Detecting graph isomorphisms using deep learning

Muzamil Yahia
University of Hawaii at Manoa

Abstract
This project aims to detect graph isomorphism from a given graph. We first generate a bunch of isomorphic graphs and non isomorphic graphs as our data. We train our neural network using these data, and then test and state the loss and accuracy.

2012 ACM Subject Classification Models of learning

Keywords and phrases Deep learning, graphs, graph automorphism, graph homomorphisms

Digital Object Identifier 10.4230/LIPIcs..2018.1

1 Introduction

A *directed* graph $G \equiv (V, E)$ can be written as a mapping $G \equiv \begin{tikzcd} E \arrow[r, shift left, "\delta"] \arrow[r, shift right, "\rho"'] & V \end{tikzcd}$ where $\delta(e)$ and $\rho(e)$ indicate the source and destination for each edge $e \in E$. A neighbour of a vertex $v \in V$ is given by the set of vertices $N(v) = \{ w \in V \mid \exists e \in E.\, \delta(e) = v \wedge \rho(e) = w \}$ and we say w is adjacent to v if $w \in N(v)$. Undirected graphs add the following restriction $w \in N(v) \implies v \in N(w)$ on E.

A graph isomorphism over G is a permutation $p\colon V \to V$ that preserves the adjacency of vertices i.e. $w \in N(v) \Leftrightarrow p(w) \in N(p(v))$. Checking whether two graphs g and g' are isomorphic amounts to find such permutation.

2 Testing for isomorphisms

Given two graphs $G \equiv \begin{tikzcd} E \arrow[r, shift left, "\delta"] \arrow[r, shift right, "\delta"'] & V \end{tikzcd}$ and $G' \equiv \begin{tikzcd} E' \arrow[r, shift left, "\rho'"] \arrow[r, shift right, "\delta'"'] & V' \end{tikzcd}$. If $V \neq V'$, or $E \neq E'$ i.e. different sizes of vertices or edges, then two graphs are trivially non-isomorphic. The case $|E| = |E'| \wedge |V| = |V'|$ is much harder and in the worst case we may try all permutations $p\colon V \to V'$ and test whether a bijection $e_p\colon E \to E'$ induced by p, will provide an isomorphism.

The Graph Isomorphism is an NP-hard problem, and it is not known if it is NP-complete. It is also not known if belongs to co-NP. The fast known result for solving graph isomorphism which stood for 3 decades is $e^{O(\sqrt{n \log n})}$ by Babai et al. [?]. However for some special cases of graphs i.e. class of graphs with a forbidden minor and classes of graphs of a topological minor are shown to be solved in polynomial time by Grohe in 2013 [?], however these methods are not very useful in practice.

Most common practical algorithms that detect isomorphisms rely on "canonical relabeling" which takes an advantage of an invariant property of graph isomorphisms; If two graphs are isomorphic then they have the same canonical relabeling. The most successful approach uses a method of fixing a vertex v and then refining the set partitions of V through a strategy called "individualization and refinement" [?]. The method was implemented by McKay in 1979 in C, in a software package named "Nauty". The software has seen many improvements through the decades and recent improvement was done by McKay and Piperno in 2014 in a new software called "Tracy" [?].

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 1; pp. 1:1–1:7
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

3 Deep learning and graph isomorphism

Since the graph isomorphism problem is NP-hard, we might want to look for heuristic methods to determine whether two graphs are isomorphic for majority of graphs. As neural network has shown recently the ability to learn complicated features hidden in a given set of data, and with careful construction of these networks, they can be generalized to unseen data input. Therefore, a classifier neural network would be a suitable choice for detecting graph isomorphism. We will specify our methods in the subsections below.

3.1 Problem statement

Given a graph $G = (V, E)$, find a neural network model that can receive as input a graph $G' = (V', E')$, check if G is isomorphic to G' with high probability.

3.2 Method

We encode the graph G using adjacency matrix, and produce m isomorphic graphs $\{G' \mid G' \cong G\}$. To do that, we take to our advantage the simple forward approach and generate a random permutation of vertices $p\colon V \to V$, and then rewrite the adjacency matrix to reflect that as shown the diagram below:

$$\begin{tikzcd} v \arrow[r, dashed, "p"] \arrow[d, "e"] & p(v) \arrow[d, "e_p"] \\ w \arrow[r, dashed, "p"] & p(w) \end{tikzcd}$$

To generate non isomorphic data, we followed a simple approach first by reducing the set E by randomly removing one edge, and later removing random number of edges.

I note that this approach of non-isomorphism is trivial, and does not tackle the heart of the problem. Future work would be to generate all isomorphisms graphs using Nauty/Tracy packages and then use these data for training and testing. For now I will focus only on the trivial case of changing the size of edges E.

3.3 Results

Sequential neural network struggled to learn effectively for some large set of edges and learnt effectively for smaller sets. Figure **??** shows loss and accuracy for a sequential neural network of 8 layers. The accuracy is higher for 10–20 edges but then get lower and lower for rest of 20–120 edges.

Convolutional neural network with two convoluted layers did the opposite. For small set of edges 10–50 the model showed little accuracy. However, for larger set of edges 60–120, the convoluted model showed better accuracy in training and also in validation.

4 Conclusion and future work

Graph isomorphism is an NP-hard problem which might be tackled with tools of deep-learning. Sequential models were effective for only small set of edges, while convolutional network worked better for graphs that are more than half-full. In future work, we would implement combinations of graph of same number of edges, graphs with different sizes, and also graph homomorphisms.

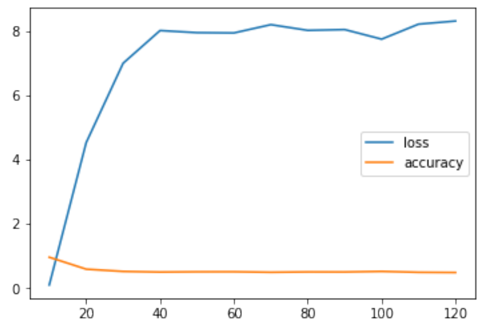

Figure 1 Loss and accuracy for a sequential neural network of 8 layers and 6 epochs. The underlying graphs consists of 16 nodes and edges from the set $\{10, 20, \ldots, 120\}$.

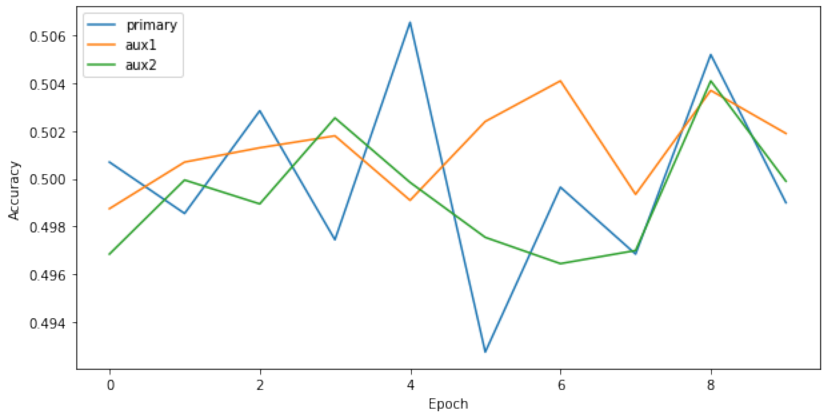

Figure 2 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 10 edges.

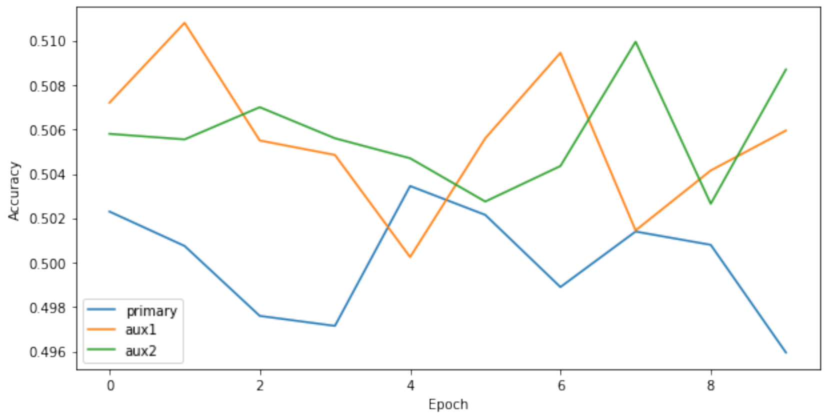

Figure 3 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 20 edges.

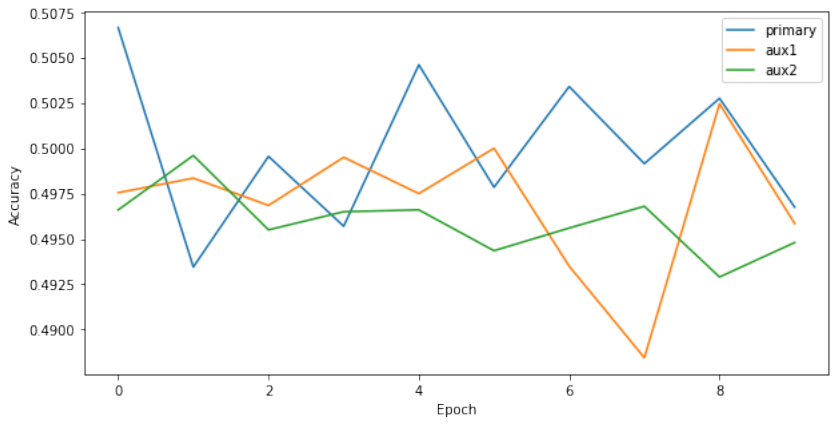

Figure 4 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 30 edges.

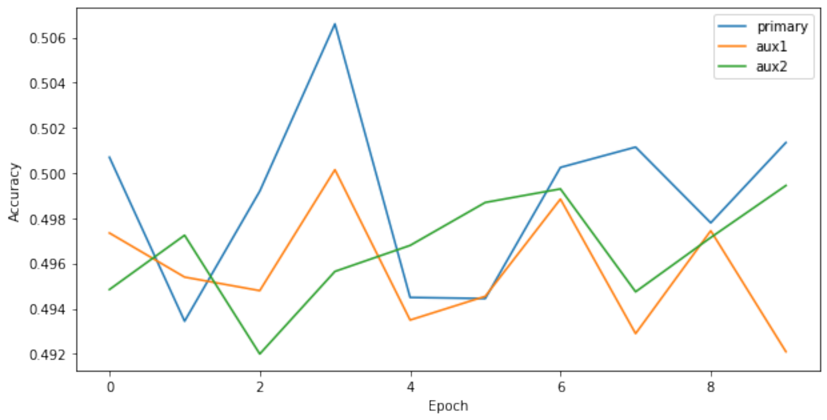

Figure 5 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 40 edges.

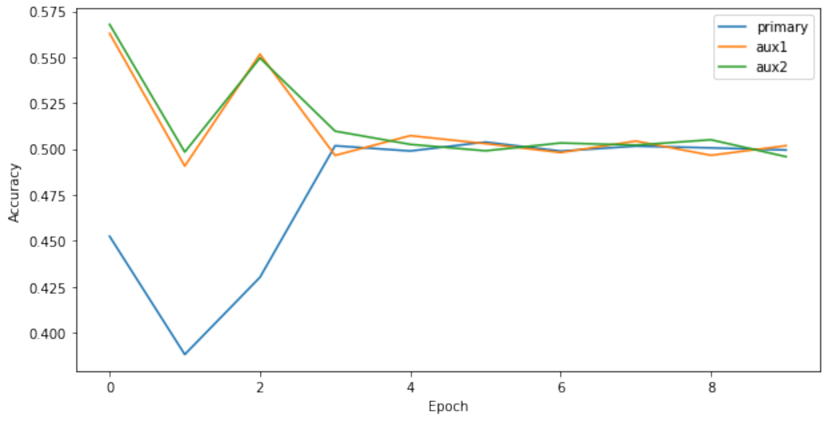

Figure 6 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 50 edges.

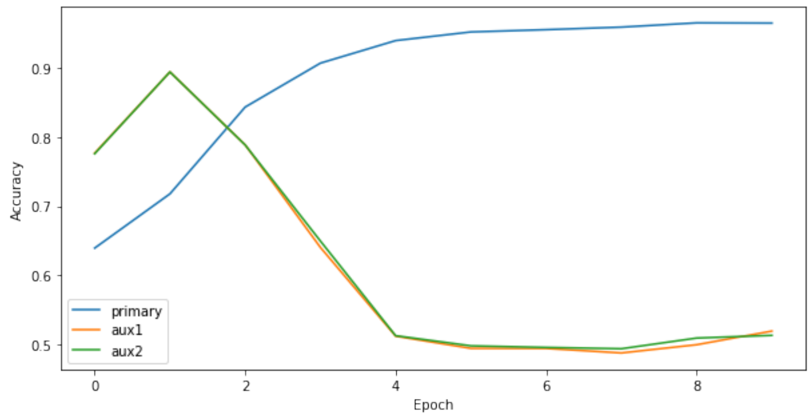

Figure 7 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 60 edges.

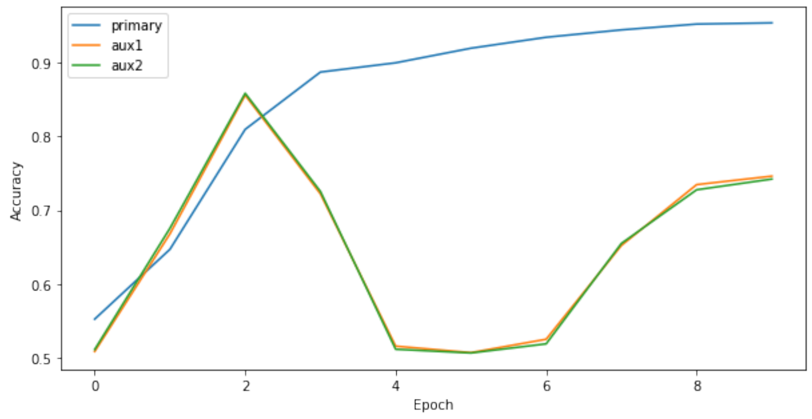

Figure 8 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 70 edges.

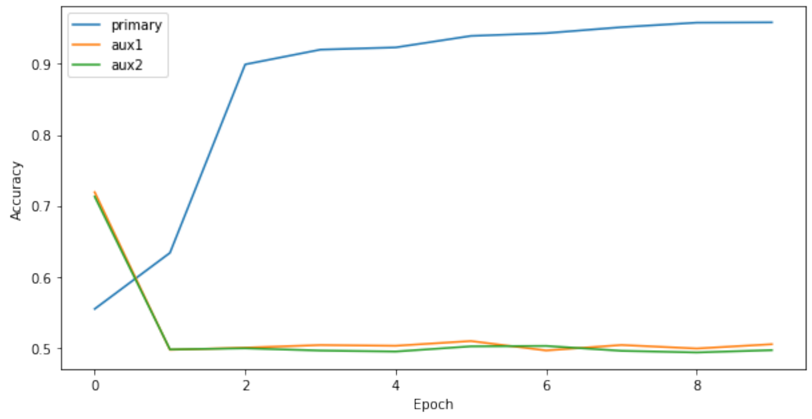

Figure 9 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 80 edges.

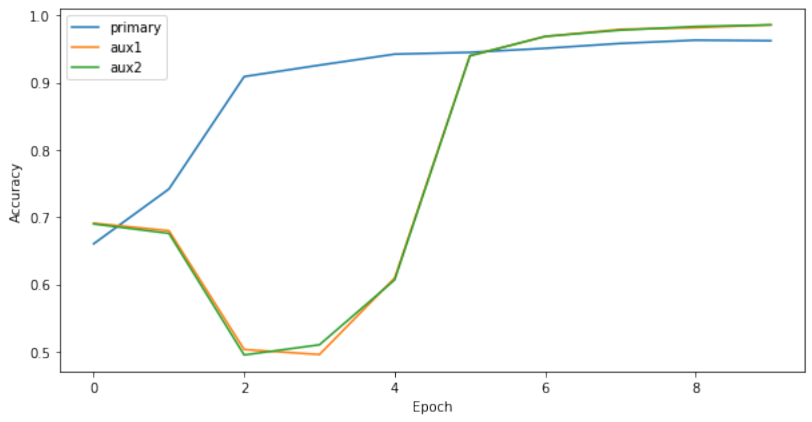

Figure 10 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 90 edges.

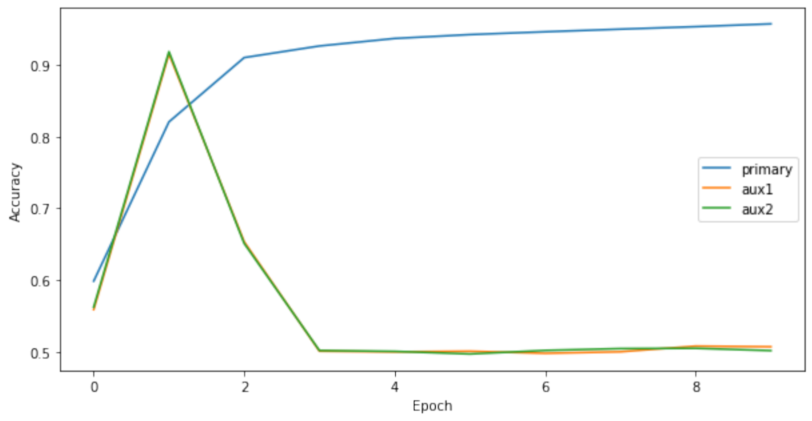

Figure 11 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 100 edges.

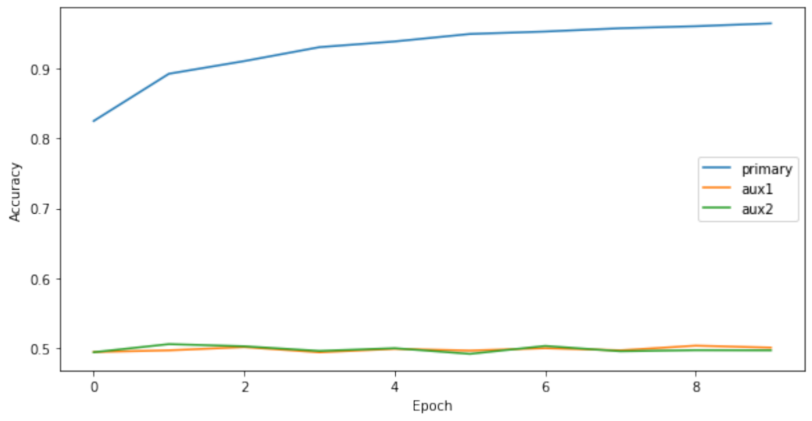

Figure 12 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 110 edges.

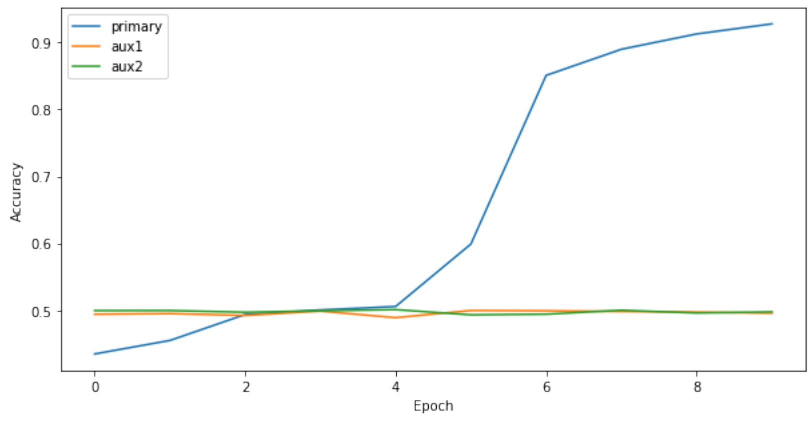

Figure 13 Loss and accuracy for a convolutional neural network of 2 convolutional layers and 10 epochs. The underlying graphs consists of 16 nodes and 120 edges.