Graph coarsening can increase learning efficiency

Marek Dědič[1,2], Lukáš Bajer[2], Pavel Procházka[2], and Martin Holeňa[3]

[1] Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague, Trojanova 13, Prague, Czech Republic
[2] Cisco Systems, Inc., Karlovo náměstí 10, Prague, Czech Republic
[3] Institute of Computer Science, Czech Academy of Sciences, Pod vodárenskou věží 2, Prague, Czech Republic

Abstract Graph based models are used for tasks with increasing size and computational demands. The paper focuses on leveraging methods for pretraining on coarser graphs. A way of producing such coarsenings is shown to be feasible and the performance-complexity characterics of these methods are studied.

1 Introduction

Graph-based models for machine learning are often used for task with millions of nodes and billions of edges. For such tasks, many machine learning algorithms may be computationally intractable. As a way to combat these growing demands, we look at the trade-off between performance, complexity and method generality. Our work in progress builds on HARP [1] and proposes a general framework for coarsening graphs and examines the performance characteristics of models trained on these coarser graphs.

In the next section, the graph learning algorithm HARP is presented, followed by a section presenting partially injective homomorphisms. Section 4 connects these ideas and Section 5 discusses the performance characteristics of HARP. Finally, Section 6 supports our theses with experimental evaluation.

2 HARP

2.1 Method overview

HARP is a method for improving the performance of graph-based learning algorithms such as DeepWalk [8], LINE [12], PTE [11] or node2vec [3]. The method is a combination of dataset augmentation and pre-training based on the general principle that graph-based models train more efficiently on smaller graphs and can thus be pre-trained on a coarsened representation of the graph task at hand. In an overview, the method consists of the following steps:

1. **Dataset augmentation**. The graph is consecutively reduced in size by the application of several graph coarsening schemas. In each step, the coarsened can be viewed as a representation of ever more global structure in the task.

After all the coarsened graphs are pre-computed, the method itself can be executed by repeating the following steps on the graphs from the coarsest to the finest:

2. **Training on an intermediary graph**. The model is trained on one of the pre-computed intermediary graphs.
3. **Embedding prolongation**. The embedding generated by the model is prolonged from a coarser graph to a finer one (with the original graph being the finest).

The first step is independent of the rest of the computation and can be done ahead of time. The last two steps can be seen as a form of pre-training for the model that is to be learnt on the original task.

2.2 Graph coarsening

Consider an undirected graph G with nodes $V(G)$ and edges $E(G)$. The aim of the graph coarsening part of the algorithm is to generate a sequence of graphs $G_0, G_1, G_2, \ldots, G_L$ where $G_0 = G$ and $L \in \mathbb{N}$ is a hyperparameter of the method. In this sequence, each graph G_i is generated from the graph G_{i-1} by coarsening it – lowering the number of nodes and edges while preserving the general structure of the graph. Following [1], let ψ_i denote the mapping between the graphs such that $G_i = \psi_i(G_{i-1})$.

In Section 4, a general framework for graph coarsening is presented. The authors of [1] instead introduce two particular coarsening relations ψ_i – **edge collapsing** and **star collapsing**. Edge collapsing is a very simple method – out of all the edges $E(G_{i-1})$, a subset E' is selected such that no two edges from E' are incident on the same node. Then, for each edge $(u, v) \in E'$, u and v are merged into a single node w, with all edges incident on u or v being replaced with edges incident on w.

The edge collapsing algorithm is a good general way of lowering the number of nodes in a graph, however, some structures are not easily collapsed by it. An example of such a structure is a "star" – a single node connected to many other nodes. To coarsen graphs with such structures effectively, the star collapsing algorithm is proposed. For each such *hub* node u, its unconnected neighbouring nodes are taken and merged pairwise. Again, all edges incident on such nodes are replaced with edges incident on the corresponding newly created nodes. These two approaches are combined in HARP, with each coarsening step being a star collapsing step followed by an edge collapsing step.

2.3 Embedding prolongation

In each coarsening step, an embedding of the graph G_i is trained by one of the embedding algorithms. To continue training with a finer graph, this embedding $\Phi_i : V(G_i) \to \mathbb{R}^d$ needs to be *prolonged* to create the finer embedding $\Phi_{i-1} : V(G_{i-1}) \to \mathbb{R}^d$. To achieve this, the representation of a node in the graph G_i is copied for each of the nodes in G_{i-1} it was created from (by the graph collapsing algorithm). That is,

$$\Phi_{i-1}(u) = \Phi_i(\psi_i(u)).$$

This is then taken as the starting point for the next training phase.

3 Partially injective homomorphism

In [10], the authors present the notion of **partially injective homomorphisms** as a bridge between the comparatively weak concept of a homomorphism and the much stronger concept of an injective homomorphism (i.e., a subgraph isomorphism). This section presents these concepts and explores some of their properties. In Section 4, graph homomorphisms are then used as a general framework for graph coarsening.

A **graph homomorphism** between graphs G and H is a mapping $\varphi : V(G) \to V(H)$ that preserves edges, thus

$$(u,v) \in E(G) \implies (\varphi(u), \varphi(v)) \in E(H).$$

A homomorphism is **injective** iff $\forall u, v \in V(G) \quad \varphi(u) = \varphi(v) \implies u = v$. Finally, a homomorphism is **partially injective** iff $\forall (u,v) \in \mathcal{C} \quad \varphi(u) = \varphi(v) \implies u = v$ for some $\mathcal{C} \subseteq (V(G))^2 \setminus E(G)$, that is, $\mathcal{C}$ is a set of *non-edges* of the graph G. Observe that for any $(u,v) \in E(G)$ the condition holds in general per the definition of a homomorphism, which justifies the limitation of $\mathcal{C}$ to non-edges of the graph.

For given graphs G and H, let $\mathcal{L}$ denote the finite set of all partially injective homomorphisms between them. Let a particular partially injective homomorphism described by $\mathcal{C}$ be denoted as $\mathrm{PIHom}(G, H, \mathcal{C})$. $\mathcal{L}$ has a natural partial order $\preceq$ where $\mathrm{PIHom}(G, H, \mathcal{C}_1) \preceq \mathrm{PIHom}(G, H, \mathcal{C}_2)$ iff $\mathcal{C}_1 \subseteq \mathcal{C}_2$ and forms a lattice with order $\preceq$. The minimum of this lattice $\mathrm{PIHom}(G, H, \emptyset)$ corresponds to an ordinary homomorphism while the maximum $\mathrm{PIHom}\left(G, H, (V(G))^2 \setminus E(G)\right)$ corresponds to an injective homomorphism.

4 HARP and partially injective homomorphisms

The connection between HARP and partially injective homomorphisms is a theoretical, yet useful one. If the HARP coarsenings could be restricted to partially injective homomorphisms or their compositions, general coarsenings could be learned for each task. In Section 6.3 we explore whether this restriction limits the performance of HARP.

When both the edge and star collapsing algorithms are used, the mapping ψ_i introduced in 2.2 is **not** a homomorphism nor a combination of homomorphisms due to its not meeting the injectivity condition on edges. However, it is met for the complementary graph (graph where edges are swapped with non-edges). For the complementary graph, the star collapsing algorithm is not a homomorphism, however, it can be replaced by homomorphisms that also collapses stars. We propose a coarsening that merges a hub node with half of its neighbours. Because merging a node with its neighbour only collapses an edge, this coarsening scheme is a composition of partially injective homomorphisms on the complementary graph. Such a model is studied in Section 6.3.

This theoretical connection gives a way of constructing more general graph coarsenings by setting a constraint on such relations and finding the maximum of a subset of $\mathcal{L}$ that satisfies such a constraint. The authors of [10] present a way of effectively (in polynomial time for bounded tree-width graphs) searching for such maximally constrained homomorphisms.

5 HARP and the performance-complexity trade-off

Graph-based methods such as node2vec typically have a large number of parameters - on the widely used OGBN-ArXiv dataset (see [4]), the SOTA node2vec model has over 21 million parameters. At the same time, recent works have started to focus more heavily on simpler methods as a competitive alternative to heavy-weight ones (see [2,5,9,13]). As the authors of [1] observed, HARP improves the performance of models when fewer labelled data are available. The proposed lower complexity models based on HARP could also improve performance in a setting where only low fidelity data are available for large parts of the graph. Coarser models could be trained on them, with a subsequent training of finer models using only a limited sample of high fidelity data.

As a core principle of HARP, lower level representations of the task are generated and a model is learnt on them. How good these models are remains in question. In order to test this, several models were compared. Each model M_i was trained on graphs $G_L, \ldots, G_i$ as with HARP, then the embeddings were prolonged on graphs $G_{i-1}, \ldots, G_0$ without training. On G_0, the models were then trained as they would be in an ordinary node2vec setup. With this schema, L models are obtained, each trained on graphs of different granularity. To examine the performance-complexity trade-off of HARP, the trade-off between decreasing predictive accuracy and decreasing amount of training data was evaluated.

6 Experimental evaluation

The experiments proposed in the previous sections were run on the OGBN-ArXiv dataset (see [4]) with node2vec as the underlying graph learning algorithm. The experiments build upon the SOTA node2vec implementation for this dataset.

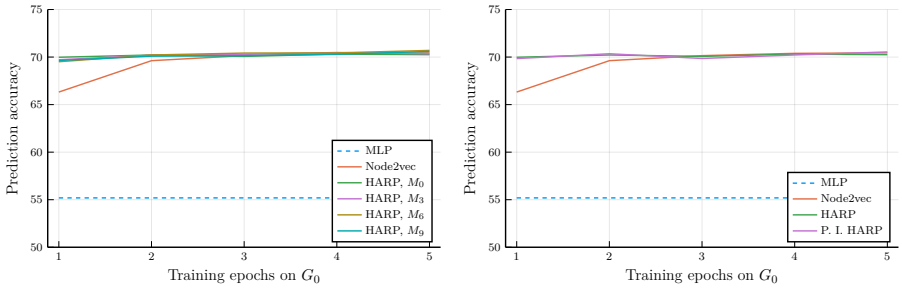
6.1 Experiment setup

The configurations of both the node2vec and the MLP models are taken from the PyTorch Geometric implementation of node2vec on the ogbn-arxiv dataset (see [7]). The node2vec model generated an embedding into $\mathbb{R}^{128}$ from 10 random walks of length 80 for each node with a context window size of 20. The optimizer ADAM [6] was used with a learning rate of 0.01 and batches of 256 samples. The MLP classifier using the embeddings featured 3 linear layers of 256 neurons with batch normalization after each layer. Finally, a linear layer was used for the class prediction. ADAM with a learning rate of 0.01 and a dropout rate of 0.5 was used for 200 epochs of training with the cross-entropy loss function. The experiments were implemented using PyTorch[4] and PyTorch Geometric [7].

6.2 Effect of HARP pretraining

To asses the effect of HARP as a pretraining, the main question is how does HARP pretraining change the behaviour of the node2vec training?

[4] https://pytorch.org/



(a) Performance over epochs of models pretrained on different coarsening levels.

(b) Standard HARP compared to the version built on partially injective homomorphisms.

Figure 1a compares the accuracy of several models over learning epochs on the original graph G_0. An MLP using just dataset features and an ordinary node2vec model are provided as a reference. The models $M_0, \ldots, M_L$ from Section 5 are then compared.

As can be seen, there is no noticeable difference in the performance of a pretrained node2vec in comparison to an ordinary one when it comes to performance after 4 or more training epochs. Our explanation is that the configuration used for node2vec training already samples large parts of the graph thanks to a relatively high number of long random walks. There is, however, a noticeable improvement in the performance of models when only very few training epochs are available.

6.3 Feasibility of partially injective transformations

In order to test the feasibility of HARP-like algorithms based purely on partially injective homomorphisms, ordinary HARP is compared to one with the coarsening proposed in Section 4. Figure 1b shows the performance of such a model compared to ordinary HARP, node2vec and purely feature-based MLP classifier. As can be seen, there is no meaningful difference between the two models.

7 Conclusion

In our work in progress, a way to merge method generality, computational efficiency and high performance was explored. HARP and partially injective homomorphisms were presented and subsequently connected as a way to, in a future work, adapt graph coarsenings to a specific task. This connection was experimentally verified not to impact prediction performance. Furthermore, the effect of HARP pretraining on learning characteristics was studied and found to reduce the need for training on fine (and therefore large) graphs, making way for a more efficient training without sacrificing performance.

The research reported in this paper has been supported by the Czech Science Foundation (GAČR) grant 18-18080S.

References

1. Chen, H., Perozzi, B., Hu, Y., Skiena, S.: HARP: Hierarchical Representation Learning for Networks. Proceedings of the AAAI Conference on Artificial Intelligence **32**(1) (Apr 2018), https://ojs.aaai.org/index.php/AAAI/article/view/11849, number: 1
2. Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., Monti, F.: SIGN: Scalable Inception Graph Neural Networks. arXiv:2004.11198 [cs, stat] (Nov 2020), http://arxiv.org/abs/2004.11198, arXiv: 2004.11198
3. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)
4. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open Graph Benchmark: Datasets for Machine Learning on Graphs. arXiv:2005.00687 [cs, stat] (Feb 2021), http://arxiv.org/abs/2005.00687, arXiv: 2005.00687
5. Huang, Q., He, H., Singh, A., Lim, S.N., Benson, A.R.: Combining Label Propagation and Simple Models Out-performs Graph Neural Networks. arXiv:2010.13993 [cs] (Nov 2020), http://arxiv.org/abs/2010.13993, arXiv: 2010.13993
6. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs] (Jan 2017), http://arxiv.org/abs/1412.6980, arXiv: 1412.6980
7. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F.d., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32. pp. 8024–8035. Curran Associates, Inc. (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
8. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710 (2014)
9. Salha, G., Hennequin, R., Vazirgiannis, M.: Keep It Simple: Graph Autoencoders Without Graph Convolutional Networks. arXiv:1910.00942 [cs, stat] (Oct 2019), http://arxiv.org/abs/1910.00942, arXiv: 1910.00942
10. Schulz, T.H., Horváth, T., Welke, P., Wrobel, S.: Mining Tree Patterns with Partially Injective Homomorphisms. In: Berlingerio, M., Bonchi, F., Gärtner, T., Hurley, N., Ifrim, G. (eds.) Machine Learning and Knowledge Discovery in Databases. pp. 585–601. Lecture Notes in Computer Science, Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-10928-8_35
11. Tang, J., Qu, M., Mei, Q.: PTE: Predictive Text Embedding through Large-scale Heterogeneous Text Networks. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1165–1174. Association for Computing Machinery, New York, NY, USA (Aug 2015), https://doi.org/10.1145/2783258.2783307
12. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th international conference on world wide web. pp. 1067–1077 (2015)
13. Zhang, Z., Cui, P., Pei, J., Wang, X., Zhu, W.: Eigen-GNN: A Graph Structure Preserving Plug-in for GNNs. arXiv:2006.04330 [cs, stat] (Jun 2020), http://arxiv.org/abs/2006.04330, arXiv: 2006.04330