{r_2-hop}(v)/G_{r_1-hop}(v), \quad r_1 < r_2, \tag{3.4}$$

which is a ring between two hyper-spheres in the graph topological space.

For node v, the main GNN encoder maps it to vector embeddings. At the same time, all nodes in context graph $G_{context}(v, r_1, r_2)$ are mapped to the same space by another context-GNN, and the mean of their embeddings is calculated as context embeddings. Then CP considers node-context embedding pairs from the same node to be positive, otherwise to be negative. This comparison scheme in the model assumes high similarity between graph nodes and the corresponding context in hidden space mapped by GNN-encoder, and learns discriminative graph representation.

Algorithm: Graph Contrastive Learning (GraphCL) [59] GraphCL is the classical contrastive framework that systematically studies the effect of augmentations in self-supervised graph representation learning. GraphCL proposes four augmentors $\{\mathcal{T}_i\}_{i=1}^4$ on graph data: discarding a certain rate of nodes, randomly adding or abandoning some edges, masking attributes partially, and building subgraph. With GNN-based encoder f_θ and InfoNCE objective $\mathcal{L}_{ssl}$, GraphCL could analyze the performance of different augmentation pairs on different datasets, and then find the best dataset-specific augmentation combination. This empirical rule helps further study the correlation highlighted by augmentation in the data domain.

Algorithm: Joint Augmentation Optimization (JOAO) [60] JOAO automatically selects the best augmentation pair for contrastive learning on graph data, by an adversarial structure. It could be viewed as the improved version of GraphCL, addressing the problem that manual selection of augmentation requires a tedious validation process on huge amounts of data or empirical rules for the specific domain.

JOAO builds its adversarial structure with two reverse sides. Firstly, it assumes the best data augmentation pair $\mathcal{T}_1, \mathcal{T}_2$ could maximize the loss

function $\mathcal{L}_{ssl}(\mathcal{T}_1, \mathcal{T}_2, f_\theta, \{G_i\}_{i=1}^N)$ corresponding to current encoder f_θ. At the same time, the loss function is expected to be minimized when the augmentations are fixed. With these assumptions, the contrastive learning problem is reformulated to be a minimax optimization problem:

$$\min_{\theta} \max_{\mathbb{P}(\mathcal{T}_1,\mathcal{T}_2)} \left(\mathcal{L}_{ssl}(\mathcal{T}_1, \mathcal{T}_2, f_\theta, \{G_i\}_{i=1}^N) - \frac{\gamma}{2} dist(\mathbb{P}(\mathcal{T}_1, \mathcal{T}_2), \mathbb{P}_{prior}) \right),$$

where $\{G_i\}_{i=1}^N$ is the training batch, $\mathbb{P}(\mathcal{T}_1, \mathcal{T}_2)$ denotes joint distribution of augmentation pair $\mathcal{T}_1, \mathcal{T}_2$, and $\mathbb{P}_{prior}$ denotes the prior joint distribution of them, $dist(\cdot, \cdot) : \mathbb{P} \times \mathbb{P} \to \mathbb{R}^+$ measures the distribution distance, γ adjusts weight on confidence of prior augmentation distribution, here $\mathcal{L}_{ssl}$ is the InfoNCE objective.

Algorithm: Local-instance and Global-semantic Learning (GraphLoG) [**57**] GraphLoG is a complicated SSL algorithm, which merges the contrast learning, hierarchical clustering, and expectation maximization (EM) into representation learning on the graph. There are two parts of the objective function in GraphLoG: local and global loss.

The local instance information is learned contrastively. The algorithm augments graph data by attributes masking, maps different views of the graph to embeddings by GNN encoder f_θ, the local objective $\mathcal{L}_{local}$ is to maximize the MI of positive pairs on graph-graph and context-context scale in the representation space. Here the context in GraphLoG is the K-hop neighborhood subgraph, which is defined in Equation (3.3).

The global semantic is structured by hierarchical prototypes $\mathcal{C}$, a L-layer tree. For node sets $\mathbf{C}^{(l)}$ in l-th layer of tree $\mathcal{C}$, the children set of node $c \in \mathbf{C}^{(l)}$ is denoted by $child(c) \subset \mathbf{C}^{(l+1)}$. Assume each graph G from the training set S_{train} owns a series of hidden state $\{\mathbf{z}_G^1, \ldots, \mathbf{z}_G^L\}$ sampled from each layer,

$$\mathbf{z}_G^1 \in \mathbf{C}^{(1)}, \quad \mathbf{z}_G^l \in \mathbf{C}^{(l)} \cap child(\mathbf{z}_G^{l-1}), \ \forall l = 2, \ldots, L.$$

Then the global loss $\mathcal{L}_{global}$ is to maximize the similarity among graph embeddings and the hidden states

$$\sum_{l=1}^{L} sim(f_\theta(G), \mathbf{z}_G^l) + \sum_{l=1}^{L-1} sim(\mathbf{z}_G^l, \mathbf{z}_G^{l+1}),$$

where the $sim(\cdot, \cdot)$ is the similarity function.

With viewing $\{\theta, \mathcal{C}\}$ as parameters, $\mathbf{Z} = \{\mathbf{z}_G^1, \ldots, \mathbf{z}_G^L\}_{G \in S_{train}}$ as hidden variables, the algorithm tries to maximize the joint likelihood $p(S_{train}, \mathbf{Z}|\theta, \mathbf{C})$

with objective $\mathcal{L}_{ssl} = (\mathcal{L}_{local} + \mathcal{L}_{global})$ by Expectation–maximization algorithm.

GraphLoG combines so many powerful machine learning tools with GNN for self-supervised learning, so it costs more computation resources than most other SSL-GNN algorithms. Therefore, when it achieves the same performance as other algorithms, others are much preferred with shorter training time.

3.2 Generative Models

Generative models focus more on strengthening the connection between hidden space mapped by the encoder and the space of the target component. The main process of generative SSL-GNNs is visualized in Figure 3.2, with similar structure of autoencoder [25].

Model description The main modules inside this structure are component extractor, decoder g_ϕ, and objective loss $\mathcal{L}_{ssl}$. Given input data $G = (V, E, \mathbf{X})$, certain parts of G is extracted directly as the target component t_G for generation, the decoder g_ϕ reconstructs the component from graph embeddings mapped by the GNN-encoder f_θ:

$$\tilde{t}_G = g_\phi \left(f_\theta(G) \right).$$

The objective $\mathcal{L}_{ssl}$ aims to minimize the distance between the target components and the corresponding reconstructed version:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}_{\text{ssl}}(g_\phi \left(f_\theta(G) \right), t_G).$$

Module: component extractor The main difference between generative SSL graph learning is the selected target component. The common components extracted for reconstruction are the small subset of graph nodes/edges attributes or the adjacency matrix, which require encoder learning local connections and global structure, respectively. After specifying the reconstructed target, encoder f_θ should accept the input graph without target information for meaningful learning. For attributes-reconstruction conditions, features are partially modified by masked or noising. When reconstructing the adjacency matrix, a certain ratio of the edges is randomly perturbed for noised topology.

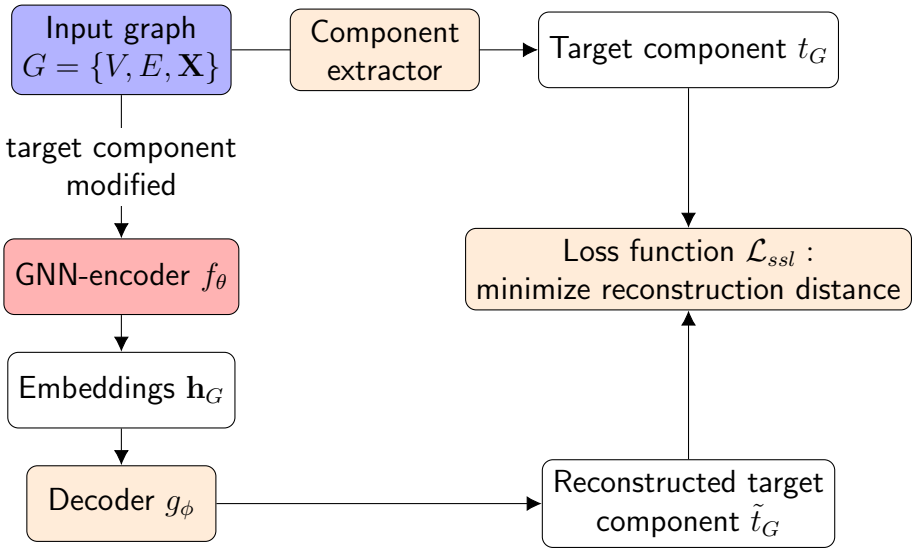


Figure 3.2: Visualization of generative SSL-GNNs. Given input data G, a certain part t_G of G is selected for reconstruction by a component extractor. The slightly modified version of t_G is mapped to embeddings $\mathbf{h}_G$ by the GNN-encoder, then the decoder g_ϕ reconstruct the target component $\tilde{t}_G$ from $\mathbf{h}_G$. Objective $\mathcal{L}_{ssl}$ aims to minimize the distance between t_G and $\tilde{t}_G$.

Modules: decoder g_ϕ and objective $\mathcal{L}_{ssl}$ Usually, the decoder g_ϕ is simply designed to avoid absorbing too much information from data, which diminishes the learning of encoder f_θ. The loss function $\mathcal{L}_{ssl}$ is often the Euclidean distance between reconstructed vectors with original ones.

Algorithm: Attribute Masking (AM) [28] The AM algorithm trained GNN-encoder by reconstructing node/edge attributes. Graphs with masked attributes are mapped to vectors by encoder f_θ. Then the target attributes are predicted by a linear classifier g_ϕ based on corresponding embeddings.

Algorithm: Edge Prediction (EP) [22] The EP algorithm aims to reconstruct adjacency matrix from node embeddings. It assumes that neighbour nodes of the same graph $G = (V, E, \mathbf{X})$ are closer in hidden space, which means high $\sigma(\mathbf{h}_u^\top \mathbf{h}_v)$ in condition that $u \in \mathcal{N}(v)$, where $\sigma(\cdot)$ is the sigmoid function. The objective function is designed as

$$\mathcal{L}_{ssl} = -\sum_{(u,v)\in E} \log(\sigma(\mathbf{h}_u^\top \mathbf{h}_v)) - \alpha \mathbb{E}_{(u,v)\sim \mathbb{P}_n} \log(\sigma(-\mathbf{h}_u^\top \mathbf{h}_v)),$$

where the $\mathbb{P}_n$ denotes the distribution of the negative edges, it is the uniform distribution on $(V \times V)/E$, the set of all possible edges not belonging to E. α scales the loss from negative pair samples.

3.3 Predictive Models

Predictive models generate informative labels instead of human annotation based on certain properties of unlabeled data, then handle the supervised task on data with these pseudo-labels as shown in Figure 3.3. Unlike the contrastive model, predictive models focus on graph representations containing intra-data information, not discriminating between different data.

Model description There are three main modules in predictive SSL-GNN: label extractor l, label classifier g_ϕ, and the obejctive function $\mathcal{L}_{ssl}$. Given input graph $G = (V, E, \mathbf{X})$, the label extractor l generates pseudo-label y_G for G. At the same time, the embeddings $\mathbf{h}_G = f_\theta(G)$ mapped by GNN encoder f_θ is classified to be label $\tilde{y}_G$:

$$\begin{cases} y_G = l(G), \\ \tilde{y}_G = g_\phi(f_\theta(G)). \end{cases}$$

The objective $\mathcal{L}_{ssl}$ maximizes the prediction accuracy:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}_{\text{ssl}}(g_\phi\left(f_\theta(G)\right), l(G)).$$

Module: label extractor l The pseudo-label is generated based on the graph information. For better performance under SSL manner, the label extraction should be reliable with low collection costs. There are two types of label extractor l. The first type is to capture geometrical data from graph structure, such as node degree, the shortest path between two nodes, and node clustering by other unsupervised algorithms. The other class is to utilize domain-related knowledge, such as statistical information on functional groups.

Module: classifier g_ϕ and objective $\mathcal{L}_{ssl}$ The g_ϕ is a simple neural network to adjust the dimension of data embeddings to label size. To measure the accuracy, the $\mathcal{L}_{ssl}$ is often defined as cross-entropy.

$$\mathcal{L}_{ssl} = \frac{1}{N_l} \sum_{i=1}^{N_l} \left(-y_i \log \tilde{y}_i - (1 - y_i) \log(1 - \tilde{y}_i)\right),$$

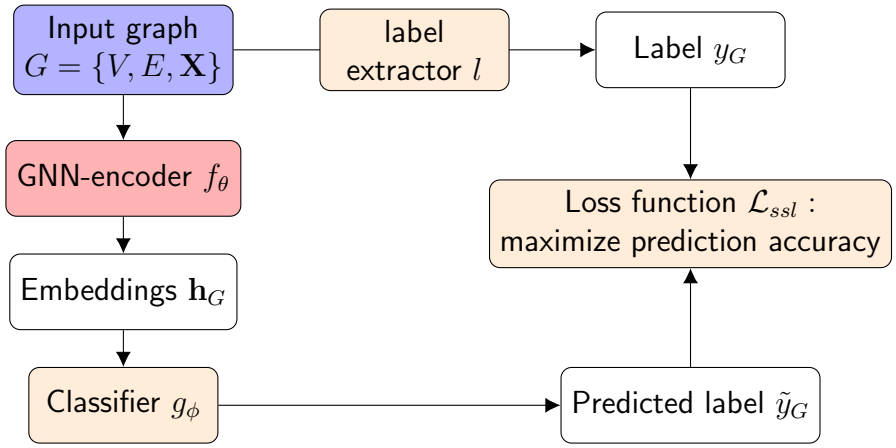


Figure 3.3: Visualization of predictive SSL-GNNs. Given input graph G, label extractor creates label y_G for it. The GNN-encoder f_θ maps G into embeddings $\mathbf{h}_G$, and the classifier g_ϕ predicts the corresponding label $\tilde{y}_G$ from $\mathbf{h}_G$. Finally, the objective function $\mathcal{L}_{ssl}$ aims to maximize the prediction accuracy.

where N_l is the label numbers, $y = (y_1, \ldots, y_{N_l}) \in \{0, 1\}^{N_l}$ is the binary multi-label, $\tilde{y}$ is the predicted label.

Algorithm: GROVER Motif Prediction (G-Motif) [43] The G-Motif algorithm combines the MPNN structure and transformer to design a predictive SSL-GNN framework: GROVER. In this framework, three dynamic message passing networks (dyMPNs) are built to generate three different embeddings of the input graph, which are used as queries, keys, and values for the following multi-head attention layer. Here the dyMPNs are GNNs with a dynamic number of layers, which is drawn from specific distribution during training. This dynamic scheme improves the adaptation of the GNN-based transformer on different datasets. For long-term memory, the input data is concatenated with the embeddings after the attention layer by a long-range residual. Finally, a feed-forward network maps the concatenated vectors to the node embeddings.

In G-Motif, the pseudo-label y_G is selected to be molecular motifs, the recurrent sub-graph structures. These molecular descriptors are obtained by the professional cheminformatics software, RDKit [34]. With the abundant domain knowledge, the unlabeled data could be trained in a supervised manner.

However, for a meaningful comparison among different SSL-GNN algorithms, the advanced GROVER GNN framework with attention and small

dyMPNs is not utilized. All SSL-GNN algorithms share the same structure of GNN, and details are introduced in Section 4.2

3.4 Summary of Methodologies

In this chapter, three branches of SSL-GNNs were introduced with corresponding algorithm examples, which are implemented in the thesis benchmark experiment. The taxonomy of SSL-GNN discussed here is just instructive and popular among many researchers [53], but it does not always work. For example, some hybrid algorithms are difficult to be classified because of their multi-pretext tasks [42]. Sometimes, the predictive models are also called auxiliary property-based SSL-GNNs [37].

The GNN-encoder pre-trained by different types of SSL-GNNs embeds graph data into different hidden spaces. The contrastive models highlight the discrimination between data points, generative models focus reconstruction of certain parts of the original graph, and the predictive models rely on the pseudo labels extractor, which captures the topological or domain-expertise information. When selecting the algorithms, the essential part is to consider whether the algorithm could bridge the connections between pretext and downstream tasks for success.

Chapter 4

Experiments

This chapter presents the detailed workflow of pre-training and fine-tuning scheme, and the settings and results of experiments on the benchmark built in this thesis.

4.1 Training Scheme and Workflow

Training scheme: pre-training & fine-tuning In the experiment, all algorithms are trained in the pre-training and fine-tuning scheme for the following three reasons.

Firstly, motivated by successful transfer learning on natural language processing [13], many researchers extend the pre-training and fine-tuning scheme to graph domain with increasing interest, and gain models outperforming the original models without pre-training.

Secondly, this scheme fits the application requirements more than other aforementioned schemes in Section 2.4. Usually, in real-world scenarios, the pre-training graph datasets could be biological proteins, chemical molecules, or social networks, which are huge. It leads to high computation costs under the JL scheme. And URL requires huge amounts of labels on datasets, and it brings new challenges to manual annotation.

Thirdly, the pre-training task requires careful design mitigates the degree of negative transfer on new domains [28], which is caused by the harmful knowledge learned in the previous domain. Therefore, high performance under the PT&FT scheme provides a convincing comparison between different SSL-GNNs.

Training workflow Each SSL-GNN algorithm $\mathcal{A}$ contains a GNN encoder $f_\theta^{\mathcal{A}}$ and a pretext decoder $g_\phi^{\mathcal{A}}$, for discriminatory comparison objective $\mathcal{L}_{ssl}$.

All the algorithms share the structure of the GNN encoder, so here the algorithm-related superscript of encoder f_θ is omitted.

In the pre-training stage, given the pre-train data $S_{pt} = \{G_i\}_{i=1}^N$, the randomly initialized GNN-encoder f_θ is trained by the algorithm $\mathcal{A}$ in a self-supervised way:

$$\theta^{\mathcal{A}}, \phi^* = \arg\min_{\theta,\phi} \mathcal{L}_{ssl}(\{G_i\}_{i=1}^N, f_\theta, g_\phi^{\mathcal{A}}), \quad \theta_{init} = \theta_{random}.$$

In the fine-tuning stage, the trained encoder $f_{\theta^{\mathcal{A}}}$ is re-tuned with an extra linear layer h_ψ for downstream tasks. In the experiments, the downstream tasks are all binary graph classification, the simplest task. Given the labeled fine-tuning data $S_{ft} = \{\tilde{G}_i, \tilde{y}_i\}_{i=1}^M$, the model is optimized in the supervised way:

$$\theta^{\mathcal{A},S_{ft}}, \psi^{\mathcal{A},S_{ft}} = \arg\min_{\theta,\psi} \mathcal{L}_{sl}\left(\left\{h_\psi\left(f_\theta(\tilde{G}_i)\right), \tilde{y}_i\right\}_{i=1}^M\right), \quad \theta_{init} = \theta^{\mathcal{A}}.$$

Where M is the size of S_{ft} , $\mathcal{L}_{sl}$ denotes objective function which measures the prediction accuracy.

Finally, the performance of a SSL-GNN algorithm $\mathcal{A}$ depends on the downstream task dataset S_{ft}, and it is evaluated by the final tuned $f_{\theta^{\mathcal{A},S_{ft}}}$ and assisted layer $h_{\psi^{\mathcal{A},S_{ft}}}$:

$$score(\mathcal{A}, S_{ft}) = AUC\text{-}ROC\left(\left\{h_{\psi^{\mathcal{A},S_{ft}}}\left(f_{\theta^{\mathcal{A},S_{ft}}}(\tilde{G}_i)\right), \tilde{y}_i\right\}_{i=1}^M\right) \tag{4.1}$$

4.2 Experiment Settings

In this section, the settings of the benchmark experiment are listed in detail.

Datasets S_{pt} for pre-training and S_{ft} for fine-tuning For the pre-training stage, a 2,000,000 unlabeled molecules dataset is chosen from a publicly accessible chemistry dataset (ZINC-15 [47]). For the fine-tuning stage, eight binary labels datasets, in physiological and biophysical domains, are chosen from MoleculeNet [54] for model evaluation on downstream graph-classification tasks: BBBP, Tox21, ToxCast, SIDER, ClinTox, MUV, HIV, BACE. They are popular in recent SSL graph learning [28, 59, 60]. The downstream datasets are all related to molecular properties, such as molecular toxicity, penetration ability over the blood-brain barrier, and inhibition strength on virus replication. The statistical information of them is included in Appendix A.2.

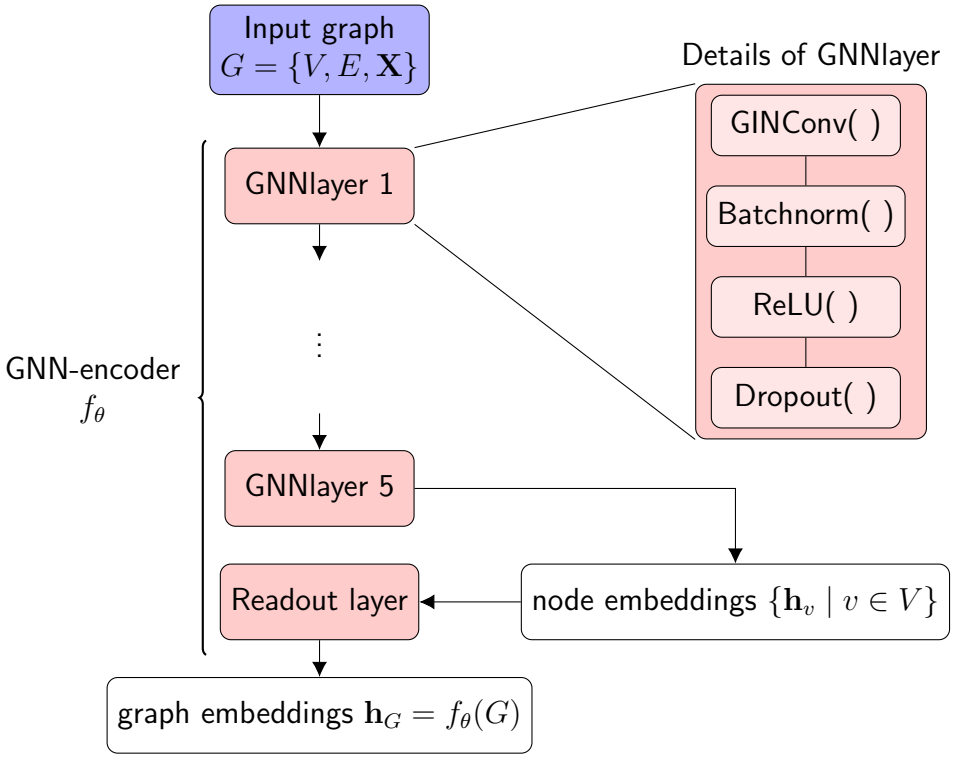


Figure 4.1: Structure of GNN encoder f_θ. The input graph G is processed by five GNNlayers to be the node embeddings $\{\mathbf{h}_v \mid v \in V\}$, then the Readout layer maps these embeddings to graph representation $\mathcal{G}$. Each GNNlayer contains four small blocks. All GNNlayers and the Readout layer make up the GNN-encoder f_θ

Algorithms $\mathcal{A}$ In the benchmark, 9 SSL-GNN algorithms are implemented for pre-training and comparison: AM [28], CP [28], DACL [51], EP [22], GraphCL [59], GraphLoG [57], IG [48], JOAO [60], G-Motif [43]. Details of these algorithms have been given in Chapter 3.

Model hyper-parameters $\Theta_\mathcal{A}$ For each algorithm $\mathcal{A}$, five model hyper-parameter $\Theta_\mathcal{A}$ configurations are selected for parameter search. The selection is based on the algorithm mechanism. And the result of the model with the best hyper-parameter represents the performance of an algorithm. Details of hyper-parameter setting are listed in Appendix A.1.

GNN encoder f_θ The structure of the GNN-encoder f_θ are shared by all algorithms. The shared structure is visualized as Figure 4.1. The GNN is a 5-layer MPNN with the same message passing scheme for each layer. It maps

graph data into a 300-dimensional latent space. The message passing scheme tested in the experiment is the GIN mentioned in Section 2.2.3, whose expressive ability is theoretically guaranteed [56]. The drop-out layers are only activated in the fine-tuning stage. The read-out function of GNN averages all node embeddings as the graph embedding.

Single task The experiment with a certain SSL-GNN algorithm $\mathcal{A}$, specific model hyper-parameter $\Theta_{\mathcal{A}}$, and specific fine-tuning dataset S_{ft}, is labelled as a single task $(\mathcal{A}, \Theta_{\mathcal{A}}, S_{ft})$ here.

In the pretext task, a graph encoder f_θ is initialized with random parameters by Xavier initialization [18], and trained by an SSL-GNN algorithm $\mathcal{A}$ for 100 epochs to be $f_{\theta^{\mathcal{A}}}$. In the downstream task, each specific downstream dataset is split into train-set, validation-set, and test-set, according to Bemis-Murcko scaffold representation [5], which contains the abstract topological structure and atomic features of molecules. The encoder $f_{\theta^{\mathcal{A}}}$ connected with an extra single-layer linear neural network h_ψ in the bottom is trained for 100 epochs on train-set. The same protocol of early stop condition is applied to all downstream classification tasks: the accuracy on test-set of specific epoch, at which the highest accuracy validation-set located, is reported to be the evaluation score of this single task: $score(\mathcal{A}, \Theta_{\mathcal{A}}, S_{ft})$. The calculation process is introduced in Equation (4.1). For both pre-training and fine-tuning stages, model parameters are updated by the Adam optimizer with a learning rate of 0.001.

4.3 Experiment Results

The experiment result is visualized in Table 4.1. Each single task (combination of pre-training algorithm, algorithm configuration, and fine-tuning dataset) is repeated 10 times with different random seeds.

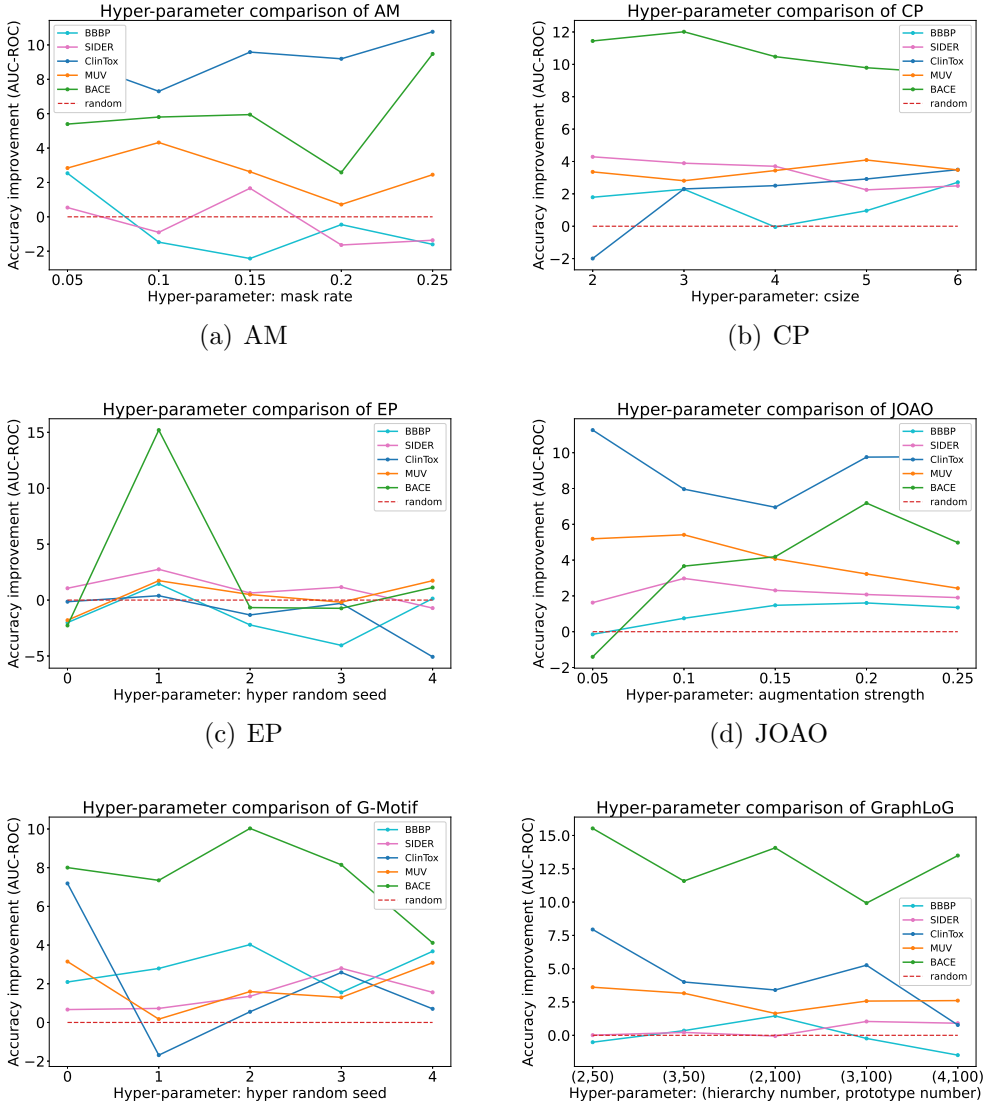
Benchmark experiment In this experiment, GIN-based GNN encoders are pre-trained by 9 SSL-GNN algorithms $\mathcal{A}$ on the ZINC-15 dataset, then fine-tuned on 8 datasets S_{ft}. For each combination of algorithm and dataset $(\mathcal{A}, S_{ft})$, the best result searched from 5 model hyper-parameters $\Theta_{\mathcal{A}}$ is reported. For each configuration $(\mathcal{A}, \Theta_{\mathcal{A}}, S_{ft})$, the result is the average on 10 random seeds. In other words, there are a total of $9 \times 8 \times 5 \times 10 = 3600$ single tasks in this benchmark experiment. Table 4.1 shows the experiment result. The details of datasets and algorithms are explained in Section 4.2. The row index indicates the algorithms applied to pre-train the GNN-encoder on the ZINC-15 dataset. Especially, the 'random' provides a baseline algorithm

Table 4.1: Benchmark experiment result with pre-training and fine-tuning scheme. The value records the fine-tuning accuracy, measured by the mean (and 100 times scaled standard derivation) of AUC-ROC over 10 random seeds. **Bold** number represents the best result on each dataset.

Algorithm $\mathcal{A}$	Fine-tuning datasets S_{ft}								Avg
	BBBP	Tox21	ToxCast	SIDER	ClinTox	MUV	HIV	BACE	
random	66.7(2.4)	75.0(0.5)	61.5(0.8)	58.9(1.1)	61.6(3.9)	72.8(1.7)	75.4(1.0)	69.5(3.1)	67.7
CP	69.4(1.0)	75.5(0.4)	**63.8(0.3)**	**63.2(0.4)**	65.1(2.7)	76.9(1.6)	**78.3(1.1)**	81.5(0.8)	71.7
AM	69.2(2.8)	**76.3(0.3)**	63.5(0.3)	60.5(0.5)	72.4(4.6)	77.1(0.9)	77.5(1.0)	79.0(5.4)	**71.9**
DACL	65.6(2.2)	71.6(0.9)	61.1(0.5)	58.7(0.9)	64.2(2.4)	68.3(2.6)	72.7(1.4)	75.5(2.2)	67.2
EP	68.1(1.9)	75.3(0.5)	62.7(1.1)	61.6(0.5)	62.0(2.6)	74.5(2.7)	75.8(0.6)	84.7(0.7)	70.6
GraphCL	68.4(1.8)	75.9(0.4)	63.5(0.3)	61.5(0.5)	71.1(3.0)	76.0(2.5)	76.7(0.6)	77.7(2.8)	71.3
GraphLoG	68.1(1.1)	74.0(0.6)	63.3(0.3)	59.9(0.9)	69.6(1.6)	76.4(1.4)	75.6(1.0)	**85.0(0.6)**	71.5
IG	69.4(1.6)	75.5(0.4)	63.8(0.3)	59.7(1.0)	70.4(3.6)	77.4(2.2)	75.8(1.0)	77.5(1.2)	71.2
JOAO	68.3(1.2)	75.1(0.3)	63.2(0.4)	61.9(0.6)	**72.9(2.6)**	**78.2(1.3)**	76.8(1.0)	76.7(1.3)	71.6
G-Motif	**70.7(0.5)**	75.1(0.3)	63.5(0.2)	61.7(0.6)	68.8(2.8)	75.9(2.4)	77.2(0.7)	79.5(1.4)	71.6

without the pre-training stage. The column index indicates datasets that serve for downstream classification tasks.

Algorithms hyper-parameter comparison In Table 4.1, only the result of an algorithm with the best hyper-parameter is reported. To analyze the effect of model hyper-parameters, the comparison of 8 algorithms are plotted in Figures 4.2 and 4.3. Only the most discriminative downstream datasets S_{ft} are selected for the comparison. Here the discriminative dataset refers to the datasets, on which the best algorithm (its accuracy is **boldly** noted in Table 4.1) can achieve 3% better performance than the random baseline. Different sub-figures displays the parameter comparison of different algorithms $\mathcal{A}$, the x-axis denotes the value of model hyper-parameters, and the y-axis denotes the accuracy difference between the $\mathcal{A}$ and random baseline for a clearer comparison. In each subfigure, the accuracy of the random baseline is plotted to be a red dotted line.

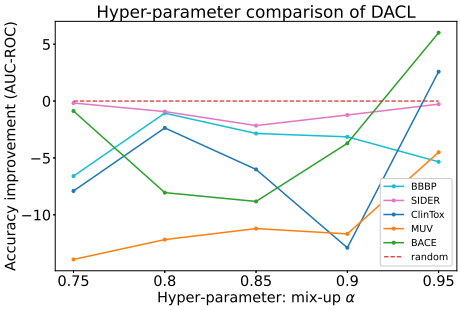


(a) AM (b) CP

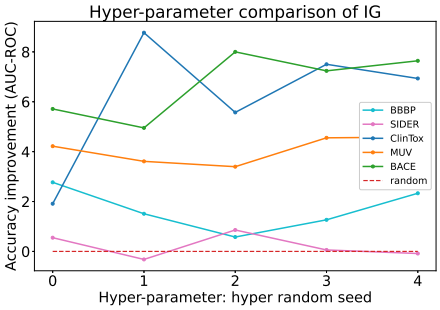
(c) EP (d) JOAO

(e) G-Motif (f) GraphLoG

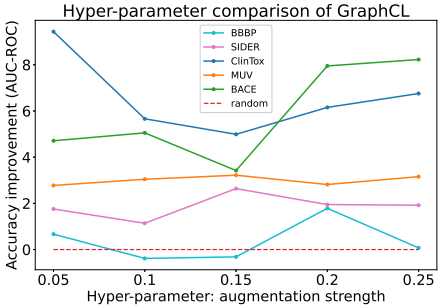
Figure 4.2: Algorithms hyper-parameters comparison of AM, CP, EP, JOAO, G-Motif, and GraphLoG on datasets BBBP, SIDER, ClinTox, MUV, and BACE. The performance of AM and JOAO on ClinTox are always better than other downstream datasets. And the performance of CP, G-Motif, and GraphLoG on BACE are always better than other downstream datasets.



(a) DACL



(b) IG



(c) GraphCL

Figure 4.3: Algorithms hyper-parameters comparison of DACL, IG, and GraphCL on datasets BBBP, SIDER, ClinTox, MUV, and BACE. The performance of IG and GraphCL on BACE and ClinTox are always better than other downstream datasets. The performance of DACL improves obviously when α increases to 0.96.

Chapter 5

Discussion

In the Experiment chapter, nine different algorithms are tested on eight datasets for comparison in the benchmark with unified set, as shown in Table 4.1 and Figures 4.2 and 4.3. Here the results are discussed in the joint view and algorithm-specific views separately.

Overall evaluation In general, SSL-GNN algorithms successfully achieve transfer learning tasks under FT & PT scheme: According to Table 4.1, most of the SSL-GNN algorithms tested in the benchmark experiment outperform 3% better accuracy than random baseline on average, with model hyper-parameter set of size five for exhaustive searching.

Prospective future of SSL-GNNs At the same time, the PT& FT training scheme adopted in the benchmark experiment is popular with meaningful application scenarios in many domains, such as natural language processing. For example, the language translation model pre-trained by a well-designed dictionary works better on many downstream translation tasks [13]. Under this scheme, corresponding models could be pre-trained on a huge dataset in advance by high-performance computing clusters. When the downstream tasks appear, the pre-trained model is fine-tuned and finally achieves higher accuracy with lower time consumption compared with the random baseline without the pre-training stage.

Contrast is powerful. The contrastive models attract more attention than the other two types of SSL-GNNs [37, 53, 55], with more methods proposed and more flexible replaceable modules. It is also proved in the experiment result: In Table 4.1, HIV and MUV are the only two big datasets with over 30,000 molecules. The best algorithms on these datasets are CP and JOAO, which are all contrastive models.

The relationship among datasets S_{pt}, S_{ft} and algorithm $\mathcal{A}$ The performance of the model should depend on two factors: how much correlation there is between the distribution of pre-training and fine-tuning datasets S_{pt}, S_{ft}, and how much correlation can be learned by the SSL-GNN algorithm $\mathcal{A}$. In the aforementioned example of translation task, assume the downstream task is to analyze the emotion of Chinese poems. If the dictionary is in Finnish, then there is little correlation between S_{pt}, S_{ft}. Even if the dictionary shares the same language with the downstream task, but the algorithm $\mathcal{A}$ cares little about the words expressed emotion, then $\mathcal{A}$ could bring limited improvement, even cause the negative transfer. Therefore, each specific downstream problem needs a comprehensive dictionary that contains the necessary information, and a well-designed SSL algorithm to obtain improvement in the pre-training stage.

Certain downstream datasets fit the pre-training dataset more than others on many algorithms Different algorithms obtain different advantages on different datasets, based on the Figure 4.2 and 4.3. For example, Figures 4.2(b), 4.2(e) and 4.2(f) show that algorithms G-Motif, GraphLoG, and CP always gain much more accuracy on the BACE dataset compared to other downstream datasets. It means the molecule related to inhibitors of human β-secretase 1 are sensitive to graph context, motif labels, and graph embeddings clustering. Figures 4.2(a) and 4.2(d) show that algorithms JOAO, and AM always perform better on the ClinTox dataset compared with other downstream datasets. It implies that molecular toxicity could be reflected by the difference in masked attributes and the corresponding augmentation pairs selected finally by JOAO. Figures 4.3(b) and 4.3(c) show that algorithms GraphCL and IG can achieve more improvement both on BACE and ClinTox than other datasets. Combined with the two factors discussed before, the pre-training dataset S_{pt} ZINC-15 is considered as sharing more correlated information with BACE and ClinTox datasets.

This conclusion also implies some future directions of benchmark except performance comparison among different SSL-GNN algorithms:

- Recommend relative better pre-training dataset S_{pt} for different downstream tasks based on specific dataset S_{ft} by benchmark methods.
- For algorithms that always obtain the best result on a specific downstream task, it is meaningful to analyze the distribution correlation between two datasets S_{pt} and S_{ft} by the mechanisms utilized specifically in these algorithms.

Algorithm Specific Views

No algorithm is reliable on all datasets. Mostly, each algorithm only works on specific datasets. However, it is beneficial to understand the algorithm design details by model hyper-parameters to analyze the performance: Patterns of model performance as a function of model-specific hyper-parameters could better reveal the model mechanism.

In this benchmark experiment, algorithms are evaluated on how many downstream datasets on which it gains the best accuracy among all algorithms in Table 4.1.

Analysis of CP CP is the best algorithm with the best performance on three downstream datasets. The hyper-parameter 'csize' represents the size of the context. Mathematically, for context graph $G_{context}(v, r_1, r_2)$ in Equation (3.4), $csize = r_2 - r_1$, and the $r_1 = 4 < 5$, which guarantees there is intersection between context graph $G_{context}(v, r_1, r_2)$ and $G_{5-hop}(v)$. From Figure 4.2(b), the size change of the graph ring brings stable improvement to the ClinTox dataset, from around -2% to 4%. It implies small ring radius is possibly harmful to catching toxicity properties. However, when the ring radius is large enough, an increase of it brings little change to the algorithm performance.

Analysis of JOAO and GraphCL JOAO is the second best algorithm with the best performance on two downstream datasets. While JOAO is the improved version of GraphCL by selecting the best augmentations pair from adversary learning, they should capture similar information shared with datasets S_{pt} and S_{ft}. Figure 4.2(d) shows that JOAO's pattern of accuracy change over augmentation strength is indeed very similar to it of GraphCL. Here the augmentation strength is the public hyper-parameter of these two algorithms. It measures the ratio of modification when the graph is augmented. It is expected that when augmentation strength is small, an increase of it brings better results, and when it is big enough, the increase of it becomes harmful. There is a minimum point on 15% augmentation strength for JOAO on the ClinTox dataset, which is counter-intuitive. More experiments with broader parameter intervals and sampling density could maybe bring the answer to this.

Analysis of DACL DACL only augments data in hidden space, while most SSL-GNN algorithms need graph-related or domain-related knowledge to design the pretext task. In other words, DACL requests less domain

information. Thus it is more difficult for DACL to achieve the same or better results than other algorithms. However, even though there is poor performance under the current experiment conditions, there is an obvious trend that the performance of DACL on all datasets improves when the mix-up α gets larger. More experiments with large mix-up *alpha* of DACL are needed for further analysis.

Chapter 6

Conclusion

In this thesis, an SSL-GNN algorithm benchmark platform was built with replaceable building blocks. And selected advanced algorithms were tested on this benchmark, with the same encoder structure and evaluation protocol. Then the experiment results were discussed in different views.

At first, the concepts of SSL and GNN, and three different SSL training schemes were introduced. Secondly, contrastive, generative, and predictive SSL-GNN models were discussed in the following details: model description, model modules, and corresponding algorithm examples. Then nine SSL-GNN algorithms were compared in the benchmark experiment, with one pre-training dataset, eight fine-tuning datasets, and five hyper-parameters under a pre-training and fine-tuning scheme. In this experiment, the CP and JOAO algorithms outperformed others under experiment settings in Section 4.2. The discussion chapter concludes that most algorithms tested achieve improvement on GNN-encoder training and analyzes the patterns with hyper-parameters change on certain algorithms and datasets.

Future Directions

With current work, many possible topics related to it deserve further research.

Benchmark experiments with more algorithms, datasets, and hyper-parameters. In fact, the tedious brute-force method by grid search could be annoying. However, the relatively small experiment could bring clear direction for further benchmark experiments as discussed in Section 5.

Develop better SSL-GNN algorithms with machine learning (ML) theory and domain knowledge.

- Borrow ML tools from other domains. Many graph algorithms are developed with the combination of MPNN framework and traditional ML tools, such as transformer [43], mix-up [51], adversarial learning [60], hierarchical clustering [57]. It encourages the researchers to try more possibilities on graph representation learning with SSL-GNNs by borrowing subtle designs from other domains. For example, combines the graph wavelet transform [23] with augmentations in the contrastive model.

- Merge more domain knowledge into algorithm structure with careful design. Take the molecules as an example: randomly modifying edges means to add or drop some chemical bonds, which possibly leads to non-existed augmented molecules and brings a negative influence on training. And for graph data with huge sizes, such as macromolecule, GNN with few layers cannot transfer information globally, and the message passing scheme becomes inefficient. Actually, most SSL-GNNs focus on the abstract graph topology, such as randomly drop nodes [59] or mask attributes [28]. Even some algorithms utilize the domain knowledge, such as G-Motif [43]. However, it just uses molecular motifs as labels rudely.

Discover correlated datasets and analyze the correlation pattern. Outstanding SSL-GNN should capture the correlation between pre-training and fine-tuning datasets, but it only works on the condition that these datasets share enough information. Benchmark experiment provides another perspective to analyze the correlation patterns between datasets: When specific datasets pair S_{pt}, S_{ft} show obvious fitness on several algorithms $\mathcal{A}$, they are assumed to be correlated, and the specific mechanisms of these algorithms provide analysis perspectives. (The fitness means that the result of combination $(\mathcal{A}, S_{pt}, S_{ft})$ is superior to random baseline.)

Bibliography

[1] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016.

[2] D. Bahdanau, K. H. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[3] J. Bastings, I. Titov, W. Aziz, D. Marcheggiani, and K. Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1957–1967, 2017.

[4] D. Beck, G. Haffari, and T. Cohn. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 273–283. Association for Computational Linguistics, 2018.

[5] G. W. Bemis and M. A. Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of Medicinal Chemistry*, 39(15):2887–2893, 1996.

[6] R. v. d. Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. *ArXiv Preprint ArXiv:1706.02263*, 2017.

[7] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[8] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and deep locally connected networks on graphs. In *2nd International Conference on Learning Representations (ICLR)*, 2014.

[9] S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 1, 2016.

[10] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1597–1607. PMLR, 2020.

[11] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[14] B. L. Douglas. The weisfeiler-lehman method and graph isomorphism testing. *ArXiv Preprint ArXiv:1101.5211*, 2011.

[15] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson. Benchmarking graph neural networks. *ArXiv Preprint ArXiv:2003.00982*, 2020.

[16] M. Fey and J. E. Lenssen. Fast graph representation learning with pytorch geometric. *ArXiv Preprint ArXiv:1903.02428*, 2019.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.

[18] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. PMLR, 13–15 May 2010.

[19] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, pages 729–734, 2005.

[20] I. Gulrajani and D. Lopez-Paz. In search of lost domain generalization. In *International Conference on Learning Representations (ICLR)*, 2021.

[21] M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304. PMLR, 2010.

[22] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017.

[23] D. K. Hammond, P. Vandergheynst, and R. Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[24] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *ArXiv Preprint ArXiv:1506.05163*, 2015.

[25] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[26] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations (ICLR)*, 2019.

[27] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, pages 22118–22133. Curran Associates, Inc., 2020.

[28] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[29] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2021.

[30] S. Ji, S. Pan, E. Cambria, P. Marttinen, and S. Y. Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514, 2021.

[31] L. Jing and Y. Tian. Self-supervised visual feature learning with deep neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):4037–4058, 2021.

[32] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012.

[34] G. Landrum. Rdkit: Open-source cheminformatics software. 2016. URL `https://github.com/rdkit/rdkit/releases/tag/Release_2016_09_4`.

[35] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia. Learning deep generative models of graphs, 2018. URL `https://openreview.net/forum?id=Hy1d-ebAb`.

[36] Q. Liu, M. Allamanis, M. Brockschmidt, and A. Gaunt. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018.

[37] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and P. Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2022.

[38] D. Marcheggiani and I. Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1506–1515. Association for Computational Linguistics, 2017.

[39] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations (ICLR)*, 2013.

[40] F. Monti, M. Bronstein, and X. Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in Neural Information Processing Systems*, 30, 2017.

[41] S. Nowozin, B. Cseke, and R. Tomioka. F-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016.

[42] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang. *Graph Representation Learning via Graphical Mutual Information Maximization*, pages 259–270. Association for Computing Machinery, 2020.

[43] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang. Self-supervised graph transformer on large-scale molecular data. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

[44] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[45] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[46] K. Sohn. Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016.

[47] T. Sterling and J. J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015.

[48] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations (ICLR)*, 2020.

[49] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. In *International Conference on Learning Representations (ICLR)*, page 4, 2019.

[50] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, page 20, 2018.

[51] V. Verma, T. Luong, K. Kawaguchi, H. Pham, and Q. Le. Towards domain-agnostic contrastive learning. In *Proceedings of the 38th International Conference on Machine Learning*, pages 10530–10541. PMLR, 2021.

[52] L. M. Weber, W. Saelens, R. Cannoodt, C. Soneson, A. Hapfelmeier, P. P. Gardner, A.-L. Boulesteix, Y. Saeys, and M. D. Robinson. Essential guidelines for computational method benchmarking. *Genome Biology*, 20(1):1–12, 2019.

[53] L. Wu, H. Lin, Z. Gao, C. Tan, and S. Z. Li. Self-supervised learning on graphs: Contrastive, generative,or predictive. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[54] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. Moleculenet: A benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

[55] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji. Self-supervised learning of graph neural networks: A unified review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2022.

[56] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019.

[57] M. Xu, H. Wang, B. Ni, H. Guo, and J. Tang. Self-supervised graph-level representation learning with local and global structure. In *Proceedings of the 38th International Conference on Machine Learning*. PMLR, 2021.

[58] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, pages 974–983. Association for Computing Machinery, 2018.

[59] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020.

[60] Y. You, T. Chen, Y. Shen, and Z. Wang. Graph contrastive learning automated. In *Proceedings of the 38th International Conference on Machine Learning*, pages 12121–12132. PMLR, 2021.

[61] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *Proceedings of the 38th International Conference on Machine Learning*, pages 12310–12320. PMLR, 18–24 Jul 2021.

[62] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, pages 19365–19376, 2018.

Appendix A

Appendix

A.1 Model hyper-parameters list

CP 'csize' $\in [2, 3, 4, 5, 6]$.

- The 'csize' controls the radius length of ring shape context graph $G_{context}(v, r_1, r_2)$ in Equation (3.4), and '$cisze$' $= r_2 - r_1$.

AM 'mask_rate' $\in [0.05, 0.1, 0.15, 0.2, 0.25]$

- The 'mask_rate' controls the rate of atoms to be masked. Here only node attributes are masked.

DACL 'mixup_alpha' $\in [0.75, 0.8, 0.85, 0.9, 0.95]$

- The 'mixup_alpha' is hyper-parameter α which controls the mixup strength. When creating augmented views of input graph by the linear mixup, the parameter λ in Equation (3.2) is sampled from uniform distribution $U(\alpha, 1)$.

EP 'random_seed' $\in [0, 1, 2, 3, 4]$

- There is no algorithm-specific hyper-parameter, thus more randomness provided by the hyper random seeds here.

GraphCL 'aug_strength' $\in [0.05, 0.1, 0.15, 0.2, 0.25]$

- The 'aug_strength' adjusts the augmentation strength, which is the degree that structure will be augmented.

GraphLoG
('hierarchy', 'num_proto') $\in [(2, 50), (3, 50), (2, 100), (3, 100), (4, 100)]$

- The 'hierarchy' is the number of hierarchy in prototypes $\mathcal{C}$.
- The 'num_proto' is the number of initial prototypes $\mathcal{C}$.

IG 'random_seed' $\in [0, 1, 2, 3, 4]$

- There is no algorithm-specific hyper-parameter, thus more randomness provided by the hyper random seeds here.

JOAO 'aug_strength' $\in [0.05, 0.1, 0.15, 0.2, 0.25]$

- The 'aug_strength' adjusts the augmentation strength, which is the degree that structure will be augmented.

G-Motif 'random_seed' $\in [0, 1, 2, 3, 4]$

- There is no algorithm-specific hyper-parameter, thus more randomness provided by the hyper random seeds here.

A.2 Datasets information

Dataset	ZINC15	BBBP	Tox21	ToxCast	SIDER	ClinTox	MUV	HIV	BACE
Num_graphs	2000000	2039	7831	8576	1427	1477	93087	41127	1513
Avg_nodes	26.63	24.06	18.57	18.78	33.64	26.16	24.23	25.51	34.09
Avg_edges	57.74	51.91	38.59	38.52	70.72	55.77	52.56	54.94	73.72

Table A.1: Information of pre-training and fine-tuning datasets. This table contains three statistical indexes of datasets S_{pt} and S_{ft} mentioned in Section 4.2: the number of graphs, the average of nodes number per graph, and the average of edges number per graph