Sparse Graph Neural Networks with Scikit-network

Thomas Bonald, Simon Delarue

Sparse Graph Neural Networks with Scikit-network

Simon Delarue[1], Thomas Bonald[1]

[1]Institut Polytechnique de Paris, Palaiseau, France
simon.delarue@telecom-paris.fr

Abstract. In recent years, Graph Neural Networks (GNNs) have undergone rapid development and have become an essential tool for building representations of complex relational data. Large real-world graphs, characterised by sparsity in relations and features, necessitate dedicated tools that existing dense tensor-centred approaches cannot easily provide. To address this need, we introduce a GNNs module in Scikit-network, a Python package for graph analysis, leveraging sparse matrices for both graph structures and features. Our contribution enhances GNNs efficiency without requiring access to significant computational resources, unifies graph analysis algorithms and GNNs in the same framework, and prioritises user-friendliness.

Keywords: Graph Neural Networks, Sparse Matrices, Python

1 Introduction

Graph Neural Networks (GNNs) are an extension of traditional deep learning (DL) methods for relational data structured as graphs [4]. In the past few years, GNN-based methods have gained increasing attention thanks to their impressive performance on a wide range of machine learning tasks, such as node classification, graph classification, or link prediction [19,16,25,28]. GNNs derive internal graph element representations using entity relationships and associated features, making them highly valuable for real-world data, where information frequently spans across multiple dimensions.

Real-world graphs, exemplified by large social networks or web collections with millions or billions of elements, each potentially having numerous attributes, pose substantial challenges for GNNs training [12,13]. Tremendous efforts have been made to tackle this challenge and scale up GNNs [16,8,30,6,7,29]. However, several existing approaches rely on parallelisation or approximation techniques such as sampling or batch training, to reduce memory consumption. A key feature of real-world graphs is that they are sparse: in a graph with n nodes, the number of edges m is very low compared to the number of possible edges $n \cdot (n-1)$. It therefore makes sense to have a data representation and algorithms that takes sparsity into account, allowing for a low memory footprint and efficient computation times.

Numerous Python packages already provide GNN implementations, including PyTorch Geometric [11], Deep Graph Library [27], Spektral [14], Stellar Graph [10] or Dive Into Graphs library [20]. But to allow natural integration with existing DL frameworks, such as PyTorch [23] or TensorFlow [1], and benefit from differentiable operators, these libraries rely upon a dense tensor-centred paradigm which does not align with graph sparsity. This dramatically hinders the use of such libraries on large real-world graphs, by requiring access to servers with large quantities of RAM.

To address this gap, we propose a GNNs module implementation relying on sparse matrices for both graph adjacency and features. Our implementation aligns seamlessly with Scikit-network[1] [3], a Python package inspired by Scikit-learn [5] for graph analysis. Scikit-network leverages the sparse formats provided by SciPy [26] for encoding graphs. It already provides diverse state-of-the-art graph algorithms, including ranking, clustering, and embedding, in a simplified framework while achieving computational efficiency compared to similar tools, *e.g.* graph-tool [24], IGraph [9] or NetworkX [15]. By only relying on NumPy [17] and SciPy [26], the development of a GNN module in Scikit-network stays true to the core principles of the package: performance and ease of use.

To summarise, our contributions encompass three key aspects. Firstly, we introduce an efficient GNN module within Scikit-network, harnessing the power of sparse matrices for both graph and features to address the characteristics of real-world graphs. Secondly, our package bridges the gap between traditional graph analysis methods and GNNs, offering a unified platform that operates within the same sparse graph representation. Lastly, we prioritise simplicity by designing our package to rely solely on foundational Python libraries, specifically NumPy and SciPy, sparing users the complexity associated with larger tensor-based DL frameworks.

The rest of the paper is organised as follows. We start by reviewing the existing related work in Section 2. Then, we formulate the computation of the GNNs message passing scheme using sparse matrices in Section 3 . In Section 4 we briefly describe our GNNs module design and we show the implementation's performance compared to other GNNs libraries in Section 5.

2 Related Work

Several Python packages already exist to help with GNNs development and usage. Pytorch Geometric (PyG) [11] proposes a general message passing interface in which all recent GNN-based aggregation schemes can be integrated. In order to reduce the computation time when running on large complex networks, they propose several dataset processing tools such as sampling or batch training. Spektral [14] is built upon the same gather-scatter paradigm as PyG, but implements GNNs on top of the user-friendly API Keras. Stellar Graph [10], like Spektral is based on Keras, but uses its own custom graph representation.

[1] https://github.com/sknetwork-team/scikit-network

Deep Graph Library [27] involves a combination of user-configurable message passing functions and sparse-dense matrix multiplications to provide the user with a GNNs framework. However, in all these libraries, the implementation design is driven by the necessity to integrate with existing Deep Learning (DL) frameworks, namely PyTorch [23] or TensorFlow [1], that provide differentiable operators. This implies the use of dense tensors for the encoding of graph elements, which comes at the expense of the sparse nature of real-world graphs. Our work differs from these approaches in the sense that we leverage sparse format representation for both the structure and the features of the graph, and only rely on Python fundamental libraries.

Other libraries, such as Dive Into Graphs library [20] offer a high-level toolbox for deep learning on graphs. However, its goal is slightly different from the previous libraries; it goes beyond the implementation of elementary tasks and includes advanded research-oriented methods such as benchmarks, graph generation or 3D graphs. In Scikit-network, we mainly focus on efficiency and ease-of-use rather than extensive features for the users.

3 Message Passing With Sparse Matrices

3.1 Message Passing Overview

Traditional GNNs models rely on an architecture that propagates the signal (or features) information across the network, through a serie of iterations (or layers). At each layer l, this architecture involves two steps for computing the representation of a node u, h_u^l; (i) the aggregation of this node's neighbourhood information into a *message* and (ii) the update of the node's representation using this aggregated information and the previous embedding of node u. These two steps can be written as:

$$h_{\mathcal{N}(u)}^l = \phi(\{h_v^{l-1}, \forall v \in \mathcal{N}(u)\}) \tag{1}$$

$$h_u^l = \psi\left(h_u^{l-1}, h_{\mathcal{N}(u)}^l\right) \tag{2}$$

where $\mathcal{N}(u)$ denotes node u's neighbourhood, ϕ is an aggregation function and ψ is an update function.

3.2 Using Sparse Matrices in Scikit-network

In Scikit-network, we represent a node attributed graph $G = (V, E, X)$ by its adjacency matrix A and its node-feature matrix X, both encoded in SciPy's Compressed Sparse Row format (CSR). This format uses three three arrays to represent a matrix: two arrays of size m, where m is the number of non-zeros elements in the matrix, and one array of size $n + 1$, where n is the number of rows in the initial matrix. Among various SciPy sparse formats (such as COO, DOD, etc), the CSR format was initially chosen for Scikit-network as it is the

most memory-efficient for common algebraic operations like listing neighbours and performing matrix-vector product [21].

The concept of *message passing scheme* within a graph convolutional layer, as seen in GCN [19], involves the gathering and scattering of information from a node's neighbourhood to compute the node's updated representation. In scikit-network, these operations are executed using the following layer-wise propagation rule:

$$H^{l+1} = \sigma(\tilde{A}H^l W^l + b^l) \tag{3}$$

where $\tilde{A}$ is the (normalised) adjacency matrix of the graph (with or without self-loops), H^l denotes the matrix of node representations at layer l, with $H^0 = X$, W^l and b^l represent trainable weight matrix and bias vector, and σ is a non-linear activation function. In our Scikit-network implementation, we employ sparse matrix multiplication (SpMSpM) to compute the term $\tilde{A}H^l$ when $l = 0$. Subsequently, for $l \geq 1$ and thus when the dimension of the H^l matrix is significantly reduced to the hidden-layer dimension, we use sparse-dense matrix multiplications (SpMM). In the same manner, the backward propagation only requires SpMM multiplications between reduced-sized matrices.

4 Scikit-network GNN Design

As illustrated in Figure 1, initialising a GNN model in Scikit-network is straightforward. Like other graph analysis algorithms in the package, GNN training is built upon the `.fit_predict()` method. This method hides forward and backward propagation's complexity to the user, providing them directly with the final predictions. Additional training details, like loss, accuracy and temporary layer outputs, are easily accessible using the `history` parameter.

5 Performance

To assess Scikit-network's GNN implementation, we compare the training time of a GCN [19] model for a node classification task with the two most widely used libraries: PyG and DGL[2]. The computer used to run all the experiments is a Mac with OS 12.6.8, equipped with a M1 Pro processor and 16 GB of RAM. For fair comparison, all experiments are run on a CPU device.

[2] We rely on the number of *forks* associated with each package on GitHub as a metric to gauge library usage. Please note that this metric provides only a partial view of actual project usage.

Code Listing 1.1. Node classification using Graph Neural Network in Scikit-network.

```
# Graph Neural Network
hidden_dim = 16

gnn = GNNClassifier(
    dims=[hidden_dim, n_labels],
    layer_types='Conv',
    activations='ReLu',
    verbose=True)

labels_pred = gnn.fit_predict(
    adjacency, features, labels,
    n_epochs=100, history=True)
```

Code Listing 1.2. Graph analysis algorithms in Scikit-network.

```
# PageRank
pagerank = PageRank()
scores = pagerank.
             fit_predict(
             adjacency)

# Louvain clustering
louvain = Louvain()
labels = louvain.
             fit_predict(
             adjacency)
```

Fig. 1. Graph methods using Scikit-network. On the left, node classification using a GNN model. On the right, Louvain clustering [2] and PageRank scoring [22]. Traditional graph algorithms and Deep Learning based models use the same API.

5.1 Datasets

We use three real-world datasets of varying sizes considering their structure and attributes. The datasets include Wikipedia-based networks[3], `Wikivitals` and `Wikivitals+`. Nodes in these datasets represent Wikipedia articles, and there is a link between two nodes if the corresponding articles are referencing each other through hypertext links on Wikipedia. Additionally, each article comes with a feature vector, corresponding to the number of occurrences of each word in its text. `OGBN-arxiv` [18] models a citation network among Computer Science papers from arXiv. For this dataset, we use node connections as the feature matrix. We detail the dataset characteristics in Table 1.

Table 1. Dataset statistics. Adjacency matrix A is summarised with its number of nodes $|V|$, edges $|E|$ and density δ_A. For the feature matrix X, we show the number of unique attributes d, and the number connections m and density δ_X of the of node-attribute matrix. For dataset marked with $\star$, we use the adjacency matrix as features.

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	δ_A	d	m	δ_X
`Wikivitals`	1.00×10^4	8.24×10^5	1.64×10^{-2}	3.78×10^4	1.363×10^6	3.59×10^{-3}
`Wikivitals+`	4.51×10^4	3.94×10^6	3.86×10^{-3}	8.55×10^4	4.78×10^6	1.24×10^{-3}
`OGBN-arxiv` $\star$	1.69×10^5	1.66×10^6	8.13×10^{-5}	1.69×10^5	1.66×10^6	8.13×10^{-5}

[3] https://netset.telecom-paris.fr/

5.2 Running Time

Table 2 displays training process running time for the different GNNs implementations. For DGL and PyG, the dense feature matrix format hinders training models on large graphs without additional tricks, *e.g.* sampling or batch training (which we did not use for fair comparison). Therefore, these implementations trigger an out-of-memory (OOM) error when used on the `OGBN-arxiv` dataset. In contrast, Scikit-network does not require these extra steps to achieve good performance on this dataset. Furthermore, we can observe the benefits of using Scikit-network regarding the characteristics of the graph: the sparser the graph, the more efficient the sparse representation is.

Table 2. Average computation times and standard deviations (3 runs) for 100 epochs model training on 3 real-world datasets.

Package	`Wikivitals`	`Wikivitals+`	`OGBN-arxiv`
`DGL`	76.6 ± 4.2	2396.1 ± 36.5	OOM
`PyG`	$\mathbf{28.7 \pm 0.1}$	1242.8 ± 28.6	OOM
`Sknetwork`	139.5 ± 0.6	$\mathbf{632.9 \pm 7.9}$	$\mathbf{60.1 \pm 0.4}$

5.3 Impact of Graph Characteristics

Graph density We further validate this observation in Figure 2, where we initialise a random graph (Erdos–Rényi) with a fixed number of nodes $|V| = 1 \times 10^4$ and varying density. In this setup, node connections are used as features. We can notice that for highly sparse graphs, *i.e.* low density, both DGL and PyG implementations face challenges due to their reliance on dense matrices. On the opposite, Scikit-network implementation shows great performance improvements compared to these methods once below a density threshold (approximately 1×10^{-3}).

Number of nodes To evaluate the impact of the number of nodes on the performance of our implementation, we fix the densities of graph matrix δ_A and feature matrix δ_X, such as $\delta_A = \delta_X = 5 \times 10^{-4}$, and vary the number of nodes in the random graphs generated. In Figure 3 we show Scikit-network's benefits in terms of training model computation time, compared to DGL and PyG implementations.

6 Conclusion

We presented the Scikit-network Graph Neural Network module. Our implementation achieves great performance on real-world graphs both in terms of accuracy

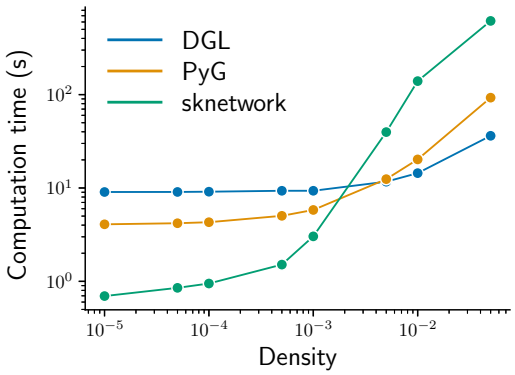

Fig. 2. Computation time for several GCN [19] implementations, according to adjacency and feature matrix densities. *Notice the log-scale on both axis.*

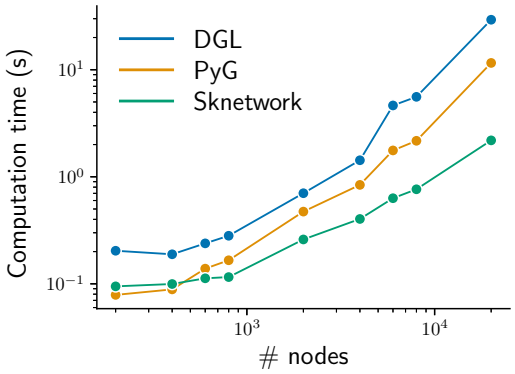

Fig. 3. Computation time for several GCN [19] implementations, according to the number of nodes in the graph. *Notice the log-scale on both axis.*

and running time, by making use of sparse encoding and operations in the learning process. Moreover, our design relies solely on foundational Python libraries, NumPy and SciPy, and does not require the use of tensor-centred traditional Deep Learning frameworks. With this module, Scikit-network offers a unified platform gathering traditional graph analysis algorithms and Deep Learning-based models. In the future, we plan to extend the current module by adding additional state-of-the-art GNNs models and layers, as well as optimizers.

Acknowledgements. The authors would like to thank Tiphaine Viard for the numerous discussions, as well as her insightful comments and suggestions about the paper.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16), pp. 265–283 (2016)
2. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **2008**(10), P10,008 (2008). DOI 10.1088/1742-5468/2008/10/p10008
3. Bonald, T., De Lara, N., Lutz, Q., Charpentier, B.: Scikit-network: Graph analysis in python. The Journal of Machine Learning Research **21**(1), 7543–7548 (2020)
4. Bronstein, M.M., Bruna, J., LeCun, Y., Szlam, A., Vandergheynst, P.: Geometric deep learning: Going beyond euclidean data. IEEE Signal Processing Magazine **34**(4), 18–42 (2017). DOI 10.1109/MSP.2017.2693418
5. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., Varoquaux, G.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013)
6. Chen, J., Ma, T., Xiao, C.: Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv preprint arXiv:1801.10247 (2018)
7. Chen, J., Zhu, J., Song, L.: Stochastic training of graph convolutional networks with variance reduction. arXiv preprint arXiv:1710.10568 (2017)
8. Chiang, W.L., Liu, X., Si, S., Li, Y., Bengio, S., Hsieh, C.J.: Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 257–266 (2019)
9. Csardi, G., Nepusz, T., et al.: The igraph software package for complex network research. InterJournal, complex systems **1695**(5), 1–9 (2006)
10. Data61, C.: Stellargraph machine learning library. https://github.com/stellargraph/stellargraph (2018)
11. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
12. Fey, M., Lenssen, J.E., Weichert, F., Leskovec, J.: Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In: International conference on machine learning, pp. 3294–3304. PMLR (2021)
13. Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., Monti, F.: Sign: Scalable inception graph neural networks. arXiv preprint arXiv:2004.11198 (2020)
14. Grattarola, D., Alippi, C.: Graph neural networks in tensorflow and keras with spektral [application notes]. IEEE Computational Intelligence Magazine **16**(1), 99–106 (2021)
15. Hagberg, A., Swart, P., S Chult, D.: Exploring network structure, dynamics, and function using networkx. Tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008)
16. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)
17. Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., et al.: Array programming with numpy. Nature **585**(7825), 357–362 (2020)

18. 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 preprint arXiv:2005.00687 (2020)
19. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
20. Liu, M., Luo, Y., Wang, L., Xie, Y., Yuan, H., Gui, S., Yu, H., Xu, Z., Zhang, J., Liu, Y., et al.: Dig: A turnkey library for diving into graph deep learning research. The Journal of Machine Learning Research **22**(1), 10,873–10,881 (2021)
21. Lutz, Q.: Graph-based contributions to machine-learning. Theses, Institut Polytechnique de Paris (2022). URL https://theses.hal.science/tel-03634148
22. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bring order to the web. Tech. rep., Technical report, stanford University (1998)
23. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
24. Peixoto, T.P.: The graph-tool python library. figshare (2014). DOI 10.6084/m9.figshare.1164194
25. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
26. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al.: Scipy 1.0: fundamental algorithms for scientific computing in python. Nature methods **17**(3), 261–272 (2020)
27. Wang, M., Zheng, D., Ye, Z., Gan, Q., Li, M., Song, X., Zhou, J., Ma, C., Yu, L., Gai, Y., Xiao, T., He, T., Karypis, G., Li, J., Zhang, Z.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
28. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018)
29. Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W.L., Leskovec, J.: Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, pp. 974–983 (2018)
30. Zeng, H., Zhou, H., Srivastava, A., Kannan, R., Prasanna, V.: Graphsaint: Graph sampling based inductive learning method. arXiv preprint arXiv:1907.04931 (2019)