March 2019

Deep Learning Method for Social Networks

Daniel Bowen McKay
Worcester Polytechnic Institute

James Andrew Corse
Worcester Polytechnic Institute

Manuel Santana Gonsalves
Worcester Polytechnic Institute

Deep Learning Method for Social Networks

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
In partial fulfillment of the requirements for the
Degree in Bachelor of Science
In
Computer Science
By

_____
James Corse

_____
Manuel Gonsalves

_____
Daniel McKay

Date: 3/21/2019
Project Advisors:

_____
Professor Xiangnan Kong, Advisor

Abstract

The goal of this project was to find a better solution for the influence maximization problem in a faster and more accurate manner than the current leading algorithms. We utilized deep learning models to accomplish this goal. The results revealed a significant improvement in time efficiency, while maintaining a slightly better accuracy.

Table of Contents

Executive Summary

The goal of this project was to create a deep learning model to solve the influence maximization problem in a faster, more efficient, and more accurate manner than current leading algorithms. The influence maximization problem can be used to help with Viral marketing. Viral marketing is a business strategy relying on people suggesting a product to their friends over a social network. If a company has a certain number of free products they could hand out to customers, then this model can predict optimal targets to receive these products in order to maximize the spread of influence of the product over the given network. The three main algorithms we tested as comparisons to our model were a random selector algorithm, a sum of edges algorithm, and a greedy algorithm. The greedy algorithm is the best in maximizing the number of people, or nodes, that were influenced enough to be activated among the three algorithms. This can be seen in Figure 4.1 where all the algorithms are compared over the same network with a range of the amount of seed nodes, number of nodes budgeted to start the influencing process.

Here we can lay out some specifics of the model and how it improved upon the time and efficiency over greedy.

In making our model, we designed it to take in the network and iteratively change the initial influence distributed to the seed users that it deems most effective in order to accomplish our goal. As far as the total influence spread, how effective the model is can be seen in Figure 4.1. It can also be seen the trend of effectiveness with how efficient the model is as the network size increases in Figure 4.2 where the time taken to complete is compared to the greedy algorithm. From this data, we were able estimate the time requirement to run the greedy algorithm on our real data set that we look at in Chapter 4 Section 2 would be several days. From the findings of our comparisons, we discovered the true effectiveness of our deep learning model.

Chapter 1: Introduction

Viral marketing is a relatively newer solution to advertisement within social networks. While the concept of "word of mouth" sharing is timeless, the utilization of it has not been fully explored. Several methods for solving this model have arisen such as randomly choosing the seed users, choosing the seed users based on how many connections or friends they have, and selecting the seed users based on who has the most immediate influence of the remaining possible options, and more. Many newer methodologies for maximizing the spread of influence by the selection of seed users have done so by either mix and matching parts of the baseline methods or even attempting to alter them in an attempt to make them perform even better. Despite the interest in being able to maximize the effectiveness of viral marketing, we are still far from the optimal solution to the problem of selecting the seed users.

The goal of this project is to create a deep learning model that will solve the influence maximization problem with regards to viral marketing on a social network, and beat out the other leading algorithms used to solve the same problem with respect to time efficiency and influence spread in the network. For the purposes of this project, we built artificial networks in order to test our model against the other algorithms (Section 3.1) and then used a real network from DBLP, a computer science bibliography website, in order to obtain results on real data comparing our model and the best of the alternative algorithms (Section 4.2). The alternative algorithms we use for comparison in this project are a best-of random algorithm, a sum of edges algorithm, and a greedy algorithm (Section 2.4).

Many of the sources we found during our initial research solved the influence maximization problem using the aforementioned algorithms, occasionally adding another alternate algorithm by trying to combine the best aspects of each one. However, we found no sources looking to utilize machine learning to find a more efficient solution. In order to accomplish our goal, we designed our model to take in the network and iteratively change the initial influence distribution to the seed users that it deems most effective.

In the following chapters, we will walk through our process of designing and testing our deep learning model to solve the influence maximization problem. In Chapter 2, we talk about what is being done in the viral marketing field currently to solve the same problem and what steps other people have done to improve the current methods. Then, in Chapter 3, we will go through the process we took to create our artificial networks, design our model, and then utilize our model to analyze a real social network. In Chapter 4, we look at the comparisons of the alternate algorithms to find the best, and use said algorithm as a baseline comparison point for our learning model. We will also cover the results from using our model on the real data as well

as a small case study that we have included in our analysis. Finally, in Chapter 5, we will cover a final analysis and summary of the entire project and list potential future work.

Chapter 2: Background

2.1: What is a neural network

A neural network (NN) is a nexus of interconnected nodes used to perform a task. There are thousands or even millions of nodes in a NN, designed to roughly emulate the neurons of a human brain. These nodes are organized into layers where data can flow one direction at a time. Data first flows forward during the feed forward procedure, and then backwards during back-propagation. Every node is connected to every node in the previous layer and every node in the next layer. Every connection is an edge that has a weight value associated with it. When a value is sent via an edge, it is multiplied by the weight value before it reaches the next node. Once the feed forward procedure is complete, the NN then begins its back-propagation process. During the back-propagation, the NN sees the values that should have been outputted and compares them to what it actually outputted. Then the NN goes back through the nodes and edges and adjusts the values of the weights that are impacting the end values incorrectly. This process, when repeated over all of the training data, leads to the neural network being able to correctly perform a task a large percentage of the time (Hardesty, 2017).

2.2: Standard uses of a Neural Network

Neural networks are extremely prominent in the machine learning field. The ability to handle large datasets has become more feasible with the use of NNs. One of the largest uses for NNs is in image classifying. The reason that this setup is so powerful for image classifying is

that it is possible to use all sorts of images for the model. For example, if a model was made to identify chairs, then you traditionally would run into the problem of hard coding a machine to look for a wide range of characteristics that are common between most chairs. The problem is that there are always different looking chairs. With a NN, it can be shown thousands of images of all sorts of different looking chairs and it will learn how to identify them by itself and will then be able to extrapolate the important details in order to identify a strange looking chair it had never seen before (ujjwalkarn, 2016).

Another popular use for NNs is in designing artificial intelligences for games. One of the most notable cases for this usage is known as AlphaGo. AlphaGo was a deep learning model using two NNs in combination. AlphaGo utilized the NNs in order to employ the Monte-Carlo tree search to run through the game to look at potential outcomes and decide on the most optimal move for the turn. During a game, AlphaGo has one NN running through possible moves while the other NN runs through future game states in order to evaluate the moves. AlphaGo was so successful that it eventually earned the highest of Grandmaster ranks within the Go ranking system. This followed AlphaGo winning a match four to one against Lee Se Dol, who is considered as one of the best Go players of our era (Google Research, 2016). Neural networks are a powerful tool that when given enough training examples can be used to perform tasks very quickly and accurately.

2.3: Viral Marketing and Social Networks

Viral marketing is a “form of word-of-mouth marketing that relies on social networks” to spread word of a product amongst their friends (Viral Marketing. 2017). Word of mouth has always been used to spread the word of products, but now with the internet being so prevalent you have faster, potentially global, coverage. The basic idea of viral marketing is that a product

is given to a number of seed users and then those seed users spread word of the product to their friends. Then those friends spread the influence to their friends and the cycle continues, similar to a virus (Viral marketing. 2017). This is an effective technique because, as shown in a 2014 study, 92% of people said that they trust recommendations by friends and family. One example of viral marketing is with hotmail. By making their email service free and having a message at the bottom of every email that advertised their free service, they gained 12 million users in 18 months (Viral Marketing. 2017).

In this project we looked at online social networks, first using artificial networks as described in Section 3.1, and then on real network data. For the purposes of our specific project, each person is represented as a node and each node is a part of a graph, similar to Figure 2.1. This graph represents a real social network. Each node in the social network has a threshold value which represents the amount of influence they need to receive to become activated. Once a node is activated it will then spread its influence to all nodes that it is connected to. All nodes have their own amount of influence over their neighboring nodes. For example, if there is node A and node B, the amount of influence node A has over node B can be different from the influence node B has over node A. This is to represent the fact that friends may not react equally to being recommended a product from each other. Additionally, there is a starting budget to give out the product to seed nodes. This products given to the starting nodes can be any value, they do not have to be whole products. Instead of just giving one seed node a product for free, we use the starting budget as a discount on the product. This allowed us to split the budget up further among users in order to further optimize the propagation of the network.

Figure 2.1: Image of Example Social Network

2.4: Alternative Algorithms

2.4.1: Random Algorithm

The goal for using a random algorithm is to set a baseline as to how well the simplest algorithm could determine an answer. The algorithm would randomly select people in the network to give free products to and see how the propagation went. Because it takes a consistent amount of time to find a group of people, the runtime for random to find an answer is O(p) where p equals the number of initial products. This means that the algorithm can be run very quickly on any size graph and can be run multiple times until a good answer is found. If a good answer can be found randomly than there would be no reason to continue to find another algorithm to get a consistently better answer because you could run random until some acceptable answer is found.

2.4.2: Sum of Edges Algorithm

The sum of Edges Algorithm applies a heuristic to every node and gives it an initial value. The way we used a heuristic in this problem was by looking at the total influence that a person had on all of their friends and choosing the individuals that had the greatest influence.

The reasoning for this was simple, by giving the most influential people free products they would initially spread the most influence into the network. The runtime for the algorithm would be O(n), where n equals the number of nodes in the network. This is because the heuristic must be applied to every node and finding the top n values in a list can be done in O(n) time as well. There has been a lot of research into developing better heuristics because of how quickly this algorithm can run, but from our research none of consistently beat the greedy method.

2.4.3: Greedy Based Algorithm

Greedy algorithms work great for other problems. Greedy algorithms will choose the optimal choice for the problem it is working on at the time it comes across the decision. A good example of where a greedy algorithm works perfectly is Kruskal's algorithm for generating a minimum spanning tree from a graph. The beauty of the algorithm and this problem is that it guarantees an optimal answer at the time the algorithm terminates (Cormen, Leiserson, Rivest, & Stein, 2009). The problem of influence maximization, however, is it does not have a greedy algorithm that can guarantee an optimal answer.

The greedy algorithm is simple. At every iteration it finds which node would activate the most people in the network with the current set of seed nodes. Once it finds that node, it permanently adds that node to the list of seed nodes. Then the algorithm continues to find the most optimal node at each step until the budget has been exhausted. This can not guarantee an optimal answer because there is no way to prove that the optimal choices have been made at every step of the iteration. The graph is just too complicated and there are too many permutations to check if the best answer had been found. This solution to the problem is one of the more popular ways of solving it. For small graphs the runtime is very small and the answer that is produced is usually better than random and most heuristic based approaches.

2.5: Related works

The alternative algorithms that we used are the most common algorithms that are used when trying to maximize the influence in a social network graph. In one of the most influential papers on the matter, Maximizing the Spread of Influence through a Social Network, the random, degree-based, and distance centrality algorithms are used as baselines to compare to the greedy algorithm that is analyzed in the paper (Kempe, Kleinberg, Tardos, 2003). From the results of the paper, it becomes apparent that the greedy algorithm was most effective in activating the highest number of nodes when compared to the other algorithms. It is acknowledged that in using real data sets, it is nigh impossible to find the absolute solution to the influence maximization problem. That is why in the results, the algorithms have to be compared to each other as opposed to potentially looking at the difference between the results and the "solution."

From papers we read, the leading idea of improvement for influence maximization is to improve the greedy algorithm. In Efficient Influence Maximization in Social Networks specifically, the authors looked to reduce the running time for the greedy algorithm as a way of improving it (Chen, Wang, Yang, 2009). However, the improved greedy algorithm produces equal results to the original greedy algorithm in terms of activated nodes. This lead to the idea of making a model that would beat greedy in running time in addition to the spread of influence using machine learning.

Chapter 3: Methodology

3.1: Graph Maker

The first part of the problem was creating test data, or in this case, test graphs to run the different algorithms on. The graphs that are used for influence maximization problems are

weighted bi-directional graphs with a scale free distribution (Albert, Baraba´si, 2002). Each node in the graph would have a certain threshold, the amount of influence needed for activation, and a list of friends with how much influence they have on that friend. Random numbers generated from a normal distribution were used to create the weights on each edge and the value of the threshold on each node.

The Baraba´si-Albert model is the most popular way of modeling a scale free distribution. The algorithm that goes behind generating this model is built on two properties, growth and preferential attachment. The algorithm starts with m_0 nodes in the network. Then a specific number of edges is added to the network with m_0 nodes at random while not duplicating any edges. Nodes are then added to the network one by one. When a new node is added to the network the node starts with m edges, $m <= m_0$, connected to other nodes that are already in the graph. When choosing the other node in the graph to connect to is where the preferential treatment comes into play. The probability a specific node is chosen is equal to $K_i / \sum_j K_j$, where K is equal to the degree of that specific node. Once the last node is added into the network the algorithm terminates and a network with a scale free distribution will have been built. Figures 3.1 and 3.2 show the distribution of friends in a network with 100 people. With these networks we could test different algorithms without having to try and find real world data and would allow us to test on many different sized networks.

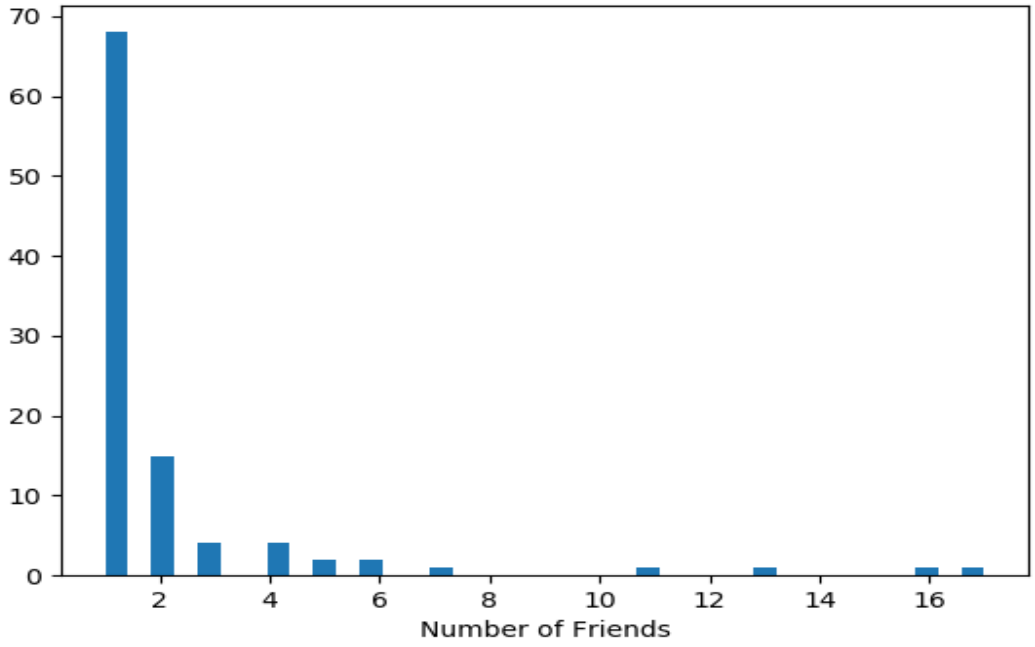

Figure 3.1: Scale Free Distribution $m_0 = 6$ and $m = 1$

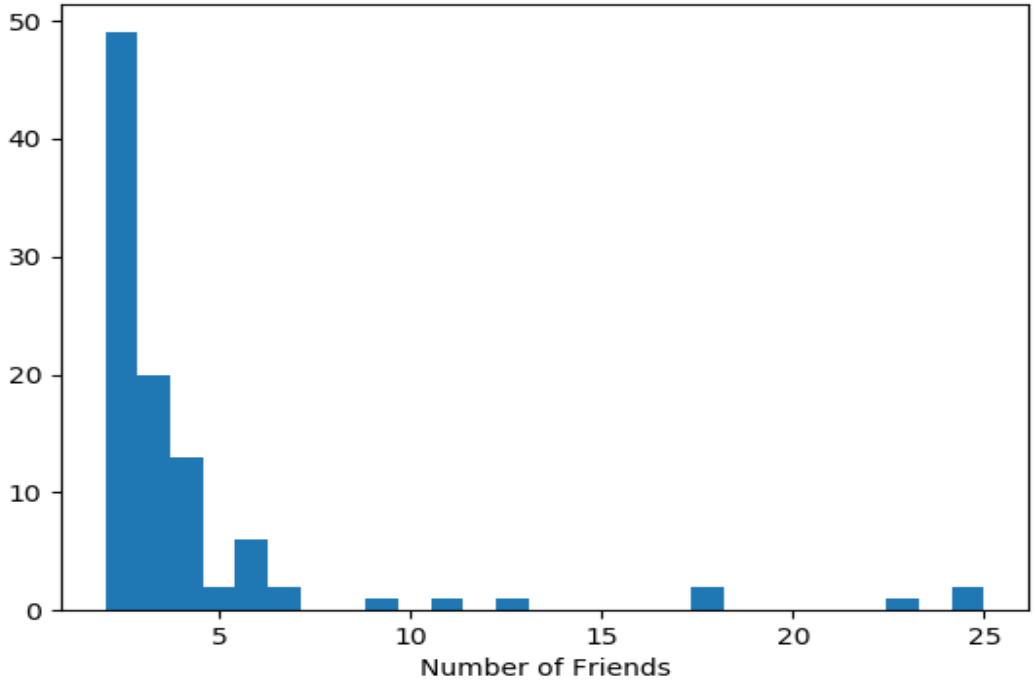

Figure 3.2: Scale Free Distribution $m_0 = 6$ and $m = 2$

3.2: Creating the Neural Network

The goal of the NN here is to simulate propagation through the network. This means that a very specific network is needed to be made for each graph. There are two main parts to the network. The first one is a tensor that holds the values that are going to be put into the neural network, and the second is the normal looking NN. A tensor is a type of list that holds more information, such as the gradient for each value in the list and backwards hooks as to where the gradient should continue when back propagating. This layer is a tensor with a budget for each person. Initially the budget is divided randomly between everyone. This tensor then leads into the normal NN layers by applying a softmax function[1] to the values and multiplying those values by the total budget. This guarantees the right amount of initial influence is in the system.

The second layer looks like a normal NN. For the number of layers in the second part, we used seven layers because the average degree of separation between two internet users is optimally 3.43, but we wanted to try and reach even distant connections (Bahkhshandeh, 2011). Within each layer of the NN, there is a node for each person. There is a negative bias on every node that is equal to the threshold for that particular person. The connections between each layer are equal to the weight of the edge between those two nodes. So if person A influences person B with a weight of 0.69 then the weight on the edge connecting those two nodes between layers will be 0.69. Due to the fact that the bias subtracts the threshold at every layer, the person needs to push their previous value to the next layer. This means that the connection between the same nodes in different layers is 1. If there is no connection between the 2 people then the connection in the neural network is 0 between those nodes and layers. Each node also goes through a

[1] $f(x_j) = e^{xj}/\sum_{k=1}^{K} e^{xk}$ Given a list, all elements in the list are reproportioned so that the sum of the list is 1

sigmoid function[2] before propagating to the next layer because no one will be negatively impacted by not having heard of the product.

Lastly, the weights and biases are for simulating propagation through the network. If these values changed then the network would no longer represent the original graph we were looking at. This means the only value that is going to change after the loss has been calculated and the gradient has been created is the initial budget layer. Figure 3.3 shows what a small 4 person network could look like. The value in the circle with the letter represents the threshold for that node. Figure 3.4 shows the neural network that would be created from the small network. The initial values in the first layer are just random numbers. It also shows how the output from this network will feed into the loss function.

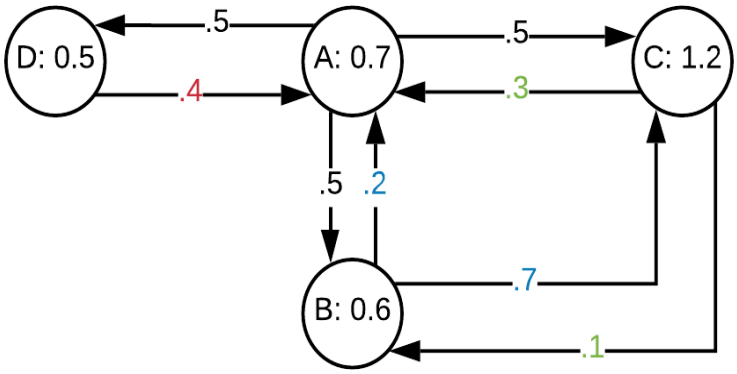

Figure 3.3: Sample original graph

[2] $f(x) = 1 / (1+e^{-x})$ Given an x, returns a value between 0 and 1 based on the x value

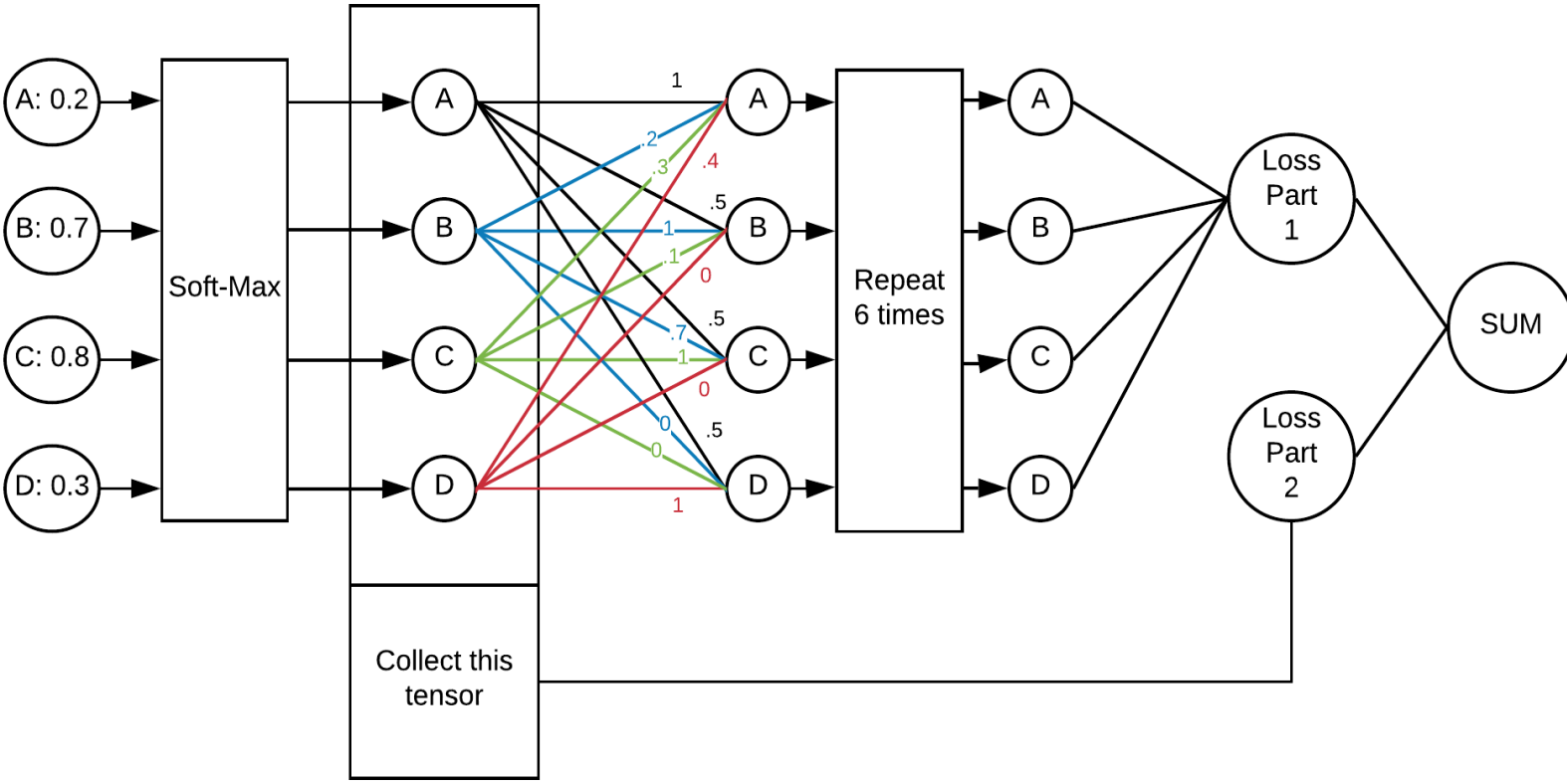

Figure 3.4: sample neural network

3.4: Loss function

The key part for any successful neural network to work is the loss function. This function is a way of scoring how well the network performed on a given iteration. Normally neural networks would use a built in loss function like a cross entropy loss function, but this problem required that we build our own loss function. This loss function is built on two separate parts as Figure 3.4 shows.

The first part of our loss function looks at the output tensor of the network after going through the seven layers of propagation. To see how well the network performed, we created a type of absolute value function. We first subtracted 0.5 from every value in the tensor because

this is what 0 is after going through a sigmoid function. Afterwards, the values in the tensor would represent whether or not the node would be active, negative if they were not activated and positive if they were. The next part was to weight the positive and negative values as to create a score. The function is as follows:

$$f(x) = \begin{cases} -80x & \text{if } x <= 0 \\ x & \text{if } x > 0 \end{cases}$$

The function gives a very large weight to negative values so the nodes are given a larger portion of the budget the next iteration. There was also a small penalty for going over budget at the end to prevent over budgeting. Over budgeting is giving more resources in the budget than is needed to activate a node initially.

The second part of the loss function also dealt with over budgeting. As figure 3.4 shows, it collects the values of how the budget was divided. From the budget tensor, the loss function subtracts each individual's threshold value. Any value that is positive would indicate that there has been over budgeting and a negative value would show that there was not enough budget used to activate the target node. Then the loss function uses a Relu[3] function on this new tensor to eliminate any negative values since we only care about over budgeting here.

After both functions have been applied to their respective parts, the loss function then gets the total sum of both tensors and this represents the score for that iteration of the algorithm. The -80x was found through our own user optimization steps.

[3] f(x) = max(0,x) Given an x, returns x if x is above 0, or returns 0 if x is below 0.

3.5: Moving to Real Networks

The next task was being able to apply our learning algorithm to real world networks. Originally when generating practice networks we looked at graphs with a maximum of 8,000 nodes to get a general time estimation for how well the algorithm scales. When the model tried to take on the task of a real network with 28,702 people, we found that there was a memory error and that the library we had been using could not handle tensors of that size. So we altered the algorithm slightly. Like in the degree based approach, nodes with more connections are more likely to be spots where some of the budget should be allocated. So we created a smaller network from the original larger network that removed anyone with 3 or less connections. This allowed for a new network that is very similar to the original one but could be analyzed by a neural network. After the budget was allocated to the smaller network we just applied that budget to the larger network. The next section will show the results of this algorithm on this larger network and a comparative analysis of this algorithm compared to others.

Chapter 4: Results and Analysis

In this chapter we present the results of our neural network in comparison to other algorithms traditionally used for this problem. We compare the algorithms with regards to runtime and our own scoring method, the percent of the total thresholds activated of the graph. Normally the total number of nodes activated is used for how well an algorithm performed, however, the percent threshold activated is used here for several reasons. One reason is that a node's activation threshold is not absolute, it is impossible to know someone's exact threshold in real life. Similarly, it is impossible to know the exact influence that people hold over each other. Due to these factors, it is better to look at how much influence is spread through a network rather

than the specific number of nodes activated. In this chapter, we will present the results of our learning model and analyze the comparison between it and the traditional algorithms used to solve the influence maximization problem.

4.1 Comparison of Algorithms

4.1.1 Greedy Algorithm compared to other traditional methods

The greedy algorithm, when used on the graph of 500 people that we created, greatly outperformed the other methods. As shown in figure 4.1, random is not an effective algorithm for this problem. The series ‘Best Random’ is plotted by choosing random seed nodes 1000 times for each amount of products, and choosing the best result out of the 1000 trials. Even then, random still performs the worst, so it is clearly not a good algorithm to use. Sum of edges performs similarly to the best random algorithm, peaking at around activating 30% of the network’s threshold. Therefore, sum of edges is not a good algorithm to use for this problem. Compared to sum of edges and best random, greedy activates much more of the network’s threshold, and therefore will be used in our analysis of the learning algorithm.

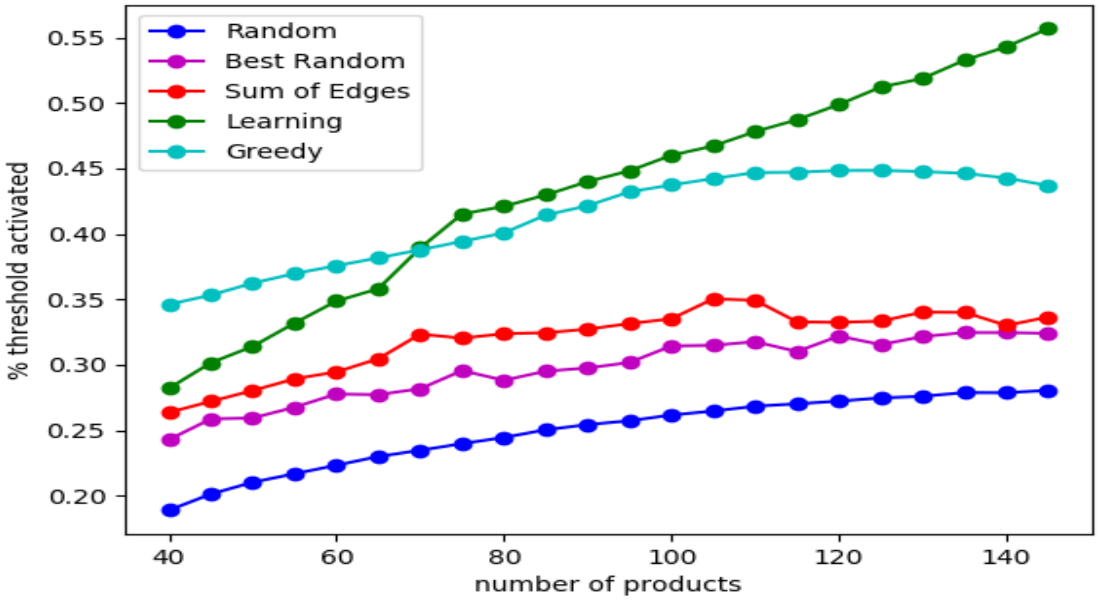

Figure 4.1: Percent Threshold Activation of Algorithms

4.1.2 Learning Algorithm compared to Greedy Algorithm

As greedy is the best performing algorithm of the traditional methods for this problem, we will be using it as a baseline to compare our algorithm. Shown in figure 4.1, the learning algorithm does worse with a low number of products, but after 75 products are given out initially, the learning algorithm jumps ahead. Interestingly, with greedy, after a certain point its percent threshold activated starts to decrease. However, with our algorithm, the percent threshold activated continues to increase at a near steady rate. Shown here, the learning algorithm performs better than greedy, as it activates a higher percent threshold on the 500 person graph, and as the number of products grows, the gap between the two algorithms grows as well.

Not only does the learning algorithm perform better in terms of percent threshold activated, but it also runs significantly faster than the greedy algorithm as shown in figure 4.2, where the x axis is the number of nodes in the graph, and the y axis is the runtime in seconds.

This data was all generated on graphs that we made with 10% of the network being activated initially. As seen, greedy has an exponential scale as the network increases in size, where the learning algorithm stays nearly flat for networks up to 1000 people in size. For this influence maximization problem, this result shows that the greedy algorithm is impractical to use for real network data. Real network data can have tens of thousands of people in it, and with the greedy algorithm exponentially scaling in time, it could take days or weeks to run on large networks. The learning algorithm is able to easily handle networks of 1000 people, so it will be scalable to larger networks. Shown in these graphs, the learning algorithm outperforms greedy, both in percent threshold activated of a network, and in the time it takes to do so.

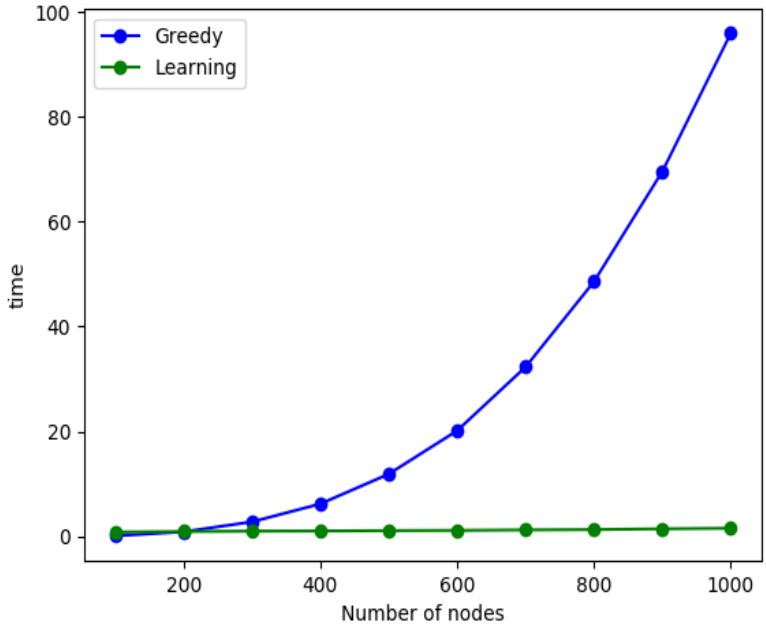

Figure 4.2: Runtime Comparison of Learning Algorithm and Greedy Algorithm 10% Initially Active

4.2: Case Study

To compare the learning algorithm and greedy algorithm, we first looked at a smaller dataset as a case study. The dataset used is a 1589 person coauthorship network of scientists working on network theory (Newman, 2006). Due to the runtime complexity of the greedy algorithm, it is unable to be used for very large datasets; however, it is able to be used for this dataset. As shown below in Table 4.1, the learning algorithm greatly outperforms greedy, sum of edges, and random algorithms. The learning algorithm activates over 25% of the network's threshold, but no other algorithm activates more than 4%. Additionally, the learning algorithm was much faster than greedy. Learning took 28.23 seconds to run, and greedy took 729.62 seconds, which is over 25 times as long. Shown here, the learning algorithm is able to greatly outperform greedy both in terms of percent threshold activated, and time it takes to do so.

Algorithm	Percent Threshold Activated
Learning	25.46%
Greedy	2.98%
Sum of Edges	2.87%
Best Random	3.78%

Table 4.1: Comparison of Algorithms on a Smaller Network

4.3: Real Data Results

To test our algorithm, we used a real example of a connected network. We used data from a DBLP dataset of 28702 authors, containing how many papers each author co-authored with one another. We turned these relationships into a network which was tested with our algorithm, and compared the results with the sum of edges and random algorithms. We used a budget of 5000. The greedy algorithm was not able to be used because of runtime constraints. Using the data

from figure 4.3, we calculated a trendline to estimate how long it would take to run on this network, and it would take days. Shown below in TABLE 4.2, the learning algorithm does much better than the other algorithms with the scoring method. Compared to our artificially made network of 500 people, the results stay relatively consistent when using the same percentage of the total network as the budget. On the artificial network, sum of edges activated about 32%, and on real world data it is at 26.42%. The best random on the artificial network activated about 30%, and performed slightly better with the real world data, activating 32.58% of the total threshold of the network. The learning algorithm on the artificial network activated about 45%, and performed better on the real world data, activating 48.18% of the network.

Shown here, the results of each algorithm on the real network data are very similar to the results on the artificial network, as talked about in section 4.1. Therefore, it is safe to assume that the learning algorithm would outperform the greedy algorithm on the real world data. Not only will the learning algorithm outperform greedy in terms of percent threshold activated, but it will greatly outperform it in terms of time. The learning algorithm took 278.99 seconds to run on a network of size 11,507 where greedy would have taken days to run. Clearly the learning algorithm is superior to the greedy algorithm and is much more scalable.

Algorithm	Percent Threshold Activated
Learning	48.18%
Sum of Edges	26.42%
Best Random	32.58%

Table 4.2: Comparison of Algorithms on a large scale network

Chapter 5: Conclusion

The goal of creating an algorithm that could compete with the greedy algorithm in the influence maximization problem was completed. Even though the new learning algorithm could not compare to the greedy algorithm with the original scoring method, the learning algorithm could consistently beat greedy in time efficiency and in the total amount of influence spread through the network.

5.1: Future Work

There is still work to be done in optimizing this approach to the influence maximization problem. Currently it does not score well with the traditional measurements, which could mean that a different, more advanced loss function could be found to increase its original score. There was also a major problem of over budgeting in denser networks. Figure 5.1 shows that the learning algorithm still eventually beat the greedy algorithm in a denser network. The graph doesn't show that in the last round with a budget of 145, that the learning algorithm over budgeted 24 nodes, wasting 35.24 of the initial influence. So there is still more work to be done in limiting over budgeting while still being able to get the same results.

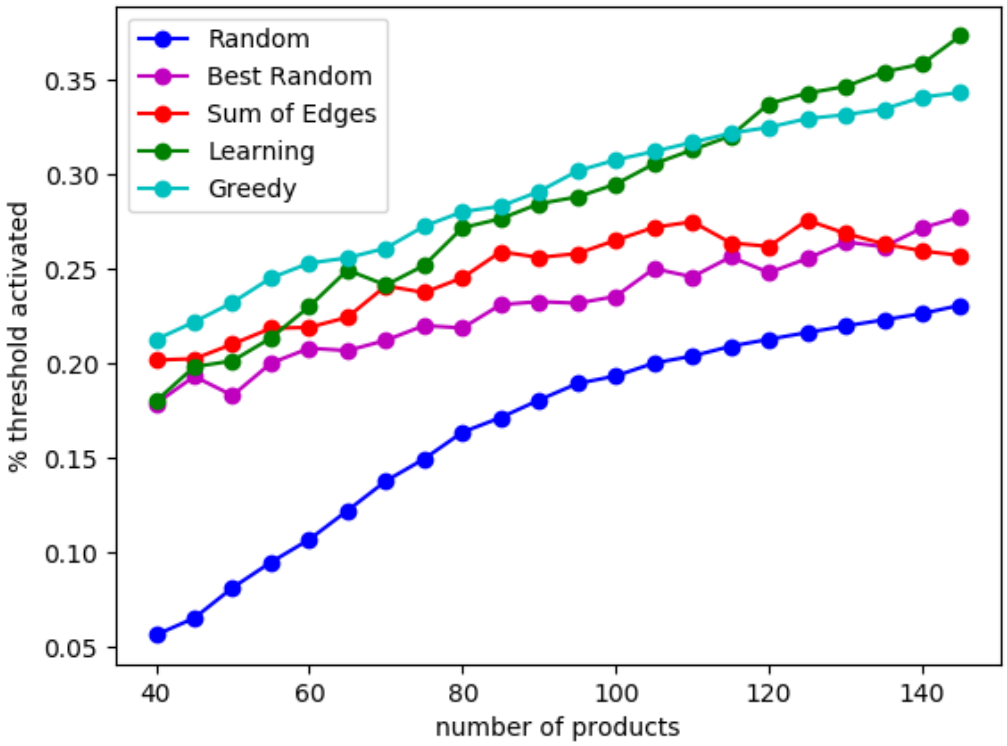

Figure 5.1: 500 person network with $m_0 = 6$ and $m = 2$

References

1. ujjwalkarn. An intuitive explanation of convolutional neural networks. The Data Science Blog Web site. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. Updated 2016. Accessed Oct 4, 2018.

2. Newman MEJ. Finding community structure in networks using the eigenvectors of matrices. *Phys Rev E*. 2006;74(3):036104. https://link.aps.org/doi/10.1103/PhysRevE.74.036104. Accessed Feb 28, 2019. doi: 10.1103/PhysRevE.74.036104.

3. Bakhshandeh R, Samadi M, Azimifar Z, Schaeffer J. Degrees of separation in social networks. *AAAI Publications*. 2011.

4. Albert R, Baraba´si A. Statistical mechanics of complex networks. *Reviews of modern Physics*. 2002;74:48-94.

5. Cormen TH, Leiserson CE, Rivest RL, Stein C. *Introduction to algorithms.* ; 2009.

6. Blockchain can make social networks more private — and profitable for you. TheNextWeb.com Web site. http://scholar.aci.info/view/1464f09cecc6cb40146/15e04ee49f4000115fc9cec. Updated 2017. Accessed Feb 24, 2019.

7. Google's AlphaGo gets 'divine' go ranking. *The Nation*. Mar 15, 2016. Available from: https://search.proquest.com/docview/1773600286.

8. Google Research. AlphaGo: Mastering the ancient game of go with machine learning. Google Research Blog Web site. http://scholar.aci.info/view/146b055aae209180128/152855fedd90012000a. Updated 2016. Accessed Feb 2, 2019.

9. Chen W, Wang Y, Yang S. Efficient influence maximization in social networks. *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining*. Jun 28, 2009:199-208. http://dl.acm.org/citation.cfm?id=1557047. doi: 10.1145/1557019.1557047.

10. Kempe D, Kleinberg J, Tardos É. Maximizing the spread of influence through a social network. *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining*. Aug 24, 2003:137-146. http://dl.acm.org/citation.cfm?id=956769. doi: 10.1145/956750.956769.

11. Viral marketing. In: Burton VL, III, ed. *Encyclopedia of small business.* Vol 2. 5th ed. ed. Farmington Hills, MI: Gale; 2017:1145-1147. http%3A%2F%2Flink.galegroup.com%2Fapps%2Fdoc%2FCX6062700612%2FGVRL%3Fu%3Dmlin_c_worpoly%26sid%3DGVRL%26xid%3D1b571e61. Accessed Jan 31, 2019.

12. Viral marketing. http://www.marketing-schools.org/types-of-marketing/viral-marketing.html.

13. Silver D, Huang A, Maddison CJ, et al. Mastering the game of go with deep neural networks and tree search. *Nature*. 2016;529(7587):484-489. https://www.nature.com/articles/nature16961. Accessed Jan 18, 2019. doi: 10.1038/nature16961.

14. Hardesty L. Explained: Neural networks. MIT News Web site. http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414. Updated 2017. Accessed Jan 16, 2019.