Influence Propagation for Linear Threshold Model with Graph Neural Networks

Francisco Santos, Anna Stephens, Pang-Ning Tan, and Abdol-Hossein Esfahanian
Dept. of Computer Science and Engineering
Michigan State University
East Lansing, United States
{santosf3, steph496, ptan, esfahanian}@msu.edu

***Abstract*—Influence propagation is a network phenomenon governing how information is diffused in a network. With the advent of deep learning, there has been growing interest in applying graph neural networks to extract salient feature representation of the nodes for a variety of network mining tasks, such as forecasting the virality of information cascade. Given the importance of social influence, this paper presents a novel deep learning framework called IP-GNN for simulating the information propagation process in a complex network and learning a node representation that embeds information about the diffusion process under the linear threshold model. Our framework employs a modified graph convolutional network architecture with adaptive diffusion kernel to capture long-range propagation of information along with an entropy-regularized mixture of loss functions to ensure accurate prediction and faster convergence of the learning algorithm. Experimental results on 4 real-world datasets show that the model accurately mimics the output of the linear threshold model, achieving an average accuracy that exceeds 90% on all datasets.**

***Index Terms*—Influence propagation, Deep Learning, Graph Neural Network**

I. Introduction

The study of influence propagation, also known as diffusion of innovation, in complex networks has attracted significant interest over the last two decades, motivated by its applicability to diverse disciplines, from viral marketing [1] [2] to understanding fake news propagation [3] [4] and epidemic spread [5] [6]. Previous studies have focused on myriad issues, such as proposing a diffusion model that characterizes the influence propagation process [7]–[9], identifying the k-most prominent seed nodes that would maximize the information spread [10], [11], and forecasting the virality of information cascade such as fake news, memes, and disease pandemic [12], [13].

In the meantime, graph neural networks [14] have also emerged as a popular machine learning approach for modeling social networks as well as the spatio-temporal graphs associated with the traffic [15] and weather [16] forecasting tasks. These approaches would employ a deep neural network architecture to learn the representation of different entities (nodes) of the domain and their relationships, providing an informative set of features for subsequent learning tasks. Given the prominent role influential nodes play in a network, this begs the question: ***Is it possible to incorporate the influence propagation process explicitly into the deep learning formulation?*** In doing so, this would help enrich the features learned for the downstream tasks given the domain-guided knowledge about the information diffusion process.

While there have been several previous works on integrating node influence into representation learning, e.g., via the random walk approach [17], they generally focused on *homophily*-driven diffusion, which is the tendency of similar individuals to form ties with one another, instead of social influence-based contagion [18]. In particular, two social actors who are directly linked may not necessarily influence the behavior of each other. As the latter requires understanding of the information propagation process, *integrating the diffusion model explicitly into the deep learning formulation* is thus an important research problem. Our paper aims to fill this gap by presenting a novel deep neural network framework capable of robustly sumulating the information diffusion process.

Specifically, our influence propagation network is guided by a user-specified diffusion model, which governs how the influence will spread and which users will be affected. Though there are numerous diffusion models available [8], [10], this workshop paper presents our preliminary results using the linear threshold model [19], which is a discrete-time model that considers the use of thresholds to determine whether a user will likely be influenced by the collective behavior of its neighbors. Figure 1 illustrates an example of how the linear threshold model works. Assume each node can be in one of two states—activated (green) or deactivated (white). Initially, assume there is only node (A) that is activated. In every round, each node will sum up the edge weights of its activated neighbors. If the sum of the edge weights exceeds its threshold, then the node will be activated. In this case, node A will activate node B since the edge weight is equal to the threshold for B. However, it will not be able to activate node D, whose threshold (0.9) is more than the edge weight (0.5). In turn, node B will then influence node C as the edge weight is greater than the threshold. This example distinguishes social influence from homophily, in which not all neighbors are equally influenced.

Designing a deep neural network framework to simulate the linear threshold model has its challenges. First, one has to reconcile the discrete-valued input/output of the linear threshold model with the continuous nature of deep neural network formulation. Second, the neural architecture must be able to simulate multiple steps of information propagation,

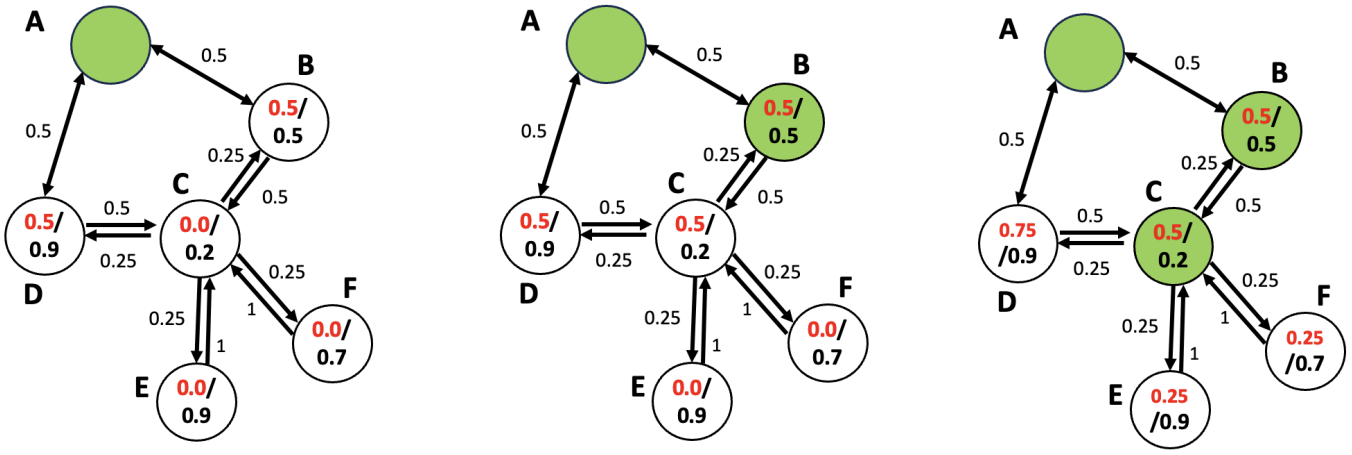

Fig. 1: **Linear Threshold Model Example.** The green nodes represent that they are activated, otherwise they are deactivated. The black number in the center of the node represents the threshold. The red number represents the sum of edges of the activated neighbors. The number next to edges represents the weight of the edge.

taking into account the threshold associated with each user. A trivial solution would be to use conventional Graph Convolutional Network (GCN) architecture [14], which leverages the adjacency matrix to perform message passing between neighboring nodes. Unfortunately, this approach is susceptible to over-smoothing problem, where the model outputs start to saturate when extended beyond two or three graph convolutional layers. This leads to significant drop in its model performance. Another challenge is training the model to achieve good performance and fast convergence. Note that the influenced nodes depend on the choice of initial seed, i.e., the set of initially activated nodes. A different initial seed will lead to different set of activated nodes, even though the graph topology and node thresholds remain unchanged. Given the exponential number of possible initial seed configurations, designing an efficient yet effective training procedure on mini-batches of seed samples is a challenge. The deep neural network must generalize well even to initial seed configurations not seen during training.

To overcome such challenges, we propose a novel framework called **IP-GNN** (Influence Propagation with Graph Neural Networks) which aims at learning the influence propagation process using a deep neural network. Our framework uses a combination of a modified GCN with an adaptive diffusion kernel and a stack of fully connected layers to simulate the influence propagation process. The adaptive diffusion kernel enables us to capture the information propagation for multiple time steps using learnable kernel weight parameters. Our framework also employs degree-based sampling for initial seed selection and an entropy-regularized loss function to train the network for higher accuracy and faster convergence.

In summary, the main contributions of our paper are as follows:

- We develop an approach for simulating the influence propagation process using deep neural networks. Our framework employs a modified GCN with adaptive diffusion kernel and entropy-regularized loss to improve its performance and help the model converges faster.
- We design a training procedure that performs degree-based sampling to mitigate the exponential search space of initial seed configuration.
- We performed experiments on numerous real-world datasets to demonstrate the efficacy of our approach.

II. Related Works

A. *Influence propagation*

The linear threshold model is one of the most used influence propagation models. It is based on the idea of each node having a threshold, which represents the difficulty of influencing the node. The idea of node threshold first came from Granovetter [19] and Schelling [20]. Others such as Pautasso et al. [21] have looked into generating more realistic simulations for real-world applications such as disease outbreaks using the linear threshold model. The authors in [21] use the probability of infection transmission between connected nodes and the probability of infection persistence in an infected node to calculate the node influence threshold.

Independent cascade is another prevalent diffusion model, in which each activated node has certain probability to influence its immediate neighbor. If it fails, then it will not make any further attempts to influence the neighbor. Kempe et al. [10] showed that the generalized versions of the independent cascade model and linear threshold model are equivalent. Note that both independent cascade and linear threshold are considered user-to-user influence models.

Barbieri et al. [9] presents a diffusion model that focuses more on user authoritativeness and interest in a topic instead of a user-to-user influence model. Their diffusion model is more designed towards viral marketing. Aral et al. [22] argues that having a single threshold is insufficient. Their model considers

an influence parameter and a susceptible parameter, which measure how influential the user is and how easy the user is to convince. The two calculated metrics are then combined to create an edge weight, which is used for influence propagation.

B. Graph Neural Networks

Graph neural networks are typically used to learn the feature representation for various network mining tasks including node classification [14], [23], [24], link prediction [25]–[27], and graph clustering [28] [29]. For node classification, some popular methods include GCN [14], Graph Attention Network [23], and GraphSage [24]. GCN employs a message-passing mechanism to aggregate the embedding of its neighbors in order to learn the representation of a node. The approach requires stacking l graph convolutional layers to perform feature aggregation from neighbors located l-hop away. Graph Attention Network [23] uses multi-head attention layers to give higher importance to certain neighbors of a node. GraphSage [24] uses a sampling approach to improve efficiency of the node representation learning task.

III. Preliminaries

A. Problem Statement

Consider a graph $\mathcal{G} = (V, E, X)$ where V is a the set of nodes, $E \subseteq V \times V$ is the set of edges, and $X \in \mathbb{R}^{|V| \times d}$ is the node features. Let A be the adjacency matrix representation of E where $A(v_i, v_j) = 1$ if $(v_i, v_j) \in E$, otherwise $A(v_i, v_j) = 0$. Assume the weight for each edge is given by $W = \hat{D}^{-1}A$. Let s_v^t be the binary state of node v at time step t, where $s_v^t = 1$ if v is activated and 0 otherwise. Here, the definition of a node in an activated state is domain-dependent. For example, it may refer to the adoption of an idea, decision to share or retweet a social media post, or an individual in an infected disease state. The initial state of each node in the graph is denoted as $\boldsymbol{s^0} \in \mathbb{R}^{|V|}$. The number of activated nodes at time t can be computed as $K_t = \sum_{v \in V} s_v^t$. For instance, in influence maximization problems [10], K_0 is assumed to be given by users and the objective is to find an initial set of activated nodes, s^0, that will maximize K_t as $t \to \infty$.

The activation of the nodes depends on a threshold vector $\boldsymbol{\tau} \in \mathbb{R}^{|V|}$. Given the current state of the nodes, s^t, and their respective thresholds, $\boldsymbol{\tau}$, a diffusion model is a function of the form $g : \{0,1\}^{|V|} \times \mathbb{R}^{|V|} \to \{0,1\}^{|V|}$ with the output being the next state of the node, i.e., $\mathbf{s}^{t+1} \equiv g(\mathbf{s}^t, \tau)$. For brevity, we denote the T-step information propagation process as

$$s^T \equiv g^T(\mathbf{s}^0, \boldsymbol{\tau}) = g\left[g\left(\cdots g\left(\mathbf{s}^0, \boldsymbol{\tau}\right)\cdots\right), \boldsymbol{\tau}\right]$$

Definition 1 (Influence Propagation): Given a graph $\mathcal{G} = (V, E, X)$, a threshold vector $\boldsymbol{\tau}$, an initial state vector, $\mathbf{s}^0 \in \{0,1\}^{|V|}$, and a diffusion model g, compute the final state of the nodes after T time steps, i.e., $s^T \equiv g^T(s^0, \boldsymbol{\tau})$.

The goal of this paper is to train a deep neural network $f_\theta(\cdot)$ that accurately simulates the output of the diffusion model after T time steps given the initial seed, $\mathbf{s}^0$.

B. Background

1) Linear Threshold Model: In this model, at every time step, each node will sum up the edge weights of its activated neighbors to determine whether it should be activated. If the sum is greater than or equal to the threshold, then the node becomes activated, as shown by the equation below:

$$s_i^{t+1} = \begin{cases} 1 & \text{if } \sum_j W_{ij} s_j^t \geq \tau_j \\ 0 & \text{otherwise} \end{cases}$$

where W is the edge weight matrix. Furthermore, a node that has been activated (e.g., an initial seed node) remains activated even if none of its neighbors are activated. One way to enforce this constraint is by setting its threshold $\tau_v = 0$ if $s_v^0 = 1$.

The linear threshold model can be implemented as the following function to predict the next state of the nodes:

$$\mathbf{s}_i^{t+1} = \begin{cases} 1 & \text{if } s_i^{t-1} = 1 \\ \texttt{Step}(\sum_j W_{ij} \boldsymbol{s}_j^t - \boldsymbol{\tau}_i) & \text{otherwise} \end{cases} \quad (1)$$

where $\texttt{Step}(\cdot)$ is an element-wise Heaviside step function

$$\texttt{Step}(x_i) = \begin{cases} 1 & \text{if } x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

2) Graph Convolutional Network (GCN): This is a graph neural network that uses a message-passing mechanism to learn the feature embedding for each node. The message-passing takes the current embedding of each neighbor and aggregates them to create the new node embedding. The formulation of a GCN can be formally written as follows:

$$H^{(l+1)} = \sigma(\tilde{A} H^l \theta^{(l)}) \quad (2)$$

where $H^{(l+1)}$ is the node embedding for $(l+1)$ layer, σ is a nonlinear activation function (e.g., ReLU), θ is the model parameters and $\tilde{A}$ is the normalized adjacency matrix, which is defined as follows:

$$\hat{A} = A + I \quad \text{and} \quad \tilde{A} = \tilde{D}^{-\frac{1}{2}} \hat{A} \tilde{D}^{-\frac{1}{2}} \quad (3)$$

where I is the identity matrix and $\tilde{D}$ is a diagonal matrix containing the degree of each node.

3) Random Walk.: A random walk traversal on a graph starts from a source node u_i and randomly chooses an immediate neighbor of node u_i to visit next. This process of randomly selecting a node to visit next is repeated until the length of the traversal is equal to d, a user-specified parameter. Formally, the probability of visiting a node u is given by the following equation:

$$P(u_i = u | u_{i-1} = v) = \begin{cases} \pi_{vu}, & \text{if } (i,j) \in E \\ 0, & \text{Otherwise} \end{cases}$$

where $\pi_{i,j}$ is a normalized probability between the node i and j. Note that a node can potentially be visited multiple times during the random walk process.

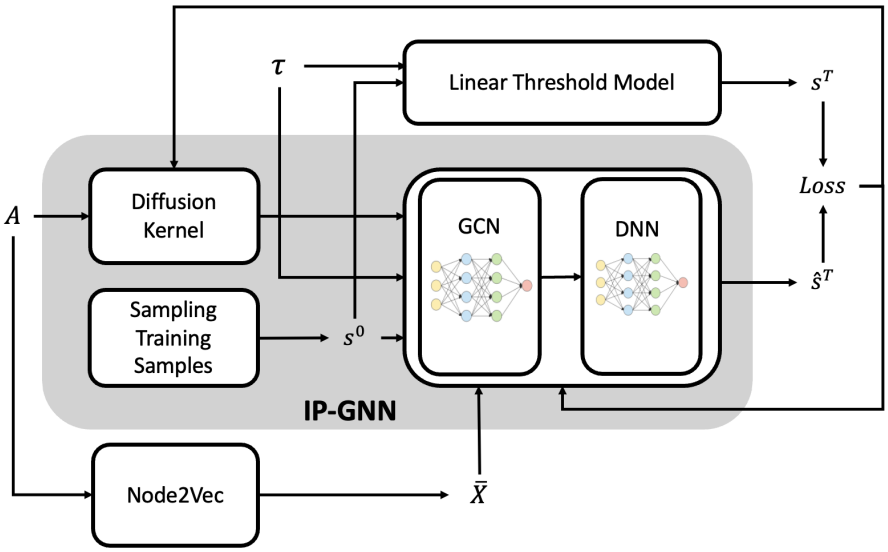

Fig. 2: A schematic illustration of the **IP-GNN** framework. Here, s^0 denotes the initial condition, τ denotes the threshold and $\overline{X}$ is the initial feature embedding obtained from Node2Vec. The framework consists of two major components—a modified graph neural network with adaptive diffusion kernel (shaded area) and a linear threshold model used to generate the ground truth activated nodes for each given seed. The output of **IP-GNN** corresponds to s^T, which is the predicted state of the nodes after T time steps.

IV. Proposed IP-GNN Framework

Figure 2 contains a visual representation of our proposed framework. The framework is designed to simulate the T-step information propagation process of the linear threshold model. As shown in the diagram, the framework employs a modified GNN with adaptive diffusion kernel and a stack of fully connected layers to learn the information propagation process. During training, **IP-GNN** would query the linear threshold model to generate a ground truth list of activated nodes for validation. It will compare its output $\hat{\mathbf{s}}^T$ against the ground truth $\mathbf{s}^T$ and uses the loss to update its model parameters. Once the network has been trained, the learned feature representation of the **IP-GNN** can be utilized for any subsequent downstream tasks. A detailed explanation of the framework is given in the subsections below.

A. Design of *IP-GNN* Architecture

Learning to simulate the output of the linear threshold model using deep neural networks is not a trivial task. Our objective here is to simulate the information propagation after multiple time steps. A naive attempt would be to use a stack of fully connected layers, but such a model would not be able to learn well given the lack of information about the network topology. For this reason, our framework uses a graph convolutional network (GCN) but a fundamental problem still remains. A vanilla GCN may only perform message passing effectively up to neighbors that are two or three hops away due to its well-documented over-smoothing effect [30]. As a result, the vanilla GCN will not have access to a larger neighborhood size information (i.e., T-hop neighbors) which is needed for effective modeling of the diffusion process. Furthermore, the relative strength of influence by the neighbors may depend on how far they are located from each other.

To allow the model to have access to a larger neighborhood size, we modified the GCN to employ an adaptive diffusion kernel [31] instead of a weighted adjacency matrix. This gives us the flexibility to consider the influence of neighbors located l-hops away, with learnable parameters $\{\lambda_l\}$ associated with each hop l. The adaptive kernel is defined as follows:

$$Q = \sum_{l=0}^{L} \frac{\lambda_l}{l!} \tilde{A}^l, \tag{4}$$

where $\tilde{A}$ is the normalized adjacency matrix defined in Eqn. (3) and λ_l is a learnable parameter that represents the influence of nodes located l hops away. The framework still uses a two-layer GCN, with its adjacency matrix replaced by the adaptive diffusion kernel. The forward pass of our modified GCN framework performs the following computation:

$$Z = Q^{(2)} \sigma \left(Q^{(1)} X \theta_1 \right) \theta_2 \tag{5}$$

where the $Q^{(i)}$'s correspond to the adaptive kernels (with potentially different set of weight parameters λ_l's), θ_1 and θ_2 are the GCN parameters while σ refers to the ReLU activation function. During training, the values of the learnable parameters λ's are initially set to 1 before they are updated during backpropagation.

Conventional GCN requires an input matrix X corresponding to the node features. However, since the linear threshold model does not require access to the node features, we provide

a structural embedding of the nodes as the input feture matrix X of the GCN. Specifically, we use the embedding generated by the Node2Vec [32] algorithm. Node2Vec performs truncated biased random walks to learn the feature embedding of the nodes. The node embedding obtained is then concatenated with the initial condition vector s^0 as well as the threshold vector τ before providing them as input to the GCN. Finally, the GCN's output Z will be sent through a stack of fully connected layers, which uses a sigmoid function in its final output layer, to generate predictions of the activation states of the node at time step T. Specifically, the predicted output of **IP-GNN** is as follows:

$$\hat{\mathbf{s}}^T = \text{FCN}(Z) \in [0,1]^{|V|}, \tag{6}$$

where Z is given by Eqn. (5).

B. Training Procedure

The **IP-GNN** framework will be trained to simulate the information propagation given an initial seed, $\mathbf{s}^0$. However, given the combinatorial number of possible seeds, ensuring the trained network generalizes well to any given input seed is a challenge. To help with the training process, our framework must be trained to use different initial seed configurations, where each seed contains a subset of nodes assumed to be activated. A naïve approach would be to use a uniform distribution to select the seed nodes for activation. Unfortunately, due to the scale-free property of most networks, this approach may not be effective as it tends to choose low degree nodes that have little influence on the information propagation. To circumvent this issue, we employ a degree-based sampling approach, in which the probability a node is chosen to be a seed is proportional to its node degree. The higher the degree, the more likely it will be selected as part of the initial seed. This allows the algorithm to learn the feature embedding of highly influential nodes in the network.

We created mini-batches of initial seed samples to train our graph neural network. First, we query the linear threshold model to obtain the T-step information propagation for each initial seed sample $\mathbf{s}^0$. The output of the linear threshold model will be used as the ground truth list of activated nodes, s^T. This process is repeated for every seed sample in the mini-batch to obtain their respective ground truth values. To improve its efficiency and reduce the number of queries, a new mini-batch is created only after the network has been trained for a certain number of epochs[1].

The mini-batches are used to train **IP-GNN** by updating the parameters of our deep neural network (which includes the learnable parameters of our adaptive diffusion kernel) in such a way that its predicted output $\hat{\mathbf{s}}^T$ is consistent with the ground truth $\mathbf{s}^T$. To ensure accurate prediction of activated nodes, we employ the following cross-entropy loss function:

$$L_{CE} = \frac{1}{|V|}\sum_{b=1}^{B}\sum_{i=1}^{|V|} s_{b,i}^T \log(\hat{s}_{b,i}^T) + (1-s_{b,i}^T)\log(1-\hat{s}_{b,i}^T) \tag{7}$$

[1]We use 200 epochs for each new mini-batch of samples in our experiments.

where $|V|$ is the number of nodes in the graph, B is the mini-batch size, $\hat{s}_{b,i}^T$ is the prediction for node i being activated, and $s_{b,i}^T$ is its ground truth value for the seed sample b in the mini-batch.

Additionally, to ensure that the model can accurately learn initial seed configurations that can produce a large number of activated nodes, the following weighted mean square error is also calculated:

$$L_{mse} = \sum_{b=1}^{B}\frac{K_b}{|V|}\sum_{i=1}^{|V|}(\hat{s}_{b,i}^T - s_{b,i}^T)^2 \tag{8}$$

where $K_b = \sum_i s_{b,i}^T$ is the actual number of activated nodes after T propagation steps for sample b in the mini-batch. The weight factor $\frac{K_b}{|V|}$ gives a higher penalty if the model incorrectly predicts the number of activated nodes when K_b is large for the given sample b.

Another challenge in training the network is to reconcile the discrete-valued ground truth activation states against the continuous-valued prediction of our deep neural network. In particular, a small change in the predicted sigmoid-layer output may not significantly alter the loss, causing the gradient of the network to saturate easily. To overcome this challenge, we employ the following entropy regularization to bias the predicted output towards either 0 or 1 when compared against the ground truth activation states of the nodes:

$$L_{ER} = -\frac{1}{|V|}\sum_{b=1}^{B}\sum_{i=1}^{|V|}\hat{s}_{b,i}^T \log(\hat{s}_{b,i}^T) \tag{9}$$

Note that the entropy is maximized when all the $\hat{s}_{b,i}$ are equal to 0.5 and minimized when $\hat{s}_{b,i}$ goes to 0 or 1.

The overall loss function to be optimized by the **IP-GNN** network is

$$L_{total} = \alpha L_{CE} + \beta L_{mse} + \gamma L_{ER} \tag{10}$$

where α, β and γ are hyper-parameters. Note that the loss function above is defined for one seed sample in the mini-batch. After the loss is calculated for all seed samples in the mini-batch, the gradient of the total loss will be used to update the model parameters during backpropagation.

V. Experiments

This section gives a detailed description of the experiments performed as well as the datasets used. The code implementation is available at https://github.com/frsantosp/IP-GNN.

A. Data Description

The experiments were performed using the following 4 real-world networks datasets. A summary of the characteristics for each dataset can be found in Table I.

- **Facebook [33]**: This network dataset was obtained from the Stanford Network Analysis Project (SNAP). The nodes represent Facebook users while the edges represent friendship relations among the users.
- **German Credit [34]**: This dataset contains information about customers of a German bank. Each node represents

TABLE I: Summary description of datasets.

Dataset	$\|V\|$	$\|E\|$
Wikipedia	889	2,900
Facebook	1045	53498
FB Food	620	2100
German	1000	24970

a bank customer while each edge represents a pair of customers with similar paying habits.

- **Facebook Food [35]**: This dataset contains verified Facebook pages that focus mainly on the topic of food. The nodes represent Facebook pages while the edges represent mutual likes between the pages.
- **Wikipedia** [35]: This dataset corresponds to the Wikipedia voting data. The data was collected since the beginning of Wikipedia until 2008. The nodes represent Wikipedia users while the edges are votes between users.

B. Experimental Setup

Our model consists of a two-layer GCN with 4096 hidden neurons in each layer along with a 4-layer fully-connected network with a hidden representation of size 512. We used Adam as our optimizer for both GCN and the fully connected layers. The learning rate was set to 0.01 for the GCN and 0.0001 for the fully connected layers. The model was trained using a batch size of 64 for a maximum of 2000 epochs. The switching to a new mini-batch of seed samples occured at every 200 epochs. The hyperparameters associated with our loss function were set to $\alpha = 1$, $\beta = 0.1$, and $\gamma = 0.0001$. Each initial seed configuration was designed to have $K_0 = 50$ activated nodes.

C. Metrics

After the deep neural network had been trained, for evaluation purposes, we randomly generated another 500 samples of initial seed configurations, $\mathbf{s}^0$, for each dataset. We then applied the linear threshold model to obtain the ground truth set of activated nodes, $\mathbf{s}^T$, associated with each seed sample. We compared the ground truth activation against the predicted output of **IP-GNN**, $\hat{\mathbf{s}}^T$, using the following metrics:

- Precision:

$$\text{Precision} = \frac{\text{\# True Positives}}{\text{\# True Positives + \# False Positive}}$$

- Recall:

$$\text{Recall} = \frac{\text{\# True Positives}}{\text{\# True Positives + \# False Negatives}}$$

- F1-score:

$$\text{F1-score} = 2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall + Precisions}}$$

- Root Mean Square Error:

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{\sum_{i=0}^{N-1}(y_i - \hat{y}_i)^2}{N}}$$

- Correlation:

$$r(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Precision, recall, and F1-score were used to evaluate the correctness of the activated nodes. Specifically, these metrics would compare $\mathbf{s}^T$ against $\hat{\mathbf{s}}^T$ for each initial seed configuration. The reported results correspond to the average value of these metrics computed over the 500 samples. RMSE and correlation compare the actual number of activated nodes against the predicted number. To do this, we first sum up the predicted activated nodes, $\hat{K}_T = \sum_{i=1}^{|V|} \hat{\mathbf{s}}_i^T$ and the actual activated nodes $K_T = \sum_{i=1}^{|V|} \mathbf{s}_i^T$ for each sample. We then create a vector containing the K_T and $\hat{K}_T$ values for all the samples before calculating their RMSE and correlation values.

D. Results

Table II summarizes the performance of **IP-GNN** on all 4 datasets. For the Wikipedia dataset, **IP-GNN** achieved a precision value of 0.9538, which means 95% of the nodes predicted to be activated were supposed to be active according to the linear threshold model. The recall for **IP-GNN** was found to be 0.9317, meaning 93% of the true activated nodes were correctly predicted to be active. The F1-score was also high, 0.9415, which suggests that our model was able to correctly predict the activated nodes in the Wikipedia dataset with a low number of false positives and false negatives.

The RMSE shows that, on average, around 31.65 nodes were misclassified for this specific dataset. Considering that the graph size is 889, the number of misclassified nodes is relatively small, which is about 3.6%. The correlation value is also high at 0.8816. As can be seen from Figure 4, there is a clear linear trend when plotting the predicted versus actual number of activated nodes for the 500 test samples.

The same trend was observed for the other 3 datasets. The precision, recall, and F1-scores were consistently above 0.93 for all datasets. Though the RMSE values for Facebook and German datasets were slightly higher, if we consider the size of the graphs, the error rate remained to be around 3%, which is similar to the magnitude observed in the Wikipedia dataset. The correlation values were above 0.8 for three out of the four datasets. The only dataset with a correlation value below 0.8 was the Facebook dataset. As shown in 4, a linear trend is still detectable for the Facebook dataset, though it is not as clear as the trend observed in other datasets. Despite its lower correlation, the RMSE shows that the model still performs quite well as its prediction error is only 36.57 nodes on average.

Figure 3 shows the true positive and false positive rates of **IP-GNN** on the four datasets. For the Wikipedia, Facebook and German datasets, their false positive rates were mostly below 0.2 while the true positive rates were mostly above 0.8. The Facebook food dataset struggles slightly by having a higher false positive rate (above 0.2) for some of the seed samples but still maintains a high true positive rate.

TABLE II: Performance evaluation for **IP-GNN** on four real-world datasets.

	Precision	Recall	F1	RMSE	Corr
Wikipedia	0.9538 $\pm$ 0.0378	0.9317 $\pm$ 0.0362	0.9415 $\pm$ 0.0194	31.65	0.8816
Facebook	0.9507 $\pm$ 0.0442	0.9305 $\pm$ 0.0349	0.9392 $\pm$ 0.0213	36.57	0.7008
FB Food	0.9383 $\pm$ 0.0577	0.9668$\pm$ 0.0210	0.9475 $\pm$ 0.0303	27.58	0.8180
German Credit	0.9439 $\pm$ 0.0398	0.9634 $\pm$ 0.0298	0.9524 $\pm$ 0.0148	33.68	0.8377

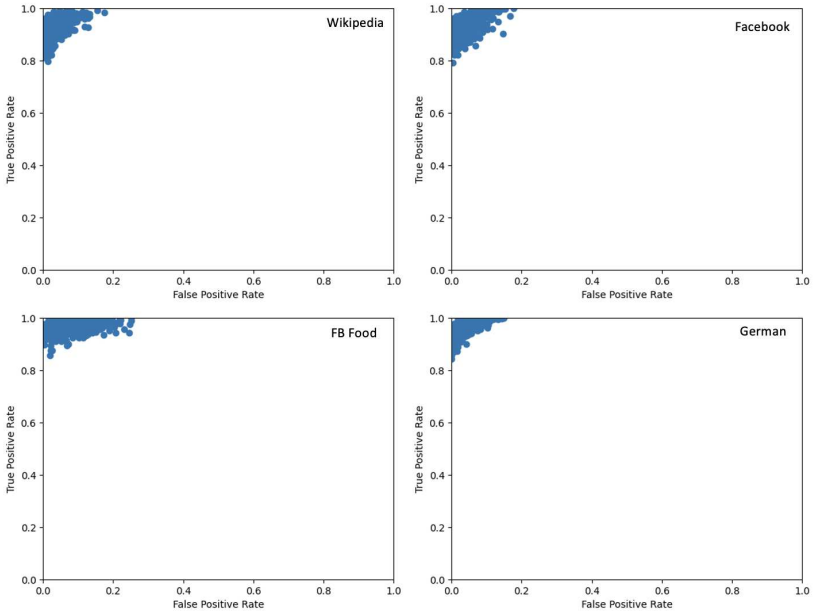

Fig. 3: Comparison of true positive and false positive rates of **IP-GNN** on 500 test samples.

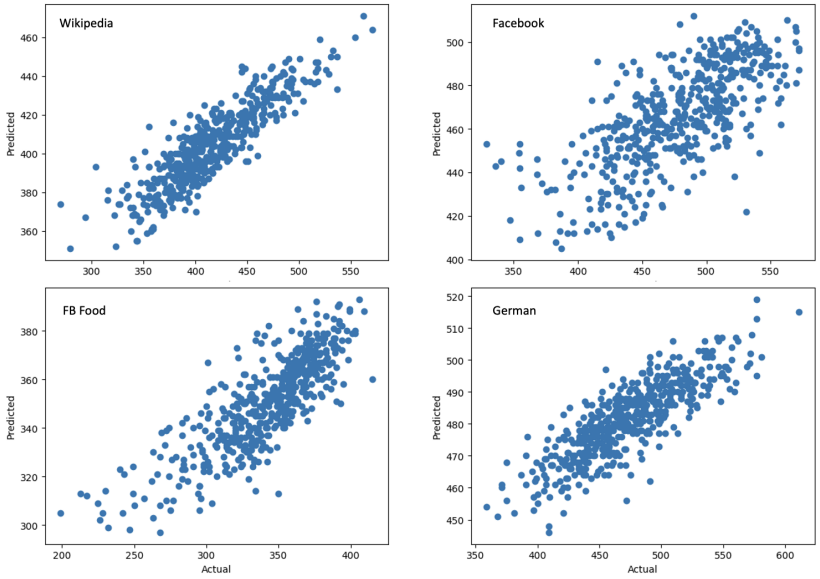

Fig. 4: Comparison of actual and predicted number of activated nodes by **IP-GNN** on 500 test samples.

VI. Conclusions

This paper presents a novel method of using deep neural networks to simulate influence propagation. We designed a framework that uses an adaptive diffusion kernel to overcome the oversmoothing problem of GCN, thus allowing **IP-GNN** to learn the influence propagation process beyond the 2-hop message passing of GCN. We also present an effective way to effectively train the model so it can achieve higher accuracy and fast convergence. Empirical results on 4 real-world datasets verified the effectiveness of our approach.

The results of this research open the door for many different possibilities. One possible research direction is to extend the framework to other diffusion models such as Independent Cascade. Another possible research direction is to investigate the value of the learned embedding for other downstream tasks such as forecasting information cascade, influence maximization, node classification, etc.

ACKNOWLEDGMENT

This material is based upon work supported by the NSF Program on Fairness in AI in collaboration with Amazon under grant #IIS-1939368. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or Amazon.

REFERENCES

[1] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 1029–1038.

[2] T. N. Dinh, H. Zhang, D. T. Nguyen, and M. T. Thai, "Cost-effective viral marketing for time-critical campaigns in large-scale social networks," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 2001–2011, 2013.

[3] J.-H. Cho, S. Rager, J. O'Donovan, S. Adali, and B. D. Horne, "Uncertainty-based false information propagation in social networks," *ACM Transactions on Social Computing*, vol. 2, no. 2, pp. 1–34, 2019.

[4] J.-H. Cho, T. Cook, S. Rager, J. O'Donovan, and S. Adali, "Modeling and analysis of uncertainty-based false information propagation in social networks," in *GLOBECOM 2017-2017 IEEE Global Communications Conference*. IEEE, 2017, pp. 1–7.

[5] Z. Wang, C. Xia, Z. Chen, and G. Chen, "Epidemic propagation with positive and negative preventive information in multiplex networks," *IEEE transactions on cybernetics*, vol. 51, no. 3, pp. 1454–1462, 2020.

[6] C. Xia, L. Wang, S. Sun, and J. Wang, "An sir model with infection delay and propagation vector in complex networks," *Nonlinear Dynamics*, vol. 69, pp. 927–934, 2012.

[7] F. Xiong, Y. Liu, Z.-j. Zhang, J. Zhu, and Y. Zhang, "An information diffusion model based on retweeting mechanism for online social media," *Physics letters A*, vol. 376, no. 30-31, pp. 2103–2108, 2012.

[8] A. Guille, H. Hacid, C. Favre, and D. A. Zighed, "Information diffusion in online social networks: A survey," *ACM Sigmod Record*, vol. 42, no. 2, pp. 17–28, 2013.

[9] N. Barbieri, F. Bonchi, and G. Manco, "Topic-aware social influence propagation models," *Knowledge and information systems*, vol. 37, pp. 555–584, 2013.

[10] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 137–146.

[11] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1852–1872, 2018.

[12] S. Krishnan, P. Butler, R. Tandon, J. Leskovec, and N. Ramakrishnan, "Seeing the forest for the trees: new approaches to forecasting cascades," in *Proceedings of the 8th ACM Conference on Web Science*, 2016, pp. 249–258.

[13] L. Weng, F. Menczer, and Y.-Y. Ahn, "Virality prediction and community structure in social networks," *Scientific reports*, vol. 3, no. 1, pp. 1–6, 2013.

[14] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[15] K.-H. N. Bui, J. Cho, and H. Yi, "Spatial-temporal graph neural network for traffic forecasting: An overview and open research issues," *Applied Intelligence*, vol. 52, no. 3, pp. 2763–2774, 2022.

[16] T. Wilson, P.-N. Tan, and L. Luo, "A low rank weighted graph convolutional approach to weather prediction," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 627–636.

[17] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[18] S. Aral, L. Muchnik, and A. Sundararajan, "Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks," *Proceedings of the National Academy of Sciences*, vol. 106, no. 51, pp. 21 544–21 549, 2009.

[19] M. Granovetter, "Threshold models of collective behavior," *American journal of sociology*, vol. 83, no. 6, pp. 1420–1443, 1978.

[20] T. C. Schelling, *Micromotives and macrobehavior*. WW Norton & Company, 2006.

[21] M. Pautasso and M. J. Jeger, "Epidemic threshold and network structure: The interplay of probability of transmission and of persistence in small-size directed networks," *Ecological Complexity*, vol. 5, no. 1, pp. 1–8, 2008.

[22] S. Aral and P. S. Dhillon, "Social influence maximization under empirical influence models," *Nature human behaviour*, vol. 2, no. 6, pp. 375–382, 2018.

[23] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[24] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

[25] F. Masrour, T. Wilson, H. Yan, P.-N. Tan, and A. Esfahanian, "Bursting the filter bubble: Fairness-aware network link prediction," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 841–848.

[26] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[27] Z. Zhu, Z. Zhang, L.-P. Xhonneux, and J. Tang, "Neural bellman-ford networks: A general graph neural network framework for link prediction," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 476–29 490, 2021.

[28] E. Müller, "Graph clustering with graph neural networks," *Journal of Machine Learning Research*, vol. 24, pp. 1–21, 2023.

[29] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *International conference on machine learning*. PMLR, 2020, pp. 874–883.

[30] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," 2018.

[31] J. Zhao, Y. Dong, M. Ding, E. Kharlamov, and J. Tang, "Adaptive diffusion in graph neural networks," *Advances in neural information processing systems*, vol. 34, pp. 23 321–23 333, 2021.

[32] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[33] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[34] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[35] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *AAAI*, 2015. [Online]. Available: https://networkrepository.com