Analysis and Optimization of Security Infrastructure with Deep Learning Methods

Franklin E. Diaz

January 29, 2022

Abstract

Our teams rely heavily on the Terraform software made by Hashicorp as a means of automating network infrastructure deployments in our labs and customer environments. Our teams also develop Open Source Terraform modules. These deployments and modules take the form of graph-structured data. This paper describes efforts to apply Graph neural networks (GNNs), a form of deep learning, to perform analysis of this infrastructure automation and determine the likelihood of computationally improving it.

1 Introduction

A cursory consideration of the technology we are using to define and deploy network infrastructure reveals a declarative use of graphing as a descriptive language and in a definitive manner. Given this inherent use of graphs, and a seemingly meteoric rise in the availability of Artificial Intelligence research and tooling, it feels like a logical next step to transform these designs into a computationally suitable format. The use of Python, open source modules, and the proliferation of scholarly papers and tutorials is a powerful combination indeed. As the common refrain goes, gathering and making data suitable for analysis may be the largest challenge facing the would-be data scientist.

With these ideas in mind, this paper describes attempts to realize improvements in security and efficiency related to automated deployment of network infrastructure by bringing powerful technology to bear on their source. Capturing attempts to understand what's possible and find the edges of the problem space while learning new things is the true goal of this work. Usable, useful or even interesting results are a happy accident. Using AI to uncover inferences, and realizing there are more, better questions we can be asking is a mindset we can only cultivate by an undertaking like the one this paper describes.

Where possible, code samples and screenshots of resultant execution are provided in an attempt to make the material accessible to a greater number of readers. Studying and working in what is considered a sub-field of artificial intelligence reveals a limited number of folks engaging in these methodologies. Mass market books on the field of artificial intelligence are typically targeted towards helping folks get started in machine learning. While these books are very interesting and educational, they don't get us far enough down the path of AI to help us with what seem to be niche problems in deep learning (see figure 1).

What graph neural network layering can we discover that will lead to useful features of the nodes? Node classification for making predictions about the nodes. Predictions on edges and their features. Learning shared functions about nodes and edges to make predictions. Is the regularity of security infrastructure (or perhaps certain sub-graphs) enough to realize solutions to inductive problems?

Finally, the mathematics that underpin this effort, although off-putting for some, must be recognized and understood by folks wishing to have questions answered with artificial intelligence. There is elegance and even beauty in this math.

> *No human investigation can be called real science if it cannot be demonstrated mathematically.* - Leonardo Da Vinci

2 A Very Brief Primer on Graphing and AI

While an in depth explanation of graph theory is beyond the scope of this paper, some background will be provided here to make the paper accessible to a wider audience. A collection of related references will be included at the end of this paper for those who may be interested.

2.1 About Graphs

Use of the term graph in this paper simply means a representation of information in a way that describes a group of things, which we will refer to as "nodes". A relationship between nodes is called an "edge". Graphs have many properties, most relevant of which to us include whether a graph has directionality, As with many things in computer science, we are affected by problems of scale. It often makes sense to constrain our scope to a given sub-graph or "neighborhood".

Most likely when this paper refers to a "graph" the reader is safe to assume a reference to an image like figure 3. The nodes are represented as colored circles, the edges are represented by arrows. The use of arrows instead of lines means the graph is "directional" (rather than "undirected") and will be referred to as a "digraph" hereafter.

2.2 About Deep Learning with Graph Neural Networks

The first item of note is Consider figure 1.

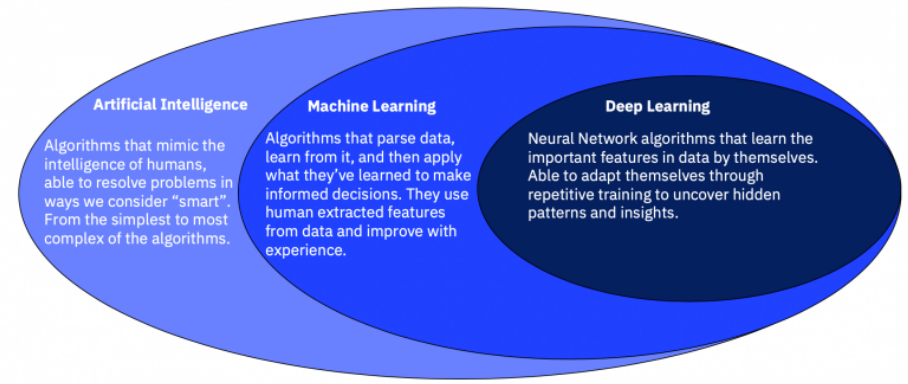

Figure 1: From AI, machine learning and deep learning: What's the difference? by Rodrigo Ceron.

3 Collecting Infrastructure Code as a Data Source

Some level of familiarity on the reader's part with Terraform and by extension, Infrastructure as Code is assumed.

Terraform users create files that define network infrastructure. When this infrastructure is deployed via Terraform, a state file is created to keep track of the current deployment. This state file is essentially graph structured data. We can export this graph data from the actual (or planned) state and ingest this with our deep learning pipeline.

3.1 The Data Collection Process

A tool for gathering and uploading from data sources is used to send relevant data from various sources to a storage bucket in Google Cloud. Raw data is processed and tagged with a UUID as a unique key to associate data with it's source. The source code for this Python tool is available in the Github repository for this project.

Identifying and versioning data as we would any other code or documentation artifact eases the burden of handling our data and allows us to more easily work with larger data sets.

While the collection process is relatively straight forward to implement and maintain for this project on a small hobbyist/researcher scale, going forward it would make more sense to use a community supported framework such as dvc to leverage the efforts of folks doing similar work.

4 Using Infrastructure Code as Data

Once a Terraform code base has been initialized, the user has the option to generate a directed graph of the infrastructure using Dot. Dot is a language used to describe graphs. Consider the following declaration of a Cloud Function for Google Cloud.

4.1: Terraform Declaration Example

```
resource "google_project_service" "cloud_function" {
  project                    = var.project_id
  service                    = "cloudfunctions.googleapis.com"
  disable_on_destroy         = false
  disable_dependent_services = false
}
```

Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs

and networks. Conversion of Terraform HCL to a directed graph representation is trivial thanks to Graphviz and Dot. It is possible to reproduce the digraph displayed in figure 2 using the shell script provided.

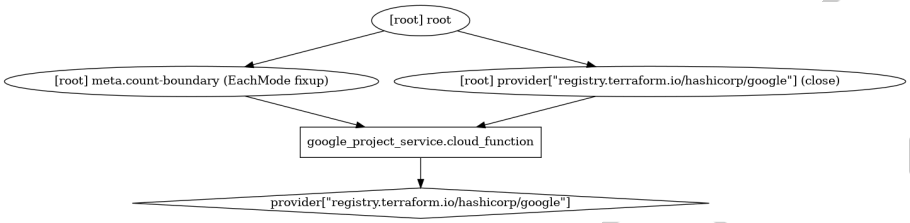

Figure 2: DOT output from Terraform Files as a digraph

5 Data Collection

Run a small tool to collect data and pass results to a Google Cloud storage bucket.

5.1 Conversion of a Digraph

Once our Terraform infrastructure has been output as a directed graph, we need to bring it in to Python so we can perform operations on it. To run Terraform commands in an automated fashion, the python-terraform module is used.

Directed graphs are ingested into Python as an object using the networkx module. Now we are prepared to perform computational operations on the digraph.

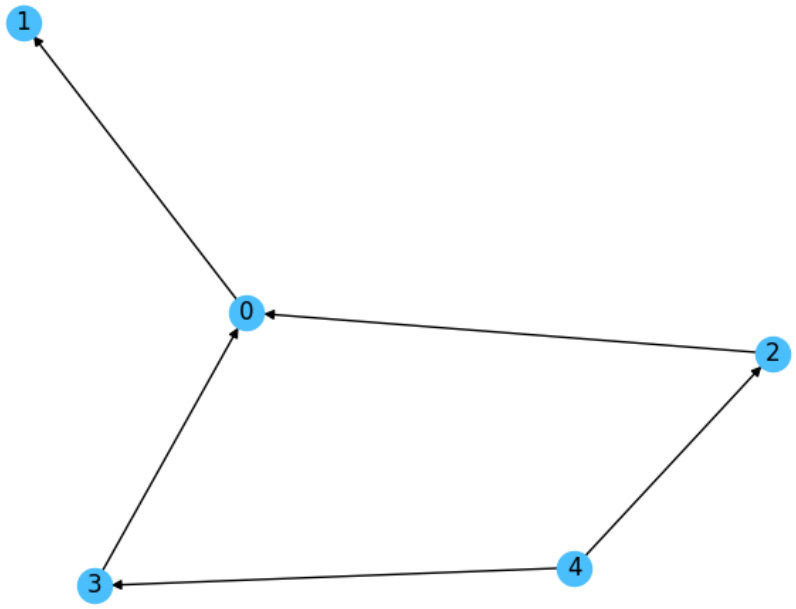

Figure 3: Directed graph as ingested by Networx

Note that new numerical labels are displayed in place of the original labels. These are stored in the node objects in case we need them later. The root node of the adjacency matrix is labeled as the 0th element in the graph. Also of note is that the graphs displayed in figure 2 and figure 3 can be described as isomorphic.

Cleaning data using the pyjanitor module.

5.2 The Adjacency Matrix

The adjacency matrix is a mathematical representation of our graph. The nodes are represented in a columnar format known as a matrix. Continuing with our previous example, the representation of our

Add the identity matrix to the adjacency matrix using an update rule. The goal is to prevent the central node from being excluded. Doing this without increasing the scale requires something like multiplication by the inverse of the degree matrix, aka "mean pooling". The most popular graph convolutional layer is the GCN update rule. We will not be able to attach complex features to the edges.

5.3 Node Embedding

Assigning weights and features to the edges of the graph. Labeling.
Attach feature vectors to nodes.
Compute arbitrary message vectors between neighbor nodes.

5.4 Convolutional Layers

6 Resources

Here is a quick list of things that I found interesting and relevant.

1. There is an excellent talk on YouTube by Petar Veličković, a research scientist at DeepMind.
2. The Kipf and Welling GCN repo on GitHub.

Revision History

Revision	Date	Author(s)	Description
v0.1	Nov. 20th, 2021	Franklin Diaz	Initial Draft

References

[1] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *CoRR*, abs/1912.12693, 2019.

[2] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[3] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[4] Ruslan Kuprieiev, Saugat Pachhai, Dmitry Petrov, Pawel Redzynski, Casper da Costa-Luis, Peter Rowlands, Alexander Schepanovski, Ivan Shcheklein, Batuhan Taskaya, Jorge Orpinel, FÃ¡bio Santos, Gao, Aman Sharma, David de la Iglesia Castro, Zhanibek, Dani Hodovic, Nikita Kodenko, Andrew Grigorev, Earl, Nabanita Dash, George Vyshnya, maykulkarni, Max Hora, Vera, Sanidhya Mangal, Wojciech Baranowski, Clemens Wolff, and Kurian Benoy. Dvc: Data version control - git for data & models, December 2021.