A Survey on Unsupervised Graph Representation Learning

Yuanhao Xiong, Xiangning Chen, Li-Cheng Lan, Lucas Tecot
{yhxiong,xiangning,lclan}@cs.ucla.edu, lucastecot@gmail.com

Abstract

Recently, machine learning algorithms have achieve great success in tasks related to graph-structured data, such as citation networks and protein molecules. Learning good latent representations from raw features is key to the final performance. When the task is specified and labels are accessible, those representations are usually derived from the end-to-end training. However, it is more common when no label information is provided. In that case, a variety of unsupervised graph representation learning methods have been proposed. We select several typical algorithms ranging from DeepWalk to Graphical Mutual Information, and summarize them with a general framework. Besides, we conduct a thorough evaluation of these methods on three benchmarking datasets, Cora, Citeseer, and PubMed. Experimental results have shown the progresses in unsupervised GRL, and indicated great potentials in fine-tuning the model from pre-trained representations.

1 Introduction

The structure of graph is ubiquitous in the real word: social networks among people, biological pathways, protein structure, and so on. All these sets of entities that interact between each other can be described as graphs. With rapid development of machine learning in domains dealing with images and texts, increasing attention has been drawn to graph-structured data, to solve related tasks like node classifcation and link prediction. However, a large number of approaches rely on specific assumptions about the graph and the task at hand, and limit the model's generalization to other graphs. For example, the common neighbor heuristic assumes that two nodes are more likely to connect if they have many common neighbors [28]. It works correctly in social networks while not applicable to protein structures. Then a question arises naturally: can we just extract latent features from raw graph data automatically for different downstream tasks? This is exactly the goal of representation learning [3]. A more practical scenario is that representations are learned in an unsupervised setting, where no labels or targets are given during the process. In this way, more general latent features can be obtained to capture the intrinsic properties of the graph. Great progresses in computer vision such as MoCo [12] and SimCLR [5] and natural language such as word2vec [17] and BERT [8] have advanced representation learning in graphs without supervision. Therefore, in this paper we aim to study existing methods and make a thorough comparison.

Specifically, we investigate several representative methods, including DeepWalk [21], node2vec [10], GraphSage [11], Adversarially Regularized Graph AutoEncoders (ARVGA + ARGA) [19], Deep Graph Infomax (DGI) [23], Graphical Mutual Information (GMI) [20], and GNN with self-supervised learning [27]. They are then evaluated on three benchmarking datasets, Cora, Citeseer, and PubMed for two downstream tasks, node classification and link prediction.

This survey is organized as follows: Section 2 introduces the related work of unsupervised graph representation learning; Section 3 gives the problem formulation; Section 4 presents a general framework for representation learning in the unsupervised setting, together with a discussion about pros and cons of selected methods; Section 5 describes the details of representative algorithms;

Section 6 demonstrates evaluation results on benmarking datasets; Section 7 discusses the future direction in graph representation learning; finally Section 8 draws a brief conclusion.

2 Related Work

In real-world applications where labels are not available, we can employ unsupervised learning to distill useful knowledge from the graph. The learned embedding (for node [10] or the whole graph [18]) can then serve as a strong prior knowledge from downstream tasks. Generally, most of unsupervised approaches leverage the idea of contrastive learning. These methods work by defining a score for pairs of samples that follow observations in the data, also known as positive samples, and a score for negative samples that deviate from observations. The objective of representation learning then can be formulated as minimizing a loss function designed to maximize the score for positive samples, and minimize it for negative samples [7].

Contrastive learning then have been implemented by a variety of algorithms. At the beginning, an intuitive assumption is that nodes with similar structural information (e.g. neighbourhood) should have similar embeddings in the high dimension. Based on that intuition, some methods simply sample a fixed number of neighbors for each node [11], which can merely capture local information. Other works convert the graph into node sequences via random walk [21, 29, 10, 11]. Such methods are adapted from word2vec [17], modeling nodes as words and random walks as sentences. There are also papers proposing clustering algorithm to define such similarity [6]. On the other hand, similarity between nodes can be implicitly captured by reconstructing the graph, especially adjacency information. One simple and effective method is the encoder-decoder model: using an encoder to embed the graph into a high-level latent representation, then a decoder manages to reconstruct the original graph based the latent embedding [16, 19].

With the development of graph neural networks, stronger and more effective architectures have been proposed to extract the graph structure. For example, there are improvements based on graph convolution and graph attention [24] to better aggregate neighbors [22]. Since the previous assumption that neighboring nodes have similar representations has already been utilized in GNNs, another line of work attempts to maximize the mutual information [23, 20] between the hidden representation and the whole graph, which is shown to better capture the global structural information.

Recently, some methods based on self-supervised learning are proposed [25, 27, 15]. They employ pre-trained representations as initialization, and then fine-tune the model on different downstream tasks, similar to BERT fine-tuning.

3 Problem Formulation

In this survey, we assume a generic graph-based unsupervised representation learning setup: we are provided with an undirected graph $\mathcal{G} = \{V, E\}$, where V is a set of nodes and E is a set of edges. Each edge e_k takes the form of a pair (v_i, v_j) with $v_i, v_j \in V$. We define feature matrix as $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where N is the number of nodes in the graph and $\mathbf{x}_i \in \mathbb{R}^F$ represents the raw features of node i. We also have access to relational information between these nodes in the form of an *adjacency matrix*, $\mathbf{A} \in \mathbb{R}^{N \times N}$. While $\mathbf{A}$ may consist of arbitrary real numbers (or even arbitrary edge features), we only assume the graphs to be *unweighted*, i.e. $A_{ij} = 1$ if there exists an edge (v_i, v_j) in the graph and $A_{ij} = 0$ otherwise.

Our objective is to learn an *encoder*, $\mathcal{E} : \mathbb{R}^{N \times F} \times \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times F'}$, such that $\mathcal{E}(\mathbf{X}, \mathbf{A}) = \mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N\}$ represents high-level representations $\mathbf{z}_i \in \mathbb{R}^{F'}$ for each node i. These representations may then be retrieved and used for downstream tasks, such as node classification and link prediction. When learning those representations, no label information is leveraged.

4 Representation Learning Framework

As mentioned in Section 2, most representation learning methods are based on contrastive learning, and therefore can be summarized by a general framework, shown in Figure x. As we can see, the first procedure is to obtain positive and negative samples for each node of interest. Then an encoder is utilized to generate latent features, or embeddings for different nodes. With positive and negative

pairs sampled in the first stage, corresponding similarity scores can be calculated and then used for loss computation. In Table, we provide a comparison among representative methods of their modules in the framework. More details of these algorithms will be discussed in Section 5.

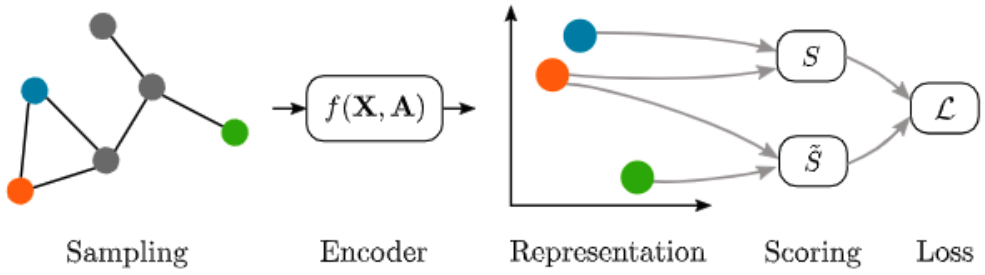

Figure 1: A general framework for unsupervised graph representation learning.

Table 1: A comparison of different methods.

Method	Encoder	Score	Loss	Sampling
DeepWalk (node2vec)	Embedding	$\sigma(\mathbf{z}_i^{\mathrm{T}}\mathbf{z}_j)$	$-\log(S)-\log(1-\widetilde{S})$	+: random walk neighbors −: non-neighboring nodes[1]
GraphSAGE	GCN	$\sigma(\mathbf{z}_i^{\mathrm{T}}\mathbf{z}_j)$	$-\log(S)-\log(1-\widetilde{S})$	+: random walk neighbors −: non-neighboring nodes
ARGE	GCN	$\sigma(\mathbf{z}_i^{\mathrm{T}}\mathbf{z}_j)$	$-\log(S)-\log(1-\widetilde{S})$	+: prior distribution −: graph encoder
ARVGE	GCN	$\sigma(\mathbf{z}_i^{\mathrm{T}}\mathbf{z}_j)$	$-\log(S)-\log(1-\widetilde{S})+KL(p(\cdot)\|\|q(\cdot))$	+: prior distribution −: graph encoder
DGI	GCN	$\sigma(\mathbf{z}_i^{\mathrm{T}}\mathbf{W}\mathbf{s})$	$-\log(S)-\log(1-\widetilde{S})$	+: original graph −: corrupted graph
GMI	GCN	$\sigma(\mathbf{z}_i^{\mathrm{T}}\mathbf{W}\mathbf{x}_j)$	See Eq. (16)	+: original graph −: corrupted graph

4.1 Pros and Cons

For DeepWalk and node2vec, they are attempts to learn node representations in the early stage. They are simple but hard to generalize to large graphs and performance is highly dependent on hyperparameter choice [10, 21]. it is unclear whether random-walk objectives actually provide any useful signal, as these encoders already enforce an inductive bias that neighboring nodes have similar representations.

For ARGE, the objective is to reconstruct the adjacency matrix, which makes it more suitable for link prediction while less competitive in node classification.

DGI maximizes the mutual information between local node representations and global summary representation of the graph. This property guarantees that DGI can capture local and global information of the graph simultaneously. However, the choice of the corruption function has a great influence on the performance as well as the number of negative samples, based on the experimental findings and analysis in [14],

Different from DGI, GMI directly focuses on maximizing the mutual information between the input and output of a graph neural encoder in terms of node features and topological structure. And GMI leverages a more stable JSD estimator to compute mutual information rather than infoNCE.

[1] node2vec is based on DeepWalk and introduces parameters to achieve biased random walks.

5 Selected Methods

In this section, details of selected algorithms of unsupervised graph representation learning are discussed.

5.1 DeepWalk

Similar to the idea of work2vec Mikolov et al. [17] that words appearing in the same context should be close in the high-dimensional embedding space, DeepWalk [21] defines the context C_i of a certain node v_i as a sequence of nodes present in a random wandering starting from v_i:

$$C_i = \{v_{i-w}, ..., v_{i-1}, v_{i+1}, ..., v_{i+w}\}. \tag{1}$$

With negative sampling, for each node the training objective is to maximize its co-occurrence probability with a positive node v_p in C_i, and minimize the probability for another randomly sampled negative node v_n:

$$\min \left[-\log p(v_p \in C_i | v_i) - \log(1 - p(v_n \notin C_i | v_i)) \right]. \tag{2}$$

5.2 Node2Vec

Node2Vec provides node embedding that allow us to identify it's neighbors $N_s(v)$ which is sampled by Biased Random Walk s. Biased Random Walks is designed to balance BFS and DFS. BFS focus on local neighbors and DFS on further neighbors. Assume the embedding of node v is $f(v)$. The objective funciton is:

$$\max_f \left[\sum_{u \in V} log Pr(N_s(u) | f(u)) \right]. \tag{3}$$

The algorithm also assumes that the probability of each neighbor is independent. Hence they have:

$$Pr(N_s(u)|f(u)) = \prod_{n_i \in N_s u} Pr(n_i|f(u)) = \prod_{n_i \in N_s u} \frac{exp\ f(n_i) f(u)}{\sum_{v \in V} exp\ f(v) f(u)}. \tag{4}$$

Since the calculation of $Pr(N_s(u)|f(u))$ is too high, the algorithm uses Biased Random Walk to proximate it. Biased Random Walk is controlled by two hyper-parameters p and q. When we just pass through edge (t, v) to node v, we have $\pi_{vx} = \alpha_{pq}(t, x) * w_{vx}$. w_{vx} is the weight of edge (v, x) and $\alpha_{pq}(t, x)$ is defined as:

$$\alpha_{pq}(t,x) = \begin{cases} 1/p, & \text{if } d_{tx} = 0 \\ 1, & \text{if } d_{tx} = 1 \\ 1/q, & \text{if } d_{tx} = 2 \end{cases} \tag{5}$$

where d_{tx} is the distance between t and x. Therefore, if return parameter p is large we won't return to t and if In-out parameter q is low the algorithm will walk farther.

5.3 GraphSAGE

GraphSAGE is the short term of "Graph SAmple and aggreGatE". Different from GCN, GraphSAGE can generate the embedding for unseen nodes during training. GraphSAGE can be separated into following three steps. First, it sample its neighborhood. In the paper, it just sample nodes with the distance of two. The purpose of this step is to reduce to complexity of the algorithm.

For the second step, aggregate feature information from neighbors using GCN. This implies that its neighbor decides the embedding of a node. Hence the algorithm can generate the embedding for unseen nodes during training by using their neighbor. The paper proposed several ways to aggregate the neighbors' information, such as mean, Pooling, and LSTM. Following works such as Graph Attention Networks [24] use attention network to aggregate the information. In the paper, the aggregation starts from the nodes with 2 distances. After the first round of aggregation, we can get the temperate embedding of the nodes with 1 distance. And after the second round of aggregation, we can get the final embedding of the node we are interested in. The final embedding can be trained

by both supervised loss and unsupervised loss. In this survey, we use the unsupervised loss, which is shown as follow:

$$\mathcal{J}_{\mathcal{G}}(z_u) = -log(\sigma(z_u^T z_v)) - Q * \mathbb{E}_{v_n \sim P_n(u)} log(\sigma(-z_u^T z_{v_n})) \tag{6}$$

For the first part, v is a random neighbor of u. $z_u^T z_v$ means the similarity of v and u. Since they are neighbor, they should be very similar. For the second part, $v_n \sim P_n(u)$ is the negative sampling, and Q is the number of negative sampling. We will want v_n and u as different as possible. Therefore there is a negative before $z_u^T z_{v_n}$.

Finally, GraphSAGE use the trained embedding to predict graph context and label using aggregated information.

5.4 ARGE + ARVGE

Adversarially regularized graph autoencoders are an iteration on previous graph autoencoder work [16] extended with an adversarial learning scheme to help enforce the latent variable's distribution. The goal of all of these models is to create a per-node embedding of the graph that allows for the adjacency matrix to be accurately reconstructed.

5.5 GAE + VGAE

For both model types, the encoder is constructed with GCN layers as follows:

$$f(z^{(l)}, A|W^{(l)}) = \phi(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}Z^{(l)}W^{(l)}) \tag{7}$$

$$\widetilde{D} = \sum_j \widetilde{A}_{ij} \tag{8}$$

$$\widetilde{A} = A + I \tag{9}$$

In the paper introducing this method [19], they use two layers with a Relu non-linearity on the first layer. For ARGE, the output of this network is simply the embedding. In ARVGE, they instead use this encoder to produce the means and variances that the embedding is sampled from. Specifically:

$$q(z_i|X, A) = \mathcal{N}(z_i|\mu_i, diag(\sigma^2)) \tag{10}$$

In the original paper μ and σ share their first GCN layer but have separate second layers.

For the decoder, both models use a sigmoid on the inner product of the node embeddings as follows:

$$p(A_{ij} = 1|z_i, z_j) = sigmoid(z_i^T z_j) \tag{11}$$

For the ARGE, we simply minimize the reconstruction error of the adjacency matrix. For ARVGE we use the variational lower bound. Specifically the loss for each of these is:

$$L_{ARGE} = -\mathbb{E}_{q(Z|X,A)}[\log p(A, Z)] \tag{12}$$

$$L_{ARVGE} = -\mathbb{E}_{q(Z|X,A)}[\log p(A, Z)] + KL[q(Z|X, A)\|p(Z)] \tag{13}$$

Where $p(Z)$ is the unit Gaussian prior.

5.6 Adversarial Regularization

The difference between GAE/VGAE [16] and ARGE/ARVGE [19] is the use of adversarial regularization on the embedding space. During the training of the encoder/decoder models, they additionally train a binary classifier. This classifier predicts whether or not the input came from the prior distribution (p_z, the unit Gaussian in the original paper) or from the graph encoder. The loss for this classifier is:

$$-\frac{1}{2}\mathbb{E}_{z \sim p_z} \log D(Z) - \frac{1}{2}\mathbb{E}_X \log(1 - D(G(X, A))) \tag{14}$$

Where D is the binary classifier discriminator and G is the encoder from the previous section. In turn, the encoder is optimized to maximize this loss (in tandem with minimizing the original reconstruction loss).

5.7 Deep Graph Infomax

Deep Graph Infomax (DGI) [23] is a general approach for learning node representations within graph-structured data in an unsupervised manner. The objective of DGI is to maximize mutual information between patch representations, i.e., node embeddings, and corresponding high-level global representations of the graph. With notations in Section 4, to obtain a summary vector $\mathbf{s}$, a readout function *readout function*, $\mathcal{R} : \mathbb{R}^{N\times F} \rightarrow \mathbb{R}^F$ is leveraged to aggregate all patch representations into a graph-level one: $\mathbf{s} = \mathcal{R}(\mathcal{E}(\mathbf{X}, \mathbf{A}))$. On the other hand, GMI follows Mutual Information Neural Estimation (MINE [2]) to use a discriminator $\mathcal{D} : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}$ as a proxy to estimate mutual information. Specifically, $\mathcal{D}(\mathbf{z}_i, \mathbf{s})$ represents the probability scores assigned to a node-summary pair. Since training an encoder to be contrastive between representations that capture statistical dependencies of interest and those that do not are shown to be effective, a noise-contrastive objective is employed here:

$$\mathcal{L} = \frac{1}{N+M}\left(\sum_{i=1}^{N}\mathbb{E}_{(\mathbf{X},\mathbf{A})}\left[\log\mathcal{D}\left(\mathbf{z}_i,\mathbf{s}\right)\right] + \sum_{j=1}^{M}\mathbb{E}_{(\widetilde{\mathbf{X}},\widetilde{\mathbf{A}})}\left[\log\left(1-\mathcal{D}\left(\widetilde{\mathbf{z}}_j,\mathbf{s}\right)\right)\right]\right), \tag{15}$$

where $\widetilde{\mathbf{z}}_j$ is a negative sample from an alternative graph, $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}})$. For a single graph, a *corruption function*, $\mathcal{C} : \mathbb{R}^{N\times F} \times \mathbb{R}^{N\times N} \rightarrow \mathbb{R}^{M\times F} \times \mathbb{R}^{M\times M}$ is required to obtain a negative example from the original graph, i.e. $(\widetilde{\mathbf{X}}, \widetilde{\mathbf{A}}) = \mathcal{C}(\mathbf{X}, \mathbf{A})$.

5.8 Graphical Mutual Information

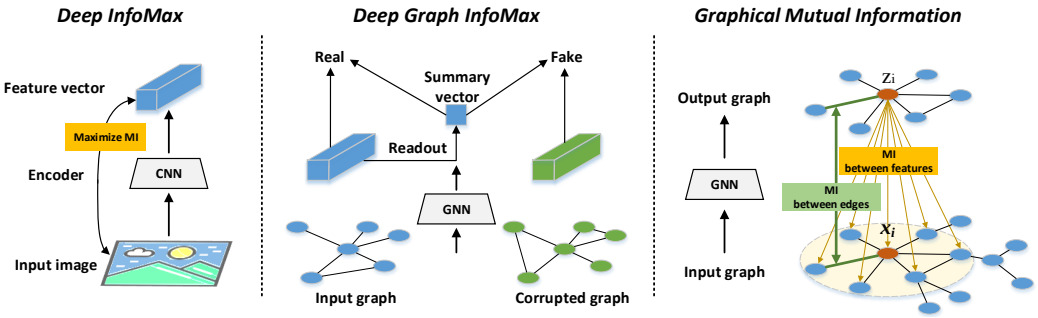

Figure 2: Illustration of DGI and GMI. This figure is adapted from [20].

Different from DGI whose goal is to maximize mutual information between node representations and graph-level summary representations, Graphical Mutual Information (GMI) adopts a more straightforward approach by comparing the input (i.e., the sub-graph consisting of the input neighborhood) and the output (i.e., the hidden representation of each node) of the encoder. Before introducing the details of GMI, we define $\mathbf{X_i}$ and $\mathbf{A_i}$ for node i as respectively the features of its neighbors and the corresponding adjacency matrix conditional on the neighbors. we call the sub-graph expanded by $\mathbf{X_i}$ and $\mathbf{A_i}$ as a *support graph* for node i, denoted by $\mathcal{G}_i$.

Definition 1 (Graphical Mutual Information)**.** *The MI between the hidden vector* $\mathbf{z}_i$ *and its support graph* $\mathcal{G}_i = (\mathbf{X}_i, \mathbf{A}_i)$ *is defined as*

$$\begin{aligned} I(\mathbf{z}_i; \mathcal{G}_i) \quad &= \quad \sum_{j}^{i_n} w_{ij} I(\mathbf{z}_i; \mathbf{x}_j) + I(w_{ij}; a_{ij}), \\ &\textit{with} \quad w_{ij} = \sigma(\mathbf{z}_i^{\mathrm{T}}\mathbf{z}_j), \end{aligned} \tag{16}$$

where a_{ij} *is the edge weight/feature in the adjacency matrix* $\mathbf{A}$*, and* $\sigma(\cdot)$ *is a sigmoid function.*

In particular, we calculate $I(\mathbf{z}_i; \mathbf{x}_j)$ in the first term of Eq. (16) by

$$I(\mathbf{z}_i; \mathbf{x}_j) \;=\; -sp(-\mathcal{D}(\mathbf{z}_i, \mathbf{x}_j)) - \mathbb{E}_{\tilde{\mathbb{P}}}[sp(\mathcal{D}(\mathbf{z}_i, \mathbf{x'}_j))], \tag{17}$$

where $\mathcal{D}$ is a discriminator constructed by a neural network. $\vec{x'}_j$ is an negative sampled from $\tilde{\mathbb{P}} = \mathbb{P}$, and $sp(x) = log(1 + e^x)$ denotes the soft-plus function. We maximize $I(w_{ij}; a_{ij})$ via calculating its cross-entropy

$$I(w_{ij}; a_{ij}) \ = \ a_{ij} log w_{ij} + (1 - a_{ij}) log(1 - w_{ij}). \tag{18}$$

By maximizing $I(\mathbf{z}_i; \mathcal{G}_i)$ with the sum of Eq. (17) and Eq. (18) over all hidden vectors $\mathbf{H}$, we arrive at our complete objective function for GMI optimization. Besides, we can further add trade-off parameters to balance Eq. (17) and (18) for more flexibility.

5.9 GNN-Specific Self-Supervised Learning

Self-supervision has attracted a surge of interest to make use of rich unlabeled data in the computer vision domain [12, 9]. One typical paradigm of self-supervised learning is to first pretrain the network based on some pretext tasks, and then fine-tune it for downstream tasks. Another way is to use the self-supervision loss as an auxiliary regularization term added to the supervised loss, which is shown to improve generalization ability Hendrycks et al. [13]. You et al. [27] presents the first systematic study on how to incorporate self-supervision in Graph Neural Networks. In particular, they study *how to select self-supervision scheme* and *how to choose pretext tasks*.

Self-Supervision Scheme. The most straightforward way is to pretrain the GNN with certain pretext tasks, whose parameters can serve as the initialization for downstream tasks. In the pretraining process, the objective is given as:

$$\min L_{ss}(Z, \ Y_{ss}), \tag{19}$$

$$Z = \hat{A} \, \text{ReLU}(\hat{A} X W_0) \, W_1, \tag{20}$$

$$\hat{A} = D^{-1/2}(A + I)D^{-1/2}, \tag{21}$$

where D is the degree matrix of $A + I$, Z is the output of a 2-layer GCN model, L_{ss} is the pretext task loss, and Y_{ss} is the pseudo-labels assigned by the pretext task. However, such scheme just brings little performance gain, especially for large datasets. The learned information about graph structure and features during pretraining may be largely lost during fine-tuning while targeting the target supervised loss alone. The shallow GNNs can be more easily overwritten when switching from L_{ss} to L_{sup} than CNNs. Another way is to use self-supervision loss as a regularization term and optimize it together with the supervised one:

$$\min \ \alpha_1 L_{sup} + \alpha_2 L_{ss}. \tag{22}$$

The effectiveness of the regularization term has been shown in graph Laplacian regularizer (GLR) [1], and the self-supervised task can act as the regularizer learned from unlabeled data under the minor guidance of human prior. Empirically, such regularization manner is more effective than the pretraining & finetuning manner.

Pretext tasks. A variety of GNN-specific pretext tasks by utilizing the rich node and edge information have been shown to benefit downstream tasks. One straightforward task is **node clustering**, which simply clusters the nodes based the feature matrix X and assign the cluster indices as the pseudo-label. This task can group distant nodes with similar features together. However, this task is less informative when the dataset is large and the feature dimension is low; Another exemplary pretext task is called **graph partitioning**. Unlike node clustering that groups nodes with similar attributes, this task group nodes based on the graph topology. Intuitively, it partitions the nodes into roughly equal subsets, ensuring that the number of connecting edges between subsets is minimized. This pretext task is shown to be general and effective to benefit GNNs.

6 Experiment

6.1 Evaluation Metric

Following previous studies, we plan to evaluate the quality of representations learned from different algorithms on two classical downstream tasks: link prediction and node classification. For link prediction, our goal is to predict whether there is an edge between a pair of nodes, given their

embeddings and we use Average Precision (AP) scores as a metric. For node classification, we leverage learned features to predict the label of a node. This can be achieved by training a simple classifier, such as a logistic regression model with the embedding as input. Accuracy is the metric for node classification.

As to the benchmark datasets, we select several commonly used graphs in the literature with varying sizes. Here is a table summarizing these datasets.

Table 2: Statistics of the datasets used in experiments.

Dataset	**Type**	**#Nodes**	**#Edges**	**#Features**	**#Classes**
Cora	Citation network	2,708	5,429	1,433	7
Citeseer	Citation network	3,327	4,732	3,703	6
PubMed	Citation network	19,717	44,338	500	3

6.2 Training Details

For both node classification and link prediction, we use default hyperparameters in original papers. Each model is trained with the Adam optimizer for 300 epochs. For node classification, datasets are split into training set and testing set following [26]. As to link prediction, each dataset is separated into a training, testing set and validation set. The validation set contains 5% citation edges for hyperparameter optimization, the testing set holds 10% citation edges to verify the performance, and the rest are used for training. Split datasets contain all the nodes in the original graph but different edges to ensure that the model will not observe edges to be predicted in the testing set during training.

6.3 Results

We present results for node classification and link prediction in Table 3 and 4 respectively. It can be observed that all these learned representations outperform raw features to a great degree. In addition, sophisticated algorithms with strong theoretical support like DGI and GMI are consistently better than others. Unsupervised methods sometimes can even surpass fully-supervised ones. Generally, fine-tuning the pre-trained node representations on specific tasks can achieve the best performance and is the most flexible and effective method.

Table 3: Summary of results in terms of classification accuracies. In the first column, we highlight the kind of data available to each method during training ($\mathbf{X}$: features, $\mathbf{A}$: adjacency matrix, $\mathbf{Y}$: labels). "GCN" corresponds to a two-layer GCN encoder trained in a supervised manner.

Available data	**Method**	**Cora**	**Citeseer**	**Pubmed**
$\mathbf{X}$	Raw features	$47.9 \pm 0.4\%$	$49.3 \pm 0.2\%$	$69.1 \pm 0.3\%$
$\mathbf{A}$	DeepWalk	$67.2 \pm 1.4\%$	$43.2 \pm 1.4\%$	$65.3 \pm 0.3\%$
$\mathbf{A}$	Node2Vec	$70.9 \pm 1.0\%$	$48.1 \pm 1.4\%$	$68.7 \pm 1.0\%$
$\mathbf{X}, \mathbf{A}$	GraphSAGE	$76.6 \pm 1.6\%$	$67.4 \pm 1.0\%$	$78.7 \pm 0.7\%$
$\mathbf{X}, \mathbf{A}$	ARGE	$78.6 \pm 0.2\%$	$70.2 \pm 0.7\%$	$76.7 \pm 0.6\%$
$\mathbf{X}, \mathbf{A}$	ARVGE	$80.5 \pm 0.9\%$	$70.0 \pm 1.3\%$	$77.7 \pm 0.9\%$
$\mathbf{X}, \mathbf{A}$	DGI	$81.8 \pm 0.6\%$	$70.7 \pm 0.8\%$	$76.8 \pm 0.8\%$
$\mathbf{X}, \mathbf{A}$	GMI	$\mathbf{82.0 \pm 0.1\%}$	$\mathbf{71.2 \pm 0.4\%}$	$\mathbf{79.5 \pm 0.4\%}$
$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	Planetoid	75.7%	64.7%	77.2%
$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	GCN	81.5%	70.3%	79.0%
$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	GCN + SS (Clustering)	$81.6 \pm 0.6\%$	$70.7 \pm 0.8\%$	$78.8 \pm 0.4\%$
$\mathbf{X}, \mathbf{A}, \mathbf{Y}$	GCN + SS (Partition)	$\mathbf{81.8 \pm 0.7\%}$	$\mathbf{71.3 \pm 0.7\%}$	$\mathbf{80.0 \pm 0.7\%}$

Table 4: Results of the link prediction task in citation networks. Average precision is reported.

Method	Cora	Citeseer	Pubmed
DeepWalk	92.0 ± 0.6	91.3 ± 0.5	91.3 ± 0.3
Node2Vec	91.7 ± 0.2	85.1 ± 1.0	76.3 ± 0.3
GraphSAGE	88.6 ± 1.1	84.8 ± 1.3	83.8 ± 0.5
ARGE	92.1 ± 0.3	88.0 ± 0.6	96.4 ± 0.9
ARVGE	92.5 ± 0.2	89.4 ± 0.7	$\mathbf{96.6 \pm 0.4}$
DGI	91.7 ± 1.3	92.4 ± 0.6	94.9 ± 0.1
GMI	$\mathbf{95.5 \pm 1.0}$	$\mathbf{94.3 \pm 0.5}$	96.0 ± 0.2

7 Future Direction

Despite the fact that unsupervised graph representation learning have been shown effective in some downstream tasks, there are still many challenges we need to deal with:

- **A more fair evaluation protocol.** Currently, benchmarking different unsupervised GRL methods is usually conducted on small datasets like citation networks in this survey. Performance can be very sensitive to the choice of hyperparameters [4], which is also a common scenario in other machine learning tasks such as image classification. It is worthwhile to design a fair evaluation protocol to take hyperparameter tuning into consideration, as well as leverage more robust and diverse datasets.
- **Negative sampling strategies.** Negative mining, i.e., How to obtain effective negative samples remains an important problem in contrastive learning. Existing methods only use simple sampling strategies, like shuffling the original graph.
- **Generalization to large and realistic graphs.** Although unsupervised representation learning is effective in small datasets, it is hard to apply them to large and realistic graphs due to memory issues. Besides, if pre-trained embeddings of molecules can be obtained, it will contribute significantly to the area of biology and chemistry.

8 Conclusion

In this survey, we have dived into recent progresses in unsupervised graph representation learning, which is a fundamental and crucial topic in graph-structured data. Contrastive learning plays an important role in generating powerful representations and using pre-trained embeddings for fine-tuning can even surpass fully-supervised methods. On the other hand, there are still some challenges such as negative mining and evaluation protocol we have to face with.

Task Distribution

Table 5: Task distribution for each group member.

Task	People
1. Survey and implement DGI and GMI	Yuanhao Xiong
2. Survey and implement DeepWalk and GCN with SS	Xiangning Chen
3. Survey and implement node2vec and GraphSAGE	Li-Cheng Lan
4. Survey and implement ARGA and ARVGA	Lucas Tecot
5. Write the report	All members

References

[1] Rie Kubota Ando and Tong Zhang. Learning on graph with laplacian regularization. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, page 25–32, Cambridge, MA, USA, 2006. MIT Press.

[2] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. Mine: mutual information neural estimation. *arXiv preprint arXiv:1801.04062*, 2018.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

[4] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *arXiv preprint arXiv:2005.03675*, 2020.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 257–266, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330925. URL https://doi.org/10.1145/3292500.3330925.

[7] Daniel Fernando Daza Cruz. A modular framework for unsupervised graph representation learning. 2019. URL https://dfdazac.github.io/assets/dd_thesis.pdf.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Priya Goyal, Dhruv Mahajan, Abhinav Gupta, and Ishan Misra. Scaling and benchmarking self-supervised visual representation learning. *arXiv preprint arXiv:1905.01235*, 2019.

[10] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[11] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*, 2017.

[12] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[13] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/a2b15837edac15df90721968986f7f8e-Paper.pdf.

[14] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[15] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2020.

[16] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[18] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs, 2017.

[19] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407*, 2018.

[20] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph Representation Learning via Graphical Mutual Information Maximization. In *Proceedings of The Web Conference*, 2020. doi: https://doi.org/10.1145/3366423.3380112.

[21] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[22] Ke Tu, Peng Cui, Xiao Wang, Philip S Yu, and Wenwu Zhu. Deep recursive network embedding with regular equivalence. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2357–2366. ACM, 2018.

[23] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *ICLR (Poster)*, 2019.

[24] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[25] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rklz9iAcKQ`.

[26] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[27] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. When does self-supervision help graph convolutional networks?, 2020.

[28] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *arXiv preprint arXiv:1802.09691*, 2018.

[29] Chenyi Zhuang and Qiang Ma. Dual graph convolutional networks for graph-based semi-supervised classification. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, page 499–508, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356398. doi: 10.1145/3178876.3186116. URL `https://doi.org/10.1145/3178876.3186116`.