Embedding design during information cascade

by

Shubham Sharma (18i190002)
Pintu Kumar(194193002)

Under the guidance of

Prof. Abir De

Indian Institute of Technology Bombay
Powai, Mumbai 400076

Contents

0.1 Introduction

With emergence of social network, it has motivated a large area of research. Various task such as node classification, link prediction and community detection has been explored in context of social networks. Another important research topic in the area is temporal diffusion of information. The aim is of the study is to capture how interaction between users effect the spread of information (information here refers to things like pictures, some written post/information shared by a user).

This kind of study can find direct application in stopping spread of rumour on social network, or to target right kind of audience to display certain type of information (like advertisement) so that it reaches the intended audience with least number of initial nodes. When a problem is modelled correctly using graph this can also be used in the field of epidemiology. It can help break the chain of highly infectious disease.

Many propagation method rely on probabilistic modelling of information, these models are based on explicit relationship between nodes in the network (based on links and structural information between the nodes). Most of the time the underlying model can be much more complex. Consider a case where 2 people have exactly same set of friends, i.e., in graphical representation of data these two nodes have same structural property. If information diffusion is modelled based on relationship between nodes then these two nodes is expected to posses similar property in sense of information diffusion. now in real life consider one person is not so active on social network and seldom comment or share some post, whereas the other person is a social media extrovert and gives his reaction to almost every post he comes across. Then both these people have very different information diffusion model which in previous model can be considered similar to each other. Some work in the area suggested to discover implicit structure of diffusion from user's behaviour before modelling with respect to the extracted graph.

Due to complexity of real world problem assuming a underlying model is impractical. Zhao et al.[3] proposes a method Deep collaborative embedding (DCE), which aim to find prediction of information diffusion without requirement of knowledge about the underlying diffusion mechanism. This algorithm will be discussed in detail in this report. DCE tries to learn embedding such that nodes which are expected to get infected from each other are close in embedding space. Loss function also incorporates structural proximity of nodes and hence preserves structural information too. Embedding obtained from DCE not only reflect the structural infromation about nodes but also captures the information diffusion in the network.

0.2 Structural Node Embedding

Graphs can not be passed directly to a machine learning algorithm. One way to tackle this is to map every node to a vector in $\mathbb{R}^d$ ($d << |V|$) and apply algorithm on this embedded space. This mapping/function should preserve similarity between nodes. What is meant by similarity varies

from algorithm to algorithm. In broader sense nodes similar to each other on graph should be mapped close in embedding space.

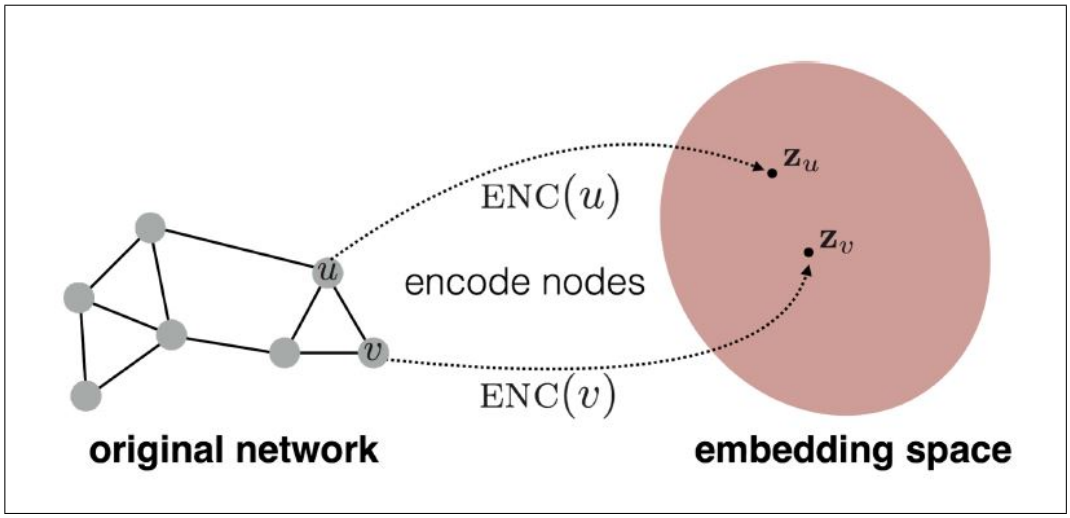

Figure 1: [1]Illustration for graph embedding, goal is to learn a embedding function which preserves similarity.

0.2.1 Encoder-Decoder approach

To transform a graph into embedding space we need following 4 component:

- A function $f : V \to \mathbb{R}^d$, this maps every node to a vector. This function is called Encoder function, this need to be learned (approximated) to be able to embed a graph.
- A notion of similarity between nodes in original space. Formally $S_G : V \times V \to \mathbb{R}^+$ is similarity between nodes. One need to define this similarity before hand.
- A notion of similarity between vectors in embedding space. This function is called decoder function. Formally $Dec : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$
- A loss function/ reconstruction error. $l : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$ such that $l(Dec(z_i, z_j), S_G(v_i, v_j))$

One intend to minimize the loss function l by updating the encoder function.
We want $Dec(enc(v_i), enc(v_j)) \approx S_G(v_i, v_j)$.

For a set of training node D we minimise the empirical loss by minimizing

$$L = \sum_{(v_i,v_j)\in D} l(Dec(z_i, z_j), s_G(v_i, v_j)) \tag{1}$$

Node embedding techniques are very successful when we try to preserve the structural property of graph. By structural proximity one mean nodes that are related to each other (either have similar structural role or there is a low distance path between nodes) are mapped close to each other in embedding space. Methods like GraRep, HOPE are matrix factorization method, whereas methods like Deepwalk, Node2vec, LINE are based on random walk approach to find similarity in origanl space. These methods have performed quite well in capturing the structural information of the graph

0.3 Autoencoder

An Autoencoder is an unsupervised artificial neural network that is trained to efficiently compress (encode) data into a lower dimensional representation and again reconstruct the original feature vector using the encoded vector. This is used widely for dimensionality reduction and get rid of noise. If we hand engineer feature for every node and train a autoencoder to reconstruct this feature then the bottleneck layer (compressed vector/ encoded vector) can be interpreted as an embedding for the node.

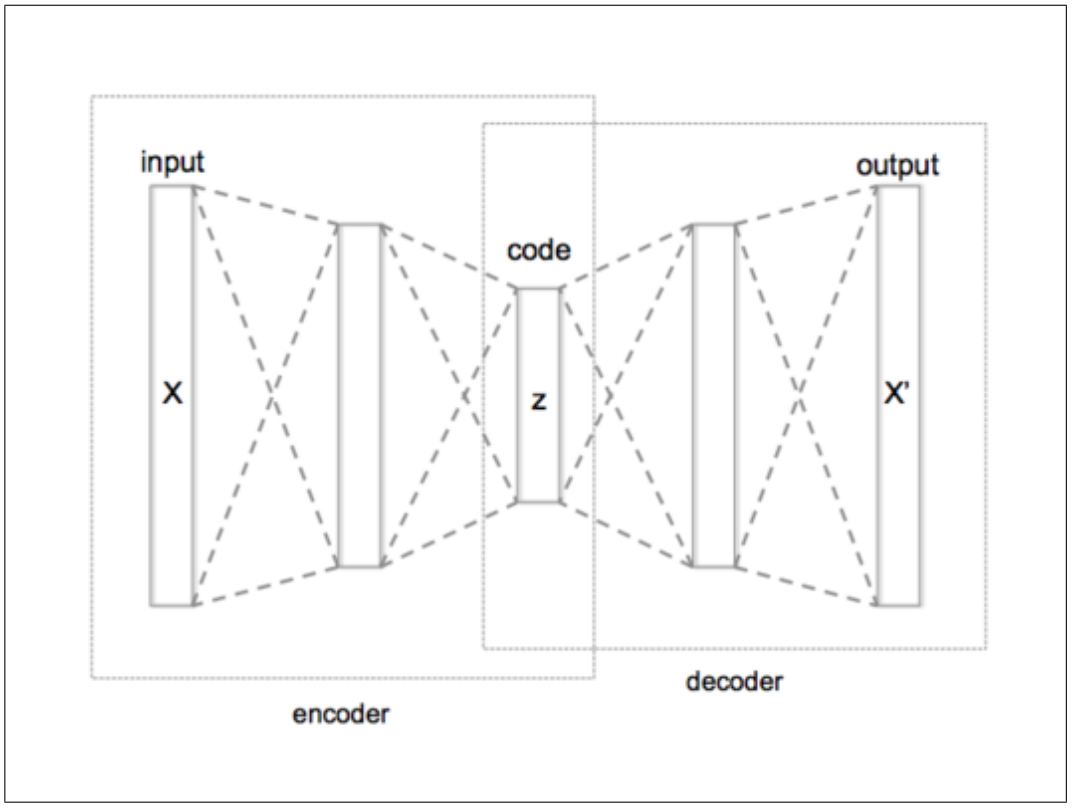

Figure 2: Illustration of a autoencoder, x is the input, z is the compressed vector representation of x and x' is reconstructed vector using the embedded vector

Check figure 2. The aim of autoencoder is to reconstruct input from the encoded vector, so to learn the parameter of a autoencoder we minimize the reconstruction loss of the network. This means loss function should penalise if x and x' are far from each other. So the loss function is given by.

$$\mathcal{L} = ||x - x'||^2 \tag{2}$$

0.4 Problem Definition

Deep Collaborative Embedding aims at prediction of node infection without knowledge about underlying diffusion model. With the help of cascade information and node collaboration it proposes an autoencoder based embedding. This helps embedding capture both information diffusion as well as structural information in a single embedding.

0.4.1 Basic Definitions

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, where $\mathcal{V}$ is the set of nodes(say N nodes) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Let $\mathcal{C} = \{C_1, C_2, C_3, C_4, ..., C_M\}$ be the set of M information cascades. We define an information cascade $C_m (1 \leq m \leq M)$ observed on a social network $\mathcal{G}$ as a set of timestamped infections, i.e., $C_m = \{(v, t_v^{(m)}) | v \in \mathcal{V} \wedge t_v^{(m)} < \infty\}$, where $(v, t_v^{(m)})$ is node v infected by cascade C_m at time $t_v^{(m)}$. We say $v_i \in C_m$ if node v_i participates in cascade C_m. Additionally, $C_m(t) = \{(v, t_v^{(m)}) | v \in \mathcal{V} \wedge t_v^{(m)} < t\}$ is the set of all nodes infected by cascade C_m before time t. $\bar{C}_m(t) = \mathcal{V} \backslash C_m$ is the set of nodes that have not been infected before t.

0.4.2 Problem Definition

The target problem is: Given a set of cascades $\mathcal{C} = (C_1, C_2, ..., C_M)$ observed on given network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we want to learn embeddings for nodes in $\mathcal{V}$, where the embeddings are capturing not only the structural properties of the graph but also the cascading characteristics of the nodes.

0.5 Modeling Cascading Characteristics

Cascading characteristics of a node reveal its relation to other nodes in information cascades, which are crucial to the prediction of node infection and infection order. Let us define two cascading characteristics, cascading context and the cascading affinity.

0.5.1 Cascading Context

Cascading Context of a node captures its temporal relation to other nodes in that cascade. It includes the potential influence imposed by other nodes in that cascade.

Definition 1. ***(Cascading Context)****: Given the set of M cascades on a network $\mathcal{G}$ of N nodes,$\mathcal{C} = \{C_1, C_2, C_3, C_4, ..., C_M\}$, the cascading context of the nodes involved in cascade $C_m (1 \leq m \leq M)$ is defined as a matrix $X^{(m)} \in \mathbb{R}^{N \times N}$. The entry at the $u-th$ row and the $v-th$ column of $X^{(m)}$ represents the potential influence from node v to u, which is defined as*

$$x_{u,v}^{(m)} = \begin{cases} \exp\left(-\frac{t_u^{(m)} - t_v^{(m)}}{\tau}\right) & , t_u^{(m)} < t_v^{(m)} \\ 0 & , t_u^{(m)} \geq t_v^{(m)} \end{cases} \tag{3}$$

where $t_u^{(m)}$ is the infection time of u in cascade C_m and τ is the decaying factor. The cascading context of node u in cascade C_m is defined as the row vector $x_u^{(m)} = X_{u,*}^{(m)}$.

0.5.2 Cascading Affinity

Cascading Affinity of two nodes measures the similarity of them with respect to the cascades, which can be defined in terms of their co-occurrences in historical cascades as follows:

Definition 2. ***(Cascading Affinity)****: Given the set of M cascades ona a social network $\mathcal{G}$ of N nodes, i.e., $\mathcal{C} = \{C_1, C_2, C_3, C_4, ..., C_M\}$, the Cascading Affinity of two nodes u and v is represented by entry at the u-th row and v-th column of the cascading affinity matrix $A \in \mathbb{R}^{N \times N}$, which is defined as the ratio of the cascades involving both u and v, i.e.,*

$$a_{u,v} = \frac{|\{C_k | u \in C_k, v \in C_k, C_k \in \mathcal{C}\}|}{|\mathcal{C}|} \tag{4}$$

0.6 Deep Collaborative Embedding

The embeddings are auto-encoder based Deep Collaborative Embedding (DCE) model, which can learn embeddings for nodes in a given social network, based on the M cascades $C_1, ..., C_M$ observed on the network, so that the learned embeddings can be used for cascade prediction without knowing the underlying diffusion mechanisms and the explicit diffusion networks.

0.6.1 Architecture of DCE

The architecture of DCE is shown in 3. As we can see the embeddings are learnt from two different collaborations, the cascade collaboration and the node collaboration. DCE uses M different auto-encoders to extract the cascading context information of nodes for M different

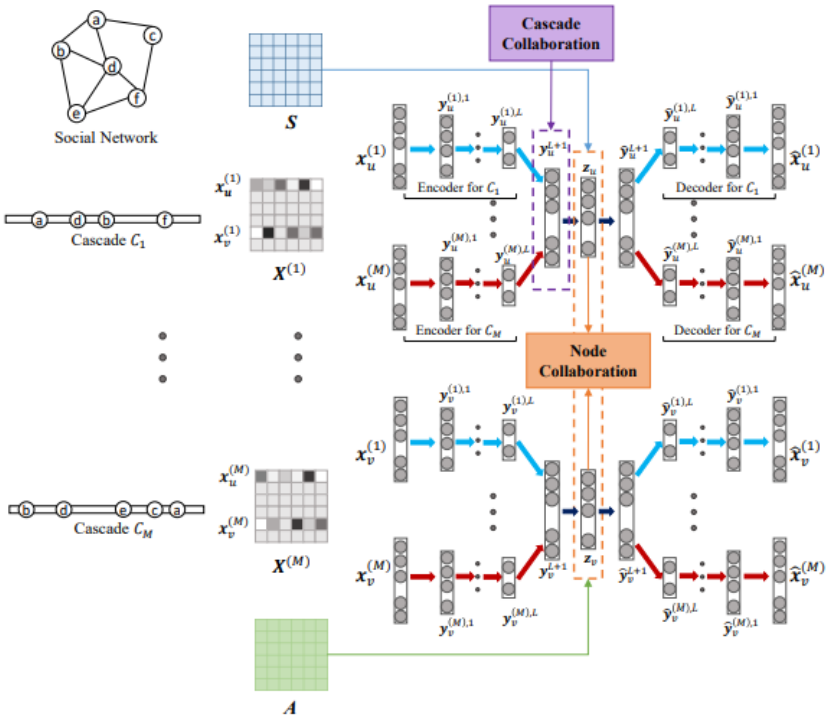

Figure 3: Architecture of DCE

cascading contexts and then use them to find the embedding for a node. The encoder part consisting L non-linear hidden layers is defined by the following equations:

$$y_v^{(m),1} = \sigma(W^{(m),1} x_v^{(m)} + b^{(m),1}), \tag{5}$$

$$y_v^{(m),l} = \sigma(W^{(m),l} x_v^{(m)} + b^{(m),l}), \quad \forall l \in \{2, 3, ..., L\}, \tag{6}$$

where $y_v^{(m),l}$ is the output vector of l-th hidden layer of m-th auto-encoder taking $x_v^{(m)}$ as input, $W^{(m),l}$ is the parameter matrix of that layer and $b^{(m),l}$ is the corresponding bias. The resulting embedding z_v is generated by fusing the M intermediate embeddings $y_v^{(m),l} (1 \leq m \leq M)$ through the following non-linear mappings:

$$y_v^{L+1} = \sigma(\sum_{m=1}^{M} (W^{(m),L+1} y_v^{(m),L} + b^{(m),L+1})) \tag{7}$$

$$z_v = \sigma(W^{L+2} y_v^{L+1} + b^{L+2}) \tag{8}$$

The decoder part of the autoencoders is similar to the encoder part.

0.6.2 Optimization Objective of DCE

There are three main parts of the loss function. Let us see them one by one.

Loss Function for Cascade Collaboration

The first loss function is the reconstruction MSE loss for M different autoencoders that we have. We want to minimize the reconstruction error between $x_v^{(m)}$ and $\hat{x}_i$. the loss fuction for this part is :

$$\mathcal{L}_x = \sum_{m=1}^{M} \sum_{v \in \mathcal{V}} ||(x_v^{(m)} - \hat{x_v}^{(m)})||_2^2 \tag{9}$$

$$= ||(X^{(m)} - \hat{X}^{(m)})||_2^F \tag{10}$$

where $X_v^{(m)}$ and $\hat{X_v^{(m)}}$ are the original context matrix and the reconstructed cascading context matrix of cascade C_m, respectively.
In practice, this cascading matrix is sparse, which leads to undesired) vectors in the embeddings z_v and the reconstructed $x_v^{(m)}$ if the sparse vectors $x_v^{(m)}$ are directly fed into the networks. To overcome this issue, a matrix P is defined and the $\mathcal{L}_x$ can be redefine as

$$\mathcal{L}_x = \sum_{m=1}^{M} \sum_{v \in \mathcal{V}} ||(x_v^{(m)} - \hat{x_v}^{(m)}) \odot p_v^{(m)}||_2^2 \tag{11}$$

$$= \sum_{m=1}^{M} ||(X^{(m)} - \hat{X}^{(m)}) \odot P^{(m)}||_2^2 \tag{12}$$

where $\odot$ denoted the Hadamard product, and the u-th column vector of the matrix $P^{(m)} \in \mathbb{R}^{N \times N}$ is the weight vector $p_u^{(m)} = \{p_{u,v}^{(m)}\}_{v \in \mathcal{V}}$ assigned to cascading context $x_u^{(m)}$. $p_{u,v}^{(m)} = \rho > 1$ if $x_{u,v}^{(m)} \neq 0$, otherwise $p_{u,v}^{(m)} = 1$.

Loss Functions for Node Collaboration

Let us now see the loss function for node collaboration. As we want the embedding z_i to preserve the cascading affinity of nodes in cascades and also the structural proximity of the nodes. The loss function to preserve cascading affinity of the corresponding nodes is :

$$\mathcal{L}_a = \sum_{u,v \in \mathcal{V}} a_{u,v} ||z_u - z_v||_2^2. \tag{13}$$

where $a_{u,v}$ is the cascading affinity between u and v. Let us now see the part of the loss function to preserve the structural proximity of their corresponding nodes:

$$\mathcal{L}_s = \sum_{u,v \in \mathcal{V}} s_{u,v} ||z_u - z_v||_2^2. \tag{14}$$

This is similar to the $\mathcal{L}_a$. We have taken S matrix to be the adjacency matrix. Let us now write $\mathcal{L}_a$ and $\mathcal{L}_s$ in the form of matrix multiplications. Let $L^{(a)}$ be the laplacian matrix of A, i.e., $L^{(a)} = D^{(a)} - A$, where $D^{(a)}$ is diagonal and $D_{u,u}^{(s)} = \sum_{v \in \mathcal{V}} a_{u,v}$. Let $L^{(s)}$ be the laplacian matrix

of A, i.e., $L^{(s)} = D^{(s)} - S$, where $D^{(a)}$ is diagonal and $D^{(s)}_{u,u} = \sum_{v \in \mathcal{V}} s_{u,v}$. Thus, we can rewrite $\mathcal{L}_a$ and $\mathcal{L}_s$ as :

$$\mathcal{L}_a = 2tr(ZL^aZ^T), \tag{15}$$

$$\mathcal{L}_s = 2tr(ZL^sZ^T), \tag{16}$$

where z is the embedding where $i-$th column is z_i.

Total Loss Function

By conbining $\mathcal{L}_x, \mathcal{L}_a$ and $\mathcal{L}_s$, the complete loss function is follows:

$$\mathcal{L} = \mathcal{L}_x + \alpha\mathcal{L}_a + \beta\mathcal{L}_s + \gamma\mathcal{L}_{reg} \tag{17}$$

where is the $\mathcal{L}2-$norm regularization term to avoid overfitting, and α, β and γ are non-negative parameters used to control the contribution of the terms.

Algorithm 1: Algorithm 1

Input: The set of cascading context matrices $\mathcal{X} = (X_1, X_2, ..., X_M)$, cascading affinity matrix A, structural proximity matrix S, and the parameters α, β and γ.
Output: Node Embedding Z 1. Initialize $W, \hat{W}, b$ and $\hat{b}$.
2. **Repeat**
3. Compute $Z, \hat{X}$ as per the equations defined above.
4. Compute the total loss $\mathcal{L}$.
5. Update $W, \hat{W}, b$ and $\hat{b}$ using backpropagation.
6. **Until** $\mathcal{L}$ converges.

0.7 Experiments

0.7.1 Dataset

The dataset that we have used is Digg[2] dataset. The dataset extracted from Digg contains 3,553 stories, 139,409 users, and 3,018,197 votes with timestamps. A vote for a story is treated as an infection of that story, and the votes for the same story constitute a cascade. In addition, a social link exists between two users if one of them is watching or is a fan of the other one. Due to the lack of resources to train such a big dataset, we have used a subset of the dataset to train the embeddings. We have considered top 500 highly participating nodes with 50 random cascades. These cascades are then split into 80% training and 20% test cascades.

0.7.2 Training

The hyperparameters α, β and γ are taken as $\alpha = 0.1, \beta = 0.9$ and $\alpha = 0.2, \beta = 0.8$ and $\alpha = 0.3, \beta = 0.7$ and $\alpha = 0.4, \beta = 0.6$ and $\alpha = 0.5, \beta = 0.5$ with $\gamma = 0.002$. The value of decaying factor τ is taken as 100000. The value is tau is taken by hit and trial by using values like 10, 100, 1000, 10000, 100000 etc. The value of ρ matrix is taken as by hit and trail by using values like 2, 10, 100. The training has been done of google colab. The optimizer used is adam with a learning rate of 0.05. The network has been trained for 10000 epochs.

0.7.3 Testing

Given a cascade C and some seed nodes which are already infected, we try to predict probabilities of infection of remaining nodes. Generally and in our experiment we have assumed initial 1% of nodes as seed nodes.

$p(u|v)$ is the probability that u would be infected given v is already infected. u would be infected by v if their embedding are close to each other. If z_u, z_v denote embedding for u and v respectively then

$$p(u|v) = \frac{1}{1 + exp(||z_u - z_v||^2)} \tag{18}$$

Let V_t the set of seed nodes then the probability that node $u \in \mathcal{V}_t$ will be infected is given by

$$p(u|V_t) = 1 - \prod_{v \in V_t} (1 - p(u|v)) \tag{19}$$

For each uninfected node probability is calculated according to equation 19 and sort it in descending order to obtain $\hat{R}_c$.
If the cascade has large number of nodes then the probability in equation 19 can become arbitrarily small. So rather than sorting probability from equation 19 in descending order we rather sort $\sum_{v \in V_t}(log(1 - p(u|v)))$ in ascending order to obtain $\hat{R}_c$ then we compare it with ground truth R_c to obtain MAP.

Top n-precision of $\hat{R}_c$ is the hitting rate of first n nodes over the ground truth

$$P_{C,n} = \frac{|\hat{R}_{C,n} \cap R_C|}{|\hat{R}_{C,n}|} \tag{20}$$

Where $\hat{R}_{C,n}$ is set of first n nodes of $\hat{R}$.

Average precision for a cascade is defined as

$$AP_C = \frac{\sum_v \in R_c P_{C,r_{C,v}}}{|R_C|} \tag{21}$$

here r_C, v denote rank of node v in $\hat{R}_C$.

We set the size k of the predicted list $\hat{R}_C$ in $\{50, 70, 100, 120\}$ to compute $AP_C@k$, then over a testing set C_t mean average precsion over all cascade is defined as

$$MAP@k = \frac{1}{|C_t|} \sum_{C \in C_t} AP_C@k \tag{22}$$

0.7.4 Results

Using the embedding obtained On training with different α, and β value, we record MAP on testing cascade and obtained results that are displayed in figure 4.

α	β	K=50	K=70	K=100	K=120
0.1	0.9	0.36202	0.36121	0.35060	0.34422
0.2	0.8	0.276097	0.290107	0.298267	0.298453
0.3	0.7	0.26222	0.27085	0.26329	0.26719
0.4	0.6	0.3316	0.33293	0.33813	0.34353
0.5	0.5	0.31986	0.31182	0.29985	0.29214

Figure 4: MAP@k obtained for different α and β

Python code that was used for performing these experminets are available at : https://github.com/shubham1166/Embedding_design_during_information_cascade

0.8 Inference

- As the algorithm learns different embedding using different training cascade, there is no use of underlying diffusion model.
- Cascade context matrix $(X^{(m)})$ and $(P^{(m)})$ is of size $|V|^2$ and we need 1000's of such matrix (no. of cascade) and requires large storage capacity, hence limit its capability of being trained on real world network normal systems.

0.9 Future Work

- Number of nodes in network was reduced substantially in order to be able to train the network on the resources available to us. Because of this we lost important structural information about the graph. Zhao et el.[3] do not compare result on task that incorporates structural information (link prediction or node classification). Though model discussed in this report is cable of performing good on these task, with the limitation to only work on pruned network did not allowed us to test and compare these task. Comparing these task when one has ability to train complete graph can be a future direction of work.

- Matrix S that captures the structural proximity in this example is nothing but the adjacency matrix. There are many Scores that can be used instead of adjacency matrix, Common neighbours, Katz measure, Preferential attachment can be few of them. Adjacency matrix captures property of one hop neighbours and hence misses the global structure of graph. A method incorporating higher order properties may be expected to perform better.

- As mentioned in inference, one may look at modification of algorithm so that it can be applied to large real world data even with limited computational resources.

Bibliography

[1] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[2] Tad Hogg and Kristina Lerman. Social dynamics of digg. *EPJ Data Science*, 1(1):5, 2012.

[3] Yuhui Zhao, Ning Yang, Tao Lin, and S Yu Philip. Deep collaborative embedding for information cascade prediction. *Knowledge-Based Systems*, 193:105502, 2020.