Unsupervised Learning of 3-colorings using Simplicial Higher-Order Neural Networks

Anonymous Authors[1]

Abstract

We propose Higher-Order Networks (HONs) for historically challenging problems for Graph Neural Networks (GNNs), such as Constraint Satisfaction Problems (CSPs). We apply a simple extension of GNNs to HONs and show its advantages for solving 3-coloring.

1. Introduction

GNNs have demonstrated state-of-the-art performance in molecular property estimation (Satorras et al., 2021), 3D object classification (Hanocka et al., 2018; Gong et al., 2019; de Haan et al., 2020), protein folding (Jumper et al., 2021), and query plan optimization (Marcus and Papaemmanouil, 2018). However, GNNs have also been shown to have fundamental limitations arising from their neighborhood message-passing. There is a deep connection between GNNs and the Weisfeiler-Lehman(1-WL) isomorphism test, a powerful test that distinguishes broad classes of graphs. Similar to GNNs, the 1-WL test iteratively updates a node features by aggregating information from its 1-hop neighbors. 1-WL has been shown to have equivalent expressivity as GNNs resulting in GNNs inheriting the limitations of 1-WL.

In addition, GNN model depth affects the receptive field, how far node information propagates. For a GNN with depth d, node information propagates up to d-hops away. While deeper GNNs have higher modeling capacity, increasing depth does not monotonically improve GNN performance. Deep GNNs are prone to ”over-smoothing” where the learned features for nearby nodes are nearly identical (Oono and Suzuki, 2021). Over-smoothing results in GNNs which cannot distinguish neighboring nodes and lose information about the local topology.

These structural limitations and practical challenges of GNNs cause them to struggle on many common tasks such as clique detection, shortest path approximation, and graph coloring (Loukas, 2019; Morris et al., 2018). Some methods have been proposed that augment a standard GNN with non-local information (Li et al., 2020; Morris et al., 2018; Bodnar et al., 2021) but they suffer from possibly combinatorial computational complexity and explicitly remove locality, a key property of GNNs.

Recent work has proposed using higher-order interactions while still maintaining locality through a special higher-order network (HON) called a simplicial complex (Glaze et al., 2021; Hajij et al., 2020). Simplicial complexes are useful because they enforce a geometry that has nice topological properties. By combining higher-order interactions and maintaining locality, simplicial complexes have been able to outperform GNNs on path approximations, and can distinguish between graphs that are indistinguishable by GNNs (Glaze et al., 2021). It is not well understood when HONs are capable of outperforming GNNs.

In this work, we propose using HONs to learn graph 3-colorings, a historically challenging problem for GNNs. This paper begins by defining message-passing and how it is performed for simplicial complexes. We then formulate a node contrastive loss and define an unsupervised learning approach for 3-coloring. The performance of HONs and GNNs are evaluated and compared on unsupervised learning 3-colorings. The experiments test how both models perform across number of layers, size of training dataset, and number of training epochs. We find that HONs significantly outperform GNNs, reducing the number of coloring collisions by 37% on average.

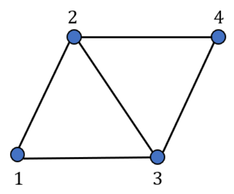
2. Background

2.1. Simplicial Complexes

A simplicial complex is a topological structure that generalizes graphs. In a graph, vertices are adjacent if they are connected by an edge. Conversely, an edge can be thought of as a line where the vertices are its endpoints. In this way, a vertex is 0-dimensional since it is a point whereas an edge is 1-dimensional since it is a line. Simplicial complexes extend this relationship to higher dimensions by generalizing vertices and edges to simplexes. I will first begin with an

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.



B^1	(12)	(13)	(23)	(24)	(34)
1	1	1	0	0	0
2	1	0	1	1	0
3	0	1	1	0	1
4	0	0	0	1	1

B^2	(123)	(234)
(12)	1	0
(13)	1	0
(23)	1	1
(24)	0	1
(34)	0	1

Figure 1. Example of boundary matrices between nodes and edges, and between edges and triangles.

intuition and then formally define simplicial complexes.

Drawing upon the earlier intuition, an edge is a line which is bounded by two vertices. This idea can be extended to 2-dimensions where a triangle is a 2-dimensional face bounded by three edges. Two vertices were adjacent because they formed the boundaries of an edge. Similarly, three edges could be adjacent because they formed the boundaries of a triangle. This idea can be extended to higher dimensions. For example a tetrahedron is a 3-dimensional polygon bounded by 4 triangular faces. This geometric intuition is the core foundation behind how simplicial complexes generalize the topology of graphs.

Formally, let V be a set of vertices. A simplicial complex $\mathcal{S}$ is a collection of subsets of V. For any subset $\sigma \in \mathcal{S}$, all subsets of σ are also in $\mathcal{S}$. A subset that fulfills this condition is called a simplex. A simplex σ is called a k-simplex if $|\sigma| = k + 1$. Connecting back to graphs, a vertex is a 0-simplex, and an edge is a 1-simplex, a triangle is a 2-simplex, and so on. Connecting to graphs, a k-simplex is a $(k + 1)$-clique.

The simplexes in a simplicial complex are connected through boundary relations. A k-simplex will have a boundary formed by $(k - 1)$-simplexes. For example, a 1-simplex(edge) has two 0-simplexes(vertices) as its boundary. Similarly a 2-simplex has a boundary of three 1-simplexes(edges). The boundary relations are contained within boundary matrices or lower incidence matrices. The boundary matrix $B^k_\downarrow$ is a binary matrix where $B^k_{\downarrow ij} = 1$ if $k-1$-simplex i is part of the boundary of k-simplex j. See Figure 1 for an example of boundary matrices.

In addition to boundaries, simplexes have co-boundary relations. The co-boundary relation is intuitively the reverse of the boundary. For example, an edge is in the co-boundary of the two vertices it connects. Similar to the boundary matrix, a co-boundary matrix $B^k_\uparrow$ is a binary matrix where $B^k_{\uparrow ij} = 1$ if simplex i is in the co-boundary of j. Since the boundary and co-boundary are opposite relations, their matrices are related by transpose: $B^k_\uparrow = Transpose(B^{k+1}_\downarrow)$.

3. Methods

3.1. Higher-order networks and simplicial message-passing

Message-passing over simplicial complexes is very similar to graphs. In a graph, messages are passed between adjacent nodes and aggregated over neighborhoods. Instead of adjacency, a simplicial complex passes messages through the boundaries and co-boundaries. The messages are then aggregated over the boundaries and co-boundaries separately. See Figure 2 for an illustration of simplicial message-passing.

More formally, let i be a k-simplex in simplicial complex $\mathcal{S}$. Let $N_\downarrow(i)$ be the set of simplexes in the boundary of i and $N_\uparrow(i)$ be the set of simplexes in its co-boundary. Simplex i will receive a message $m_{\downarrow ij}$ from simplex $j \in N_\downarrow(i)$ in the boundary of i and a messages simplex $m_{\uparrow ig}$ from simplex $g \in N_\uparrow(i)$. These messages are computed similarly to the message in graph message-passing:

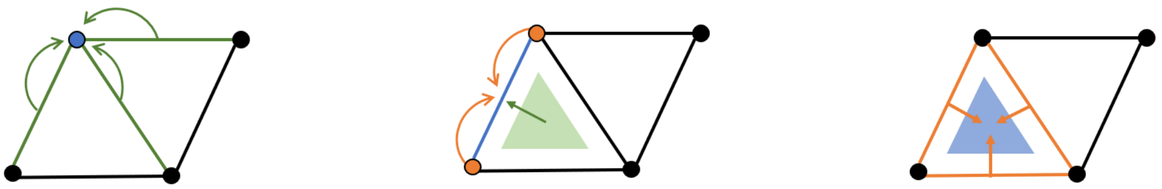
$$m_{\downarrow ij} = \Phi^k_\downarrow(h_i, h_j, b^k_{\downarrow ij})$$
$$m_{\uparrow ig} = \Phi^k_\uparrow(h_i, h_g, b^k_{\uparrow ig})$$

In this definition $\Phi^k_\downarrow, \Phi^k_\uparrow$ are both functions which take the simplex feature vectors and the boundary connection weight as inputs. These two messages are both aggregated separately into vectors:

$$m_{\downarrow i} = \sum_{j \in N_\downarrow(i)} m_{\downarrow ij}$$
$$m_{\uparrow i} = \sum_{g \in N_\uparrow(i)} m_{\uparrow ig}$$

Both of these messages are then combined into a single incoming message m_i by concatenating them. Then just like with graphs, the feature vector h_i is updated using the function $\Phi^k_h(h_i, m_i)$.

HONs use this message-passing scheme but they learn the functions $\Phi^k_\downarrow, \Phi^k_\uparrow, \Phi_{h,k}(h_i, m_i)$ by parameterizing them as multi-layer perceptrons(MLPs). Each layer of an HON performs an iteration of message-passing. The depth of an HON is the number of layers it has. The update equations



(a) Messages from nodes to edges. (b) Messages from nodes and triangles to edges. (c) Messages from edges to triangles.

Figure 2. Simplicial message-passing: Messages from the boundaries (in orange) and messages from the co-boundaries (in green).

for layer t are formally defined as:

$$\begin{aligned}
m^t_{\downarrow ij} &= \Phi^{k^t}_{\downarrow}(h^t_i, h^t_j, b^k_{\downarrow ij}) \\
m^t_{\uparrow ig} &= \Phi^{k^t}_{\uparrow}(h^t_i, h^t_g, b^k_{\uparrow ig}) \\
m^t_{\downarrow i} &= \sum_{j \in N_\downarrow(i)} m^t_{\downarrow ij} \\
m^t_{\uparrow i} &= \sum_{g \in N_\uparrow(i)} m^t_{\uparrow ig} \\
m_i &= \mathsf{Combine}(m^t_{\downarrow i}, m^t_{\uparrow i}) \\
h^{t+1}_i &= \Phi^{k^t}_h(h^t_i, m^t_i)
\end{aligned}$$

This message-passing layer is used to learn features for all of the simplexes in the complex. For graph-coloring the final node features were set to be 3-dimensions and passed through a softmax to convert them into a color distribution.

3.2. Unsupervised Learning of 3-colorings

In graph colorings each node must be assigned a color so that no two adjacent nodes have the same color. This can be achieved by using a GNN or HON to learn color probability distributions for each node as feature vectors by setting the output feature vectors to be 3-dimensional. The softmax is taken to normalize the output node features into color probability distributions c_i. A node will be colored as the color that has the maximum probability. We used a node contrastive loss based on cosine similarity to perform unsupervised learning. The loss function is defined as

$$\text{loss} = \frac{1}{|E|} \sum_{(i,j) \in E} c_i \cdot c_j$$

The loss function measures the cosine similarity between the learned color distributions c_i, c_j for adjacent nodes. In a fully correct solution, adjacent nodes will be assigned different colors so their color distributions will be orthogonal. The minimum cosine similarity between color distributions is achieved when they are orthogonal so this loss function correctly matches the goal of 3-coloring.

4. Experiments on 3-coloring

4.1. Generating 3-colorable Graph Dataset

We generated a dataset of 26,000 3-colorable graphs with between 40 and 60 nodes using the algorithm from (Lemos et al., 2019). This algorithm was chosen because it produces difficult coloring instances. The graph instances are constructed by constructing graphs on the verge of phase-transition: a 3-colorable graph G where there exists an edge that can be added to G that breaks 3-colorability. This is achieved by generating a random graph and using a CSP solver to verify it is 3-colorable. Edges are then added one at a time until the CSP solver cannot find a 3-coloring. The last 3-colorable instance is then returned.

The input node features were set to be a color distribution vector with equal $\frac{1}{3}$ for each color. For the HON, the data was pre-processed by constructing the simplicial complex up to rank 2 (triangles) of the input graphs. The rank was chosen because a 3-colorable graph cannot contain any simplexes of rank greater than 2. All of the edge and triangle features were set to 1 as an uninformative input.

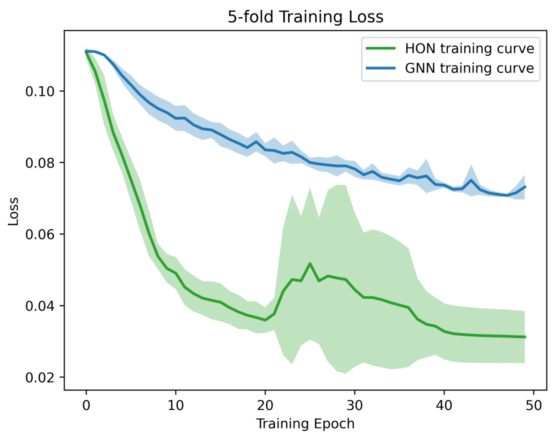
4.2. Model Configurations

The GNN used 32 dimensions for the hidden node features before outputting a final 3-dimensional color distribution. The HON used 8 dimensions for the hidden features for the nodes, edges, and triangles with the final layer outputting a 3-dimensional node color distribution. These dimensions were chosen to keep the number of trainable parameters per layer equal between the two models.

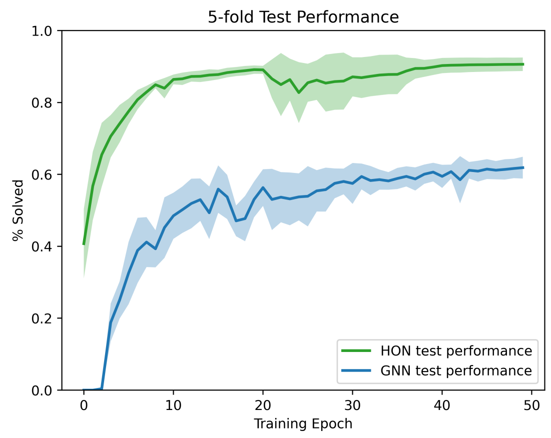
4.3. Experiments

For 3-coloring, we measure the task performance as the proportion of nodes which do not share a color with their neighbors. This constitutes how close the model was to achieving a valid 3-coloring. The performance of the HON and GNN are compared in two experiments.

In the first experiment we swept over model depths $[4, 8, 12, 16]$ both the HON and GNN. These models were trained for 30 epochs using a subset of 5000 training examples and evaluated using 500 test examples. The highest



(a) HON and GNN training loss



(b) HON and GNN test performance.

Figure 3. Comparison of HON and GNN in 5-fold cross validation experiment. HON reaches lower training loss and achieves higher test performance compared to GNN.

performing depth for both was then used in the second experiment.

In the second experiment we performed a 5-fold cross validation which partitions the full dataset into 5 sets. Both models are then trained for 50 epochs using 4 of the sets with the remaining one being used as a test set. We repeated this 5 times choosing a different test partition each time and report the average performance achieved by the HON and GNN models over these 5 runs. We also report the average training epoch time for both the HON and GNN.

5. Results

5.1. 5-Fold Cross Validation

Model	Training Loss	Test % Solved	sec per epoch
GNN	$0.071 \pm 4.1e{-4}$	0.62 ± 0.03	3.38s
HON	$0.031 \pm 6.7e{-3}$	0.91 ± 0.02	17.32s

Table 1. 5-Fold Experiment results. HON outperforms GNN but requires almost 6x training time.

The results shown in Figure 3 largely match those from the depth study. When using the full dataset, the HON increases its test performance to 91% as seen in Table 1. The variance in the HON results are caused by one of the runs where the model training loss and test performance suddenly drop after 20 epochs before rising back up again. It is not clear what caused this as the other 4 runs were very stable. Despite this, the HON continues to significantly outperform the GNN but it does incur almost 6x the training time per epoch as seen in Table 1.

6. Conclusion

In this paper we have demonstrated the improved capability of HONs to tackle the graph coloring problem as archetypical example of a constraint satisfaction problem (CSP). Simplicial complexes allow HONs to propagate information through higher-order structures while still maintaining the core property of locality. We studied the impacts of both model depths and compared their highest performing configurations in a 5-fold cross validation. In these tests the HON consistently outperformed the GNN and converged with fewer training epochs.

These improvements do come at a significant computational cost. The HON takes almost 6x the training time when compared to the GNN despite similar numbers of trainable parameters. The increased costs may become a bottleneck for HONs when scaled to larger problems where concerns about GPU memory usage and training complexity are more relevant.

In this work we restricted the task to 3-colorings and 3-colorable graphs which allowed us to limit the HON to triangles. It could be that the performance of my HON models degrade when used to solve coloring graphs with higher chromatic numbers. This could be an interesting direction of future analysis to better understand how HONs can be scaled up to more complex structures.

References

Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Liò, and Michael M. Bronstein. 2021. Weisfeiler and Lehman Go Topological: Message Passing Simplicial Networks. *CoRR* abs/2103.03212

(2021). arXiv:2103.03212 https://arxiv.org/abs/2103.03212

Pim de Haan, Maurice Weiler, Taco Cohen, and Max Welling. 2020. Gauge Equivariant Mesh CNNs: Anisotropic convolutions on geometric graphs. *CoRR* abs/2003.05425 (2020). arXiv:2003.05425 https://arxiv.org/abs/2003.05425

Nicholas Glaze, T. Mitchell Roddenberry, and Santiago Segarra. 2021. Principled Simplicial Neural Networks for Trajectory Prediction. *CoRR* abs/2102.10058 (2021). arXiv:2102.10058 https://arxiv.org/abs/2102.10058

Shunwang Gong, Lei Chen, Michael M. Bronstein, and Stefanos Zafeiriou. 2019. SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator. *CoRR* abs/1911.05856 (2019). arXiv:1911.05856 http://arxiv.org/abs/1911.05856

Mustafa Hajij, Kyle Istvan, and Ghada Zamzmi. 2020. Cell Complex Neural Networks. *CoRR* abs/2010.00743 (2020). arXiv:2010.00743 https://arxiv.org/abs/2010.00743

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2018. MeshCNN: A Network with an Edge. *CoRR* abs/1809.05910 (2018). arXiv:1809.05910 http://arxiv.org/abs/1809.05910

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (01 Aug 2021), 583–589. https://doi.org/10.1038/s41586-021-03819-2

Henrique Lemos, Marcelo O. R. Prates, Pedro H. C. Avelar, and Luís C. Lamb. 2019. Graph Colouring Meets Deep Learning: Effective Graph Neural Network Models for Combinatorial Problems. *CoRR* abs/1903.04598 (2019). arXiv:1903.04598 http://arxiv.org/abs/1903.04598

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. 2020. Distance Encoding - Design Provably More Powerful Graph Neural Networks for Structural Representation Learning. *CoRR* abs/2009.00142 (2020). arXiv:2009.00142 https://arxiv.org/abs/2009.00142

Andreas Loukas. 2019. What graph neural networks cannot learn: depth vs width. *CoRR* abs/1907.03199 (2019). arXiv:1907.03199 http://arxiv.org/abs/1907.03199

Ryan Marcus and Olga Papaemmanouil. 2018. Towards a Hands-Free Query Optimizer through Deep Learning. *CoRR* abs/1809.10212 (2018). arXiv:1809.10212 http://arxiv.org/abs/1809.10212

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. 2018. Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. *CoRR* abs/1810.02244 (2018). arXiv:1810.02244 http://arxiv.org/abs/1810.02244

Kenta Oono and Taiji Suzuki. 2021. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. arXiv:1905.10947 [cs.LG]

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. 2021. E(n) Equivariant Graph Neural Networks. *CoRR* abs/2102.09844 (2021). arXiv:2102.09844 https://arxiv.org/abs/2102.09844