Whitepaper: Graphnet for Diseasespread

By Varun Krishnan and Robert Lin

May 2019

1 Abstract

This whitepaper and the accompanying code demonstrates our model for understanding the flow of the spread of a disease. We use a slightly modified SIR Model to generate simulated disease spreads, and then we try to learn this information using two different methods. First, we use Google Deepmind's recently released Graph Nets library, which has the capacity to store (and dynamically update) disease capabilities such as its infectivity rate, transition power between edges, and the number of sick (and healthy) people in different nodes of the graph. We then compare the results of this simulation to a regular Feed-Forward Neural Network, and show that using GraphNets to store more data results in significantly better train and test accuracy - with the mean squared error being 3 times using Graphnet than with a regular neural network.

This is a short, 10 page whitepaper meant to accompany the code - it provides basics of the model and model results. Cases where the code goes into more depth on results are indicated clearly, and the code is explained in much more depth within the Collaboratory Notebook. We also present a powerpoint which represents these results in a neater fashion. Running the Google Collab Notebook takes about 10 minutes, which is why we present key results here.

2 Simulation Variables

First, we provide a means for understanding how our simulation works. (This is explored in more depth in the Google Collaboratory Notebook, where we examine the variables used in determining this simulation).

Here are the major variables that we control when determining how this model works: We made these tradeoffs in order to correspond with how, in real - life, a disease might spread.

1) The number of cities/nodes (in this case, we have 3 nodes, in order to account for memory-related problems)

2) The number of globals (in this case, 1 global variable corresponding to the disease infectivity rate).

3) The number of edges (proportional to the number of nodes $\hat{2}$, because each city is connected to each other city).

4) The maximum distance between two nodes.

5) The initial number of sick people (1000 in every node).

6) A global attribute corresponding to the infectivity constant (In this case, we initialize it to 1 * 10ˆ8).

7) A random update variable, through which edge values update.

Through these, we generate the Nodes, Edges, and Global variable at timestep 0.

3 SIR Model: Introduction

The SIR Model has two major components when diseases spread - transportation and transmission.

During Transport, the disease spreads as follows:

$$h_i(n') = h_i(n) + \sum_{j=1}^{n}(h_j(n) - h_i(n) * w_{ij}$$

$$s_i(n') = s_i(n) + \sum_{j}(s_j(n) + s_i(n) * w_{ij}$$

Where n is a timestep, h and s are the number of healthy and sick people, i is the relevant node being updated, and w is a weight from node i to node j. (Mnemonically, this can be viewed as inflow minus outflow)

During Transmission, the disease spreads as follows:

$$s_i(n+1) = s_i(n') + a * s_i(n') * h_i(n')$$

$$h_i(n+1) = h_i(n') - a * s_i(n') * h_i(n')$$

Where a is the infectivity constant.

4 Generating Graph using these parameters

.

In order to generate the graph, we needed the following variables at each timestep. We first determined them for timestep 0, and then determined them for the rest of the graph:

1. We set the **Node Values** = to sick people per node, and dynamically updated it.

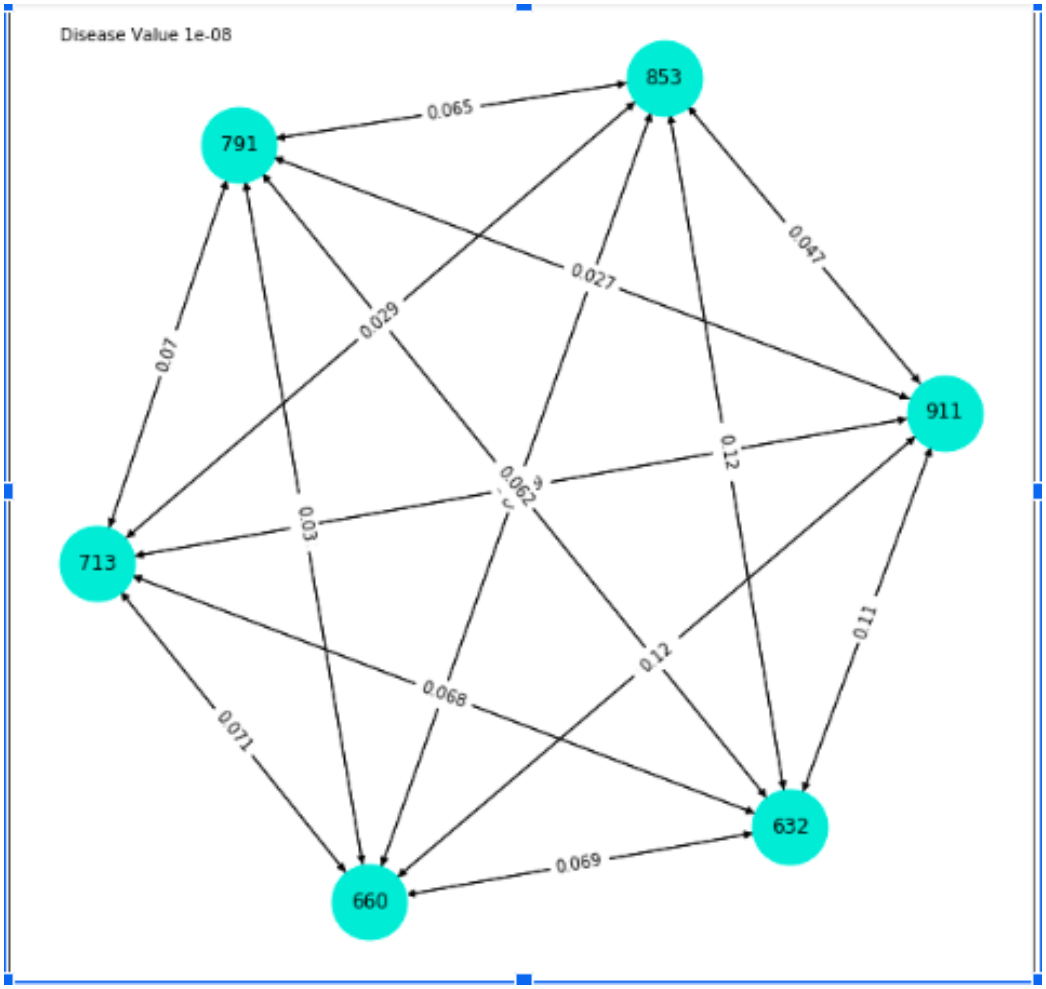
2. We set the **Edge Weights** = to a F(Population Values, Distances between Nodes, Random Variable, Update Parameter), where the edge weights were directly proportional to population value and inversely proportional to distances between nodes (in order to model how edge weights would likely look during the spread of a disease).

3. We set a **Global Variable** = to the infectivity constant.

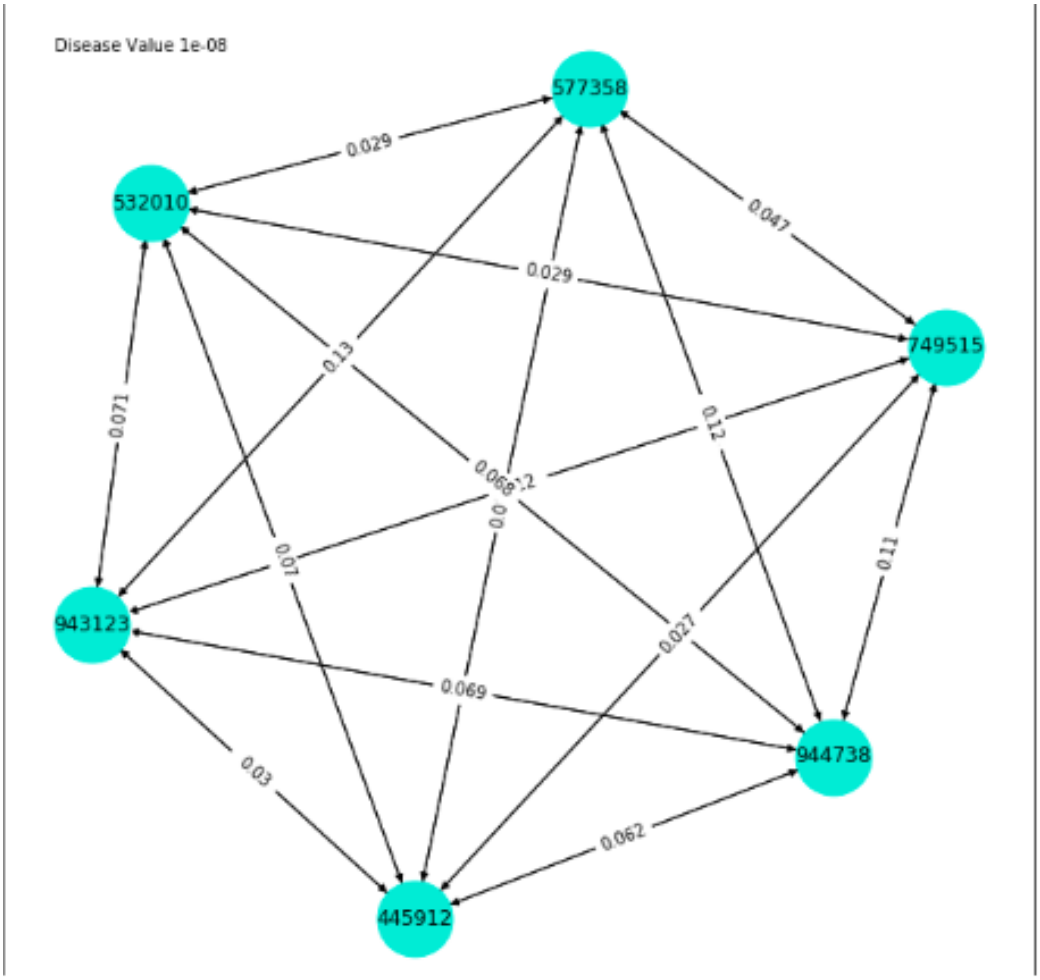
We then used the SIR Model to generate 100 time steps worth of data, and randomly updated the edge weights according to a constant random weight. This allowed us to create new node values according to each timestep (as visualized in the graph below).

5 Visualizing the Graph

Here are a couple visualizations of what these graphs look like, in a graph with 6 nodes, to show how the graph changes from timestep 1 to timestep 100. Note that the values in blue are the number of sick people in a node, and the edges contain directed edge weights corresponding to disease transitivity (These are directed, but this image hides a few of the directed edge weights under the node values).
Timestep 1:


Disease Value 1e-08
853
791
911
713
632
660
0.065
0.047
0.027
0.029
0.07
0.12
0.062
0.03
0.11
0.12
0.068
0.071
0.069

Timestep 100:



In both these cases, the edges are similar (but change slightly due to a changing constant), and the node values change from about 1000 to slightly over 1 million.

6 Training and Testing Component of Project

The next task in this project was to train a Neural Network (using Graphnet) that could learn this data. In essence, we were trying to feed Graphnet our SIR Model, and allow it to use these parameters to learn the SIR Model. We ran 100 simulations of the data with 100 timesteps used an 80/20 train-test split to split the dataset, and then we used an AdamOptimizer to fit a Neural Network according to the values in the Graph Dictionary. This allowed us to predict, for any given timestep, the estimated number of sick people at the next timestep, and we calculated a loss function corresponding to this.

The intuition of a neural network using graphnet is that it dynamically updates each of the three parameters (Nodes, Edges, and Global Variables) during iteration. This allows for much more information to be conveyed than during a regular neural network.

We also ran into various issues while doing this Train/Test Function, most notably memory leaks - these are outlined in the Google Collab Notebook. This made it necessary for us to do our simulation with only 3 nodes (even this used 4 GB of memory - simulations with more nodes almost always caused the Google Collab Notebook to crash).

7 Results

First, here are two graphs of our model's predictions. The first graph contains the predicted vs actual number of sick people at each node, according to our model.

The second graph contains deltas in these two values. Note that, while these get higher over time, our model is virtually always able to predict this within at least 5 %.

Figure 1: Raw Sick Populations

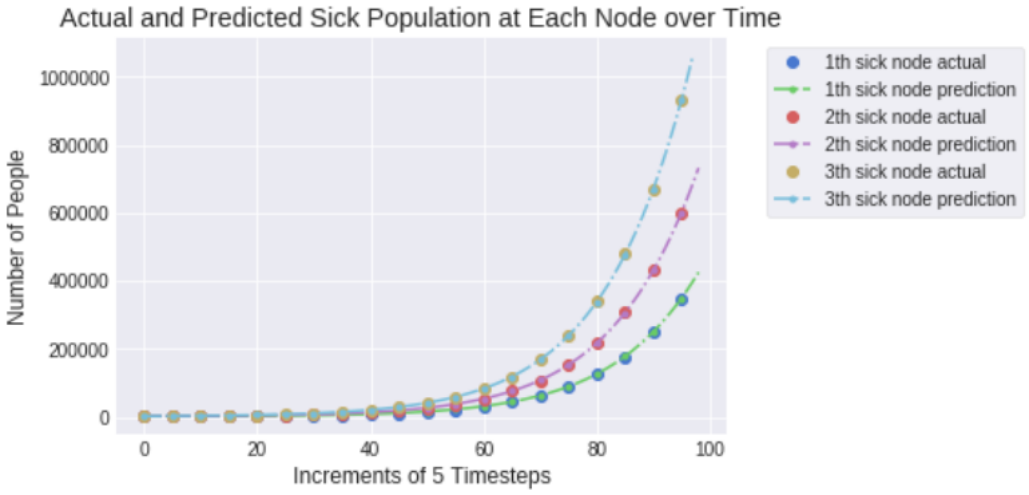
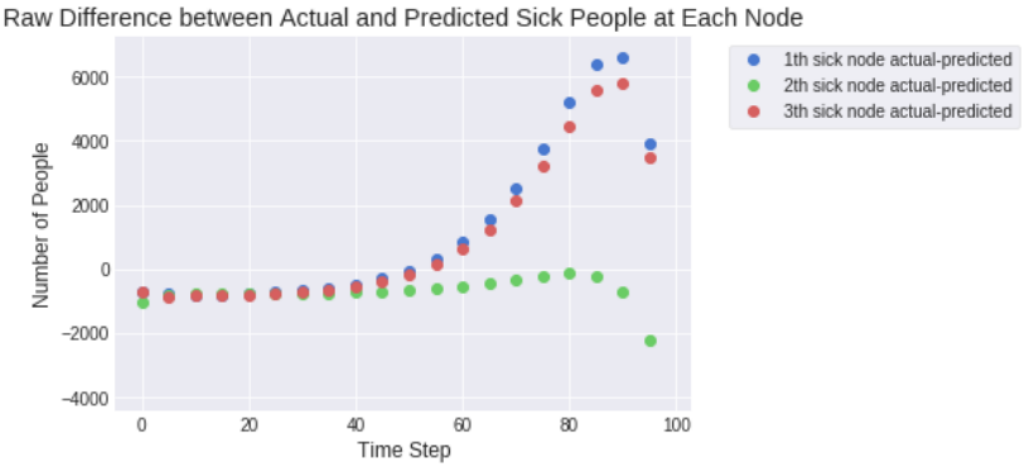


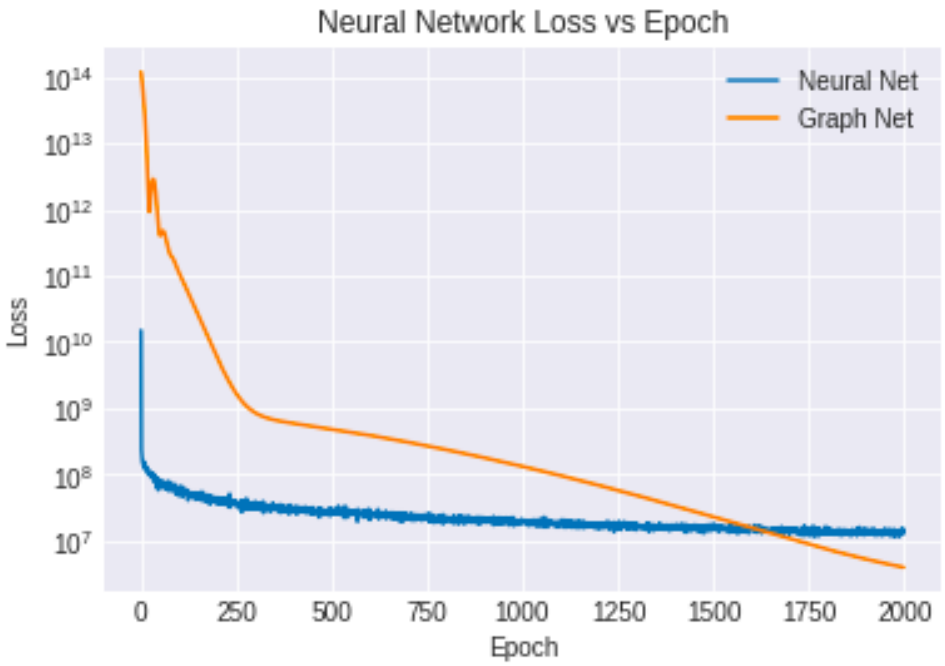
Figure 2: Raw Deltas



8 Regular Convolutional Neural Net Comparison

We also compared our model to a regularly trained Neural Network. In a regularly trained feed forward convolutional neural network, we did not take into account factors such as edge weights - we simply trained the network based on mappings of node values from one timestep to another. In this case, we used an RMS optimizer, as this worked better than an Adam Optimizer in the case of a regular neural network.

Below is a loss of Mean Squared Error training loss over time after 2000 epochs of the simulation. The regular neural network actually does better after a few epochs (due to bad initializtion within Graphnet) but Graphnet eventually does better over many iterations (and converges much better).



9 Discussion and Conclusions

These results seem to indicate that our approach, using Graphnet, represents a relatively strong improvement over a regular Neural Network, as is evidenced by its ability to store more information over time. It represents a unique way to train data on graphs, and it was very exciting to learn about it during this project.It still does have issues, however, such as bad initialization of nodes during training.

9.1 Challenges

We ran into many different challenges over time - among them, choosing model parameters pseudorandomly, deciding between various training methods, and trying to make sense of our data! We also had to deal with various logistical issues such as memory leaks and Google Collab (fun but at times extremely painful)!

9.2 Potential areas of future work

The simplest method of future work is to add more parameters to our model. We could take into account factors such as recovery and deaths to make our model slightly more complex - though we decided to represent a system where these do not occur.

We also have a couple of potential different ideas. One is Professor Brenner's suggestion of simply solving the SIR model as a differential equation, without even needing to run simulation data. We also would love to try some more libraries, such as the Graph Convolutional Network (GCN) library recently released, but which takes in data in a different format from Graphnet.

10 References

1. Battaglia, P., Hamrick, J., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., . . . Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks.

2. Google Collaboratory Notebooks: GraphNet

3. Keras, SKLearn Documentation Libraries