Evaluating Cascade Prediction via Different Embedding Techniques for Disease Mitigation

Abhinav Choudhury[1], Shubham Shakya[2], Shruti Kaushik[1], and Varun Dutt[1]

[1]Indian Institute of Technology Mandi
[2]National Institute of Technology Kurukshetra
{abhinav_choudhury, shruti_kaushik}@students.iitmandi.ac.in,
shubby98@gmail.com, varun@iitmandi.ac.in

Abstract. The diffusion of innovations (new medication or updated treatment methods) between physicians inside a social network over time (diffusion cascades) may help mitigate diseases in society. Most researchers have tried to predict the future diffusion cascade size by manually extracting graph features. However, less attention has been given to predicting the next adopter of innovation in a diffusion cascade. In this paper, we address this literature gap by predicting the next physician in a diffusion cascade via graph embeddings. For our analyses, different graph embedding techniques and embedding dimensions were fed into a number of machine learning (ML) algorithms to predict the next physician in the cascade. DeepWalk, Node2Vec, and Structural Deep network Embedding (SDNE) algorithms were evaluated as different graph embeddings; 16, 32, and 64 were assessed as different embedding dimensions; and, multilayer perceptron (MLP) and long short-term memory (LSTM) algorithms were developed for predicting the future cascades. Results indicated that SDNE with 32 dimensions gave the highest performance, followed by SDNE with 64 dimensions and SDNE with 16 dimensions. Overall, MLPs performed better compared to LSTMs. Through this research, we highlight the utility of high dimensional graph embedding algorithms and dimensions in diffusion cascade prediction in the healthcare domain. Such diffusion cascades may help rapidly spread medical innovations and guidelines among physicians.

Keywords: Cascade prediction, Social Network, Graph embedding, embedding dimension, machine learning.

1 Introduction

Understanding how the diffusion of innovation takes place inside a social network can help create strategies for better and faster dissemination of information (Choudhury, Kaushik, & Dutt, 2017). This understanding is highly relevant in the healthcare domain, where it is paramount that medical innovations and guidelines are disseminated appropriately and swiftly (Kasthuri, 2018). That is because the rapid dissemination of innovations may likely help provide appropriate treatment options and in-time treatments may probably help mitigate diseases in society. The adoption

of innovations (e.g., medications) inside a physician's social network may differ from physician to physician. While some physicians may adopt an innovation early, others may embrace it late or not at all. This innovation adoption by a physician is highly influenced by the interpersonal communication with the members of her peer network (Choudhury, Kaushik, & Dutt, 2018). Most diffusion studies conclude that active users (who adopted a medication) exert an influence on their inactive neighbors to perform an action (i.e., adopting the medication). The diffusion process that takes place inside a social network assumes that people tend to perform an action (adopting a medication) if they see their social contacts (friends, family, or acquaintances) performing that action in the network (Domingos, & Richardson, 2001). Overall, understanding the underlying dynamics of the social network may lead to better dissemination of medical information and guidelines.

Information cascades, which describe the flow of information inside a social network, play a pivotal role in the process of information diffusion. The modeling of information cascades has many applications, such as identifying opinion leaders, predicting growth of a cascade, detecting diffusion sources of a particular story, post, or a tweet (Bao, Cheung, & Liu, 2016; Cheng et al., 2014; Zhu, Chen, & Ying, 2017; Kempe, Kleinberg, & Tardos, 2003). If information cascades can be predicted, one can make informed decisions in the above scenarios. For example, by predicting the next user that will get infected, healthcare professionals may help stop the spread of diseases (Wang et al., 2016). Also, the prediction of a potential influence of a rumor may enable administrators to take safeguards against the spread of misinformation (Li et al., 2017).

Most prior works have investigated the prediction of diffusion cascades using generative models based on epidemiology (Cheng et al., 2014) or using a mixture of structural, content, and temporal features (Matsubara et al., 2012). Some recent works have also relied upon machine learning models (Bourigault, Lamprier, & Gallinari, 2016; Li et al., 2017; Qiu et al., 2018; Wang et al., 2017; Yang et al., 2019). In most social networks, information is usually diffused over the underlying social graph representing the users' social structure. For predicting cascade, researchers have assumed strong correlations between the spread of content and structural properties of the underlying social graph (Cheng et al., 2014). Therefore, most of the early attempts towards predicting cascades relied on manually extracted features from the underlying network structure and the cascade itself (Kefato et al., 2018). Unfortunately, manually extracting structural features from a social graph needs extensive domain knowledge; and, it is a time-consuming process that may not lead to good feature extraction. (Kefato et al., 2018)

In recent years, certain graph embedding techniques (Cai, Zheng, & Chang, 2018) have been developed that aim to represent a graph as a low dimensional (embedding's dimensions) vector, where the graph structures are preserved (i.e., embeddings of connected nodes are closer to each other). This representational learning method helps to obtain data representations that make it easier to extract useful structural

information from graphs (Cai, Zheng, & Chang, 2018). Prior researchers have used graph embeddings to assist in information diffusion prediction on graphs (Li et al., 2017, Bourigault, Lamprier, & Gallinari, 2016; Wang et al., 2017). For example, Bourigault, Lamprier, and Gallinari (2016) generated node embeddings of users based on the closeness of these embeddings and the predicted diffusion between users. Similarly, DeepCas (Li et al., 2017) generated embeddings of the diffusion subgraph using the Gated Recurrent Unit (GRU) (Cho et al., 2014) to predict the future cascade size of Twitter tweets. Topo-LSTM (Wang et al., 2017) used a recurrent neural network (RNN) (Rumelhart et al., 1985), which took dynamic directed acyclic graphs (DAGs) as input to generate embeddings for each node for the diffusion prediction. However, all the models used by Li et al. (2017), Bourigault, Lamprier, and Gallinari (2016), and Wang et al. (2017) have used variants of RNNs, which are memory-based networks, which may not be the appropriate choice when there is a scarcity of data (Feng, Zhou, & Dong, 2019).

Furthermore, most graph embedding techniques may share different design choices and hyperparameters. One important hyperparameter is the number of embedding dimensions (Patel & Bhattacharyya, 2017). The embedding dimensions are usually decided via a rule of thumb, mostly between 16 and 256 based on graph size, or by trial and error (Patel & Bhattacharyya, 2017). However, there has been a lack of studies done on the number of embedding dimensions that could be used during training different graph embeddings.

The primary objective of this research is to bridge the literature gaps highlighted above and evaluate different graph embedding techniques with different embedding dimensions via two machine learning (ML) models in the healthcare domain. Specifically, in this research, different ML models rely upon different embedding techniques and dimensions for predicting the next set of physicians that will be activated in an information cascade inside a physician's social network.

In what follows, we first provide a brief review of related work involving cascade prediction and graph embeddings. Then, we explain the data used in this research and describe the diffusion cascade prediction problem. In the next section, we explain the methodology of the three embedding techniques and implement machine learning models to predict the next set of users that become activated in an information cascade. Furthermore, we present our experimental results and discuss the implications of this research and its possible extensions.

2 Background

In social network analysis (SNA) (Scott, 1991), the structure of social interactions is viewed as a network composed of nodes (people) interconnected by edges (social relations, friendship, or advice). SNA is an ideal approach for describing interaction patterns to study how social influence is transmitted among physicians and how it

affects their contingent behavior (Choudhury, Kaushik, & Dutt, 2018). Social networks are important channels for the diffusion of innovations and social influence (Scott, 1991). Most diffusion processes can be viewed as a collection of interaction traces (e.g., prescription logs of physicians) indicating when people create a post, tweet a piece of information, or buy a particular product (Bourigault, Lamprier, & Gallinari, 2016). Most often, one only observes the when and where the actions had taken place rather than why it happened (Bourigault, Lamprier, & Gallinari, 2016). Therefore, modeling the diffusion process mostly boils down to approximating this mechanism based on the interaction traces. As such, the diffusion cascades of particular types of information like Tweets/microblogs, photos, videos, and academic papers have been empirically proven to be predictable to some extent (Li et al., 2017).

In literature, cascade prediction is formulated either as a classification problem, (popularity of a piece of information) or as a regression problem, (future size of a cascade) (Li et al., 2017). But prior work has not attempted to predict the next set of people to whom the information is going to be diffused. Moreover, prior research towards predicting information cascades have relied on hand-crafted features manually extracted from the underlying network structure and the cascade itself (Kefato et al., 2018). Manually extracting structural features from a social graph is a time-consuming exercise, and it may not lead to good feature extraction. Additionally, hand-crafted features are not able to fully represent both the local and the global structure of a graph and the complex interaction between local and global properties (Kefato et al., 2018). However, with the recent development of graph embedding techniques for representing graph structures, we can preserve the underlying structure of the social network and get a better representation of the local and global structures. Graph embedding represents a graph as low dimensional vectors where the graph structures are preserved. Researchers have used graph embeddings techniques like Node2Vec (Grover, & Leskovec, 2016) and DeepWalk (Perozzi, Al-Rfou, & Skiena, 2014) for predicting interests of users in social networks as well as link prediction tasks like predicting future relationships of users in social networks. Node2Vec and DeepWalk are deep-learning methods, where the embeddings are generated from random walks sampled from the graph. Wang, Cui, and Zhu (2016) relied upon the structural deep network embedding (SDNE), which uses autoencoders (AE) (Salakhutdinov & Hinton, 2009) to generate graph embeddings to perform multi-label classification of nodes. While there are multiple graph embedding techniques, not much research has been done on the effect of size of embedding dimensions' on these embeddings. The embedding dimensions are usually decided via a rule of thumb (Patel, & Bhattacharyya, 2017).

Graph embeddings have also been successfully implemented in cascade prediction tasks. For example, the Embedded-IC algorithm (Bourigault, Lamprier, & Gallinari, 2016) takes a cascade-based modeling approach, which assumes that an active node can activate an inactive node. It differentiates the active nodes as senders and inactive nodes as receivers. The receivers (inactive nodes) receive information from the sender (active nodes) in a diffusion cascade. The algorithm learns an embedding vector for

each role. Then, it models an activation taking place based on the closeness of the receivers' embedding vector and the senders' embedding vectors. Similarly, another approach named DeepCas (Li et al., 2017) is designed to predict the future cascade size. It creates a subgraph of the activated nodes based on the cascade at each time. Then, it decomposes the subgraph into certain random walk paths, and it uses a Gated Recurrent Unit (GRU) to learn an embedding vector of the subgraph. Based on this subgraph embedding vector, it predicts the cascade size in the future. Moreover, prior studies used only memory-based networks like LSTM or GRU to make cascade predictions. But memory-based networks may not perform well when there is a scarcity of data (Feng, Zhou, & Dong, 2019). Furthermore, to the best of author's knowledge, there is no research on performing cascade predictions of diffusion of medical information (e.g., new medications) inside a physician's social network using different embedding techniques. Furthermore, for cascade prediction tasks, memory-based networks (e.g., LSTMs or GRUs) have not been compared against memory-less networks (e.g., MLPs).

In this research, we try to overcome these limitations. For learning the representation of the underlying network structure, we evaluate three different graph embedding techniques: DeepWalk (Perozzi, Al-Rfou, & Skiena, 2014), Node2Vec (Grover, & Leskovec, 2016), and SDNE (Wang, Cui, & Zhu, 2016). We also vary the embedding dimensions between 16, 32, and 64 to investigate the effect of higher dimensions on the prediction task. Lastly, we generate embeddings that are feed into a memoryless Multi-layer Perceptron (MLP) model (Friedman, Hastie, & Tibshirani, 2009) and a memory-based long short-term memory (LSTM) model (Hochreiter, & Schmidhuber, 1997). Both these models predict the next set of physicians that will adopt a new medication.

3 Method

3.1 Dataset

We used a real-world physician dataset PhysicianSN (Choudhury, Kaushik, & Dutt, 2018) to predict the diffusion cascade of certain medications prescribed in a social network. The PhysicianSN is a physician social network dataset, where the physicians are the nodes, and edges represent the relationship between them. We used a medical-prescriptions dataset and the Healthcare Organization Services (HCOS) (IMS health, 2012) physician-affiliation dataset to extract physicians' attributes. The graph had 288 nodes and 38795 edges. The relationship between the physicians was created based on the similarity between physicians' attributes. We relied upon the prescription record of 45 medicines (prescribed between April 1996 and April 2017) as our cascade data, of which 35 (80%) were used for training, and 10 (20%) were used for testing. Then, we converted the diffusion cascade of each medicine in the training and test sets into smaller supervised sets (mini-batches) containing the chosen look-back period. After the supervised training set had been created, we used 10% of the training data as a validation set. The first prescription of the medicine by a physician was considered as the time of adoption. The average length of a cascade was 95 physicians, i.e., on an average, a new medication was diffused to 95 physicians in the social network.

3.2 Generating graph embeddings

Graph Embedding. Given the input graph $G = (V,E)$, and the embedding dimension d ($d \ll |V|$), the problem of graph embedding is to convert G into a d-dimensional space, in which the graph property is preserved as much as possible. In graph embedding, either the whole graph is represented as a d-dimensional vector or parts of the graph like nodes, and edges are represented as a set of d-dimensional vectors. Most graph embedding techniques try to preserve the following properties:
First-order proximity (Cai, Zheng, & Chang, 2018): The first-order proximity, $s_{i,j}^{(1)}$, between node v_i and node v_j (where, v_i and v_j are vertices of the graph G) is the weight of the edge e_{ij}, i.e., $A_{i,j}$ (where A is the adjacency matrix detailing the connection between nodes).
Second-order proximity (Cai, Zheng, & Chang, 2018): The second-order proximity $s_{i,j}^{(2)}$ between node v_i and v_j is the similarity between v_i's neighborhood and v_j's neighborhood.

We evaluated DeepWalk (Perozzi, Al-Rfou, & Skiena, 2014), Node2Vec (Grover, & Leskovec, 2016), and SDNE (Wang, Cui, & Zhu, 2016) algorithms to generate the d-dimensional node embeddings of each node in the PhysicianSN graph. Each of these techniques relies upon deep learning approaches to preserve the first-order proximity and the second-order proximity of the embeddings (Cai, Zheng, & Chang, 2018). We also varied the embedding dimension in the range 16, 32, and 64 in each of the graph embeddings. Once the embeddings were generated, we used the t-distributed stochastic neighbor embedding (t-SNE) to plot the embeddings in a 2-dimensional

space. The t-SNE algorithm calculates a similarity measure between pairs of instances in both the high and low dimensional space and then optimizes the two similarity measures using a cost function (Maaten, & Hinton, 2008).

DeepWalk. DeepWalk (Perozzi, Al-Rfou, & Skiena, 2014) is a graph embedding technique that adopts the SkipGram language model (McCormick, 2016) for embedding the nodes. SkipGram maximizes the probability of co-occurrence among words that appear within a window w. DeepWalk takes a graph G and samples uniformly a random vertex v_i as the root of the random walk. It then samples a set of paths from the node v_i of length l. Each path is analogous to a sentence, and each node is analogous to a word. We generate r walks per each node. Then, SkipGram is applied on the paths to maximize the probability of observing a node's neighborhood conditioned on its embedding. In this way, nodes with similar neighborhoods (i.e., having a high second-order proximity value) shared similar embeddings, thereby preserving the 2nd order proximity between the nodes.

Node2Vec. Node2Vec (Grover, & Leskovec, 2016) is similar to Deepwalk in that it also adopts the SkipGram model for graph embeddings. However, the random walk generation process is different. In Node2Vec, a random walk of fixed length l is simulated from a source node (v). Let n_i denote the i^{th} node in the walk, starting with $n_0 = v$. Nodes n_i are generated based on equation 1:

$$P(n_i = x | n_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{z}, & if\ (v,x) \in E \\ 0, & otherwise \end{cases} \quad (1)$$

where, π_{vx} is the unnormalized transition probability between nodes v and x, and Z is the normalizing constant.

In Node2Vec, $\pi_{vx} = \alpha_{p,q}(t,x).w_{v,x}$ where $w_{v,x}$ is the static edge weight and $\alpha_{p,q}(t,x)$ is a bias term defined as follows:

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p}, if\ d_{tx} = 0 \\ 1, if\ d_{tx} = 1 \\ \frac{1}{q}, if\ d_{tx} = 2 \end{cases} \quad (2)$$

where d_{tx} denotes the shortest path distance between nodes t and x (see Fig. 1). Fig. 1 illustrates how Node2Vec performs a random walk. The random walk starts from a source node and transitions to each of its neighbor node based on the transition probability $\pi_{vx} = \alpha_{p,q}(t,x).w_{v,x}$ where α is the search bias and $w_{v,x}$ is the static edge weight. Moreover, α is calculated based on the shortest path distance between nodes denoted by d_{tx}. As can be seen from Fig 1, currently, the walk has transitioned from t to v and it is evaluating its next node to select from node v. The edge labels indicate the search biases α and the distance between nodes d_{tx}. Since the walk has already transitioned from node t to reach v, the shortest distance from t to v (d_{tv}) is equal to zero. As per equation 2, if $d_{tv} = 0$, $\alpha_{p,q}(t,v) = \frac{1}{p}$, i.e., the walk has $\frac{1}{p}.w_{v,x}$ probability of transitioning back to t from v. Next, X_1 is a neighbor to both t and v, as such the shortest distance from t to X_1 (d_{tx_1}) is equal to 1, thereby $\alpha_{p,q}(t,x_1) = 1$, i.e., the walk has 1. $w_{v,x}$, probability of transitioning to X_1. Lastly, both X_2 and X_3 are neighbors of v but not of t. So, the shortest path

from t to X_2 and X_3 is through v, as such d_{tx_2}= 2, which according to equation 2 implies that $\alpha_{p,q}(t,x_2) = \frac{1}{q}$, i.e., the walk has $\frac{1}{q}.w_{v,x}$ probability of transitioning from v to X_2 or X_3.

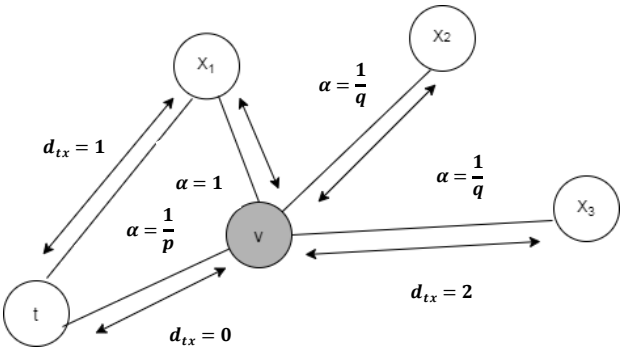


Fig. 1: Random walk in Node2Vec

The in-out parameter q determines how far or close does the walk traverse from the source node t. If q > 1, the random walk is more inclined towards nodes closer to t, whereas if $q < 1$, the walk is more prone towards node further away from t. The return-parameter p controls the likelihood of backtracking to a visited node. A high value of p $(> max(q,1))$ ensures that one is less inclined to backtrack to an already visited node in the following two steps. While a low value of p (< min (q, 1)), indicates that walk is inclined to backtrack a step, thus keeping the walk local. Node2Vec also preserves the 2nd order proximity between nodes.

Structured Deep Network Embedding (SDNE). SDNE (Wang, Cui, & Zhu, 2016) uses a semi-supervised model comprising of two autoencoders (Salakhutdinov, & Hinton, 2009) that preserves the first-order proximity and the second-order proximity of the node embeddings by minimizing the following objective function:

$$L_{sdne} = L_{2nd} + \alpha L_{1st} + \nu L_{reg} \qquad (3)$$

where L_{2nd} is the reconstruction loss of the autoencoder, L_{1st} is the 1st order proximity loss, and L_{reg} is an L2-norm regularizer term to prevent overfitting. The parameter α balances the weight of the first-order proximity and second-order proximity between vertices and ν is the regularization constant. The L_{2nd} is calculated as follows:

$$L_{2nd} = \sum_{i=1}^{n} \left|\left|(\hat{x_i} - x_i) \odot b_i\right|\right|_2^2 \qquad (4)$$

where x is the input and $\hat{x_i}$ is the reconstructed output, $\odot$ is the Hadamard operator, and $b_i = \{b_{i,j}\}_{j=1}^{n}$ is a penalty term. If $s_{i,j} = 0, b_{i,j} = 1$ $else$ $b_{i,j} = \beta$, β > 1,

where $s_{i,j}$ is an entry in the adjacency matrix S. An auto-encoder (AE) aims to minimize the reconstruction error of the output and input samples by using the encoder and decoder networks. The encoder maps input data to a representation space (embedding dimension), and the decoder maps the representation space (embedding dimension) to a reconstruction space. The reconstruction loss L_{2nd} helps preserve the second-order proximity. Due to the sparsity of graphs (e.g., social networks), the number of non-zero elements in the adjacency matrix S is far less than that of zero elements. Thus, if S is used as the input to the traditional AE, it is more prone to reconstruct the zero elements in S. To address this problem, a penalty term is imposed on the reconstruction error of the non-zero elements than that of zero elements.

The L_{1st} loss is calculated as follows:

$$L_{1st} = \sum_{i,j=1}^{n} s_{i,j} \left|\left|\left(y_i - y_j \right)\right|\right|_2^2 \quad (5)$$

where y_i and y_j are the embeddings of vertex v_i and v_j, respectively. The SDNE consists of two AEs (left and right networks), where each AE receives as input the node adjacency vectors s_i of all pairs of nodes connected by edges on the input graph. In an adjacency vector (a row from adjacency matrix, S), positive values indicate a connection between the node and the selected node. For each node, the adjacency vector is defined as follows:
$s_i = \{s_{i,j}\}_{j=1}^{n}, s_{i,j} > 0$ if and only if there exists a link between v_i and v_j.

Since vertices v_i and v_j are neighbors, their embeddings should be similar (i.e., first-order proximity). Thus, a distance loss is added to the objective function, which computed the distance between embeddings of v_i and v_j.
The L_{reg} loss is defined as follows:

$$L_{reg} = \frac{1}{2}\sum_{k=1}^{K}(\| W^{(k)} \|_F^2 + \| \widehat{W}^{(k)} \|_F^2) \quad (6)$$

where $W^{(k)}$ and $\widehat{W}^{(k)}$ are the k^{th} layer encoder and decoder weight matrices.
We used the backpropagation algorithm for training the AE and we minimized the L_{sdne} loss using stochastic gradient descent.

Model Calibration. Each embedding technique generated d-dimensional node embeddings for the 288 nodes social network. We generated node embedding of 3 different dimensions: 16, 32, and 64. This was done to understand the effect of embedding dimensions on the prediction result. The parameter settings used for Node2Vec and DeepWalk were similar. Specifically, we set window-size w = 10, walks per node r = 500, and maximum length l = 95 (same as the average cascade length). The p and q parameters in Node2Vec were set to 1 and 0.5, respectively, based on Grover and Leskovec (2016). In the case of SDNE, the encoder and decoder in the AE model consisted of two fully connected layers with 256 and 128 neurons, and d neurons in the latent layer, where d is the embedding dimension. The AE was trained for 1000 epochs with a batch size of 1. We ran each model 5 times and calculated the mean and standard deviation of the training and test evaluation metric for each of the models over these five runs.

3.3 Cascade prediction

We used both memoryless and memory-based models to predict the diffusion cascades inside the social network. In this paper, we used a Multilayer Perceptron model as the memoryless model and a Long Short-Term Memory model as the memory-based model. All models forecasted one-step ahead with walk-forward validation (Kaastra & Boyd, 1996). In one-step-ahead walk-forward validation, a model uses training data to predict the next time-step. This prediction is then evaluated against the actual value. Next, the actual value corresponding to the prediction is added to the training data, and the process is repeated by predicting the value for the next time-step.

Multi-layer Perceptron (MLP). An MLP is a variant of the original perceptron model (see Fig. 2(A)) proposed by Rosenblatt (Rosenblatt, 1961).

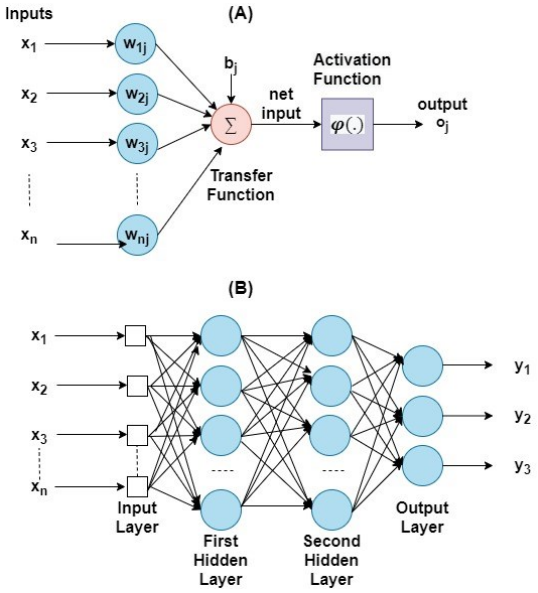


Fig. 2: (A) Rosenblatt's perceptron and (B) Architectural graph of an MLP with two hidden layers.

A neuron (represented as $\sum$ in Fig. 2A) computes a weighted sum of the inputs, followed by a non-linear activation φ of the calculated sum, as shown in Equation 3. Neural network architectures are considered as the universal function approximators because of the presence of activation functions. An activation function helps generate mappings from inputs to outputs, and it provides the neural network model the ability to learn complex data representations (Chung, Lee, & Park, 2016). There are a number of activation functions proposed in the literature, which include the sigmoid, tanh, and ReLU (Chung, Lee, & Park, 2016; Krizhevsky, Sutskever, & Hinton, 2012).

Krizhevsky, Sutskever, and Hinton (2012) have shown that the sigmoid and tanh activation functions suffer from the problem of vanishing gradient, while the ReLU activation function overcomes the vanishing gradient problem and provides faster convergence. Also, the function is computationally efficient to calculate. Thus, based on the literature, we used the ReLU activation function in the MLP model. The output o_i of a neuron in the MLP was defined as per the following equation:

$$o_i = \varphi\left(\sum_{j=1}^{d}(x_j w_{ij} + b_j)\right) \quad (6)$$

where d is the length of the input vector, x_i is a single instance of the input vector, and b_j and w_{ij} are the bias and weights associated with each x_j. Figure 2B shows the architecture of an MLP with two hidden layers. A typical MLP is composed of multiple hidden layers, with multiple neurons in each layer where every neuron in a layer (say i) is fully connected to every other neuron in the next layer (i.e., $i + 1$).

*Long Short-Term Memory (LSTM)***.** An LSTM (Hochreiter, & Schmidhuber, 1997) model is a recurrent neural network (RNN) model with the capacity of remembering the values from earlier stages in the network (i.e., the model possesses memory). The architecture of an LSTM consists of units called memory cells. Fig. 3 shows an LSTM memory cell containing self-connections and special multiplicative units called gates. These connections remember the temporal state of the memory cells, and the gates control the flow of information. Each memory cell contains an input gate, an output gate, and a forget gate. The flow of input activations is controlled by the input gate, while the flow of cell activations into the remaining network is controlled by the output gate. Furthermore, the internal state of the cell is scaled by the forget gate and is added back to the cell as input through a self-recurrent connection. In Figure 3, c_{t-1} is the previous cell state, h_{t-1} is the previous cell output, x_t represents the input to the memory cell, c_t represents the new cell state, and h_t represents the output of the hidden layer at time t.

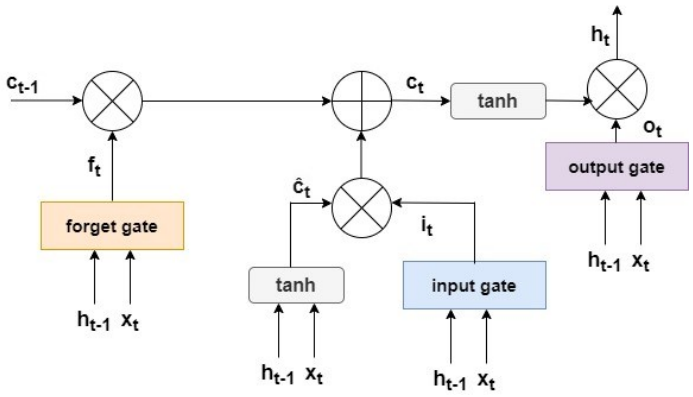


Fig. 3: An illustration of an LSTM memory cell. (Source: Kaushik et al., 2020)

Model Calibration. Both the MLP and LSTM were trained on the training set and validated on the validation set. The LSTM model comprised of a single LSTM layer followed by a dropout layer and a fully connected output layer with 288 nodes with the softmax activation function. We also used a 20% recurrent dropout in the LSTM layer. The MLP model consisted of 2 fully connected layers, followed by a fully connected output layer with 288 nodes with a softmax activation function. Both dropout and recurrent dropout were used only in the LSTM model, while no dropout was used in the MLP model. This arrangement of dropouts in LSTM and MLP models was made because the LSTM model was more likely to overfit due to a higher number of parameters (# parameter: 189744) compared to the MLP model (# parameter: 87216). All hidden layers in both models had ReLU activation. We used the cross-entropy loss as the loss function. The number of layers was kept small so as to reduce overfitting. We also used early stopping (Chollet, 2015); i.e., training will stop when the chosen performance measure stops improving). In this paper, we have used validation loss as the performance measure for early stopping, i.e., if the validation loss stops decreasing, then the model's training is stopped. Furthermore, we used a genetic algorithm (Whitley, 1994) with a 5% mutation and an 80% crossover rate to tune the following hyper-parameters using training data: number of neurons in a layer, batch size, epochs, and dropout rate. The hyper-parameters were varied in the following ranges: the number of neurons in a layer (16, 32, 64, 128, 256, and 512), batch size (16, 32, 64, 128, 256, 512, and 1024), and number of epochs (5 to 20) in increments of 1. For both the LSTM and MLP model, we set the look-back period to be 5 i.e. for predicting t we look at 5 ($t-1$,…, $t-5$) prior time-steps. After training, all models were tested on the test dataset. During training, we did not want to give an advantage to the memory-based LSTM model over the memory-less MLP model. Therefore, we varied the same range of hyper-parameters for both the MLP and LSTM models.

3.4 Evaluation Metrics

To evaluate our results, we have used mAP@k[1] (Wang, Cui, & Zhu, 2016) to compare our algorithms. The map@k computes the mean average precision at k for a set of predictions. So, if the algorithm predicts n nodes, only the k topmost nodes out of n predicted nodes are compared with the actual result. mAP@k is calculated as follows:

$$mAP@k = \frac{1}{|U|}\sum_{u=1}^{U}\frac{1}{m}\sum_{i=1}^{k}P(i).rel(i) \qquad (7)$$

where $|U|$ is the total number of cascades, m is the number of nodes in the cascade, P is the precision, $P(k)$ is the precision at k calculated by considering only the topmost k predictions from 1 to k, and $rel(k)$ is just an indicator that says whether that k_{th} item was relevant ($rel(k) = 1$) or not ($rel(k) = 0$).

$$precision\ (P) = \frac{\#\ correct\ positive}{\#\ predicted\ positive} \qquad (8)$$

For this research, we have selected the k value to be 10.

[1] The map@k score was calculated using the ml_metrics python library. Available at https://github.com/benhamner/Metrics

4 Results

4.1 Cascade Prediction using MLP

Fig. 4 shows the MLP model's training and test mAP@10 score for different graph embedding algorithms. As shown in Fig. 4, the best training map@10 (= 0.74) was obtained for SDNE-16 and SDNE-32 embeddings while the best test mAP@10 score (= 0.66) was obtained on the SDNE-32 embeddings (i.e., an embedding dimension of 32) by the MLP model. The MLP model comprised of 2 layers with 128 neurons in the 1st layer and 256 neurons in the second layer and was trained for 12 epochs with 128 batch size. We also found that Node2Vec gave the worst training and test mAP@10 scores among all the embeddings irrespective of the embedding dimensions.

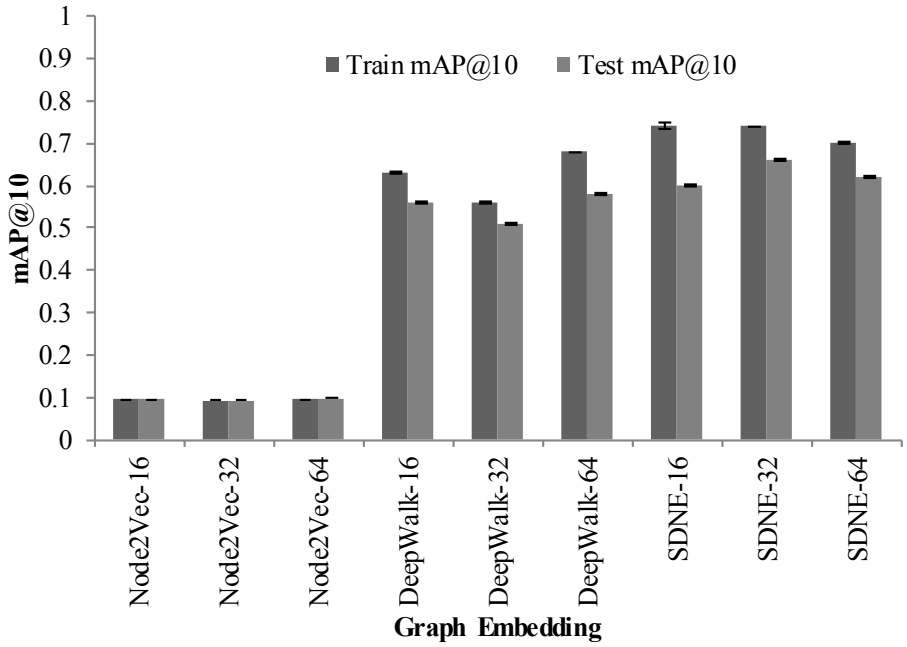


Fig. 4. The mAP@10 score of the MLP model on training and test data. The error bars indicate the 95% confidence interval around the average estimate.

4.2 Cascade Prediction using LSTM

Fig. 5 shows the LSTM model's training and test mAP@10 score for different graph embedding algorithms. As shown in Fig. 5, the best training mAP@10 score (= 0.63) was obtained on SDNE-16 embeddings, while the best test mAP@10 score (= 0.58) was obtained by the LSTM model for the SDNE-64 embeddings. The LSTM model

comprised of a 1-LSTM layer with 512-neurons and was trained for 16-epochs with a batch size of 64 iterations. Similar to the MLP model, the worst test mAP@10 was given by the Node2Vec embedding. Moreover, even the DeepWalk embedding did not give a high-test mAP@10 score on the LSTM model.

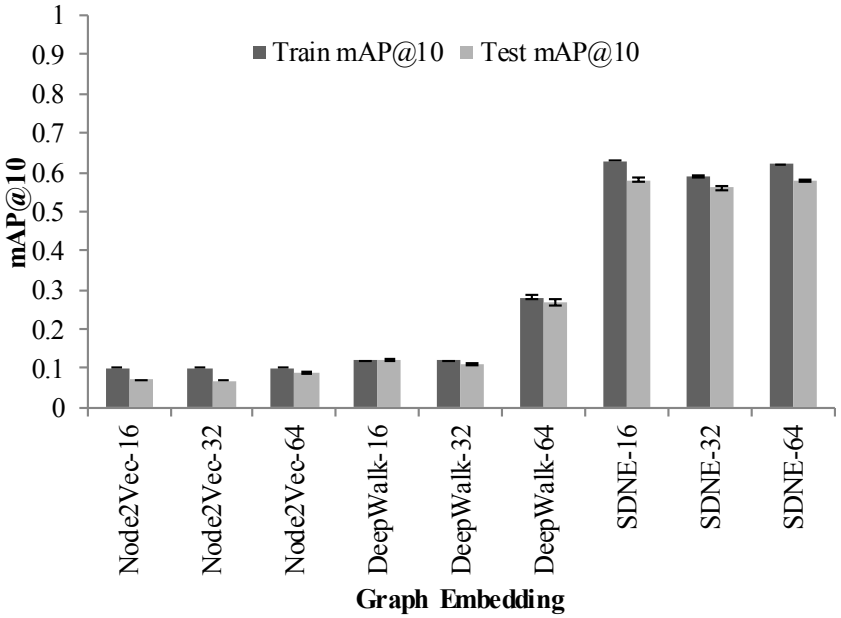


Fig. 5. The mAP@10 score of the LSTM model on training and test data. The error bars indicate the 95% confidence interval around the average estimate.

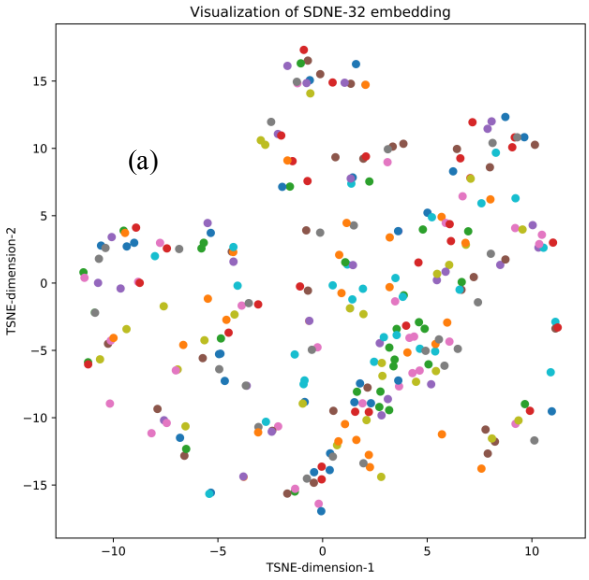
4.3 Model Comparison

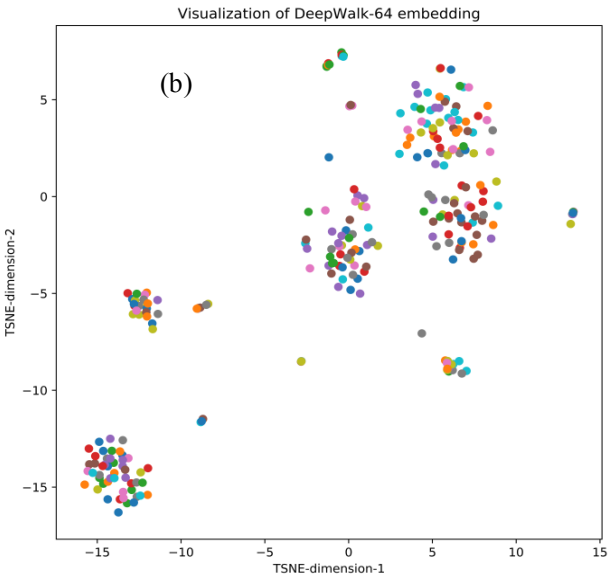
Table 1 shows the mean and the standard deviation of the training and test mAP@10 scores obtained over the 5 runs from both MLP and LSTM models. Overall the best mAP@10 score was given by the MLP model on the SDNE-32 embeddings.

Table 1. Model Comparison of different embedding algorithm and embedding dimensions.

Model	Embedding Algorithm	Embedding Dimension	Training map@10	Test map@10
MLP	SDNE	32	0.74± 0.019[2]	0.66 ± 0.02
LSTM	SDNE	64	0.62 ± 0.02	0.57 ± 0.05

[2] The standard deviation in the map@10 score


Visualization of SDNE-32 embedding
(a)
TSNE-dimension-2
TSNE-dimension-1
15
10
5
0
−5
−10
−15
−10
−5
0
5
10


Visualization of DeepWalk-64 embedding
(b)
TSNE-dimension-2
TSNE-dimension-1
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15

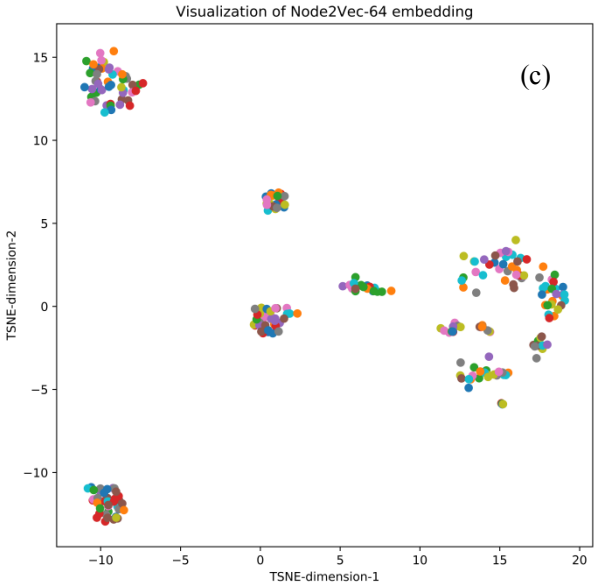


Fig. 6. Visualization of (a) SDNE-32 (b) DeepWalk-64, and (c) Node2Vec-64 embeddings using t-SNE

Fig. 6 shows the t-SNE visualization of the different embedding dimensions for each of the three embedding techniques for which the highest test mAP@10 was obtained. t-SNE visualizes the high dimensional node embeddings (32-dimension SDNE embedding, 64-dimension Deepwalk embedding, and 64-dimension Node2Vec embedding) in a 2-dimensional space. Each node in the social network is represented with a different color and similar nodes are clustered together. The denser the node clusters, more similar are the embeddings of the nodes in that cluster. As can been seen from the t-SNE visualization of DeepWalk (Fig 6(b)) and Node2Vec (Fig 6(c)) embeddings, nodes are segregated into dense clusters while in SDNE embeddings there are no such clusters. These differences in clustering could explain the poor performances of DeepWalk and Node2Vec algorithms compared to the SDNE algorithm.

5 Discussion and Conclusions

Information cascades play a pivotal role in the process of information diffusion, as it describes the flow of information inside a social network. Most diffusion processes can be viewed as a collection of interaction traces (e.g., prescription logs of physicians) indicating when people post/tweet a piece of information or buy a

particular product. Most often, one only observes the when and where the actions had taken place rather than why it happened (Bourigault, Lamprier, & Gallinari, 2016). Therefore, modeling the diffusion process mostly boils down to approximating this mechanism based on the interaction traces. Prior research had modeled the diffusion process as to how a piece of information like a tweet or meme diffused inside an online social network (e.g., Twitter or Facebook) using the action of liking or retweeting of tweets/memes/posts as an indicator of diffusion (Bourigault, Lamprier, & Gallinari, 2016; Li et al., 2017; Wang et al., 2017). Bourigault, Lamprier, and Gallinari, (2016) and Wang et al., (2017) generated graph embedding of the underlying social network and the cascade itself to predict future diffusion. However, no such analysis was done in the field of healthcare, where it is of utmost importance that accurate information is swiftly disseminated. In this research, we investigate how the diffusion of medication takes place inside a physician's social network. We performed a comprehensive evaluation of different graph embedding techniques as well as studied the effect of different embedding dimensions on future cascade prediction using both memory and memoryless networks in the healthcare domain. We tried SDNE, DeepWalk, and Node2Vec algorithms to extract low dimensional embeddings of the underlying network and used prescription logs of 45 medications as our cascade data. We then fed the embeddings into MLP and LSTM models to predict the next set of physicians who will adopt the medication in a cascade.

Firstly, we found that both MLP and LSTM gave a high-test mAP@10 score on the SDNE embeddings. This result was as per our expectations as the SDNE method preserves both the first-order proximity and the second-order proximity of the graph structure (Wang, Cui, & Zhu, 2016). We also found that Node2Vec gave the worst test mAP@10 score, followed by DeepWalk. One likely reason for this finding could be due to the presence of dense clusters in the Node2Vec and DeepWalk representation of the underlying social network. Clusters are obstacles to diffusion cascades as people are hesitant to adopt new information if they belong to a cluster (Easley, & Kleinberg, 2012). Furthermore, we also found that generally higher dimensional embeddings performed better than its lower-dimensional counterparts, specifically in the case of Node2Vec and DeepWalk, while for SDNE, the highest test mAP@10 was obtained for embedding dimension of 32 followed by 64 and 16. Although not much research has been done into the evaluation of different embedding dimensions in graphs; however, a rule of thumb from word embedding literature is that the embedding dimension may be the 4^{th} root of the size of the vocabulary (i.e., the number of the distinct word in the corpus) (Google Developers, 2017). Our results disagree with this rule as we found that higher-dimensional embeddings performed better in general in predicting diffusion inside social networks.

Next, we found that the MLP model performed better than the LSTM model on predicting the next set of users that will be activated in an information cascade inside a network (Table 1). This result is not as per our expectation as we expected memory-based models like LSTM to perform better compared to memory-less models. A likely reason for this finding could be that mostly memory based networks like LSTM

and GRU need a huge amount of data for training. Also, even though we used dropout, recurrent dropout, and early stopping, LSTMs were still overfitting and were unable to generalize on the test data due to having a significantly larger number of parameters compared to MLP.

Understanding how diffusion takes place inside a physician's social network is paramount as it may likely help in better dissemination of medical information and guidelines as well as mitigate diseases (Kwan et al., 2020). However, the availability of cascade data like prescription logs is most scarce, and the available data is mostly small. In this paper, we have tried to tackle this issue by evaluating three different graph embedding techniques (DeepWalk, Node2Vec, and SDNE) with three different embedding dimensions (16, 32, and 64) using two machine learning models (MLP and LSTM) for predicting the next set of users that will be activated in an information cascade inside a physician social network. As per our findings, we believe that by preserving both first- and second- order proximity of the underlying social network, we were able to perform better in cascade prediction tasks specifically in predicting the next set of users in the cascade. We also found that for smaller datasets, MLPs performed better than LSTMs. As information regarding the social structure of physicians as well as diffusion information relating to healthcare innovation (specifically a medication) is likely scarce, one may generate high dimensional embeddings of the social network and use MLPs to make future predictions.

One particular limitation of our work is that we only generated the embeddings of the underlying structure and not the cascade itself. This process adapted well in prediction as the underlying link structure was representative of the diffusion channels; however, in the case of online social networks, the diffusion may not always propagate through links between users. In the future, we plan to generate embeddings of both the underlying social network as well as the cascade itself to predict the next set of users as well as the size of future cascades.

Acknowledgment

The project was supported by a grant (award: # IITM/CONS/RxDSI/VD/33) to Dr. Varun Dutt.

References

Bao, Q., Cheung, W. K., & Liu, J. (2016). Inferring motif-based diffusion models for social networks. In G. Brewka (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI 2016)* (pp. 3677–3683). AAAI press.

Bourigault, S., Lamprier, S., & Gallinari, P. (2016). Representation Learning for Information Diffusion through Social Networks: an Embedded Cascade Model. *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining* (pp. 573-582). San Francisco: Association for Computing Machinery.

Cai, H., Zheng, V. W., & Chen-Chua, K. (2018). A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications. *IEEE Transactions on Knowledge and Data Engineering, 30*(9), 1616-1637.

Cheng, J., Dow, P. A., Kleinberg, J. M., & Leskovec, J. (2014). Can cascades be predicted? *Proceedings of the 23rd international conference on World wide web* (pp. 925–936). Seoul: Association for Computing Machinery.

Cho, K., Merriënboer, B. v., Bahdanau, D., & Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation* (pp. 103–111). Doha, Qatar: Association for Computational Linguistics.

Chollet, F. (2015). *Deep learning library for theano and tensorflow.* Retrieved March 12, 2020, from Keras.io: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

Choudhury, A., Kaushik, S., & Dutt, V. (2017). Social-Network Analysis for Pain Medications: Influential physicians may not be high-volume prescribers. *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (pp. 881-885). Sydney: IEEE.

Choudhury, A., Kaushik, S., & Dutt, V. (2018). Social-network analysis in healthcare: analysing the effect of weighted influence in physician networks. (R. Alhajji, & U. K. Wiil, Eds.) *Network Modeling Analysis in Health Informatics and Bioinformatics, 7*(17).

Chung, H., Lee, S. J., & Park , J. G. (2016). Deep Neural Network Using Trainable Activation. *International Joint Conference on Neural Networks (IJCNN)* (pp. 348-353). IEEE.

Domingos, P., & Richardson, M. (2001). Mining the network value of customers. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 57-66). San Francisco: Association for Computing Machinery.

Easley, D., & Klienberg, J. (2010). *Networks, crowds, and markets.* New york: Cambridge University Press.

Feng, S., Huiyu, Z., & Donga, H. (2019). Using deep neural network with small dataset to predict material defects. *Materials & Design, 162*, 300-310.

Google Developers. (2017, November 20). *Introducing TensorFlow Feature Columns.* Retrieved April 4, 2020, from https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html,

Grover, A., & Leskovec, J. (2016). node2vec: Scalable Feature Learning for Networks. *Proceedings of the 22nd ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining (pp. 855-864). San Francisco: Association for Computing Machinery.

Hastie, T., Tibshirani, R., & Friedm, J. H. (2009). *The Elements of Statistical Learning*. New York: Springer.

Hinton, G. E., Sutskever, I., & Krizhevsky, A. (2012). ImageNet classification with deep convolutional neural networks. *Advances in neural information processing*, (pp. 1097-1105). Nevada.

Hochreiter, S., & Schmidhuber, J. (1997, November). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780.

IMS Health. (2012). *Healthcare Organization Services: Professional and Organization Affiliations Maintenance Process.*

Kaastra, I., & Boyd, M. (1996). Designing a neural network for forecasting financial and economic time series. *Neurocomputing, 10*(3), 215-236.

Kasthuri, A. (2018). Challenges to Healthcare in India - The Five A's. *Indian Journal of Community Medicine, 4*(3), 141-143.

Kaushik, S., Choudhury, A., Sheron, P. K., Dasgupta, N., Natarajan, S., Pickett, L. A., et al. (2020, March 19). AI in Healthcare: Time-Series Forecasting Using Statistical, Neural, and Ensemble Architectures. *Frontiers in Big Data, 3*(4).

Kefato, Z. T., Sheikh, N., Bahri, L., Soliman, A., Montresor, A., & Girdzijauskas, S. (2018). CAS2VEC: Network-Agnostic Cascade Prediction in Online Social Networks. *Fifth International Conference on Social Networks Analysis, Management and Security* (pp. 72-79). IEEE.

Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 137-146). Washington: Association for Computing Machinery.

Kwan, T. H., Wong, N. S., Lui, G. C., Chan, K. C., Tsang, O. T., Leung, W. S., et al. (2020, January). Incorporation of information diffusion model for enhancing analyses in HIV molecular surveillance. *Emerging Microbes & Infections, 9*(1), 256-262.

Li, C., Ma, J., Guo, X., & Mei, Q. (2107). DeepCas: An End-to-end Predictor of Information Cascades. *Proceedings of the 26th International Conference on World Wide Web* (pp. 577-586). Perth: International World Wide Web Conferences Steering Committee.

Maaten, L. v., & Hinton, G. (2008., November). Visualizing data using t-SNE. *Journal of Machine Learning Research, 9*, 2579--2605.

Matsubara, Y., Sakurai, Y., Prakash, B. A., Li, L., & Faloutsos, C. (2012). Rise and fall patterns of information diffusion: model and implications. *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 6-14). Beijing: Association for Computing Machinery.

McCormick, C. (2016, April 19). Retrieved April 12, 2020, from Word2Vec Tutorial - The Skip-Gram Model.: http://www.mccormickml.com

Patel, K., & Bhattacharyya, P. (2017). Towards lower bounds on number of dimensions for word embeddings. *Proceedings of the Eighth International*

Joint Conference on Natural Language Processing (Volume 2: Short Papers) (pp. 31-36). Taipei: Asian Federation of Natural Language Processing.

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). Deepwalk: Online learning of social representations. *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 701-710). New york: Association for Computing Machinery.

Qiu, J., Tang, J., Ma, H., Dong, Y., Wang, K., & Tang, J. (2018). Deepinf: Social influence prediction with deep learning. *The 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 2110–2119). London: Association for Computing Machinery.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* New york: Spartan Books.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, & J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations* (pp. 318-362). Cambridge, MA: MIT press.

Salakhutdinov, R., & Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning, 50*(7), 969-978.

Scott, J. (1991). *Social network analysis: A handbook*. SAGE Publications.

Wang, D., Cui, P., & Zhu, W. (2016). Structural deep network embedding. *The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1225-1234). San Francisco: Association for Computing Machinery.

Wang, J., Zheng, V. W., Liu, Z., & Chen-Chang, K. (2017). Topological recurrent neural network for diffusion prediction. *The 2017 IEEE International Conference on Data Mining* (pp. 475-484). IEEE.

Whitley , D. (1994). A genetic algorithm tutorial. *Statistics and Computing, 4*, 65-85.

Yang, C., Sun, M., Liu, H., Han, S., Liu, Z., & Luan, H. (2015). Neural Diffusion Model for Microscopic Cascade Study. *IEEE Transactions on Knowledge and Data Engineering., 14*(8).

Zhu, K., Chen, Z., & Ying, L. (2017). Catch'em all: Locating multiple diffusion sources in networks with partial observations. *The Thirty-First AAAI Conference on Artificial Intelligence* (pp. 1676–1682). San Francisco: AAAI Press.