From Message Passing to Actor Graph Neural Networks

Reiko Heckel & Adam M. Machowczyk
School of Computing and Mathematical Sciences
University of Leicester, UK
rh122|amm106@le.ac.uk>

Graph Neural Networks (GNNs) [9] are neural networks operating on graphs to compute node- or graph-level properties, and predict nodes or edges. One of the basic forms are Message Passing Neural Networks (MPNNs) [2] where each node sends a message to its neighbours based on its current state, then aggregates messages received from its neighbours to update their own state. Aggregation is expressed by the multiplication of feature vectors (carrying node attributes) with the adjacency matrix of the graph.

This model has limited expressiveness because the aggregation functions used are required to commute with graph isomorphisms. This means that they cannot use node identities but only their properties. For example, MPNNs cannot detect triangles. Extensions have been proposed including Cooperative Graph Neural Networks (CoGNNs) [1] where nodes are autonomous agents that decide, based on trained probabilities, if to listen to or send messages. This allows a node to adapt the interaction with its neighbourhood and act asynchronously.

Neither MPNNs nor CoGNNs consider the graph structure itself as data. Instead, the topology provides a fixed scaffold to compute node and edge attributes. This is an obvious restriction of the types of problems that can be modelled, but also limits the expressiveness of attribute computations expressible [3].

To generalise such a model while retaining its local, agent-based operation, we consider allowing: messages to carry edges to refer to nodes; redirection or free creation and deletion of edges; creation and deletion of nodes, possibly with an upper limit on the total number of nodes.

At the most expressive level, we have a model reminiscent of actor grammars [4] where actors can send and receive messages asynchronously, change their attributes, create or delete edges to other actors, create other actors and remove themselves from the graph.

We would like to explore the following questions:

1. How can models at different levels be implemented efficiently using matrix operations?
2. For which types of problems are agent- or rule-based models competitive?
3. How can we assess the expressiveness of such models?

As a starting point for (1) we observe that the multiplication of an adjacency matrix with a vector carrying node attributes used in the implementation of MPNNs is an efficient implementation of a multi rule that, for all nodes in a graph, accesses and aggregates the attributes of all its neighbours [5]. Previous studies of matrix implementations of graph transformations have focused on representing the double-pushout construction [6] but we believe that certain forms of universally quantified parallel graph transformations are a better match to matrix multiplication.

For (2) a direct implementation of an agent-based model would be competitive in distributed systems where, due to privacy concerns or asynchronous operation, it is impossible to establish a global graph and hence no adjacency matrix is available. For (3) the Weisfeiler-Lehman isomorphism test [3] is commonly used as a measure of expressiveness for computing graph-level properties and it is known that most common types of GNNs are at most as expressive as 1-WL [7] while MPNNs are less powerful [8].

Submitted to:
GCM 2024 Post-Proceedings

However, we would like to extend such an analysis to forms of GNNs that allow to change the graph structure, computing an output from an input graph rather than just node, edge or graph properties.

These types of graph computations can be expressed as functions $GC : Graph_I \to \mathscr{R}^m$ for graph-level computations and $NC : Graph_I \to (\mathscr{R}^m)^n$ for node-level computations where m is the number of attributes per node and n is the number of nodes of the input graph. Here, computations do not change the input graph, so we only output the data in vectors of real numbers, and it only allows for homogeneous graphs where every node has the same number m of real-valued attributes. In general, we can see a deterministic graph transformation as functions $GT : Graph_I \to Graph_O$ from a class of input graphs to a class of output graphs.

We can use the first notion to compare GNN-type approaches with general graph transformations, e.g. graph reduction systems to classify and compute metrics on graphs, and the second to capture general graph transformations. Both can easily be generalised towards non-deterministic computations by using relations rather than functions.

Now we can analyse computational models such as MPNNs by the graph computations they can implement, e.g., we know that MPNNs can count the number of nodes in a graph or compute the average degree, but not count the number of triangles. If messages can refer to nodes, as in the actor model, triangles can be identified if each node sends each neighbour a message referencing all its neighbours. Then a node can detect triangles by comparing each neighbour of a neighbour to their own neighbours. A similar effect can be achieved if we allow to match two-hop paths or materialise them by transitive edges.

References

[1] Ben Finkelshtein, Xingyue Huang, Michael Bronstein & İsmail İlkan Ceylan (2024): *Cooperative Graph Neural Networks*. arXiv:2310.01267.

[2] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals & George E. Dahl (2017): *Neural message passing for Quantum chemistry*. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, JMLR.org, p. 1263–1272.

[3] Ningyuan Huang & Soledad Villar (2022): *A Short Tutorial on The Weisfeiler-Lehman Test And Its Variants*. *CoRR* abs/2201.07083. arXiv:2201.07083.

[4] D. Janssens, M. Lens & G. Rozenberg (1993): *Computation graphs for actor grammars*. *Journal of Computer and System Sciences* 46(1), pp. 60–90, doi:https://doi.org/10.1016/0022-0000(93)90049-3. Available at `https://www.sciencedirect.com/science/article/pii/0022000093900493`.

[5] Adam Machowczyk & Reiko Heckel (2023): *Graph Rewriting for Graph Neural Networks*. *CoRR* abs/2305.18632, doi:10.48550/ARXIV.2305.18632. arXiv:2305.18632.

[6] Pedro Pablo Pérez Velasco (2008): *Matrix Graph Grammars*. *CoRR* abs/0801.1245. arXiv:0801.1245.

[7] Keyulu Xu, Weihua Hu, Jure Leskovec & 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. Available at `https://openreview.net/forum?id=ryGs6iA5Km`.

[8] Bingxu Zhang, Changjun Fan, Shixuan Liu, Kuihua Huang, Xiang Zhao, Jincai Huang & Zhong Liu (2023): *The Expressive Power of Graph Neural Networks: A Survey*. *CoRR* abs/2308.08235, doi:10.48550/ARXIV.2308.08235. arXiv:2308.08235.

[9] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li & Maosong Sun (2020): *Graph neural networks: A review of methods and applications*. *AI Open* 1, pp. 57–81, doi:https://doi.org/10.1016/j.aiopen.2021.01.001. Available at `https://www.sciencedirect.com/science/article/pii/S2666651021000012`.