Unraveling the Pitfalls and Differences of Current Pre-Training Approaches for GNNs

Jonatan Frank
jonatan.frank@uni-ulm.de

1 INTRODUCTION

Pre-training has achieved great success in natural language processing (NLP) [2] and computer vision (CV) [4]. The idea is to use abundant, mostly unlabeled, data to teach the model how the data works in general e. g., for NLP models, how language works. This pre-training data differs from the downstream task data, which is usually labeled and in short supply. After pre-training, the downstream task data is used for fine-tuning the pre-trained model. The combination of pre-training and fine-tuning usually improves performance compared to training from scratch. The pre-trained model can be reused for multiple other tasks by fine-tuning. Hence, training time is saved by reusing it because fine-tuning data are usually small.

Intuitively, pre-training should also be beneficial for graphs and improve a Graph Neural Network's (GNN's) performance. However, this is not always the case because pre-training a GNN can also lead to negative transfer [19]. Negative transfer is transferring knowledge from the pre-training, which can negatively impact the target learner. There are some approaches of pre-training GNNs [5, 6, 11, 15, 23], but they can also lead to negative transfer. Therefore, it is interesting to know when this happens because if it happens, one can simply save time by not pre-training the model.

In order to answer the question *when to pre-train*, we investigate the questions of *what to pre-train* and *how to pre-train*. For that, we look into four different approaches for pre-training GNNs. To evaluate how well they answer the questions, the approaches are compared to each other. The evaluation of different GNNs can have pitfalls [16], and that is why we also consider them if we compare the pre-training approaches. Furthermore, we look into possible restrictions and pitfalls of the approaches and the evaluation of them. The following questions about these approaches arise:

- How do they work?
- How is the graph defined?
- What information is used for pre-training?
- Do they improve the performance compared to training from scratch?
- What are the similarities and differences between the approaches?
- What are the restrictions and pitfalls?

Next, we present related work. Afterward, we introduce four different pre-training approaches. We also introduce the datasets we use to compare the approaches and why we choose them. Then, we compare the performance of the approaches in our experimental apparatus.

2 RELATED WORK

The success of contrastive learning in NLP [1, 14] and CV [4, 21] is also an inspiration for pre-training GNNs. An example of that is Graph Contrastive Coding (GCC) [15]. The idea is that in self-supervised pre-training, the GNN learns to capture the universal network topological properties across multiple networks.

The advances in self-supervised learning for graphs [10] have shown that unlabeled data itself contains rich semantic knowledge. That is why a model that can capture the data distribution should be able to transfer it onto various downstream tasks. Generative Pre-Training of GNN (GPT-GNN) [6] uses exactly that for pre-training. It maximizes the graph likelihood $p(G; \theta)$ during pre-training.

The already mentioned pre-training methods focus mostly on pre-training on the whole graph, but there are also pre-train approaches on nodes [5]. It is also possible to combine pre-training on graphs with pre-training on nodes. Hu et al. [5] use two node-level pre-training methods that try to capture domain-specific regularities in graphs, namely Context Prediction and Attribute Masking. They combined them with their pre-training method called Property Prediction.

The methods mentioned so far consider the pre-training and fine-tuning steps separately. They first pre-train their model and then fine-tune it on data of the downstream task. Lu et al. [11] propose L2P-GNN, an approach that alleviates the divergence between pre-training and fine-tuning by learning to pre-train (L2P) at both node and graph level in a self-supervised manner.

To ensure a fair comparison of the pre-training methods, we also have to evaluate GNNs in a fair matter. Shchur et al. [16] already showed that there are many pitfalls in evaluating GNNs. Pitfalls are the choice of the dataset and the split of the data into validation, training, and test data. Another one is hyperparameter optimization.

3 METHODS

Here, we explain each of the pre-training methods we use.

3.1 Graph Contrastive Coding

For Graphs $G = (V, E)$ with nodes V and edges E, GCC [15] performs a pre-training by minimizing InfoNCE [18]. The edges are always considered undirected. The InfoNCE is adopted as follows:

$$\mathcal{L} = -\log \frac{\exp(q^T k_+/\tau)}{\sum_{i=0}^{K} \exp(q^T k_i/\tau)},$$

where τ is the temperature hyper-parameter, q is an encoded query, and we have a dictionary of K+1 encoded keys $\{k_0, \dots, k_K\}$. f_q and f_k are two GNNs that encode the query instances x^q and key instances x^k to d-dimensional representations. $q = f_q(x_q)$ and $k = f_k(x_k)$.

The pre-training focus is purely on structural representations without any pre-defined node attributes. An instance is the r-ego network of a node n with node features created by generalized positional embedding [15]. For a node $n \in V$, its r-neighbors are defined as $S_v = \{u|u, n \in V \text{ and } d(u, n) \leq r\}$ where $d(u, n)$ is the shortest path distance between u and n in the graph G. The r-ego network of a node n consists of its r-neighbors and the edges between them that are in E.

For deciding which query instance x^q and key instance x^k form a similar instance pair (x^q, x^k), Qiu et al. [15] created their own method using the three steps—random walks with restart [17], subgraph induction, and anonymization [8, 13]. To effectively build and maintain the dictionary, GCC either uses end-to-end (E2E) or momentum contrast (MoCo) [4].

The pre-trained GNN is the encoder f_q, and either the parameters of f_q are frozen or not. If the parameters are frozen, either logistic regression or SVM is used on top of f_q as the linear classifier. In this case, the linear classifier is fine-tuned on the downstream task. In the other case, f_q is trained on the downstream task. For our experiments, we use both with MoCo because it is not clear if frozen or unfrozen parameters are better.

3.2 GPT-GNN

GPT-GNN [6] is for graphs $G = (V, E, X)$, where X is the node feature matrix. For pre-training, it maximizes the graph likelihood $p(G; \theta)$. Hu et al. [6] define the graph distribution $p(G; \theta)$ as the expected likelihood over all possible permutations, i.e.,

$$p(G; \theta) = \mathbb{E}_\pi [p_\theta(X^\pi, E^\pi)],$$

where X^π denotes permutated node attributes, E^π is a set of edges, while E_i^π denotes all edges connected with node i^π. They assume that observing any node ordering π has an equal probability. The generative process for one permutation looks as follows: Given a permutated order, one can factorize the log-likelihood autoregressively (generating one node per iteration) as:

$$\log p_\theta(X, E) = \sum_{i=1}^{|V|} \log p_\theta(X_i, E_i | X_{<i}, E_{<i})$$

At each step i, one uses all nodes that are generated before i, their attributes $X_{<i}$, and $E_{<i}$ to generate a new node i, including both its attribute X_i and its connections with existing nodes E_i. During this process, a part of the edges has already been observed or generated. That is why the generation can be decomposed into two coupled parts by assuming X_i and E_i to be independent. In part one, node attributes are generated given the observed edges, and in part two, the remaining edges are generated given the observed edges and generated node attributes. The result is:

$$\begin{aligned} &p_\theta(X_i, E_i | X_{<i}, E_{<i}) \\ &= \mathbb{E}_o [\underbrace{p_\theta(X_i, |E_{i,o}, X_{<i}, E_{<i})}_{\text{1) generate attributes}} \cdot \underbrace{p_\theta(E_{i,\neg o}, |E_{i,o}, X_{\leq i}, E_{<i})}_{\text{1) generate edges}}], \end{aligned}$$

where o denotes that the edges have been observed for i and $\neg o$ denotes the opposite.

Attribute generation and edge generation can be done simultaneously with a GNN by dividing the set of nodes into two sets. For optimization of the node generation, GPT-GNN uses its own attribute generation loss, and for edges generation, it uses a contrastive loss. For each, GPT-GNN uses an encoder in a way that is inspired by NLP. By optimizing attribute generation and edge generation, one also optimizes the probability likelihood of the whole attributed graph. The GNN used for the optimization is the resulting pre-trained GNN.

3.3 Context Prediction

For Context Prediction [5], one can use a Graph $G = (V, E)$ with and without attributes for nodes and edges for pre-training. Context Prediction depends on the number of layers of a GNN. For a K-layer GNN, it is defined as follows: For each node $v \in V$, the K-neighorhood is $N_{Kv} = \{u | u \in V \text{ and } d(v, u) \leq K\}$, use the K-layer GNN to calculate the node embedding $h_v^{(K)}$ of v depending only on nodes that are in the K-neighorhood of v. For each node $v \in V$, create the context graph $G_{Cv} = (V_{Cv}, E_{Cv})$ with $V_{Cv} = \{u | u \in V, r_1 < d(v, u) < r_2 \text{ and } r_1, r_2 \in \mathbb{N}\}$ where $r_1 < K$ so that some nodes are shared between the neighborhood and the context graph and $E_{Cv} = \{(u, w) | (u, w) \in E,\ u, w \in V_{Cv}$ and (u, w) is in a path between v and w with a length less than $r_2\}$. Calculate the context embedding of a context graph G_{Cv} with the help of an auxiliary GNN. The auxiliary GNN is applied to obtain the node embeddings in a context graph G_{Cv}. Then average the embeddings of the context nodes to obtain a context embedding $c_v^{G_{Cv}}$ of a context graph that has the same length as the embeddings of the K-layer GNN.

For pre-training, optimize the equation

$$\sigma(h_v^{(K)T} c_{v'}^{G_{Cv'}}) \approx \mathcal{I}\{v \text{ and } v' \text{ are the same nodes}\},$$

where $\sigma(\cdot)$ is the sigmoid function, and $\mathcal{I}(\cdot)$ is the indicator function for each node. After pre-training, the K-layer GNN is then fine-tuned on the data of the downstream task.

3.4 L2P-GNN

L2P-GNN [11] can be applied to any kind of graph, including $G = (V, E, X, Z)$, where X and Z are node and edge features. Pre-training data is $D^{pre} = \{G_1, G_2, \ldots, G_N\}$. A task $T_G = (S_G, Q_G)$ is capturing structures and attributes in a graph from both local and global perspectives. The meta-learned prior from the support set S_G can be adapted to a new task or graph in the query set Q_G. L2P-GNN learns this 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. L2P-GNN involves a graph $G \in D^{pre}$, consisting of a support set S_G and a query set Q_G. T_G is designed to contain k child tasks, i. e., $T_G = (T_G^1, T_G^2, \ldots, T_G^k)$ with $T_G^c = (S_G^c = \{(u, v) \sim p_\epsilon\}, Q_G^c = \{(p, q) \sim p_\epsilon\})$ s.t. $S_G^c \cap Q_G^c = \emptyset$. Support S_G^c and query Q_G^c contain edges randomly sampled from the edge distribution p_ϵ of the graph.

L2P-GNN uses a GNN with parameters ψ to create the node embeddings and a pooling function with hyperparameters ω to calculate a graph-level representation. Lu et al. [11] use their own graph-level loss $\mathcal{L}^{graph}(\omega; S_G)$ and loss for nodes in a support subset c $\mathcal{L}^{node}(\psi; S_G^c)$ to update their parameters as follows:

$$\psi' = \psi - \alpha \frac{\partial \sum_{c=1}^{k} \mathcal{L}^{node}(\psi; S_G^c)}{\partial \psi} \text{ and}$$

$$\omega' = \omega - \beta \frac{\partial \mathcal{L}^{graph}(\omega; S_G)}{\partial \omega},$$

where α and β are the learning rates. These updated parameters are used as transferable priors to the next update using the query set to simulate a fine-tuning:

$$\psi \leftarrow \psi - \gamma \frac{\partial \mathcal{L}_{T_G}(\omega', \psi'; Q_G)}{\partial \psi'} \text{ and}$$

$$\omega \leftarrow \omega - \gamma \frac{\partial \mathcal{L}_{T_G}(\omega', \psi'; Q_G)}{\partial \omega'},$$

where γ is the learning rate and

$$\mathcal{L}_{T_G}(\omega', \psi'; Q_G) = \sum_{G \in D^{pre}} \left(\mathcal{L}^{graph}(\omega'; Q_G) + \frac{1}{k} \sum_{c=1}^{k} \mathcal{L}^{node}(\psi'; Q_G^c) \right).$$

$\mathcal{L}_{T_G}$ is the combination of their node-level and graph-level loss. After updating the GNN for the task of each graph in the pre-training data, the GNN is pre-trained.

4 EXPERIMENTAL APPARATUS

4.1 Datasets

For our pre-training, we use a preprocessed ChEMBL dataset [3, 12], containing 456K molecules with 1310 kinds of diverse and extensive biochemical assays. For fine-tuning and the downstream tasks, we use eight larger binary graph classification datasets contained in MoleculeNet [20], a benchmark for molecular property prediction. We decided on these datasets because node attributes were added by Hu et al. [5]. They generated the node attributes by using RDKit [9], and the node attributes consist of the atom number $\{1, \ldots, 118\}$ and the chirality tag {unspecified, tetrahedral cw, tetrahedral ccw, other}. They also made it publicly available[1]. Another reason is that molecules differ in structure and contain different atoms. Hence structure and node attributes can be used to distinguish them.

We split the downstream data into test, train, and validation data. 10% are used for validation, 10% are used for testing, and 80% are used for training. We use 100 train/validation/test splits and 20 random initializations for each in our experiments because Shchur et al. [16] highlighted the fragility of experimental setups for evaluating different models that consider only a single train/validation/test split of the data. They also used this number of splits for the comparison of different GNNs.

4.2 Procedure

For each pre-training method except GPT-GNN, we use the Graph Isomorphism Network (GIN) [22] architecture because it was the best or only used GNN in these methods. For GPT-GNN, we use Heterogeneous Graph Transformer (HGT) [7]. The GINs and the GPT-GNN are also used without pre-training to determine if the pre-training improves performance. Furthermore, we use the model architectures of the GINs and the GPT-GNN described in the pre-training papers and the same training procedure for each model. The training procedure is the same as the one used by Shchur et al. [16] for comparing different GNNs, except that we pre-train the models. For the evaluation, we use the mean accuracy and standard deviations of a model for the eight datasets averaged over 100 splits and 20 random initializations for each split.

REFERENCES

[1] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=r1xMH1BtvB

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[3] Anna Gaulton, Louisa J Bellis, A Patricia Bento, Jon Chambers, Mark Davies, Anne Hersey, Yvonne Light, Shaun McGlinchey, David Michalovich, Bissan Al-Lazikani, et al. 2012. ChEMBL: a large-scale bioactivity database for drug discovery. *Nucleic acids research* 40, D1 (2012), D1100–D1107.

[4] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 9726–9735. https://doi.org/10.1109/CVPR42600.2020.00975

[5] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. 2020. Strategies for Pre-training Graph Neural Networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=HJlWWJSFDH

[6] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 1857–1867. https://doi.org/10.1145/3394486.3403237

[7] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous Graph Transformer. In *WWW '20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*, Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen (Eds.). ACM / IW3C2, 2704–2710. https://doi.org/10.1145/3366423.3380027

[8] Yilun Jin, Guojie Song, and Chuan Shi. 2020. GraLSP: Graph Neural Networks with Local Structural Patterns. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 4361–4368. https://ojs.aaai.org/index.php/AAAI/article/view/5861

[9] Greg Landrum. 2006. RDKit: Open-source cheminformatics. 2006. *Google Scholar* (2006).

[10] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S. Yu. 2023. Graph Self-Supervised Learning: A Survey. *IEEE Trans. Knowl. Data Eng.* 35, 6 (2023), 5879–5900. https://doi.org/10.1109/TKDE.2022.3172903

[11] Yuanfu Lu, Xunqiang Jiang, Yuan Fang, and Chuan Shi. 2021. Learning to Pre-train Graph Neural Networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 4276–4284. https://ojs.aaai.org/index.php/AAAI/article/view/16552

[12] Andreas Mayr, Günter Klambauer, Thomas Unterthiner, Marvin Steijaert, Jörg K Wegner, Hugo Ceulemans, Djork-Arné Clevert, and Sepp Hochreiter. 2018. Large-scale comparison of machine learning methods for drug target prediction on ChEMBL. *Chemical science* 9, 24 (2018), 5441–5451.

[13] Silvio Micali and Zeyuan Allen Zhu. 2016. Reconstructing Markov processes from independent and anonymous experiments. *Discret. Appl. Math.* 200 (2016), 108–122. https://doi.org/10.1016/j.dam.2015.06.035

[14] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (Eds.). 3111–3119. https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html

[15] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. 2020. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash (Eds.). ACM, 1150–1160. https://doi.org/10.1145/3394486.3403168

[16] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *CoRR* abs/1811.05868 (2018). arXiv:1811.05868 http://arxiv.org/abs/1811.05868

[17] Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*. IEEE Computer Society, 613–622. https://doi.org/10.1109/ICDM.2006.70

[18] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. *CoRR* abs/1807.03748 (2018). arXiv:1807.03748 http://arxiv.org/abs/1807.03748

[19] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime G. Carbonell. 2019. Characterizing and Avoiding Negative Transfer. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 11293–11302. https://doi.org/10.1109/CVPR.2019.01155

[20] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay S. Pande. 2017. MoleculeNet: A Benchmark for Molecular Machine Learning. *CoRR* abs/1703.00564 (2017). arXiv:1703.00564 http://arxiv.org/abs/1703.00564

[21] Zhirong Wu, Yuanjun Xiong, Stella X. Yu, and Dahua Lin. 2018. Unsupervised Feature Learning via Non-Parametric Instance Discrimination. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 3733–3742. https://doi.org/10.1109/CVPR.2018.00393

[22] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ryGs6iA5Km

[1] https://github.com/snap-stanford/pretrain-gnns

[23] Simiao Zuo, Haoming Jiang, Qingyu Yin, Xianfeng Tang, Bing Yin, and Tuo Zhao. 2022. DiP-GNN: Discriminative Pre-Training of Graph Neural Networks. *CoRR* abs/2209.07499 (2022). https://doi.org/10.48550/arXiv.2209.07499 arXiv:2209.07499