Learning Isomorphism using Neural Networks
B659 - Cognitively Inspired AI Project

Paventhan Vivekanandan
University id - 0003397339

Indiana University Bloomington
pvivekan@iu.edu

Abstract. This project reports a detailed study of what happens when Machine learning toolkits like the neural networks are employed in solving hard mathematical problems such as detecting isomorphisms which is still not known to be solvable in polynomial time. In everyday life, we often encounter very huge networks such as the internet or the road transportation network which makes solving problems like shortest-path computationally very expensive. But if we transform them to some different space such as a hierarchical clustered network, then we can save some computational effort. If we combine this with resolving isomorphism (computationally) cheaply, then that can eliminate even more computational effort which is the topic of discussion here.

Keywords: Isomorphism, Neural Networks, Hierarchical clustering, Bijection.

Introduction

Two graphs are isomorphic if there is a complete structural equivalence between them. This means that the two graphs should differ only in the naming of their nodes and everything else such as the node degrees, and the node adjacency structure should be the same (Fig. 1). There should also be a bijection mapping between any two isomorphic graphs. In Figure 1, graphs G and H are isomorphic and has a bijection mapping $bij : 1 \rightarrow e, 2 \rightarrow h, 3 \rightarrow a, 4 \rightarrow d, 5 \rightarrow f, 6 \rightarrow g, 7 \rightarrow b, 8 \rightarrow c$ which when applied to G will give H ($bij(G) = H$). Learning whether two graphs are isomorphic or not is not known to be solvable in polynomial time. In this project, we discuss how to use neural networks to learn graph isomorphisms. Such neural networks can be applied in resolving interesting problems such as routing optimization in an internet graph, or shortest paths in road networks, or deobfuscating (reverse engineering) transformations applied to a control-flow graph[1].

To understand how big the graph isomorphism problem can grow in complexity lets look at the number of connected components for graphs of node size 1 to 16.

[1] The complete code implementation for this paper is available at https://github.iu.edu/pvivekan/Isomorphism_NNet

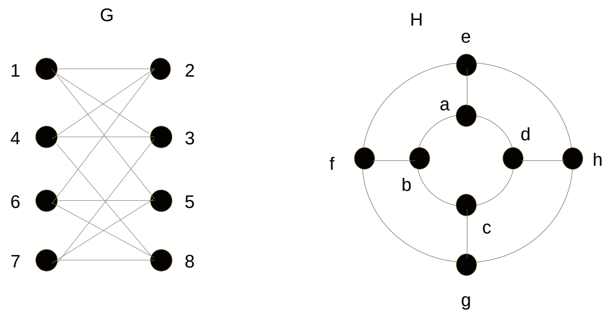


Fig. 1: Graph Isomorphism

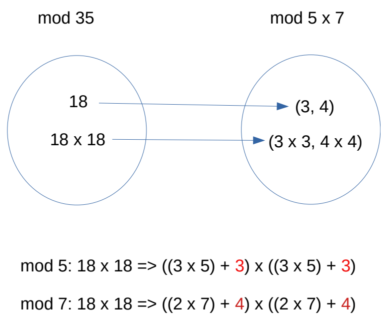
nodesize	*connectedgraphs*
1	1
2	1
3	2
4	6
5	21
6	112
7	853
8	11117
9	261080
10	11716571
11	1006700565
12	164059830476
13	50335907869219
14	29003487462848061
15	31397381142761241960
16	63969560113225176176277

The number of connected components is increasing at exponential rate. If we consider each connected component as an isomorphism class, then the total number of isomorphisms for graphs with even smaller node size can be huge. For graphs of node size up to 7, we have 992 distinct connected graphs (or isomorphism classes) and approximately 3,80,000 isomorphisms. This count gets much bigger for graphs of higher node size. The definition of isomorphism says that two isomorphic graphs should differ only in the naming of their nodes. This means we can come up with infinite isomorphisms for a given graph. But if we consider the adjacency matrix of a graph, we can restrict isomorphisms to a finite space.

Background and Motivation

The effort of this project is targeted at eliminating computations from real time applications like programming in software defined networking (SDN). This project is build as part of a declarative SDN programming language called Flange, which takes a network graph as input and performs computations like routing optimization, resource placement, and outputs a modified graph. Flange operates in real time and the goal is to eliminate as much computations as possible out of real time. The same idea can be extended to a road transportation network and incorporated by applications which can give shortest paths dynamically based on current traffic scenario.

This work is motivated by *Chinese Remainder Theorem* [1], which states that when $N = p * q$, where p and q are relatively prime numbers, we have an isomorphism (or bijection) between cyclic group $\mathbb{Z}_N$ and $\mathbb{Z}_p \times \mathbb{Z}_q$ ($\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q$). In Figure 2, the computation 18×18 is transformed into $(3 \times 3, 4 \times 4)$. A large part of the computation gets dropped in mod 5×7 space. This is possible because there is a bijection between mod 35 and mod 5×7. Using similar idea, if we can transform an input network into a hierarchical clustered network, then we can solve problems like shortest path hierarchically from top level to the bottom level. If we use graph isomorphism mapping, then we can eliminate computations for graphs belonging to the same isomorphism class.


mod 35
mod 5 x 7
18
18 x 18
(3, 4)
(3 x 3, 4 x 4)
mod 5: 18 x 18 => ((3 x 5) + 3) x ((3 x 5) + 3)
mod 7: 18 x 18 => ((2 x 7) + 4) x ((2 x 7) + 4)
Fig. 2: Chinese Remainder Theorem Example

Problem Description

The study for this project focuses on the internet graph as the target network but the future goal is to extend its application to a control-flow graph as well. If we consider a network graph (internet or a road transport network), we have a natural hierarchical structure with opportunities for clustering. If we can achieve a hierarchical clustering with clusters of limited size, then we can resolve problems like finding shortest paths from top level to the bottom level in the hierarchy. Each cluster will represent some isomorphic class[2]. Once we compute shortest

[2] Some related work [8][4][7] based on symmetry and automorphism has been investigated in the literature.

path for one member in an isomorphic class, then we can use the constructed neural networks to eliminate computations for solving again for any other members in the same class. This way we have to solve only for 992 graphs (assuming we have isomorphisms resolved up to node size 7) and eliminate computations for nearly 3,80,000 graphs. If we can extend the training for neural networks to cover graphs of node size 8 then we can save computations for nearly 4 million graphs at the cost of solving 10000 graphs. This is achievable with little more time and computational resources. But it is computationally infeasible to train the neural networks for graphs of size above 10. So, we need new techniques to extend resolving isomorphism for graphs of higher node size. One approach is to use a neural network to achieve clustering of an input graph using cluster coefficient and closeness centrality measures and resolve isomorphism separately inside the clusters. Also, we can try more sophisticated neural network architectures such as the Differentiable Neural Computer (DNC) [6] and Graph Neural Networks [10].

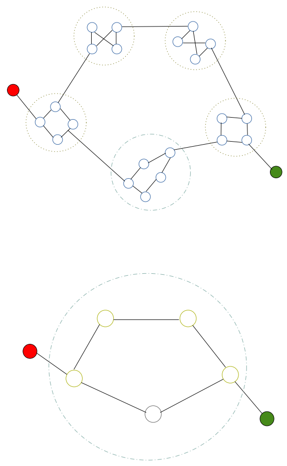


Fig. 3: Hierarchical clustering of network

By achieving a hierarchical clustering of input graph (as in fig. 3), we can resolve shortest path for each cluster independently. At a higher level in the hierarchy, we may encounter already resolved isomorphism or a new one. One limitation of this approach is that the cost involved in hierarchical clustering. But the internet or the road transportation network is mostly a static network and the cost of building a hierarchical clustering can be considered as an one-time effort.

Implementation Details and Observation

As part of this project, two feed-forward neural networks were implemented. The first neural networks is used to detect the isomorphism class of an input graph. The input to the first neural networks is an adjacency matrix representation of the graph. The second neural networks is used to produce a bijection mapping between the input graph and a graph representative of the isomorphism class. The input to the second neural networks is the Laplacian representation of the graph. Both neural networks performed with 99 percent accuracy on average[3].

In order to generate the training data, a tool called pynauty [2] was used. Pynauty is a python interface to nauty which is built in C. Nauty has two functionalities. First, on giving two graphs as input it can say whether they are isomorphic or not (it returns a Boolean value). Second, it can compute the automorphism group of an input graph (it returns the cyclic generators, group size, orbits, and the number of orbits). *Automorphism* is simply a permutation of vertices of the same graph while isomorphism is a bijection between two different graphs. The training data was generated by first permuting the adjacency matrix for different edge and node combinations of a given node size and then nauty was used to check for isomorphism before adding the graph to a specific isomorphism class. Then the training set was augmented by randomly shuffling and increasing the weights in the adjacency matrix of the graphs in each isomorphism class. The accuracy of the neural network increased with increase in the volume of the training data. Each isomorphism class will have a pre-solved computations (like shortest path) for a representative graph. The first neural network takes in an input graph as adjacency matrix and gives out the isomorphism class. The second neural network takes in the graph Laplacian of the input graph along with the neighborhood information of a specific node (a row in the graph Laplacian) and gives out the node in the representative graph to which the input node corresponds to. In this way, we can produce a bijection mapping for all nodes between the input graph and the representative graph. Note that the second neural network is trained separately for each isomorphism class. So far, the training has been done only for graphs of node size up to 7. The plan is to extend it for node size 8, and then use other techniques like trying with DNC and neural network based clustering for graphs of higher sizes.

One question that would arise is that if we have pynauty to tell us if two graphs are isomorphic then why we need a neural network to achieve the same. There are few clear and important advantages that the neural network based isomorphism detection can give us. First, the pynauty isomorphism detection has exponential running time while the neural network runs in constant time (although the training is intensive). Second, the pynauty returns only a Boolean value saying if two graphs are isomorphic or not. It does not gives us a bijection mapping and also cannot say which isomorphic class an input graph belongs to. These computations if done naively using pynauty can be computationally very intensive and inefficient. Also, using neural networks we can say how close the isomorphic structural equivalence of two input graphs are which is not possible using pynauty.

[3] The complete code implementation for this paper is available at `https://github.iu.edu/pvivekan/Isomorphism_NNet`

Applications and Future Work

From application perspective, an experimentation has been done as part of this project to analyze the structure of the internet graph. It has been observed that the internet graph is a scale-free network, i.e. the degree distribution of nodes follows power law [5]. Also, there is a natural hierarchy in the graph due to the existence of different layers of internet service providers (ISPs). The ISPs have their own autonomous systems (AS) and that provides an opportunity to cluster based on clustering co-efficient, closeness centrality, and geographic location measures.

Another network such as the road transportation network also has a well-defined geographic locations (latitude and longitude) and has a natural hierarchical clustering which is worth investigating as another application paradigm. A more challenging network would be the control-flow graph. Cryptographic tools such as $Tigress$ [3] performs obfuscation of control-flow graphs [9] to prevent intellectual theft caused be reverse engineering of softwares. Such transformations are often structure based (to some extent) which provides us an opportunity to deobfuscate them using neural networks trained on isomorphisms. If we can map a transformation to an isomorphism class then the same transformation will repeat itself at other parts of the graph and also at higher hierarchical levels. There will be some randomization involved which will make this process more difficult but even if we can achieve small success, it can be significant from cryptographic perspective which in turn can drive forward the smartness of obfuscation algorithms.

Such an experimentation using control-flow graphs is left as part of the future work. Also, generalizing for graphs of higher node sizes using sophisticated architectures such as the memory-based differentiable neural computer (DNC) [6] and the graph neural networks (GNN) [10], training neural networks to cluster input graphs based on clustering co-efficient and closeness centrality, training neural networks for detecting isomorphisms in directed graphs (required for control-flow graphs) are all planned as part of the future work.

Conclusion

As part of this paper, we have discussed the initial efforts involved in solving isomorphisms using neural networks. Even though we have used only simple feed-forward networks, if has been found that they are very effective and performed exceptionally well in detecting isomorphisms. But still we have to wait and see how they can perform on graphs of higher node sizes. One thing to keep in mind is that there is no known polynomial time algorithm to resolve graph isomorphism and we are fishing for the big blue whale using neural networks as our weapon. This experimentation is far from over and we still have plenty of ammunition (such as the DNC and GNN) left to investigate.

References

1. Chinese remainder theorem, https://mathworld.wolfram.com/ChineseRemainderTheorem.html
2. Pynauty - a python extension for nauty, https://web.cs.dal.ca/~peter/software/pynauty/html/guide.html
3. Tigress code obfuscation, https://tigress.wtf/index.html
4. Bonato, A., Del, D., Wang, C.: The structure and automorphisms of semi-directed graphs **27** (01 2016)
5. Funel, A.: The graph structure of the internet at the autonomous systems level during ten years. Journal of Computer and Communications **07**(08), 17–32 (2019). https://doi.org/10.4236/jcc.2019.78003, http://dx.doi.org/10.4236/jcc.2019.78003
6. Graves, A., Wayne, G., Malcolm Reynolds, e.a.: Hybrid computing using a neural network with dynamic external memory. Nature **538**, 471–476 (2016), https://doi.org/10.1038/nature20101
7. MacArthur, B.D., Sánchez-García, R.J., Anderson, J.W.: Symmetry in complex networks. Discrete Applied Mathematics **156**(18), 3525–3531 (Nov 2008). https://doi.org/10.1016/j.dam.2008.04.008, http://dx.doi.org/10.1016/j.dam.2008.04.008
8. Sánchez-García, R.J.: Exploiting symmetry in network analysis. Communications Physics **3**(1) (May 2020). https://doi.org/10.1038/s42005-020-0345-z, http://dx.doi.org/10.1038/s42005-020-0345-z
9. Udupa, S.K., Debray, S.K., Madou, M.: Deobfuscation: reverse engineering obfuscated code. In: 12th Working Conference on Reverse Engineering (WCRE'05). pp. 10 pp.–54 (2005). https://doi.org/10.1109/WCRE.2005.13
10. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications (2019)