Towards Graph-to-Graph Transformation Networks

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

Graph Neural Networks (GNNs) operate primarily by performing matrix operations, such as multiplication, Hadamard product, transposition and inversion, on graph adjacency matrices. Due to their essential use in machine learning, they are efficiently implemented even for large matrices. Assuming two matrices **M** and **N** of size $a \times b$ and $c \times d$ respectively, the complexity of the four operations is:

1. Matrix Multiplication: $O(abd)$. This can be optimised further for square matrices, by using Strassen's algorithm, with complexity $O(a^{2.81})$.
2. Hadamard Product: $O(ab)$, where for two matrices, $a = c$ and $b = d$.
3. Matrix Transposition (one matrix): $O(ab)$.
4. Matrix Inversion (one matrix): $O(a^3)$ with Gaussian elimination. This can be optimised further by using the Coppersmith-Winograd algorithm, with complexity $O(n^{2.376})$.

Instead, the complexity of finding a match of the LHS in the host graph, as required in rule-based graph rewriting, is $O(n^k * m^l)$, where n is the number of nodes and m is the number of edges in the host graph, and k is the number of nodes and m is the number of edges in the LHS [3].

GNNs perform a simple form of parallel graph rewriting [7] where a single rule with a central node is applied to all nodes of the graph to aggregate data from all its neighbors [6]. However, this model can only compute on node or edge attributes, not change the graph itself, and pattern matching is usually restricted to this node and its outgoing edges. We are interested in a GNN architecture that can efficiently implement general transformations from input to output graphs. We consider the following existing architectures as steps in this direction.

Message passing Neural Networks (MPNNs) define functions *message* and *update* that, when iterated, propagate graph data for up to n hops, where n is the number of layers. The message passing mechanism is critical in the Recursive, Convolutional and Autoencoder GNN categories. On their own, the objective is to aggregate information from a node neighbourhood, not to obtain a graph. There are approaches incorporating graph rewiring as a prepossessing step to improve the flow of information [2], but not to compute a new output graph.

Cooperative GNN (Co-GNN) [5] is a framework for well-established GNNs such as Graph Convolution Network (GCN), Graph Isomorphism Network (GIN), and Graph Attention Networks (GAT). Using a probabilistic action network to learn a node's behaviour in the message-passing cycle, the environment network executes the cycle. Each node can decide to exhibit the usual behaviour (both receive and propagate information), listen (but not propagate), broadcast (but not receive), and isolate (to neither receive nor propagate). The action network constructs a new adjacency matrix to improve information flow and aggregation. This is a form of dynamic rewiring that produces new graph structures, but they are not part of the output.

Submitted to:
GCM 2024 Post-Proceedings

Graph Transformer Networks (GTNs), instead of recursively aggregating a node's neighbourhood with message passing, transform it using the attention mechanism [8]. Attention scores for pairs of nodes in a graph could be used to generate new adjacency matrices, and thus output graph structures. Instead, GTNs feed attention score matrices into a Graph Convolutional Network (GCN) to compute a node embedding.

HOPPITY is a programming language-specific code transformation model using a graph-based representation of abstract syntax trees (ASTs), to learn from input graphs and sequences of observed transformations [4]. The model utilises MPNNs, Co-GNN-supported architectures, and the attention mechanism. While it is not the goal of HOPPITY to provide a general graph transformation tool, it does graph transformation in the domain-specific context of program transformations.

Graph Autoencoders implement an encoder to obtain a latent space node embedding and a decoder to reconstruct a graph from the embedding, see e.g. [1]. This leads to the creation of a new graph over the same set of nodes by deciding for each pair of nodes if they should be linked. The decision is based on the similarity of the node embeddings, i.e., their distance in the embedding space. The autoencoder is trained to produce embeddings that reflects as closely as possible the original graph, not to change its structure. However, one could consider training a similar model to reproduce given input-output pairs of graphs.

In conclusion, there are a number if ways in which graph-to-graph transformations can be implemented based on current GNN technology but, apart from HOPPITY, most approaches are motivated by internal optimisation.

References

[1] Stef De Sabbata et al (2024): *Learning urban form through unsupervised graph-convolutional neural networks*. *ResearchGate*. Available at `https://www.researchgate.net/publication/373903509_Learning_urban_form_through_unsupervised_graph-convolutional_neural_networks`.

[2] Federico Barbero, Ameya Velingker, Amin Saberi, Michael M. Bronstein & Francesco Di Giovanni (2023): *Locality-Aware Graph Rewiring in GNNs*. Available at `https://openreview.net/forum?id=4Ua4hKiAJX`.

[3] D. CONTE, P. FOGGIA, C. SANSONE & M. VENTO (2004): *THIRTY YEARS OF GRAPH MATCHING IN PATTERN RECOGNITION*. *International Journal of Pattern Recognition and Artificial Intelligence* 18(03), p. 265–298, doi:https://doi.org/10.1142/s0218001404003228.

[4] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song & Ke Wang (2019): *HOPPITY: LEARNING GRAPH TRANSFORMATIONS TO DETECT AND FIX BUGS IN PROGRAMS*. Available at `https://openreview.net/forum?id=SJeqs6EFvB`.

[5] Xiaoxiao Guo, Liang Xu, Ling Chen, Yan Chen, Han Wu & Philip S Yu (2019): *Cooperative Graph Neural Networks*. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3547–3553.

[6] Adam Machowczyk & Reiko Heckel (2023): *Graph Rewriting for Graph Neural Networks*. *Lecture Notes in Computer Science* 13961(1), p. 292–301, doi:https://doi.org/10.1007/978-3-031-36709-0_16.

[7] Gabriele Taentzer (1999): *Distributed Graphs and Graph Transformation*. *Applied categorical structures* 7(4), p. 431–462, doi:https://doi.org/10.1023/a:1008683005045.

[8] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang & Hyunwoo J Kim (2019): *Graph Transformer Networks*. Available at `https://papers.nips.cc/paper_files/paper/2019/hash/9d63484abb477c97640154d40595a3bb-Abstract.html`.