Learning to Pre-train Graph Neural Networks

Yuanfu Lu[1, 2*], **Xunqiang Jiang**[1], **Yuan Fang**[3], **Chuan Shi**[1, 4†]
[1]Beijing University of Posts and Telecommunications
[2]WeChat Search Application Department, Tencent Inc. China
[3]Singapore Management University
[4]Peng Cheng Laboratory, Shenzhen, China
luyuanfu@bupt.edu.com, skd621@bupt.edu.cn, yfang@smu.edu.sg, shichuan@bupt.edu.cn

Abstract

Graph neural networks (GNNs) have become the *de facto* standard for representation learning on graphs, which derive effective node representations by recursively aggregating information from graph neighborhoods. While GNNs can be trained from scratch, pre-training GNNs to learn transferable knowledge for downstream tasks has recently been demonstrated to improve the state of the art. However, conventional GNN pre-training methods follow a two-step paradigm: 1) pre-training on abundant unlabeled data and 2) fine-tuning on downstream labeled data, between which there exists a significant gap due to the divergence of optimization objectives in the two steps. In this paper, we conduct an analysis to show the divergence between pre-training and fine-tuning, and to alleviate such divergence, we propose L2P-GNN, a self-supervised pre-training strategy for GNNs. The key insight is that L2P-GNN attempts to learn how to fine-tune during the pre-training process in the form of transferable prior knowledge. To encode both local and global information into the prior, L2P-GNN is further designed with a dual adaptation mechanism at both node and graph levels. Finally, we conduct a systematic empirical study on the pre-training of various GNN models, using both a public collection of protein graphs and a new compilation of bibliographic graphs for pre-training. Experimental results show that L2P-GNN is capable of learning effective and transferable prior knowledge that yields powerful representations for downstream tasks. (Code and datasets are available at https://github.com/rootlu/L2P-GNN.)

1 Introduction

Graph neural networks (GNNs) have emerged as the state of the art for representation learning on graphs, due to their ability to recursively aggregate information from neighborhoods on the graph, naturally capturing both graph structures as well as node or edge features (Zhang, Cui, and Zhu 2020; Wu et al. 2020; Dwivedi et al. 2020). Various GNN architectures with different aggregation schemes have been proposed (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Velickovic et al. 2018; Ying et al. 2018b; Hasanzadeh et al. 2019; Qu, Bengio, and Tang 2019; Pei et al. 2020; Munkhdalai and Yu 2017). Empirically, these GNNs have achieved impressive performance in many tasks, such as node and graph classification (Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017), recommendation systems (Fan et al. 2019; Ying et al. 2018a) and graph generation (Li et al. 2018; You et al. 2018). However, training GNNs usually requires abundant labeled data, which are often limited and expensive to obtain.

Inspired by pre-trained language models (Devlin et al. 2019; Mikolov et al. 2013) and image encoders (Girshick et al. 2014; Donahue et al. 2014; He et al. 2019), recent advances in pre-training GNNs have provided insights into reducing the labeling burden and making use of abundant unlabeled data. The primary goal of pre-training GNNs (Navarin, Tran, and Sperduti 2018; Hu et al. 2019, 2020) is to learn transferable prior knowledge from mostly unlabeled data, which can be generalized to downstream tasks with a quick fine-tuning step. Essentially, those methods mainly follow a two-step paradigm: (1) pre-training a GNN model on a large collection of unlabeled graph data, which derives generic transferable knowledge encoding intrinsic graph properties; (2) fine-tuning the pre-trained GNN model on task-specific graph data, so as to adapt the generic knowledge to downstream tasks. However, here we argue that *there exists a gap between pre-training and fine-tuning* due to the divergence of the optimization objectives in the two steps. In particular, the pre-training step optimizes the GNN to find an optimal point over the pre-training graph data, whereas the fine-tuning step aims to optimize the performance on downstream tasks. In other words, the pre-training process completely disregards the need to quickly adapt to downstream tasks with a few fine-tuning updates, leaving a gap between the two steps. It is inevitable that such divergence will significantly hurt the generalization ability of the pre-trained GNN models.

Challenges and Present Work. In this work, we propose to alleviate the divergence between pre-training and fine-tuning. However, alleviating this divergence is non-trivial, presenting us with two key challenges. (1) *How to narrow the gap caused by different optimization objectives?* Existing pre-training strategies for GNNs fall into a two-step paradigm, and the optimization gap between the two steps significantly limits the ability of pre-trained GNNs to generalize to new downstream tasks. Hence, it is vital to re-examine the objective of the pre-training step to better match that of the fine-tuning step. (2) *How to simultaneously preserve node- and graph-level*

*Part of the work was done while a visiting research student at Singapore Management University.
†The corresponding author.

information with completely unlabeled graph data? Existing methods either only take into account the node-level pre-training (Navarin, Tran, and Sperduti 2018; Hu et al. 2019), or still require supervised information for graph-level pre-training (Hu et al. 2020). While at the node level, predicting links between node pairs is naturally self-supervised, graph-level self-supervision has been seldom explored. Thus, it is crucial to devise a self-supervised strategy to pre-train graph-level representations.

To tackle the challenges, we propose L2P-GNN, a pre-training strategy for GNNs that learns to pre-train (L2P) at both node and graph levels in a fully self-supervised manner. More specifically, for the first challenge, L2P-GNN mimics the fine-tuning step within the pre-training step, and thus learns how to fine-tune during the pre-training process itself. As a result, we learn a prior that possesses the ability of quickly adapting to new downstream tasks with only a few fine-tuning updates. The proposed learning to pre-train can be deemed a form of meta-learning (Finn, Abbeel, and Levine 2017), also known as learning to learn. For the second challenge, we propose a self-supervised strategy with a dual adaptation mechanism, which is equipped with both node- and graph-level adaptations. On one hand, the node-level adaptation takes the connectivity of node pairs as self-supervised information, so as to learn a transferable prior to encode local graph properties. On the other hand, the graph-level adaptation is designed for preserving the global information in the graph, in which a sub-structure should be close to the whole graph in the representation space.

To summarize, this work makes the following major contributions.

- This is the first attempt to explore learning to pre-train GNNs, which alleviates the divergence between pre-training and fine-tuning objectives, and sheds a new perspective for pre-training GNNs.
- We propose a completely self-supervised GNN pre-training strategy for both node- and graph-level representations.
- We build a new large-scale bibliographic graph data for pre-training GNNs, and conduct extensive empirical studies on two datasets in different domains. Experimental results demonstrate that our approach consistently and significantly outperforms the state of the art.

2 Related Work

GNNs have received significant attention due to the prevalence of graph-structured data (Bronstein et al. 2017). Originally proposed (Marco, Gabriele, and Franco 2005; Scarselli et al. 2008) as a framework of utilizing neural networks to learn node representations on graphs, this concept is extended to convolution neural networks using spectral methods (Defferrard, Bresson, and Vandergheynst 2016; Bruna et al. 2014; Levie et al. 2019; Xu et al. 2019a) and message passing architectures to aggregate neighbors' features (Kipf and Welling 2017; Niepert, Ahmed, and Kutzkov 2016; Hamilton, Ying, and Leskovec 2017; Velickovic et al. 2018; Abu-El-Haija et al. 2019). For a more comprehensive understanding of GNNs, we refer readers to the literature (Wu et al. 2020; Battaglia et al. 2018; Zhang, Cui, and Zhu 2020; Zhou et al. 2018). To enable more effective learning on graphs, researchers have explored how to pre-train GNNs for node-level representations on unlabeled graph data. Navarin *et al.* (Navarin, Tran, and Sperduti 2018) utilize the graph kernel for pre-training, while another work (Hu et al. 2019) pre-trains graph encoders with three unsupervised tasks to capture different aspects of a graph. More recently, Hu *et al.* (Hu et al. 2020) propose different strategies to pre-train graph neural networks at both node and graph levels, although labeled data are required at the graph level.

On another line, meta-learning intends to learn a form of general knowledge across similar learning tasks, so that the learned knowledge can be quickly adapted to new tasks (Vilalta and Drissi 2002; Vanschoren 2018; Peng 2020). Among previous works on meta-learning, metric-based methods (Sung et al. 2018; Snell, Swersky, and Zemel 2017) learn a metric or distance function over tasks, while model-based methods (Santoro et al. 2016; Munkhdalai and Yu 2017) aim to design an architecture or training process for rapid generalization across tasks. Finally, some optimization-based methods directly adjust the optimization algorithm to enable quick adaptation with just a few examples (Finn, Abbeel, and Levine 2017; Yao et al. 2019; Lee et al. 2019; Lu, Fang, and Shi 2020).

3 Learning to Pre-train: Motivation and Overview

Our key insight is the observation that there exists a divergence between pre-training and fine-tuning. In this section, we conduct an analysis to demonstrate this divergence, and further motivate a paradigm shift to learning to pre-train GNNs.

3.1 Preliminaries

GNNs. Let $G = (V, E, X, Z)$ denote a graph with nodes V and edges E, where $X \in \mathbb{R}^{|V| \times d_v}$ and $Z \in \mathbb{R}^{|E| \times d_e}$ are node and edge features, respectively. A GNN involves two key computations for each node v at every layer. (1) AGGREGATE operation: aggregating messages from v's neighbors N_v. (2) UPDATE operation: updating v's representation from its representation in the previous layer and the aggregated messages. Formally, the l-th layer representation of node v is given by

$$\begin{aligned} \mathbf{h}_v^l &= (\psi; A, X, Z)^l \\ &= \textsc{Update}(\mathbf{h}_v^{l-1}, \\ &\quad \textsc{Aggregate}(f(\mathbf{h}_v^{l-1}, \mathbf{h}_u^{l-1}, \mathbf{z}_{uv}) : u \in N_v)), \end{aligned} \tag{1}$$

where $\mathbf{z}_{uv}$ is the feature vector of edge (u, v), and $\mathbf{h}_v^0 = \mathbf{x}_v \in X$ is the input layer of a GNN. A denotes the adjacency matrix or some normalized variant, and N_v denotes the neighborhood of node v whose definition depends on a particular GNN variant. We abstract the composition of the two operations as one parameterized function $(\ )$ with parameters ψ.

To address graph-level tasks such as graph classification, node representations need to be further aggregated into a

graph-level representation. The READOUT operation can usually be performed at the final layer as follows:

$$\mathbf{h}_G = \quad (\omega; \mathbf{H}^l) = \text{READOUT}(f\mathbf{h}_v^l j v \ 2 \ Vg), \qquad (2)$$

where $\mathbf{h}_G$ is the representation of the whole graph G, and $\mathbf{H}^l = [\mathbf{h}_v^l]$ is the node representation matrix. READOUT is typically implemented as a simple pooling operation like sum, max or mean-pooling (Atwood and Towsley 2016; Duvenaud et al. 2015) or more complex approaches (Bruna et al. 2014; Ying et al. 2018b). We abstract READOUT as a parameterized function ( ) with parameters ω.

Conventional GNN Pre-training. The goal of pre-training GNNs is to learn a generic initialization for model parameters using readily available graph structures (Hu et al. 2020, 2019). Conventional pre-training strategies largely follow a two-step paradigm. (1) Pre-training a GNN model f (A, X, Z) on a large graph-structured dataset (e.g., multiple small graphs or a large-scale graph). The learned parameter θ_0 is expected to capture task-agnostic transferable information. (2) Fine-tuning the pre-trained GNN on downstream tasks. With multiple (say, n) gradient descent steps over the training data of the downstream task, the model aims to obtain the optimal parameters θ_n on the downstream task. Note that, for node-level tasks, the GNN model is $f = (\psi; A, X, Z)$, i.e., $\theta = \psi$; for graph-level tasks, the GNN model is $f = (\omega; (\psi; A, X, Z))$, i.e., $\theta = f\psi, \omega g$.

Let D^{pre} denote the pre-training graph data, and L^{pre} be the loss function for pre-training. That is, the objective of pre-training is to optimize the following:

$$\theta_0 = \arg\min \ L^{pre}(f; D^{pre}). \qquad (3)$$

On the other hand, the fine-tuning process aims to maximize the performance on the testing graph data D^{te} of the downstream task, after fine-tuning over the training graph data D^{tr} of the task. The so-called fine-tuning initializes the model from the pre-trained parameters θ_0, and updates the GNN model f with multiple gradient descent steps over (usually batched) D^{tr}. Taking one step as an example, we have

$$\theta_1 = \theta_0 \quad \eta r \ {}_0 L^{fine}(f \ {}_0; D^{tr}), \qquad (4)$$

where L^{fine} is the loss function of fine-tuning and η is the learning rate.

3.2 Learning to Pre-train GNNs

In the conventional two-step paradigm, the pre-training step is decoupled from the fine-tuning step. In particular, θ_0 is pre-trained without accommodating any form of adaptation that are potentially useful for future fine-tuning on downstream tasks. The apparent divergence between the two steps would result in suboptimal pre-training. To narrow the gap between pre-training and fine-tuning, it is important to *learn how to pre-train* such that the pre-trained model becomes more amenable to adaptations on future downstream tasks. To this end, we propose to structure the pre-training stage to simulate the fine-tuning process on downstream tasks, so as to directly optimize the pre-trained model's quick adaptability to downstream tasks.

Specifically, to pre-train a GNN model over a graph $G \ 2 \ D^{pre}$, we sample some sub-structures from G, denoted $D^{tr}_{T_G}$, as the training data of a *simulated* downstream task T_G; similarly, we mimic the evaluation on testing sub-structures $D^{te}_{T_G}$ that are also sampled from G. Training and testing data are simulated here since the actual downstream task is unknown during pre-training. This setup is reasonable as our goal is learning how to pre-train a GNN model with the ability of adapting to new tasks quickly, rather than directly learning the actual downstream task.

Formally, our pre-training aims to learn a GNN model f, such that after fine-tuning it on the simulated task training data $D^{tr}_{T_G}$, the loss on the simulated testing data $D^{te}_{T_G}$ is minimized. That is,

$$\theta_0 = \arg\min \ \textstyle\sum_{G 2 D^{pre}} L^{pre}(f \quad r_\theta L^{pre}(f_\theta; D^{tr}_{T_G}); D^{te}_{T_G}), \qquad (5)$$

where $\theta \quad \alpha r \ L^{pre}(f; D^{tr}_{T_G})$ is the fine-tuned parameters on $D^{tr}_{T_G}$ (still part of the pre-training data), in a similar manner as the fine-tuning step on the downstream task in Eq. (4). Moreover, α represents the learning rate of the fine-tuning on $D^{tr}_{T_G}$, which can be fixed as a hyper-parameter. Thus, the output of our pre-training, θ_0, is not intended to directly optimize the training or testing data of any particular task. Instead, θ_0 is optimal in the sense that it allows for quick adaptation to new tasks in general. Note that here we only show one gradient update, and yet employing multiple updates is a straightforward extension.

Connection to other works. Interestingly, our proposed strategy of learning to pre-train GNNs subsumes the conventional GNN pre-training as a special case. In particular, if we set $\alpha = 0$, i.e., there is no fine-tuning on $D^{tr}_{T_G}$, our strategy becomes equivalent to conventional pre-training approaches. Furthermore, our strategy is a form of meta-learning, in particular, model agnostic meta-learning (MAML) (Finn, Abbeel, and Levine 2017). Meta-learning aims to learn prior knowledge from a set of training tasks that can be transferred to testing tasks. Specifically, MAML learns a prior that can be quickly adapted to new tasks by one or a few gradient updates, so that the prior, after being adapted to the so-called support set of each task, can achieve optimal performance on the so-called query set of the task. In our case, the output of our pre-training θ_0 is the prior knowledge that can quickly adapt to new downstream tasks, while $D^{tr}_{T_G}$ and $D^{te}_{T_G}$ correspond to the support and query sets in MAML, respectively.

4 Proposed Method

In the following, we introduce our approach L2P-GNN. We first present a self-supervised base GNN model for learning graph structures in the MAML setting, followed by our dual node- and graph-level adaptations designed to simulate fine-tuning during the pre-training process.

4.1 Self-supervised Base Model

At the core of L2P-GNN is the notion of learning to pre-train a GNN to bridge the gap between the pre-training and fine-tuning processes. Specifically, our approach can be formulated as a form of MAML. To this end, we define a task as

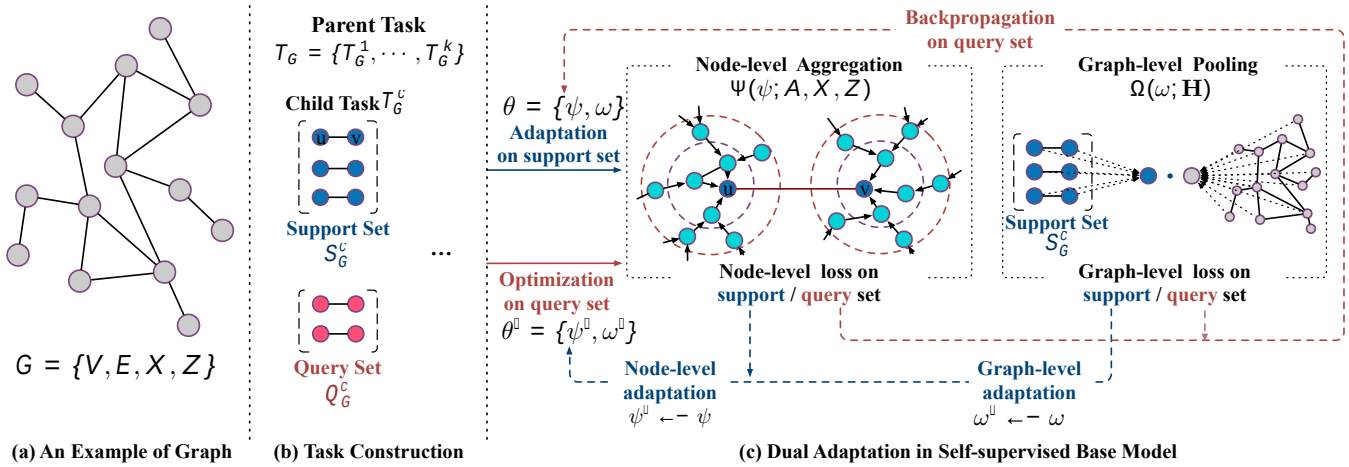

Figure 1: Illustration of L2P-GNN. (a/b) Task construction for a graph, where the graph G is associated with a parent task T_G consisting of k child tasks $\{T_G^1, \cdots, T_G^k\}$. (c) Dual node- and graph-level adaptations on the support set, and the optimization of transferable prior θ on the query set.

capturing structures and attributes in a graph, from both local and global perspectives. The meta-learned prior can then be adapted to a new task or graph.

Task Construction. Consider a set of graphs as the pre-training data, $\mathcal{D}^{pre} = \{G_1, G_2, \cdots, G_N\}$. A task $T_G = (S_G, Q_G)$ involves a graph G, consisting of a support set S_G and a query set Q_G. We learn the prior such that, after updating by gradient descent w.r.t. the loss on the support set, it optimizes the performance on the query set, which simulates the training and testing in the fine-tuning step.

As illustrated in Figs. 1(a) and (b), to promote both global and local perspectives of a graph, its corresponding task T_G is designed to contain k child tasks, i.e., $T_G = \{T_G^1, T_G^2, \cdots, T_G^k\}$. Each child task T_G^c attempts to capture a local aspect of G, which is defined as

$$T_G^c = \{S_G^c = \{(u,v) \sim p_E\}, Q_G^c = \{(p,q) \sim p_E\}\} \quad (6)$$
$$\text{s.t.} \quad S_G^c \cap Q_G^c = \emptyset,$$

where the support S_G^c and query Q_G^c contain edges randomly sampled from the edge distribution p_E of the graph, and they are mutually exclusive. In essence, child tasks incorporate the local connectivity between node pairs in a graph, and they fuse into a parent task $T_G = (S_G, Q_G)$ to facilitate a graph-level view, where $S_G = \{S_G^1, S_G^2, \cdots, S_G^k\}$ and $Q_G = \{Q_G^1, Q_G^2, \cdots, Q_G^k\}$.

Base GNN Model. Given the parent and child tasks, we design a self-supervised base GNN model with node-level aggregation and graph-level pooling to learn node and graph representations, respectively. The key idea is to utilize the intrinsic structures of label-free graph data as self-supervision, at both node and graph levels. Specifically, the base model f involves *node-level aggregation* $\Psi(\psi; A, X, Z)$ that aggregates node information (e.g., its local structures and attributes) to generate node representations, and *graph-level pooling* $\Omega(\omega; \mathbf{H})$ that further generates a graph-level representation given the node representation matrix $\mathbf{H}$.

In node-level aggregation, the node embeddings are aggregated from their neighborhoods, as defined in Eq. (1). That is, for each node $v \in G$, we obtain its representation $\mathbf{h}_v$ after l iteration of $\Psi(\cdot)$. Subsequently, given a support edge (u, v) in a child task T_G^c, we optimize the self-supervised objective of predicting the link between u and v (Tang et al. 2015; Hamilton, Ying, and Leskovec 2017), as follows.

$$\mathcal{L}^{node}(\psi; S_G^c) = \sum_{(u,v) \in S_G^c} \ln(\sigma(\mathbf{h}_u^\top \mathbf{h}_v)) + \ln(\sigma(-\mathbf{h}_u^\top \mathbf{h}_{v'})), \quad (7)$$

where v' is a negative node sample that is not linked with node u in the graph, σ is the *sigmoid* function, and ψ denotes the learnable parameters of $\Psi(\cdot)$. The node-level aggregation encourages linked nodes in the support set of the child tasks to have similar representations.

In graph-level pooling, the graph representation $\mathbf{h}_G$ is gathered from node representations with the pooling function defined in Eq. (2). As the child tasks capture various local sub-structures of the graph, we also perform pooling on the support nodes of each child task T_G^c to generate the pooled representation $\mathbf{h}_{S_G^c} = \Omega(\omega; \{\mathbf{h}_u | u, v : (u,v) \in S_G^c\})$. Given a parent task $T_G = (S_G, Q_G)$ which is a fusion of all child tasks, we define the following self-supervised graph-level objective:

$$\mathcal{L}^{graph}(\omega; S_G) = \sum_{c=1}^{k} \log(\sigma(\mathbf{h}_{S_G^c}^\top \mathbf{h}_G)) + \log(\sigma(-\mathbf{h}_{S_G^c}^\top \mathbf{h}_{G'})), \quad (8)$$

where ω denotes the learnable parameters of $\Omega(\cdot)$, and $\mathbf{h}_{G'}$ denotes the shifted graph representation by randomly shifting some dimensions of $\mathbf{h}_G$ (Velickovic et al. 2019), serving as the negative sample.

Altogether, to capture both node- and graph-level information, we minimize the following loss for a graph G:

$$L_{T_G}(\theta; S_G) = L^{graph}(\omega; S_G) + \frac{1}{k}\sum_{c=1}^{k} L^{node}(\psi; S_G^c), \quad (9)$$

where $\theta = \{\psi, \omega\}$ is the learnable parameters of our self-supervised base GNN model.

4.2 Dual Adaptation

As motivated, to bridge the gap between the pre-training and fine-tuning processes, it is crucial to optimize the model's ability of quickly adapting to new tasks during pre-training itself. To this end, we propose learning to pre-train the base GNN model: we aim to learn transferable prior knowledge (i.e., $\theta = \{\psi, \omega\}$), to provide an adaptable initialization that can be quickly fine-tuned for new downstream tasks with new graph data. In particular, the learned initialization should not only encode and adapt to the local connectivity between node pairs, but also become capable of generalizing to different sub-structures of the graphs. Correspondingly, we devise the dual node- and graph-level adaptations, as illustrated in Fig. 1(c).

Node-level Adaptation. To simulate the procedure of fine-tuning on training data, we calculate the loss on the support set S_G^c in each child task T_G^c as shown in Eq. (7). Then, we adapt the node-level aggregation prior ψ w.r.t. the support loss with one or a few gradient descent step, to obtain the adapted prior ψ^0 for the child tasks. For instance, when using one gradient update with a node-level learning rate α, we have

$$\psi^0 = \psi - \alpha\frac{\partial \sum_{c=1}^{k} L^{node}(\psi; S_G^c)}{\partial \psi}. \quad (10)$$

Graph-level Adaptation. To encode how to pool node information for representing a graph, we adapt the graph-level pooling prior ω to a parent task T_G with one (or a few) gradient descent step. Given β as the graph-level learning rate, the adapted pooling prior is given by

$$\omega^0 = \omega - \beta\frac{\partial L^{graph}(\omega; S_G)}{\partial \omega}. \quad (11)$$

Optimization of Transferable Prior. With the node- and graph-level adaptations, we have adapted the prior θ to $\theta^0 = \{\psi^0, \omega^0\}$ that is specific to the task T_G. To mimic the testing process with the fine-tuned model, the base GNN model is trained by optimizing the performance of the adapted parameters θ^0 on the query set Q_G over all training tasks or graphs in D^{pre}. That is, the transferable prior $\theta = \{\psi, \omega\}$ will be optimized through the backpropgation of the query loss given by

$$\sum_{G \in D^{pre}} L_{T_G}(\theta^0; Q_G). \quad (12)$$

In other words, we can update the prior θ as follows.

$$\theta \leftarrow \theta - \gamma\frac{\partial \sum_{G \in D^{pre}} L_{T_G}(\theta^0; Q_G)}{\partial \theta}, \quad (13)$$

where γ is the learning rate of the prior. The detailed training procedure is provided in Appendix A.

Table 1: Statistics of the two datasets.

Dataset	Biology	PreDBLP
#subgraphs	394,925	1,054,309
#labels	40	6
#subgraphs for pre-training	306,925	794,862
#subgraphs for fine-tuning	88,000	299,447

4.3 Discussion

Here we give an analysis of the proposed L2P-GNN with respect to model generality and efficiency.

Firstly, the proposed L2P-GNN is generalized and can be easily applied to different graph neural networks. Sec. 3.2 demonstrates the divergence between pre-training and fine-tuning, which is widely known in the literature (Lv et al. 2020; Gururangan et al. 2020), whether it is on the graph data, or in natural language process or computer vision. L2P-GNN directly optimizes the pre-trained model's quick adaptability to downstream tasks by simulating the fine-tuning process on downstream tasks, making it free from the architectures of graph neural networks.

Secondly, our L2P-GNN is efficient and can be parallelized for large-scale datasets. In L2P-GNN, for task construction, the time complexity is linear w.r.t. the number of edges as each task contains edges sampled from the graph. For dual adaptation, the time complexity depends on the architecture of the GNN, which is at most k (i.e., number of child tasks) times the complexity of the corresponding GNN. As the number of child task k is usually small, the complexity of L2P-GNN is as efficient as other pre-training strategies for GNNs. Besides, with on-the-fly transformation of data (e.g., task construction), there is almost no memory overhead for our L2P-GNN. Detailed pseudocode of the algorithm is in supplemental material, Appendix A.

5 Experiments

In this section, we present a new graph dataset for pre-training, and compare the performance of our approach and various state-of-the-art pre-training baselines. Lastly, we conduct thorough model analysis to support the motivation and design of our pre-training strategy.

5.1 Experimental Settings

Datasets. We conduct experiments on data from two domains: biological function prediction in biology (Hu et al. 2020) and research field prediction in bibliography. The biology graphs come from a public repository[1], covering 394,925 protein subgraphs (Marinka et al. 2019). We further present a new collection of bibliographic graphs called PreDBLP, purposely compiled for pre-training GNNs based on DBLP[2], which contains 1,054,309 paper subgraphs in 31 fields (e.g., artificial intelligence, data mining). Each subgraph is centered at a paper and contains the associated information of

[1] http://snap.stanford.edu/gnn-pretrain

[2] https://dblp.uni-trier.de

Table 2: Experimental results (mean std in percent) of different pre-training strategies w.r.t. various GNN architectures. The improvements are relative to the respective GNN without pre-training.

Model	Biology								PreDBLP							
	GCN		GraphSAGE		GAT		GIN		GCN		GraphSAGE		GAT		GIN	
No pre-train	63.22	1.06	65.72	1.23	68.21	1.26	64.82	1.21	62.18	0.43	61.03	0.65	59.63	2.32	69.01	0.23
EdgePred	64.72	1.06	67.39	1.54	67.37	1.31	65.93	1.65	65.44	0.42	63.60	0.21	55.56	1.67	69.43	0.07
DGI	64.33	1.14	66.69	0.88	68.37	0.54	65.16	1.24	65.57	0.36	63.34	0.73	61.30	2.17	69.34	0.09
ContextPred	64.56	1.36	66.31	0.94	66.89	1.98	65.99	1.22	66.11	0.16	62.55	0.11	58.44	1.18	69.37	0.21
AttrMasking	64.35	1.23	64.32	0.78	67.72	1.16	65.72	1.31	65.49	0.52	62.35	0.58	53.34	4.77	68.61	0.16
L2P-GNN	**66.48**	1.59	**69.89**	1.63	**69.15**	1.86	**70.13**	0.95	**66.58**	0.28	**65.84**	0.37	**62.24**	1.89	**70.79**	0.17
(Improv.)	(5.16%)		(6.35%)		(1.38%)		(8.19%)		(7.08%)		(7.88%)		(4.38 %)		(2.58%)	

the paper. The new bibliographic dataset is publicly released, while more detailed descriptions on the construction or processing of the datasets are included in Appendix B.

For biology data, as in (Hu et al. 2020), we use 306,925 unlabeled protein ego-networks for pre-training. In fine-tuning, we predict 40 fine-grained biological functions with 88,000 labeled subgraphs that correspond to 40 binary classification tasks. We split the downstream data with species split (Hu et al. 2020), and evaluate the test performance with average ROC-AUC (Bradley 1997) across the 40 tasks. For PreDBLP, we utilize 794,862 subgraphs to pre-train a GNN model. In fine-tuning, we predict the research field with 299,447 labeled subgraphs from 6 different categories. We randomly split the downstream data and evaluate test performance with micro-averaged F1 score. For both domains, we split downstream data with 8:1:1 ratio for train/validation/test sets. All downstream experiments are repeated with 10 random seeds, and we report the mean with standard deviation. The detailed statistics of two datasets are summarized in Table 1.

Baselines. To contextualize the empirical results of L2P-GNN on the pre-training benchmarks, we compare against four self-supervised or unsupervised baselines: (1) the original Edge Prediction (denoted by EdgePred) (Hamilton, Ying, and Leskovec 2017) to predict the connectivity of node pairs; (2) Deep Graph Infomax (denoted by DGI) (Velickovic et al. 2019) to maximize local mutual information across the graph's patch representations; (3) Context Prediction strategy (denoted by ContextPred) (Hu et al. 2020) to explore graph structures and (4) Attribute Masking strategy (denoted by AttrMasking) (Hu et al. 2020) to learn the regularities of the node and edge attributes distributed over graphs. Further details are provided in Appendix D.

GNN Architectures and Parameter Settings. All pre-training baselines and our L2P-GNN can be implemented for different GNN architectures. We experiment with four popular GNN architectures, namely, GCN (Kipf and Welling 2017), GraphSAGE (Hamilton, Ying, and Leskovec 2017), GAT (Velickovic et al. 2018) and GIN (Xu et al. 2019b). Implementation details are presented in Appendix C. We tune hyper-parameters w.r.t. the model performance on validation sets. The hyper-parameter settings and experimental environment are discussed in Appendix D.

5.2 Performance Comparison

Table 2 compares the performance of L2P-GNN and state-of-the-art pre-training baselines, w.r.t. four different GNN architectures. We make the following observations. (1) Overall, the proposed L2P-GNN consistently yields the best performance among all methods across architectures. Compared to the best baseline on each architecture, L2P-GNN achieves up to 6.27% and 3.52% improvements on the two datasets, respectively. We believe that such significant improvements can be attributed to the simulation of fine-tuning during the pre-training process, thereby narrowing the gap between pre-training and fine-tuning objectives. (2) Furthermore, pre-training GNNs with abundant unlabeled data is clearly helpful to downstream tasks, as our L2P-GNN brings up to 8.19% and 7.88% gains relative to non-pretrained models on the two datasets, respectively. (3) We also notice that some baselines give surprisingly limited performance gain and yield negative transfer (Rosenstein et al. 2005) on the downstream task (i.e., EdgePred and AttrMasking strategies w.r.t. the GAT model). The reason might be that these strategies learn information irrelevant to the downstream tasks, which harms the generalization of the pre-trained GNNs. This finding confirms previous observations (Hu et al. 2020; Rosenstein et al. 2005) that negative transfer results in limitations on the applicability and reliability of pre-trained models.

5.3 Model Analysis

Next, we investigate the underlying mechanism of L2P-GNN: the capability to narrow the gap between pre-training and fine-tuning by learning to pre-train GNNs, the impact of the node and graph-level adaptations on L2P-GNN's performance and a parameter analysis. Since similar trends are observed for different GNN architectures, here we only report the results w.r.t. the GIN model.

Comparative Analysis. We attempt to validate whether L2P-GNN narrows the gap between pre-training and fine-tuning by learning to pre-train GNNs. Towards this end, we conduct a comparative analysis of the pre-trained GNN model before and after fine-tuning on downstream tasks (named Model-P and Model-F), and consider three perspectives for comparison: Centered Kernel Alignment (CKA) similarity (Kornblith et al. 2019) between the parameters of Model-P

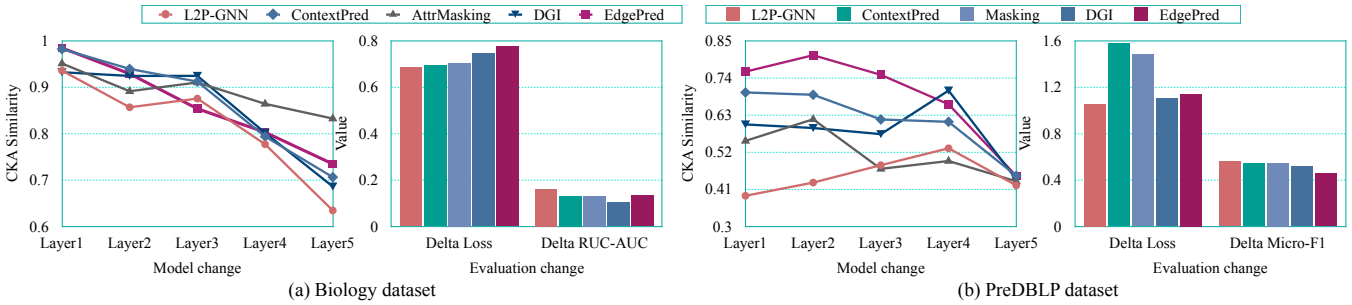

Figure 2: CKA similarity of GIN layers and changes of loss and performance on two datasets.

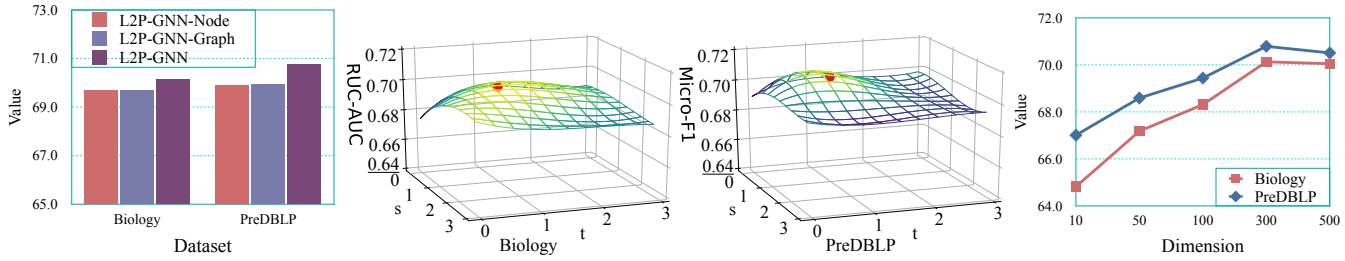

Figure 3: Model analysis w.r.t. the GIN model.

and Model-F, changes in training loss (delta loss) and testing performance on downstream tasks (delta RUC-AUC or Micro-F1). As presented in Fig. 2, we observe that the CKA similarities of our L2P-GNN parameters before and after fine-tuning are generally smaller than those of the baselines, indicating that L2P-GNN undergoes larger changes so as to become more adapted to downstream tasks. Besides, the smaller changes in training loss show that L2P-GNN can easily achieve the optimal point of the new tasks by rapid adaptations. This further implies that the objectives of our pre-training and downstream tasks are more aligned, resulting in a quick adaptation in the right optimization direction for downstream tasks and a much more significant improvement in testing performance. Thus, L2P-GNN indeed narrows the gap by learning how to make adaptations during the pre-training process.

Ablation Study. As the node- and graph-level adaptations play pivotal roles in L2P-GNN, we compare two ablated variants, namely L2P-GNN-Node (with only node-level adaptation) and L2P-GNN-Graph (with only graph-level adaptation). As reported in Fig. 3(a), L2P-GNN is superior to both variants on the two datasets. The results demonstrate that both the local node-level structures and global graph-level information are useful and it is beneficial to model them jointly.

Parameter Analysis. Lastly, we investigate the effect of the number of node- and graph-level adaptation steps (s, t), as well as the dimension of node representations. We plot the performance of L2P-GNN under combinations of $0 \quad s \quad 3$ and $0 \quad t \quad 3$ in Fig. 3(b). We find that L2P-GNN is robust to different values of s and t, except when one or both of them are zero (i.e., no adaptation at all). In particular, L2P-GNN can adapt quickly with only one gradient update in both adaptions (i.e., $s = t = 1$). Finally, we summarize the impact of the dimension in Fig. 3(c). We observe that L2P-GNN achieves the optimal performance when the dimension is 300 and is generally stable around the optimal setting, indicating that L2P-GNN is robust w.r.t. the representation dimension.

6 Conclusion

In this paper, we introduce L2P-GNN, a self-supervised pre-training strategy for GNNs. We find that with conventional pre-training strategies, there exists a divergence between the pre-training and fine-tuning objectives, resulting in suboptimal pre-trained GNN models. To narrow the gap by learning how to pre-train GNNs, L2P-GNN structures the pre-training step to simulate the fine-tuning process on downstream tasks, so as to directly optimize the pre-trained model's quick adaptability to downstream tasks. At both node and graph levels, L2P-GNN is equipped with dual adaptations to utilize the intrinsic structures of label-free graph data as self-supervision to learn local and global representations simultaneously. Extensive experiments demonstrate that L2P-GNN significantly outperforms the state of the art and effectively narrows the gap between pre-training and fine-tuning.

7 Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No. U20B2045, 61772082, 61702296, 62002029), and the Tencent WeChat Rhino-Bird Focused Research Program.

References

Abu-El-Haija, S.; Perozzi, B.; Kapoor, A.; Alipourfard, N.; Lerman, K.; Harutyunyan, H.; Steeg, G. V.; and Galstyan, A. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *Proceedings of ICML*, 21–29.

Atwood, J.; and Towsley, D. 2016. Diffusion-Convolutional Neural Networks. In *Proceedings of NeurIPS*, 1993–2001.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V. F.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; Gülçehre, Ç.; Song, H. F.; Ballard, A. J.; Gilmer, J.; Dahl, G. E.; Vaswani, A.; Allen, K. R.; Nash, C.; Langston, V.; Dyer, C.; Heess, N.; Wierstra, D.; Kohli, P.; Botvinick, M.; Vinyals, O.; Li, Y.; and Pascanu, R. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261.

Bradley, A. P. 1997. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognit.* 30(7): 1145–1159.

Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; and Vandergheynst, P. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Process. Mag.* 34(4): 18–42.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *Proceedings of ICLR*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Proceedings of NeurIPS*, 3837–3845.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*, 4171–4186.

Donahue, J.; Jia, Y.; Vinyals, O.; Hoffman, J.; Zhang, N.; Tzeng, E.; and Darrell, T. 2014. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *Proceedings ICML*, 647–655.

Duvenaud, D.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gómez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Proceedings of NeurIPS*, 2224–2232.

Dwivedi, V. P.; Joshi, C. K.; Laurent, T.; Bengio, Y.; and Bresson, X. 2020. Benchmarking Graph Neural Networks. *arXiv preprint arXiv:2003.00982* .

Fan, W.; Ma, Y.; Li, Q.; He, Y.; Zhao, Y. E.; Tang, J.; and Yin, D. 2019. Graph Neural Networks for Social Recommendation. In *Proceedings of WWW*, 417–426.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of ICML*, 1126–1135.

Girshick, R. B.; Donahue, J.; Darrell, T.; and Malik, J. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of CVPR*, 580–587.

Gururangan, S.; Marasovic, A.; Swayamdipta, S.; Lo, K.; Beltagy, I.; Downey, D.; and Smith, N. A. 2020. Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. In Jurafsky, D.; Chai, J.; Schluter, N.; and Tetreault, J. R., eds., *Proceedings of ACL*, 8342–8360.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *Proceedings of NeurIPS*, 1025–1035.

Hasanzadeh, A.; Hajiramezanali, E.; Narayanan, K. R.; Duffield, N.; Zhou, M.; and Qian, X. 2019. Semi-Implicit Graph Variational Auto-Encoders. In *Proceedings of NeurIPS*, 10711–10722.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. B. 2019. Momentum Contrast for Unsupervised Visual Representation Learning. *CoRR* abs/1911.05722.

Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V. S.; and Leskovec, J. 2020. Strategies for Pre-training Graph Neural Networks. In *Proceedings of ICLR*.

Hu, Z.; Fan, C.; Chen, T.; Chang, K.; and Sun, Y. 2019. Pre-Training Graph Neural Networks for Generic Structural Feature Extraction. *CoRR* abs/1905.13728.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of ICLR*.

Kornblith, S.; Norouzi, M.; Lee, H.; and Hinton, G. E. 2019. Similarity of Neural Network Representations Revisited. In *Proceedings of ICML*, 3519–3529.

Lee, K.; Maji, S.; Ravichandran, A.; and Soatto, S. 2019. Meta-learning with differentiable convex optimization. In *Proceedings of CVPR*, 10657–10665.

Levie, R.; Monti, F.; Bresson, X.; and Bronstein, M. M. 2019. CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters. *IEEE Trans. Signal Process.* 67(1): 97–109.

Li, Y.; Vinyals, O.; Dyer, C.; Pascanu, R.; and Battaglia, P. W. 2018. Learning Deep Generative Models of Graphs. *CoRR* abs/1803.03324.

Lu, Y.; Fang, Y.; and Shi, C. 2020. Meta-learning on Heterogeneous Information Networks for Cold-start Recommendation. In *Proceedings of KDD*, 1563–1573.

Lv, S.; Wang, Y.; Guo, D.; Tang, D.; Duan, N.; Zhu, F.; Gong, M.; Shou, L.; Ma, R.; Jiang, D.; et al. 2020. Pre-training Text Representations as Meta Learning. *arXiv preprint arXiv:2004.05568* .

Marco, G.; Gabriele, M.; and Franco, S. 2005. A new model for learning in graph domains. In *Proceedings of IJCNN*, volume 2, 729–734.

Marinka; Zitnik; Rok; Sosič; Marcus; W; Feldman; Jure; and Leskovec. 2019. Evolution of resilience in protein interactomes across the tree of life. *Proceedings of the National Academy of Sciences of the United States of America* .

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and

phrases and their compositionality. In *Proceedings of NeurIPS*, 3111–3119.

Munkhdalai, T.; and Yu, H. 2017. Meta networks. In *Proceedings of ICML*, 2554–2563.

Navarin, N.; Tran, D. V.; and Sperduti, A. 2018. Pre-training Graph Neural Networks with Kernels. *CoRR* abs/1811.06930.

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning Convolutional Neural Networks for Graphs. In Balcan, M.; and Weinberger, K. Q., eds., *Proceedings of ICML*, 2014–2023.

Pei, H.; Wei, B.; Chang, K. C.; Lei, Y.; and Yang, B. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *Proceedings of ICLR*.

Peng, H. 2020. A Comprehensive Overview and Survey of Recent Advances in Meta-Learning. *CoRR* abs/2004.11149.

Qu, M.; Bengio, Y.; and Tang, J. 2019. GMNN: Graph Markov Neural Networks. In *Proceedings of ICML*, 5241–5250.

Rosenstein, M. T.; Marx, Z.; Kaelbling, L. P.; and Dietterich, T. G. 2005. To transfer or not to transfer. In *Proceedings of NeurIPS*, 1—-4.

Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T. 2016. Meta-learning with memory-augmented neural networks. In *Proceedings of ICML*, 1842–1850.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1): 61–80.

Snell, J.; Swersky, K.; and Zemel, R. 2017. Prototypical networks for few-shot learning. In *Proceedings of NeurIPS*, 4077–4087.

Sung, F.; Yang, Y.; Zhang, L.; Xiang, T.; Torr, P. H.; and Hospedales, T. M. 2018. Learning to compare: Relation network for few-shot learning. In *Proceedings of CVPR*, 1199–1208.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *Proceedings of WWW*, 1067–1077.

Vanschoren, J. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* .

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *Proceedings of ICLR*.

Velickovic, P.; Fedus, W.; Hamilton, W. L.; Liò, P.; Bengio, Y.; and Hjelm, R. D. 2019. Deep Graph Infomax. In *Proceedings of ICLR*.

Vilalta, R.; and Drissi, Y. 2002. A perspective view and survey of meta-learning. *Artificial intelligence review* 18(2): 77–95.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Philip, S. Y. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* .

Xu, B.; Shen, H.; Cao, Q.; Qiu, Y.; and Cheng, X. 2019a. Graph Wavelet Neural Network. In *Proceedings of ICLR*.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019b. How Powerful are Graph Neural Networks? In *Proceedings of ICLR*.

Yao, H.; Wei, Y.; Huang, J.; and Li, Z. 2019. Hierarchically Structured Meta-learning. In *Proceedings of ICML*, 7045–7054.

Ying, R.; He, R.; Chen, K.; Eksombatchai, P.; Hamilton, W. L.; and Leskovec, J. 2018a. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of SIGKDD*, 974–983.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018b. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Proceedings of NeurIPS*, 4805–4815.

You, J.; Ying, R.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. GraphRNN: A Deep Generative Model for Graphs. *CoRR* abs/1802.08773.

Zhang, Z.; Cui, P.; and Zhu, W. 2020. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* .

Zhou, J.; Cui, G.; Zhang, Z.; Yang, C.; Liu, Z.; and Sun, M. 2018. Graph Neural Networks: A Review of Methods and Applications. *CoRR* abs/1812.08434.

Appendices

A Pseudocode of L2P-GNN

The pseudocode of the pre-training procedure for L2P-GNN is outlined in Algorithm 1. The training of L2P-GNN involves the initialization of parameters, task construction, as well as node-level and graph-level adaptations. At the beginning (Line 1), we randomly initialize all learnable parameters θ in our L2P-GNN, including the node-level aggregation parameters ψ and graph-level pooling parameters ω. Then, we construct child tasks and the parent task for the graph G. Each child task consists of a support set and a query set, each of which contains edges randomly sampled from the edge distribution p_E of the graph (Line 2). In each training iteration, for each child task T_G^c, we perform node-level adaptation on the support set (Line 4–8). Furthermore, we conduct graph-level adaptation with the sub-structure and whole graph representations (Line 9–12). At last, we update all learnable parameters in L2P-GNN (Line 13). The process stops when the model converges.

Algorithm 1 Pre-training of L2P-GNN

Require: a pre-training graph $G = (V, E, X, Z)$; node and graph-level update steps: s and t; node-level, graph-level and prior learning rates: α, β and γ; number of child-tasks: k; support set and query set size: m and n.

1: Randomly initialize GNN parameters $\theta = \{\psi, \omega\}$
2: Construct child tasks and the parent task for the graph G by Eq. (6): $T_G = (T_G^1, T_G^2, \quad, T_G^k)$, each child task T_G^c consisting of a support S_G^c and a query set Q_G^c, and $S_G^c = \{(u, v) \quad p_E\}, Q_G^c = \{(p, q) \quad p_E\}$.
3: **while** not done **do**
4: **for all** child task T_G^c w.r.t. graph G **do**
5: Compute node representation $\mathbf{h}_v^l$ by Eq. (1) for all nodes in support set S_G^c
6: Evaluate $L^{node}(\psi, S_G^c)$ by Eq. (7)
7: Node-level adaptation by Eq. (10) with s updates
8: **end for**
9: Compute sub-structure representation with $\mathbf{h}_{S_G^c} = \left(\omega; \{\mathbf{h}_u | \forall u, \exists v : (u, v) \in S_G^c\}\right)$
10: Compute the whole graph representation $\mathbf{h}_G$ by Eq. (2)
11: Evaluate $L^{graph}(\omega, S_G)$ by Eq. (8)
12: Graph-level adaptation by Eq. (11) with t updates
13: Update all learnable parameters θ in L2P-GNN by Eq. (9)
14: **end while**

B Details and Processing of Datasets

We conduct experiments on two large-scale datasets, including biology graphs (called Biology) and bibliographic graphs (called PreDBLP). Here we provide more details of the two datasets and any additional processing done.

Biology. Biology dataset comes from a public repository[3], covering 394,925 protein subgraphs. Following earlier work (Hu et al. 2020), we perform biological function prediction on the Biology data. Detailed information about the dataset can be found in Appendix D of the original paper (Hu et al. 2020).

PreDBLP. To enrich graph pre-training data from a different domain, we further present PreDBLP, a new compilation of bibliographic graphs. We derive the new PreDBLP data from AMiner[4] and DBLP[5]. Specifically, PreDBLP contains 1,054,309 paper subgraphs in 31 fields (e.g., artificial intelligence, data mining). Each subgraph is centered at a paper and contains the associated information of the paper.

The original Aminer/DBLP contains both the records of each paper and the implicit relations between papers, authors, venues and keywords. For each paper record in the Aminer/DBLP data, we generate a subgraph centered on the paper as follows: (1) according to the citation relationship, we perform a breadth-first search to select the subgraph nodes, with a search depth limit of 2 and a maximum number of 10 neighbors randomly expanded per node; (2) we include the selected paper nodes and all the edges between those paper nodes into the subgraph; (3) we convert the authors attached to each paper's record to nodes as well, and link them to the paper; (4) we utilize the same procedure as in (3) to incorporate the information of venues and keyword terms. As a result, each subgraph compiled contains four types of nodes (i.e., paper, author, venue and keywords) and edges (i.e., paper-paper, paper-author, paper-venue, paper-keywords).

We further utilize a set of node and edge features for the subgraph. For each subgraph, we set the node/edge features as their corresponding types. For instance, for nodes u and v connected via edge (u, v), the feature of u and v are their respective type and that of edge (u, v) is the type of (u, v).

During the pre-training process, we utilize 794,862 subgraphs that belong to 25 research fields to pre-train a GNN model. On average, each subgraph contains 262.43 nodes and 900.07 edges. In fine-tuning, we predict the research field of 299,447 labeled subgraphs from the remaining 6 research fields, including: *Artificial intelligence* (86,956 subgraphs), *Computational linguistics* (20,024 subgraphs), *Computer Vision* (95,729 subgraphs), *Data mining* (14,934 subgraphs), *Databases* (68,287 subgraphs) and *Fuzzy systems* (13,517 subgraphs).

C Implementation details of GNN Models

Here, we introduce the GNN architecture used in biological function prediction on Biology and research field prediction on PreDBLP. For both experiments, we utilize the GIN architectures (Xu et al. 2019b) as an example to explain how to incorporate the node features and edge features in the subgraphs.

Biological Function Prediction. Following previous work (Hu et al. 2020), the raw node features are uniform and the raw input edge features are binary vectors since the protein subgraphs only have edge features. We adopt the same GNN architecture as in (Hu et al. 2020) for protein function predic-

[3] http://snap.stanford.edu/gnn-pretrain

[4] https://www.aminer.cn/citation

[5] https://dblp.uni-trier.de

tion. Detailed implementation please refer to the Appendix A in (Hu et al. 2020).

Research Field Prediction. In research field prediction, the raw node features are 4-dimensional one-hot vectors, denoted as $\mathbf{x}_v \in \mathbb{R}^4$ for node v. The raw edge features are 1-dimensional type vector indicting the type of edge, denoted as $\mathbf{z}_{uv} \in \mathbb{R}^1$ (see Appendix B for details). As input features to GNNs, we first embed the feature vectors by

$$\mathbf{h}_v^0 = \mathbf{W}^{node}\mathbf{x}_u + \mathbf{b}^{node} \tag{A.1}$$

$$\mathbf{h}_{e_{uv}}^l = \mathbf{W}^{edge}\mathbf{z}_{uv} + \mathbf{b}^{edge} \quad \text{for } l = 0, 1, ..., L \quad 1, \tag{A.2}$$

where $\mathbf{W}^{node}$, b^{node}, $\mathbf{W}^{edge}$ and b^{edge} are learnable parameters. At each layer, GNNs update node representations by

$$\mathbf{h}_v^l = \textsc{ReLU}(\textsc{Mlp}^l(\textsc{Concat}(\sum_{u \in N_u \cup \{v\}} \mathbf{h}_u^{l-1}, \sum_{e_{uv}: u \in N_u \cup \{v\}} \mathbf{h}_e^{l-1}))), \tag{A.3}$$

where $\textsc{Concat}(\cdot)$ takes two vectors as input and concatenates them, and N_u is a set of nodes adjacent to node v. Note that we remove the $\textsc{ReLU}$ activation in the final layer so as to output negative values in $\mathbf{h}_v^l$.

With the aggregation and update of node/edge features, we generate node embeddings at final layer l to obtain the graph-level representation $\mathbf{h}_G$:

$$\mathbf{h}_G = \textsc{Mlp}(\textsc{Mean}(\{\mathbf{h}_v^l | v \in G\})), \tag{A.4}$$

where $\textsc{Mean}$ is the mean pooling operation and $(\cdot) = \textsc{Mlp}(\textsc{Mean}(\cdot))$ is the graph-level pooling calculation.

For other GNN architectures like GCN, GraphSAGE and GAT, we adopt the implementation in the Pytorch Geometric library[6]. More specifically, the number of GAT attention heads is set to 2 and the dimension of node/edge embeddings as well as the number of GNN layers are the same as GIN. Since these GNN models do not originally handle edge features, we incorporate edge features into them similarly to how we do it for the GIN; we add edge embeddings into node embeddings, and perform the GNN message-passing on the obtained node embeddings, as suggested in (Hu et al. 2020).

D Details of Experimental Settings

Implementation of Baselines To contextualize the empirical results of L2P-GNN on the pre-training benchmarks, we compare against four self-supervised or unsupervised baselines:

EdgePred (Hamilton, Ying, and Leskovec 2017) is a self-supervised method to predict the connectivity of node pairs, which adapts the same objective function as node-level loss in L2P-GNN.

DGI (Velickovic et al. 2019) learns node representations within graph-structured data in an unsupervised manner, which relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs—both derived using established graph convolutional network architectures.

ContextPred (Hu et al. 2020) utilizes node-level self-supervised information to explore distribution of graph structure. We use the suggested parameters to sample subgraphs to predict their surrounding graph structures.

AttrMasking (Hu et al. 2020) is also a node-level self-supervised pre-training strategy for GNNs, aiming to learn the regularities of the node and edge attributes distributed over graphs

All the above pre-training baselines and our L2P-GNN can be implemented for different GNN architectures. We experiment with four popular GNN architectures, namely, GCN (Kipf and Welling 2017), GraphSAGE (Hamilton, Ying, and Leskovec 2017), GAT (Velickovic et al. 2018) and GIN (Xu et al. 2019b). We implement these GNNs with PyTorch Geometric (PyG).

Parameter Settings We adopt Adaptive Moment Estimation (Adam) to optimize our L2P-GNN. We select the hyper-parameters that performed well across all downstream tasks in the validation sets. In pre-training procedure, for all datasets, we use a batch size of 64 and set the dimension of node representation to 300. We perform one step gradient descent update in both node-level and graph-level adaptations (i.e., $s = t = 1$). The prior learning rate, node-level and graph-level learning rates are all set to 0.001 (i.e., $\gamma = \alpha = \beta = 0.001$). We set the number of child tasks to 1 for all datasets, and set the sizes of support/query sets to 10/5 and 50/30 for Biology and PreDBLP dataset, respectively. The number of layers of GNNs is set to 5 for all datasets. The maximum number of epochs are set to 50 and 20 for pre-training GNNs on Biology and PreDBLP dataset, respectively. In fine-tuning procedure, all models are also trained with Adam optimizer with a learning rate of 0.001. For all downstream datasets, we use a batch size of 32 and train models for 50 epochs.

For baselines, we optimize their parameters empirically under the guidance of literature. Specifically, we also train the baselines with Adam optimizer with a learning rate of 0.001 and set the dimension of node representation to 300. As suggested in (Hu et al. 2020), we set the batch size to 256 for pre-training while 32 for fine-tuning Biology dataset. For PreDBLP dataset, we set the batch size and the number of epochs to be the same as in our L2P-GNN. Other baseline parameters either adopt the original optimal settings or are optimized by the validation set.

Experiment Environment All experiments are conducted on a Linux server with one GPU (GeForce RTX 2080) and CPU (Intel(R) Xeon(R) Silver 4210 CPU @ 2.20GHz), and its operating system is Red Hat 4.8.5-16. We implement the proposed L2P-GNN with deep learning library PyTorch and PyTorch Geometric. The Python and PyTorch versions are 3.7.6 and 1.4.0, respectively.

[6]https://github.com/rusty1s/pytorch_geometric