Complex & Intelligent Systems
https://doi.org/10.1007/s40747-021-00332-x

ORIGINAL ARTICLE

Temporal network embedding using graph attention network

Anuraj Mohan[1] · **K V Pramod**[2]

Received: 9 November 2020 / Accepted: 9 March 2021

Abstract
Graph convolutional network (GCN) has made remarkable progress in learning good representations from graph-structured data. The layer-wise propagation rule of conventional GCN is designed in such a way that the feature aggregation at each node depends on the features of the one-hop neighbouring nodes. Adding an attention layer over the GCN can allow the network to provide different importance within various one-hop neighbours. These methods can capture the properties of static network, but is not well suited to capture the temporal patterns in time-varying networks. In this work, we propose a temporal graph attention network (TempGAN), where the aim is to learn representations from continuous-time temporal network by preserving the temporal proximity between nodes of the network. First, we perform a temporal walk over the network to generate a positive pointwise mutual information matrix (PPMI) which denote the temporal correlation between the nodes. Furthermore, we design a TempGAN architecture which uses both adjacency and PPMI information to generate node embeddings from temporal network. Finally, we conduct link prediction experiments by designing a TempGAN autoencoder to evaluate the quality of the embedding generated, and the results are compared with other state-of-the-art methods.

Keywords Network embedding · Graph convolution · Graph autoencoder · Temporal networks

Introduction

Learning from non-euclidean data [1] has gained a lot of scientific attention in recent years. Among those data, learning from network structured data is one challenging direction which has diverse applications in fields like recommender systems [2], computational social systems [3], text mining [4], service oriented and content delivery networks [5,6], and systems biology [7]. With the success of deep learning in various domains, those methods became prominent and fruitful in learning network representations which eventually lead to the development of subdomain in machine learning named as network embedding or network representation learning (NRL) [8–11]. Initial works in this domain were based on unsupervised learning using skip-gram neural network architectures, followed by deep neural networks. Another research direction was to use conventional convolutional neural network architectures to learn network representations. Applying the traditional convolution operation on graphs is found to generate sub-optimal results as the network structure is highly irregular. Substantial developments in the field of NRL occurred with the proposal of graph convolutions which is very effective when the input is highly irregular. A plethora of works [12–15] has been proposed based on graph convolutions which can be mainly classified into spectral and spatial graph convolution-based methods. Among those methods, our work particularly focus on the direction of graph convolutional network [13] because of its wide popularity and effectiveness.

The basic principle of GCN is to learn the representations of each node by aggregating the features of its first-order neighbours through a parameterized learning mechanism. The receptive field of each node includes the immediate one-hop neighbours, which is shown for node A in Fig. 1. GCN has proved to be very successful in many state-of-the-art network mining tasks like node classification and link prediction. Some variants have been already proposed for GCN which makes the model more robust and scalable.

✉ Anuraj Mohan
anurajmohan@gmail.com

K V Pramod
pramodkv4@gmail.com

[1] Research Scholar, Artificial Intelligence Lab, Department of Computer Applications, Cochin University of Science and Technology, Kerala 682022, India

[2] Department of Computer Applications, Cochin University of Science and Technology, Kerala 682022, India

Published online: 30 March 2021

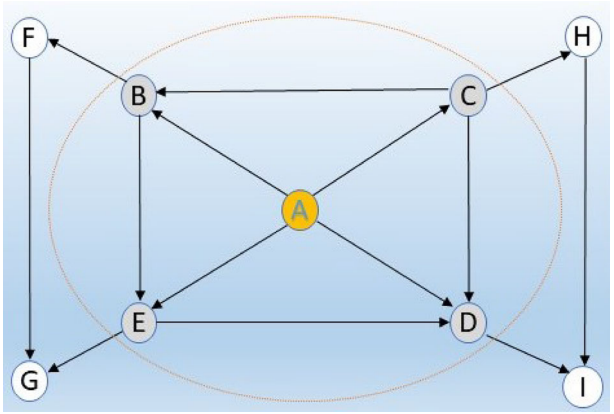

Fig. 1 Receptive field of Node A

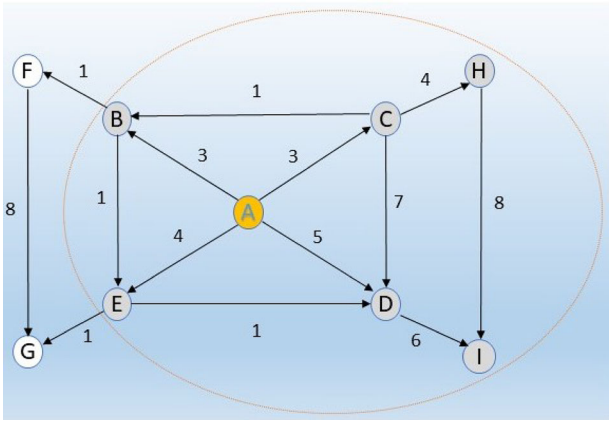

Fig. 2 A temporal network

The basic GCN model is designed only to work with static networks, where the time-varying nature of the network is not considered. However, in real world, most of the networks are either evolving in nature or they carry temporal information in their edges. We call them either as a dynamic network which is represented as network snapshots where the importance is given to their evolving nature, or a temporal network which is represented as network with time-stamped edges where the importance is given to the change in their connectivity patterns w.r.t. time. In this work, we focus on the temporal aspect where the input graph is represented with edges which carry temporal information. Telephone call networks, email communication networks, disease spread networks, etc. are some typical examples for a temporal network. The notion of temporal edge is highly sensitive when we model virus spread as a network, because the temporal ordering of contacts is an important factor to be considered while modelling the spread of disease from one person to other. The temporal relationships between the nodes are an important property to be preserved while embedding the nodes in a temporal network to the vector space.

Some works [16–18] focused on network embedding from dynamic networks which consider network snapshots at consecutive time-steps as input. However, network embedding from temporal network has received less attention apart from a few works [19] which used unsupervised approaches using random walk and skip-gram neural architectures. As graph convolutional network being the most prominent and effective method for network embedding, we aim to develop a method based on GCN which can learn node embeddings by considering the temporal information present in the network. A temporal network and the receptive field of node A (given in circle) are shown in Fig. 2. For node A, the receptive field includes the one-hop neighbours (B, C, D, E) along the nodes (H, I) which are in the temporal neighbourhood (time ordered path exist) of A. Nodes F and G are ignored as temporal neighbours as they cannot be reached using a time-respecting walk from A. Considering this temporal receptive field while learning node embeddings can generate more useful representations, which is the primary motivation of this research.

In GCN, the feature aggregation at each node assumes that all neighbours have same importance which may reduce the model capacity. Graph attention network (GAT) [20] is an approach which showed that, by providing attention to neighbours which can be learned by end-to-end training, we may build a more robust model. In this work, we follow the attention mechanism as suggested by GAT so to include the hypothesis that different temporal neighbours may contribute differently during the aggregation process. Both GCN and GAT consider the edge distribution of the network to be static and are not well suited for representation learning from temporal network whose edge distributions vary over time. Moreover, they focus on preserving the first-order neighbourhood while generating embeddings. In the case of temporal networks, temporal order proximity is another important property to be preserved while generating node embeddings.

The main contributions of the paper are as follows.

The work addresses the problem of generating node embedding from temporal networks.

The work provides a methodology to incorporate temporal information into a graph attention network for generating time-aware node embeddings.

A graph autoencoder based on proposed method is designed which can perform link prediction on real-world temporal networks .

To the best of our knowledge, this is the first work which apply GCN concepts over temporal network data represented as edge streams.

The rest of the paper is organized as follows. “Related works” and “Definitions and preliminaries” cover the related works and preliminary concepts, respectively. “Methodology” presents the proposed methodology. “Experimental

setup" and "Result and analysis" discuss the experimental setup and the results, respectively. Finally, "Conclusion" presents the conclusion and future works.

Related works

Machine learning has made tremendous improvements in various areas like speech recognition [21,22], object detection [23,24], and text mining [25,26]. With the rise of large-scale social networks, a substantial amount of research has been conducted around machine learning with network or graph-structured data. As feature (representation) learning being an important task in machine learning pipeline, methods for learning good representations of nodes in a network gained importance. The emergence of deep learning accelerated the growth of this typical research area, and various methods were proposed for network representation learning based on neural architectures. It includes works based on skip gram architectures [27,28], deep neural networks [29,30], and graph neural networks [31]. In this work, we particularly focus on graph neural networks (GNN) which has its roots in the field of signal processing with graphs [32].

The basic GNN aimed to extend the traditional neural network concepts to work with data represented in network domain. Among the variants of GNN, works based on graph convolutions [33,34] gained wide popularity. They can be mainly classified into spectral and spatial graph convolution-based methods [15]. Spectral methods include graph filtering-based approaches [35] along with some methods to reduce the computational complexity [12]. Spatial methods [14,37] perform feature aggregation using the local neighbourhood of every node which is relatively simpler compared to spectral methods. The authors of GCN [13] provided a simplified method to compute the spectral graph convolution by computing its first-order approximation which can be considered as the most practical approach towards the problem. GCN has proved to be very effective in various domains like text classification [36], recommender systems [37], relational data modeling [38], and image recognition [39], along with various other scientific areas [40,41]. FASTGCN [42] is modification over GCN which attempts to reduce the training time of GCN. Graph attention network (GAT) [20] is enhancement over GCN which uses an additional attention layer to learn the importance of node neighbourhood during feature aggregation. Another direction of research on GCN was to efficiently design convolutions that can reach higher order neighbours [43,44]. Researchers developed different variants to GCN which can work with more complex settings like heterogeneous networks [45,46], signed networks [47], and hypergraphs [48].

All the works discussed above focus on static network where the nodes and edges do not change over time. Some works are already done on time-varying network embedding where most of them aimed at embeddings network snapshots that evolved with time. Deep Embedding Method for Dynamic Graphs (DynGEM) [17] used as stacked denoising autoencoder that can incrementally learn the temporal dynamics of a dynamic network. Tempnode2vec [49] generates PPMI matrices from individual network snapshots, factorizes the PPMI matrices, and optimizes a joint loss function to align the node embeddings and captures temporal stability. Dynnode2vec [50] extends the skip-gram architecture of node2vec so as to work with dynamic network snapshots. DyRep [51] considers both topological evolution and temporal interactions, and aims to develop embeddings which encode both structural and temporal information. EvolveGCN [18] extends GCN to dynamic networks by modelling the evolution of GCN parameters using a recurrent neural network. Combining graph convolution layers with LSTM layers [52] is another direction for generating dynamic node embeddings. A brief survey on modelling of dynamic networks using dynamic graph neural networks can be found on [53]. On the other hand, temporal network [54,55] whose edge connectivity varies over continuous time has been less studied from a network embedding perspective. One state-of-the-art work in this direction is continuous-time dynamic network embeddings (CTDNE) [19,56] which is a framework to adapt random walk and skip-gram-based approaches like deepwalk to temporal networks. CTDNE optimizes a skip-gram objective, so that the node that is closer in the temporal walk occupies closer when mapped to the vector space. In the proposed work, the concept of temporal random walk [56,57] is used to capture the temporal information of the network and positive pointwise mutual information (PPMI) [58] to compute the temporal proximity between vertex pairs.

Various approaches [59–61] were proposed to perform link prediction on dynamic networks. Recent studies [18, 19,62,63] show that the link prediction performance can be improved to a great extend using network embedding methods. A graph autoencoder [64] designed using GCN as the encoder is particularly effective in reconstructing the original network and thereby predicting the missing links. A summary of some related works is presented in Table 1. In the context of temporal networks discussed in this work, the link prediction task is to predict the links that may occur as edge streams at a later point of time.

Definitions and preliminaries

In the section, we provide the various definitions used in this work along with the problem definition of temporal network embedding. We also review the preliminary design of GCN and GAT.

Table 1 Summary of related works

Method	Network type	Input	Architecture used	Applications
Node2vec [28]	Static network	Edge list	Skip-gram with negative sampling	Node classification, link prediction
Chebyshev [12]	Static network	Adjacency matrix	Graph convolutions	Text classification
GCN [13]	Static network	Adjacency and feature matrix	Graph convolutions	Node classification
GraphSAGE [14]	Static network	Adjacency and feature matrix	Graph sampling and aggregation	Node classification
GAT [20]	Static network	Adjacency and feature matrix	Graph convolutions and attention	Node classification
hpGAT [44]	Static network	Adjacency and feature matrix	Graph convolutions, attention and matrix factorization	Node classification
Graph Autoencoder [64]	Static network	Adjacency and feature matrix	Deep autoencoder	Link prediction
DynGem [17]	Dynamic network represented as snapshots	Edge list	Stacked autoencoder	Link prediction
Dynnode2vec [50]	Dynamic network represented as snapshots	Edge list	Skip-gram	Node classification, link prediction
Tempnode2vec [49]	Dynamic network represented as snapshots	Adjacency matrix	Matrix factorization	Node classification, edge prediction
EvolveGCN [18]	Dynamic network represented as snapshots	Adjacency and feature matrix	Graph convolutions and LSTM	Link prediction
CTDNE [19,56]	Temporal network represented as edge streams	Edge list	Skip-gram with negative sampling	Link prediction

Definitions

Temporal network [54] It is a graph $G = (V, E_T, T)$, where V is a set of vertices, E_T is the set of time-stamped edges between vertices in V, and T is the set of possible time-steps. Each instance of the network can be represented as the contact between the vertices as a function of time: a set of triplets (v_i, v_j, t) where t is the time of interaction between vertex v_i and v_j, with $t \in T$. At the finest granularity level, each edge may be labeled with a distinct time-stamp. Also, there can be multiple edges between nodes representing the interaction between vertices at different time-stamps. For example, an email communication may occur between two people at different time-stamps.

Temporal walk [56]: A temporal walk exists from node v_i to v_j if there exist a stream of edges $E = (v_i, v_k, t_1), (v_k, v_l, t_2), \ldots, (v_n, v_j, t_n)$ from v_i to v_jm such that with $t_1 \leq t_2 \leq \cdots \leq t_n$. Informally if we have two edges (v_1, v_2, t_1) and (v_2, v_3, t_2), a temporal walk starts at v_1 can reach v_3 only of $t_1 \leq t_2$. We can say that two vertices u and v are temporally close if there exists a time-respecting walk from u to v. We may also define the temporal order proximity between any two vertices v_i to v_j as length of the minimal length temporal walk between the vertices v_i and v_j

Therefore, in the case of temporal networks, a walk sequence is valid only if the edges are traversed in the increasing order of interaction time. If each edge represents an interaction (e.g., phone call communication) between two objects, then a temporal random walk represents an optimal path for an information transfer through the temporal network. For example, suppose we have two phone calls $e_i = (v_1, v_2, t_1)$ from v_1 to v_2 at time t_1 and $e_j = (v_2, v_3, t_2)$ from v_2 to v_3 at time t_2; then if $t_1 \leq t_2$, the call $e_j = (v_2, v_3)$ may reflect the information received during the call $e_i = (v_1, v_2)$. However, if $t_1 > t_2$, the call $e_j = (v_2, v_3)$ cannot contain any information communicated between v_1 and v_2.

Temporal network embedding: Given a temporal network $G = (V, E_T, T)$, the task is to learn a transformation function $f : V_i \rightarrow K_i \in R^d$, where $d \ll |V|$, which map the nodes of the network to a low-dimensional space by preserving the network structure and the temporal order proximity of nodes in the network. Two nodes occupy closer in the vector space if they are topologically closer in the network and there exists a high temporal closeness between the nodes.

Graph convolutional network (GCN) [13]

Convolutional neural network is a widely used concept to learn good feature representations from data that can be represented in a euclidean space. Network are inherently irregular or non-Euclidean and the traditional convolution operation as applicable in images is not directly applicable to networks. The concept of graph convolution has been evolved from the field of graph signal processing where we classify the whole domain into spectral and spacial-based graph convolutions. Under spectral domain, the node features or attributes of the graph are considered as graph signals and the Chebyshev polynomials of the diagonal matrix of Laplacian eigenvalues are considered as the graph kernel. Spectral convolutions are defined as the product of a graph signal by a kernel. The kernel defined by the Chebyshev uses the sum of all Chebyshev polynomial kernels applied to the diagonal matrix of scaled Laplacian eigenvalues for first-order to k-order (largest order) neighbourhood. GCN can be considered as a first-order approximation of spectral graph convolution as described by Chebyshev. Given a graph represented as an adjacency matrix A with n number of nodes and k number of features, the goal of GCN is to learn a function from the network which takes as input: i) an n x k feature matrix H; ii) an adjacency matrix A, and to produce an output Z which is an n x d matrix, where d is the number of dimensions per node. GCN uses the layer-wise propagation rule:

$$H_{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H_{(l)}W_{(l)}), \tag{1}$$

where W_l denote the weight matrix of l^{th} network, $\hat{A} = A+I$ and D is the diagonal node degree matrix of $\hat{A}$. Initially, the node feature matrix F can be considered as the embedding matrix, i.e., $H_0 = F$. At each convolutional layer l, the embedding matrix gets updated by following three steps which include a feaure propagation, a linear transformation, and a non-linear activation.

During feature propagation, the new features of each node become the sum of the features from the node's first-order neighbourhoods. This is followed by multiplying the result with the weight matrix which can linearly transform the feature representations to another latent space. The last step is to apply a non-linear activation function. The results of the convolutions may be fed into a softmax layer and weights can be learned using application-specific tasks like link prediction or node classification.

Graph attention network (GAT) [20]

GCN assumes equal importance to all the nodes in the same neighbourhood and, therefore, have limitations in model capacity. A GAT is designed to be a more robust model which uses an attention mechanism to assign different weights to nodes in the same neighbourhood. Moreover, the attention weights can be learned using end-to-end training which can improve the model performance. In addition to the steps followed in GCN, the core of the GAT is an attention layer which learns self-attention weights that indicate the importance of node j features to node i. GAT also takes as input an n x k feature matrix represented as F, an adjacency matrix A, and produces Z, an n x d embedding matrix. Like GCN, the initial step is to apply a linear transformation parameterized by a weight matrix w over all nodes to generate a high-level latent representation. The next step is to perform a self-attention, parameterized by a weight vector $\overrightarrow{a}$ over all nodes and learns an attention coefficient between every pair of vertices. Given the feature matrix $H = \{h_1, h_2, ...h_n\}$ (initially $H = F$) where each row h_i represents the feature vector of node i, the attention coefficient E_{ij} between every node pair i and j can be computed as:

$$E_{ij} = \overrightarrow{a}(Wh_i, Wh_j). \tag{2}$$

GAT only considers E_{ij} for those nodes $j \in N_i$, where N_i represent the first-order neighbourhood of node i in the graph. GAT uses a leaky relu function to introduce some non-linearity and a softmax function to normalize the attention coefficients. The coefficients thus computed can be represented as:

$$E_{ij} = \frac{exp(leakyRELU(\overrightarrow{a^T}(Wh_i||Wh_j))}{\sum_{l\in N_i} exp(leakyRELU(\overrightarrow{a^T}(Wh_i||Wh_l))}, \tag{3}$$

where $||$ represent the concat operation. The matrix E_{ij} constitutes the attention matrix. Furthermore, the link information in the adjacency matrix is replaced by the learned attention coefficients from the attention matrix and the features from the first-order neighbourhood are aggregated. A typical feature aggregation at node i can be represented as:

$$h_i' = \sigma\left(\sum_{j\in N_i} E_{ij}Wh_j\right). \tag{4}$$

Methodology

In this section, we discuss a methodology to incorporate temporal information to a graph convolutional network so as to generate time-aware node embeddings from a temporal network.

Extracting temporal proximity information

The first step of the process is to extract the temporal neighbourhood information of every node in the network. Here, we

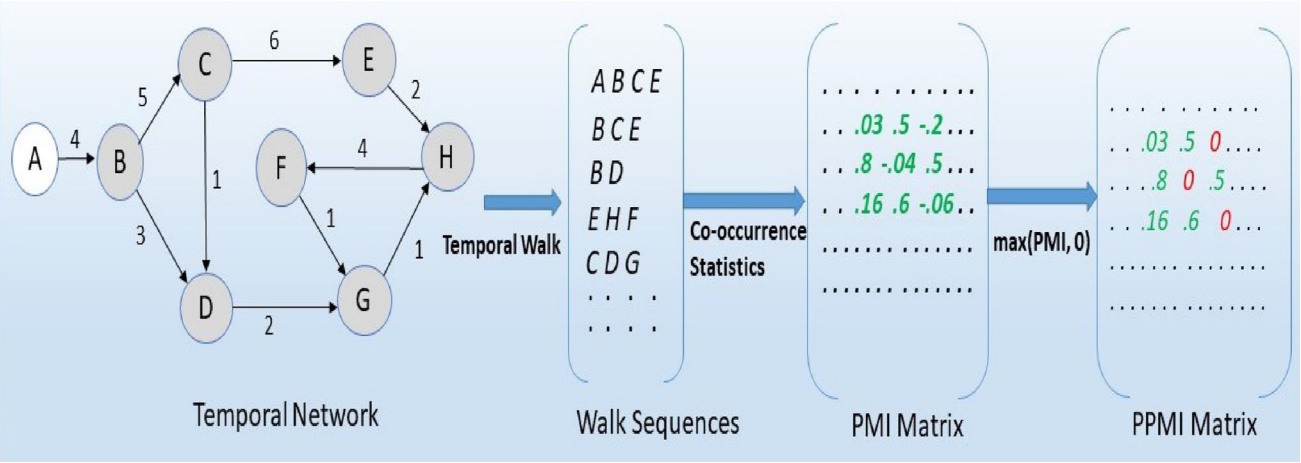

Fig. 3 Flow diagram for PPMI matrix generation

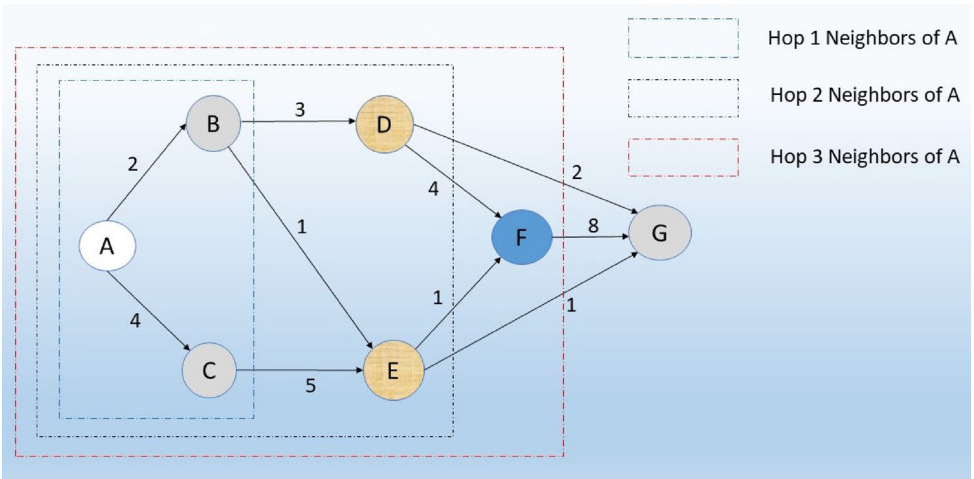

Fig. 4 Temporal Hops w.r.t. node A

propose a temporal walk and PMMI-based method to efficiently represent the temporal neighbourhood of each node. A flow diagram of the method is shown in Fig. 3. We perform a temporal random walk starting at various nodes on the network to generate walk sequences. Furthermore, we compute the co-occurrence statistics of every pair of nodes to generate the pointwise mutual information(PMI) matrix. The negative entries of PMI matrix are set to zero to form positive pointwise mutual information (PPMI) matrix.

Computing the temporal co-occurrence information between the nodes from a temporal network with large number of discrete time-steps is computationally intensive. Here, we adopt a sampling-based technique where we approximate the co-occurrence information using the walk sequences generated from a temporal random walk. A temporal walk sequence is a sequences of nodes generated from a temporal walk where l is the length of the random walk. The walk is similar to that of the truncated random walk as defined by [27], but it follows the temporal ordering as given in the definition of temporal walk.The length of the walk is a hyper-parameter which can influence order to which the temporal neighbourhood of each node is captured. For example, from Fig. 4, it can be observed that walk length l determines temporal neighbourhood of node A.

To generate walk sequences, we follow the sampling strategy as discussed in [56]. A sampling distribution based on time-steps is precomputed from the graph and is used for initial edge selection. We use a linear sampling distribution where the probability of selecting edge $e \in E_T$ as initial edge can be computed as:

$$P(e) = \frac{\gamma(e)}{\sum_{k \in E_T} \gamma(k)}; \tag{5}$$

γ is a function that sorts the edges in the ascending order of time and maps each edge to an index with $\gamma(e) = 1$ for the earliest edge e. After initial edge selection, the next step is to select a temporal neighour which will be the next node in the random walk. Here, also, we use a linear sampling distribution to provide temporal bias to the next node selection,

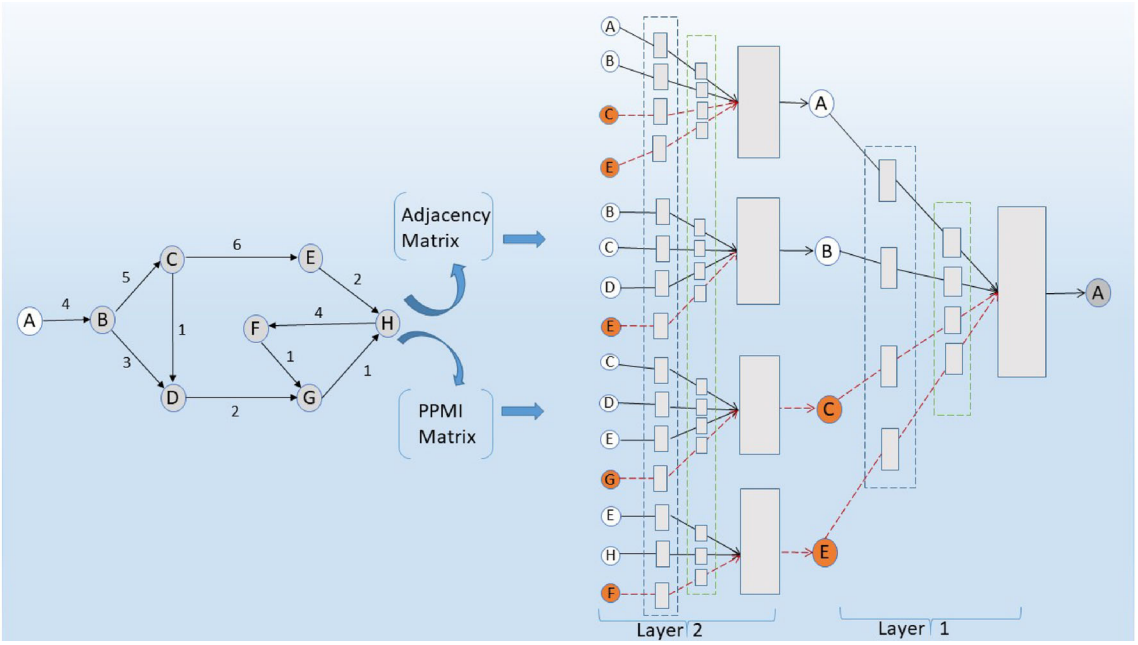

Fig. 5 A two-layer TempGAN architecture

so that walks that exhibit smaller in-between time for consecutive edges will get more bias. If β is a function that sorts the temporal neighbours in the decreasing order of time, the probability of selecting the temporal neighbour $n \in T_t(u)$ as the next node of node u in time t in the walk can be represented as:

$$P(n) = \frac{\beta(n)}{\sum_{l \in T_t(u)} \beta(l)}. \tag{6}$$

Once the walk sequences are generated, the next step is to compute point wise mutual information between the nodes which can be approximated as the co-occurrence statistics of nodes. The PMI is computed as:

$$\text{PMI}(v_i, v_j) = \log\left(\frac{P(v_i, v_j)}{P(v_i)P(v_j)}\right). \tag{7}$$

The negative entries in the PMI matrix are replaced by zero to form the PPMI matrix:

$$\text{PPMI}(v_i, v_j) = \max(PMI(v_i, v_j), 0). \tag{8}$$

Each entry in the PMMI matrix can be used to measure the temporal co-relation between the vertex pairs v_i and v_j. The value will be high when there exist more time-respecting paths between v_i and v_j and will be low if they co-occur very few times in a temporal random walk.

TempGAN

The proposed neural architecture (TempGAN) takes as input, the adjacency matrix A, the node feature matrix F, and PMMI matrix M, and generates the embedding matrix E. A two-layer TempGAN is shown in Fig. 5. First, we discuss the theoretical intuition followed by an example for a better understanding. Like GAT, TempGAN also follows two mechanisms, convolution and attention at each hidden layer of the neural network. Given an initial n x k feature matrix H ($H_0 = F$), where n is the number of nodes and k is the number of features at each node, the first step is to apply a linear transformation parameterised by weight W to generate high-level features, which can be represented as:

$$H' = (WH). \tag{9}$$

The next step is to apply self-attention over the nodes parameterized by a shared attention weight $\overrightarrow{a}$ that can compute the attention coefficient matrix E as:

$$E = (\overrightarrow{a}\, H'). \tag{10}$$

TempGAN also uses a leaky relu function to provide non-linearity and a softmax function to normalize the attention coefficients:

$$E_{ij} = softmax_j(leakyRelu(E_{ij})). \tag{11}$$

Each entry E_{ij} of E contains the attention coefficient w.r.t. every pair of vertices. We need to consider the E_{ij} of nodes

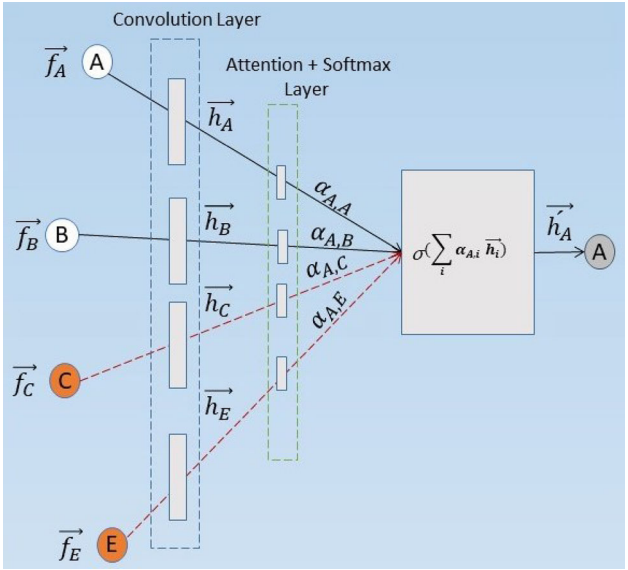

Fig. 6 Operation of tempGAN w.r.t. node A

$j \in T_i$, where T_i is the temporal neighbourhood of nodes i in the graph. The temporal neighbourhood of each node can be inferred from the PPMI matrix M. We can redefine attention matrix as:

$$\hat{E_{ij}} = \begin{cases} E_{ij}, & \text{if } M_{ij} + A_{i,j} > 0 \\ 0, & \text{otherwise;} \end{cases} \tag{12}$$

i.e., for every node i, we need to consider the attention coefficient for those nodes which are in the temporal neighbourhood of i for further propagation and aggregation process. Finally, the propagation step can be represented as:

$$H_l = \sigma(\hat{E} W H_{l-1}). \tag{13}$$

For each node i, the model propagates the transformed features f from the temporal neighbourhood T_i to i, and the learned attention weights help to differentiate the temporal neighbours based on their importance in connectivity, i.e., a distant temporal neighbour will be given lesser importance during aggregation process which will help to build a more robust model.

The operations of tempGAN across two layers can be explained using Fig. 5. The nodes which are coloured denote temporal connections (other than first-order neighbours) to the source, which can be inferred from the PPMI matrix. The nodes within the temporal proximity of A are B, C, and E. Therefore, the features of the nodes A(self), B, C, and E are used in the convolution and attention process, and are used in generating the latent representation of node A. Similarly, in layer 2, for generating representations for nodes A, B, C, and E, the features of the nodes which are in their temporal proximity are used. For better understanding, the convolution and attention operations done at node A are shown in detail in Fig. 6. The features of the nodes A, B, C, and E are first fed into a linear transformation layer and the latent representations are learned. Further, they are passed to an attention and softmax layer to learn the attention coefficients. Finally, the transformed features from the temporal neighbours A, B C, and E parameterized by the attention coefficients are aggregated to learn the latent representation of node A. The same process happens for all the nodes in the graph.

Application

Various network mining problems are used in the literature to evaluate the quality of the embeddings generated using representation learning methods. In this work, we use the link prediction problem [66] of temporal networks as the benchmark application. The task is to predict the possibility of link existence between nodes at future time intervals, given the existing links between nodes at known time intervals. We follow a variational graph autoencoder architecture to conduct experiments with link prediction problem. We use TempGAN architecture as the encoding layers and a simple inner product operator as the decoding layer. The flow diagram for TempGAN autoencoder architecture for link prediction is shown in Fig. 7, and the pseudocode of implementation is shown as Algorithm 1. Here, we use a two-layer TempGAN which takes the temporal network as input and generates the mean and log variance w.r.t. every node. The distribution thus generated will be close to $N(0, 1)$. A random sample embedding Z can be generated form distribution using reparameterization trick which can be represented as:

$$Z = \mu + \sigma * \epsilon, \tag{14}$$

where $\epsilon \sim N(0, 1)$. Furthermore, we can reconstruct the graph information using an inner product decoder which is represented as:

$$\hat{A} = \sigma(ZZ^T), \tag{15}$$

where σ is the logistic sigmoid function.

Experimental setup

In this section, we demonstrate the effectiveness of the proposed system by conducting link prediction experiments on real-world temporal network datasets. The area under receiver-operating characteristic curve (AUC) and average precision (AP) are the measures used for evaluation. The results are compared with that of the baseline methods. All experiments are conducted using a machine with Ubuntu

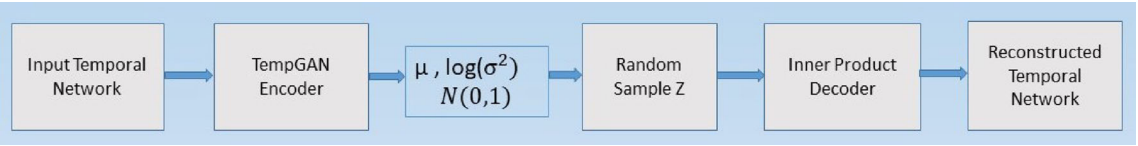

Fig. 7 TempGAN autoencoder for Link Prediction

18.04 operating system, 16 GB RAM, hexa-core processor with 3.2 GHz speed, and Geforce GTX 1050 Ti GPU. We used python packages for system implementation, which include Networkx for graph processing, Pytorch, and Scikit-learn for building the machine learning modules.

Datasets and evaluation

The temporal network datasets used in experiments are listed below. The datasets are collected from Koblenz Network Collection [65].

IA-Contacts-hypertext 2009 (hypertext) It is a temporal network which represents the face-to-face proximity between people during ACM hypertext 2009 conference. Nodes represent the attendees and the time-stamped edges represent the interaction between the people over a period of 2.5 days. It contains 113 nodes and 20.8k time-stamped edges.

IA-Enron-Employees (enron) It is an email communication network between employees of enron Inc. It contains 151 nodes and 50.5k time-stamped edges over a period of 1137 days.

FB Forum (FB) This is the data collected from an online student community where the nodes represent the students and the time-stamped edges represent the messages posted between them at a particular time-step. It contains 899 nodes and 33.7k time-stamped edges over a period of 164 days.

IA-Radoslaw-Email (radoslaw) It represents an email communication network of a manufacturing company where nodes represent employees and edges between them are email communications. The graph consists of 167 vertices and 82.9k edges over a period of 271 days.

A statistics of various datasets used is shown in Table 2.

The evaluation measures used to compare the performance of the proposed system with baselines are

AUC AUC is a widely used evaluation metric for link prediction. This metric can be interpreted as the probability that a randomly chosen missing link is given a higher score than a randomly chosen non-existent link, provided the rank of all the non-observed links. Among n independent comparisons, if there are n' times the missing link having a higher score and n'' times they have the same score, AUC score is calculated as:

$$AUC = \frac{n' + 0.5n''}{n}. \tag{16}$$

Average precision (AP) It estimates the precision of every prediction and computes the average over all precisions. It is calculated as:

$$AP = \Sigma_n (R_n - R_{n-1}) P_n, \tag{17}$$

where P_n and R_n are the precision and recall at the n^{th} threshold.

Baseline methods

A quick introduction to specific baselines is listed below:

Node2vec [28] Node2vec performs random walks on static network to generate node sequences and uses skip gram with negative sampling to generate node representations. Node2vec performs a biased random walk which provides more flexibility in exploring node neighbourhoods. The learned representations can be used for link prediction using vector-based similarity measures.

Graph convolutional network (GCN) [13] GCN is a variant of graph neural network which applies a graph convolution at each node to perform the propagation and aggregation of node features from neighbouring nodes. As GCN has been defined for only static networks, we consider the graph to be static for conducting experiments.

Graph attention network (GAT) [20] GAT is an enhancement over GCN. In addition to convolution and feature aggregation, GAT uses an attention layer to learn self-attention weights that indicate the importance of node j features to node i. As GAT has been also defined for only static networks, we consider the graph to be static for conducting experiments.

Continuous-time dynamic network embeddings (CTDNE) [19] This work first performs truncated time-respecting random walks over the temporal networks to generate temporal path sequences. Furthermore, a skip-gram objective is trained to generate node embeddings. The learned representations are used in predicting missing links.

Result and analysis

To evaluate the quality of the embeddings, we perform link prediction using TempGAN autoencoder (TempGAN-AE) architecture which is learned by end-to-end training. To

Algorithm 1: TempGAN Autoencoder

Input: (Un)directed temporal network $G = (V, E_T, T)$, which has adjacency matrix A and node feature matrix F (initally random)
Output: Reconstructed Adjacency Matrix $\hat{A}$

Initialize corpus=[ ], walk=[ ], walk length l, walk count C, Weight matrices W, Attention weight vectors $\overrightarrow{a}$, Number of TempGAN hidden layers L, Number of epochs e_p
for *c from* 1 *to* C **do**
  Sample edge (u, v) from E_T using equation 5
  $t = T(e)$
  $w = tempwalk(G, (u, v), t, l)$
  corpus.append(w)
end
PPMI Matrix M= PPMI(corpus)
$\hat{A}$=TempGANauto($A, M, F, W, \overrightarrow{a}, L, e_p$)
Procedure `tempwalk` $(G, (u, v), t, l)$
  walk= $[u, v]$
  Set $j = v$
  for *1 to* $l - 1$ **do**
    $T_t(j) = \{ w | e = (j, w, t') \in E_T \wedge T(j) > t \}$
    select node k from $T_t(j)$ using equation 6
    walk.append(k)
    $t = T(j, k)$; $j = k$
  end
  return walk
Procedure *PPMI(corpus)*
  for *each walk in corpus* **do**
    for *each u in walk* **do**
      $P(u) += 1$
      $R.append(u)$
      for *each v in walk* $- R$ **do**
        $P(u, v) += 1$
      end
    end
  end
  for *each* (u, v) *pair* **do**
    $PPMIu, v = max(\log(\frac{P(u,v)}{P(u)P(v)}), 0)$
  end
  return $PPMI$
Procedure *TempGANauto* ($A, M, H, W, \overrightarrow{a}, L, e_p$)
  $H_0 = F$
  for 1 *to* e_p **do**
    for l *from* 1 *to* L **do**
      ▷ TempGAN Encoder
      $E_{ij} = (\overrightarrow{a} W H_{l-1})$
      $E_{ij} = softmax_j(leakyRelu(E_{ij}))$
      $\hat{E_{ij}} = \begin{cases} E_{ij}, & \text{if } M_{ij} + A_{ij} > 0 \\ 0, & \text{otherwise} \end{cases}$
      $H_l = Relu(\hat{E} W H_{l-1})$
    end
    $Z = reparameterize(H_L)$
    $\hat{A} = \sigma(ZZ^T)$ ▷ Inner Product Decoder
    Objective Function L= cross entropy loss(A, $\hat{A}$)
    stochastic gradient descent(L)
  end
  return $\hat{A}$

Table 2 Statistics of various datasets used

Dataset	# of nodes	# of edges	Node average degree	Duration (days)
Hypertext	113	20.8K	368.5	2.5
Enron	150	50.5K	669.8	1137
FB	899	33.7K	75	164
Radoslaw	167	82.9K	993.1	271

test link prediction performance with TempGAN-AE, we hide 10–15 % of temporal links of the original network, generate node embeddings using TempGAN encoder, and reconstruct the original network using the inner product decoder. The whole network is trained using stochastic gradient decent (SGD). Experiment with GCN is conducted using the same procedure without considering temporal information of the links. To test the link prediction performance using Node2vec, the procedure is as follows. Hide 10–15% of the links to form the training set, generate node embeddings from the training set, use Hadamard product of the node embedding to form the edge embedding, and build a classifier based on positive and negative edges. Hidden edges are used to test the accuracy of the classifier. To test link prediction performance with CTDNE, we hide 10–15 % of temporal links from time 1 to 't-1', generate embeddings, and predict the links at time 't'. While training the classifier, existing edges are considered as positive samples and the disconnected edges are considered as negative samples. Now, we present the analysis and comparison of results obtained from conducting link prediction experiments on four real-world networks.

Performance

First, we discuss the various parameter settings used by the proposed system which gained optimum performance. We set the embedding dimension d = [128, 256, 128, 128] as optimum for hypertext, FB, enron, and radoslaw datasets, respectively. We set two hidden layers in TempGAN with sizes ENRON = [256, 128], FB = [512, 256], enron = [256, 128], and radoslaw = [256, 128]. The temporal random walk length is set as l=[6, 8, 6, 4] for hypertext, FB, enron, and radoslaw datasets, respectively. Other hyperparameters include dropout=[0.5, 0.4, 0.5, 0.4], epochs =[600, 750, 500, 600], initial learning rate=.005, and alpha for leaky relu= 0.1 for hypertext, FB, enron, and radoslaw datasets, respectively. The neuron activations are done using relu function. The training is done using zero grad optimizer with binary cross-entropy as the loss function. For GCN, we use the same parameter settings as that of the proposed system. For node2vec and CTDNE, we set walk length = 40, negative samples = 5, and context window size = 10. SVM classifier is used to predict the positive and negative links.

Now, we compare the performance of TempGAN with three different static network embedding methods (Node2vec, GCN, and GAT) along with one temporal network embedding method (CTDNE). The performance improvement of TempGAN for link prediction over the baseline methods is shown in Fig. 8 and Table 2. Figure 8 depicts the AUC comparison and it can be found that, for hypertext dataset, the proposed system gains a performance improvement of 18.7%, 15.1%, 15.1 %, and 13.4%, and for enron dataset, the proposed system gains a performance improvement of 15.0%, 10.5%, 6.3%, and 7.6% over node2vec, GCN, GAT, and CTDNE, respectively. Similarly, an AUC improvement of 11.8%, 6.2%, 3.6%, and 4.9% is obtained against the baselines for radoslaw dataset. For FB dataset, the proposed system gains an improvement of 2.5% and 6.5% and 2.5% over node2vec, GCN, and GAT, and getting a comparable performance when compared to CTDNE. We observe that the graph convolution-based methods have an advantage for networks that have a higher average node degree (dense network), whereas a truncated random walk-based strategy like CTDNE is more useful in the case of sparse networks like FB. Table 3 provides the average precision comparison of the proposed system with the baselines, which also proves the advantage of proposed system over state-of-the-art network embedding methods. Furthermore, we show the effect of attention mechanism used in the proposed system by conducting experiments with and without attention, and the results are shown in Table 4. Results show that the learned attention weights can further improve the quality of node embeddings and thereby improve the AUC score of link prediction task.

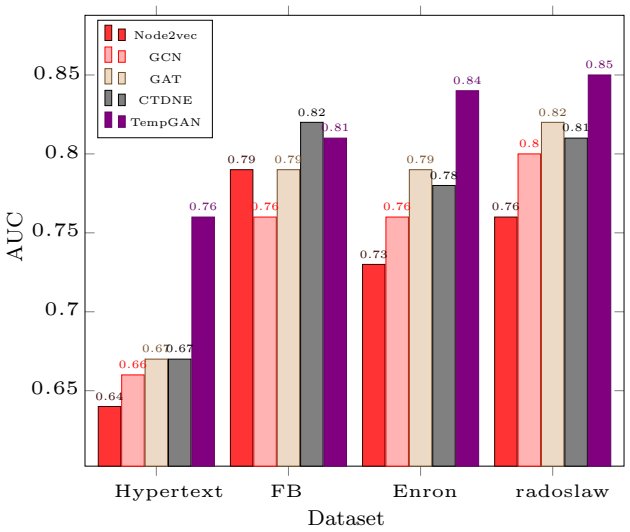

Fig. 8 AUC comparison of proposed system with baselines

Table 3 AP comparison of proposed system with baselines

Method	Hypertext	FB	Enron	Radoslaw
Node2vec	65.5	79.4	73.9	75.3
GCN	67.1	74.2	74.9	80.4
GAT	68.4	75.4	74.2	81.8
CTDNE	70.1	81.2	76.4	82.8
TempGAN	79.2	80.0	83.1	84.4

Parameter analysis

In this section, we analyse the effect of various parameter settings used in the experiments which include the length of the temporal walk, number of hidden layers for the TempGAN architecture, and the dimensionality of the node embeddings used in TempGAN autoencoder. We also show the variations in reconstruction loss and AUC at different epochs of network training along with the training time required to complete one epoch of training.

Length of the temporal walk

This parameter can decide the number of hops that a valid temporal walk can cover and therefore is an important parameter which influence the extent to which the temporal information is considered for network embedding. A very low value for walk length l only allows feature aggregation from each nodes very close temporal neighbours where larger values for l allow us to consider features from more distant temporal neighbours. The effect of l on various datasets in shown in Fig 9. For dense networks like enron and radoslaw, an optimum AUC score is obtained while considering features from four hop temporal neighbours. For FB dataset, a walk length covering eight hops provided optimum performance. Setting high values for l will introduce noise which may reduce system performance. We can conclude that the optimum value for l for each dataset depends upon connectivity patterns of the network.

Table 4 Effect of attention mechanism in the proposed system

Method	Hypertext		FB		Enron		Radoslaw	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP
Proposed system without attention	0.743	72.1	0.792	78.2	0.813	80.4	0.829	80.5
TempGAN	0.762	79.2	0.819	80.0	0.842	83.1	0.856	84.4

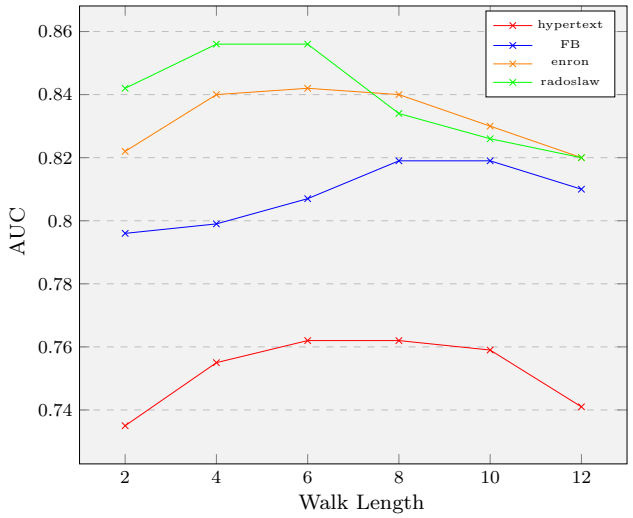

Fig. 9 Effect of walk length

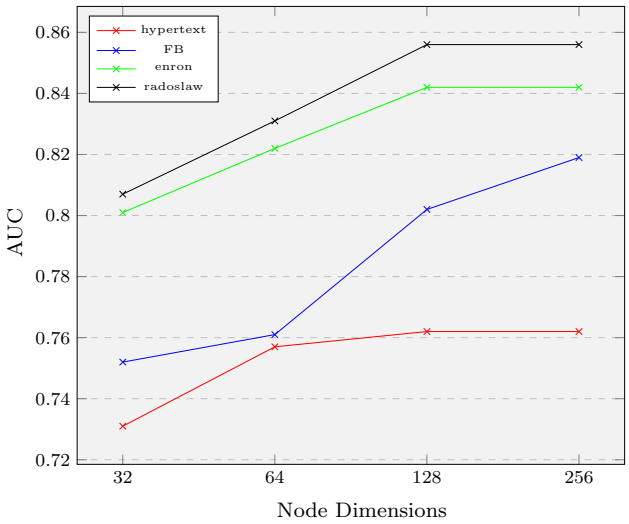

Fig. 10 Effect of dimensionality on AUC

Table 5 Effect of TempGAN hidden layers on AUC

No. of hidden layers	Hypertext	FB	Enron	Radoslaw
1	0.732	0.789	0.835	0.823
2	0.762	0.819	0.842	0.856
3	0.760	0.815	0.842	0.851

Parameters of the neural network

The effect of number of hidden layers for TempGAN architecture is shown in Table 5. We obtained the optimum values for AUC while conducting experiments with two hidden layer TempGAN. Increasing the hidden convolution and attention layers beyond two does not provide any improvements in the results. The embedding dimension is a parameter which can be tuned according to the number of nodes in the input graph. The effect of dimensionality d on AUC w.r.t. various datasets in shown in Fig 10. For hypertext, enron, and radoslaw, the optimum performance is obtained when d=128, and for FB, the value d=256 provided best AUC values.

The reduction in reconstruction loss with the increase in the number of epochs is shown in Fig. 11 and the improvement in roc score during learning is shown in Fig. 12. The training time required to complete one epoch for each dataset is shown in Table 6.

Conclusion

Embedding nodes of a network in vector space by preserving its structural properties is a challenging research problem. Among various network embedding methods, graph convolution-based approaches gained more popularity because of its simplicity and effectiveness. In this work,

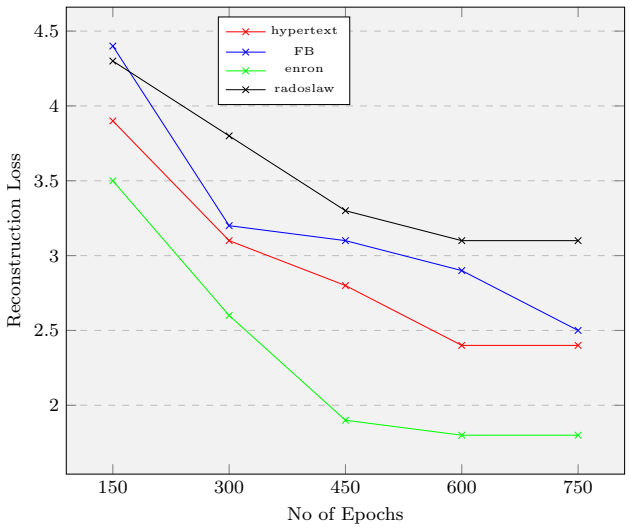

Fig. 11 Effect of epochs size on reconstruction loss

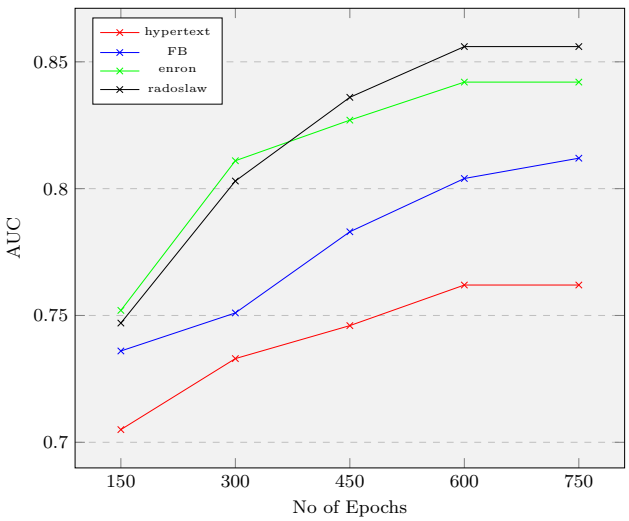

Fig. 12 Effect of epochs size on AUC

Table 6 Average training time per epoch (s)

No. of hidden layers	Hypertext	FB	Enron	Radoslaw
1	0.105	0.982	0.087	0.147
2	0.178	1.561	0.140	0.451

we address the problem of temporal network embedding which aims to map the nodes of a network to vector space by preserving the temporal information. We aim to extend the concept of graph convolution and attention to temporal network data so as to generate time-aware node embeddings. We propose an neural architecture which uses both link and temporal information of the network to generate node embeddings which can be used in many network mining tasks that require end-to-end training. We design a graph autoencoder based on the proposed architecture which performs link prediction on temporal networks. We conducted experiments with real-world temporal networks and compared the results with state-of-the-art methods.

In future, we aim to extend the temporal network embedding to more complex systems like epidemic networks which use SIS (susceptible-infected-susceptible) and SIR (susceptible-infected-recovered) modeling. The proposed approach can also be applied to other network mining tasks like node classification and anomaly detection. Extending the work to more complex settings like heterogeneous and signed networks can be another interesting direction for future work.

Funding No funds, grants, or other financial support was received.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Bronstein MM, Bruna J, LeCun Y, Szlam A, Vandergheynst P (2017) Geometric deep learning: going beyond Euclidean data. IEEE Signal Process Mag 34(4):18–42
2. Musto C, Basile P, Lops P, de Gemmis M, Semeraro G (2017) Introducing linked open data in graph-based recommender systems. Inf Process Manag 53(2):405–435
3. Mason W, Vaughan JW, Wallach H(2014) Special issue: Computational social science and social computing, Mach Learn 96:257–469
4. Yadav CS, Sharan A, Joshi ML (2014) Semantic graph based approach for text mining. In: 2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT). IEEE, pp 596–601
5. Bhadoria RS, Chaudhari NS, Samanta S (2018) Uncertainty in sensor data acquisition for SOA system. Neural Comput Appl 30(10):3177–3187
6. Srivastav MK, Bhadoria RS, Pramanik T (2020) Integration of multiple cache server scheme for user-based fuzzy logic in content delivery networks. In: Handbook of research on advanced applications of graph theory in modern society. IGI Global, pp 386–396
7. Ma'ayan A (2011) Introduction to network analysis in systems biology. Sci Signaling 4(190):tr5
8. Hamilton WL, Ying R, Leskovec J (2017) Representation learning on graphs: Methods and applications. arXiv:1709.05584
9. Cui P, Wang X, Pei J, Zhu W (2018) A survey on network embedding. IEEE Trans Knowl Data Eng 31(5):833–852
10. Goyal P, Ferrara E (2018) Graph embedding techniques, applications, and performance: a survey. Knowl Based Syst 151:78–94
11. Cai H, Zheng VW, Chang KC-C (2018) A comprehensive survey of graph embedding: problems, techniques, and applications. IEEE Trans Knowl Data Eng 30(9):1616–1637
12. Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in neural information processing systems, pp 3844–3852
13. Kipf TN, Welling M (2016) Semi-supervised classification with graph convolutional networks. arXiv:1609.02907
14. Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. arXiv:1706.02216.
15. Zhang S, Tong H, Xu J, Maciejewski R (2019) Graph convolutional networks: a comprehensive review. Comput Soc Netw 6(1):11
16. Zhu L, Guo D, Yin J, Ver Steeg G, Galstyan A (2016) Scalable temporal latent space inference for link prediction in dynamic social networks. IEEE Trans Knowl Data Eng 28(10):2765–2777
17. Goyal P, Kamra N, He X, Liu Y (2018) Dyngem: deep embedding method for dynamic graphs. arXiv:1805.11273
18. Pareja A, Domeniconi G, Chen J, Ma T, Suzumura T, Kanezashi H, Kaler T, Leisersen CE (2019) Evolvegcn: evolving graph convolutional networks for dynamic graphs. arXiv:1902.10191
19. Nguyen GH, Lee JB, Rossi RA, Ahmed NK, Koh E, Kim S (2018) Continuous-time dynamic network embeddings. Companion Proc Web Conf 2018:969–976
20. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2017) Graph attention networks. arXiv:1710.10903
21. Deng L, Hinton G, Kingsbury B (2013) New types of deep neural network learning for speech recognition and related applications: An overview. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, pp 8599–8603
22. Nassif AB, Shahin I, Attili I, Azzeh M, Shaalan K (2019) Speech recognition using deep neural networks: a systematic review. IEEE Access 7:19143–19165
23. Ramík DM, Sabourin C, Moreno R, Madani K (2014) A machine learning based intelligent vision system for autonomous object detection and recognition. Appl Intell 40(2):358–375
24. Praneel AV, Rao TS, Murty MR (2020) A survey on accelerating the classifier training using various boosting schemes within cascades of boosted ensembles. In: Intelligent Manufacturing and Energy Sustainability. Springer, pp 809–825

25. Khan A, Baharudin B, Lee LH, Khan K (2010) A review of machine learning algorithms for text-documents classification. J Adv Inf Technol 1(1):4–20
26. Allahyari M, Pouriyeh S, Assefi M, Safaei S, Trippe ED, Gutierrez JB, Kochut K (2017) A brief survey of text mining: Classification, clustering and extraction techniques. arXiv:1707.02919
27. Perozzi B, Al-Rfou R, Skiena S (2014) Deepwalk: online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp 701–710
28. Grover A, Leskovec J (2016) node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pp 855–864
29. Wang D, Cui P, Zhu W (2016) Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pp 1225–1234
30. Cao S, Lu W, Xu Q (2016) Deep neural networks for learning graph representations. In: Thirtieth AAAI conference on artificial intelligence
31. Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY (2020) A comprehensive survey on graph neural networks. IEEE Trans Neural Netw Learn Syst 32(1):4–24
32. Shuman DI, Narang SK, Frossard P, Ortega A, Vandergheynst P (2013) The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. IEEE Signal Process Mag 30(3):83–98
33. Bruna J, Zaremba W, Szlam A, LeCun Y (2013) Spectral networks and locally connected networks on graphs. arXiv:1312.6203
34. Henaff M, Bruna J, LeCun Y (2015) Deep convolutional networks on graph-structured data. arXiv:1506.05163
35. Hammond DK, Vandergheynst P, Gribonval R (2011) Wavelets on graphs via spectral graph theory. Appl Comput Harmon Anal 30(2):129–150
36. Yao L, Mao C, Luo Y (2019) Graph convolutional networks for text classification, Proceedings of the AAAI Conference on. Artif Intell 33:7370–7377
37. Ying R, He R, Chen K, Eksombatchai P, Hamilton WL, Leskovec J (2018) Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp 974–983
38. Schlichtkrull M, Kipf TN, Bloem P, Van Den Berg R, Titov I, Welling M (2018) Modeling relational data with graph convolutional networks. In: European Semantic Web Conference. Springer, pp 593–607
39. Chen Z-M, Wei X-S, Wang P, Guo Y (2019) Multi-label image recognition with graph convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 5177–5186
40. Zitnik M, Agrawal M, Leskovec J (2018) Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics 34(13):i457–i466
41. Sun M, Zhao S, Gilvary C, Elemento O, Zhou J, Wang F (2020) Graph convolutional networks for computational drug development and discovery. Brief Bioinform 21(3):919–935
42. Chen J, Ma T, Xiao C (2018) Fastgcn: fast learning with graph convolutional networks via importance sampling. arXiv:1801.10247
43. Yang C, Sun M, Liu Z, Tu C (2017) Fast network embedding enhancement via high order proximity approximation. In: IJCAI, pp 3894–3900
44. Liu Z, Liu W, Chen P-Y, Zhuang C, Song C (2019) hpgat: high-order proximity informed graph attention network. IEEE Access 7:123002–123012
45. Wang X, Ji H, Shi C, Wang B, Ye Y, Cui P, Yu PS (2019) Heterogeneous graph attention network. In: The World Wide Web Conference, pp 2022–2032
46. Yun S, Jeong M, Kim R, Kang J, Kim HJ (2019) Graph transformer networks. In: Advances in neural information processing systems, pp 11983–11993
47. Huang J, Shen H, Hou L, Cheng X (2019) Signed graph attention networks, in: International conference on artificial neural networks. Springer, pp 566–577
48. Yadati N, Nimishakavi M, Yadav P, Nitin V, Louis A, Talukdar P (2019) Hypergcn: A new method for training graph convolutional networks on hypergraphs. In: Advances in neural information processing systems, pp 1511–1522
49. Haddad M, Bothorel C, Lenca P, Bedart D (2019) Temporalnode2vec: Temporal node embedding in temporal networks. In: International conference on complex networks and their applications. Springer, pp 891–902
50. Mahdavi S, Khoshraftar S, An A (2018) dynnode2vec: scalable dynamic network embedding. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE, pp 3762–3765
51. Trivedi R, Farajtabar M, Biswal P, Zha H (2019) Dyrep: learning representations over dynamic graphs. In: International Conference on Learning Representations
52. Manessi F, Rozza A, Manzo M (2020) Dynamic graph convolutional networks. Pattern Recogn 97:107000
53. Skarding J, Gabrys B, Musial K (2020) Foundations and modelling of dynamic networks using dynamic graph neural networks: a survey. arXiv:2005.07496
54. Holme P, Saramäki J (2012) Temporal networks. Phys Rep 519(3):97–125
55. Li A, Cornelius SP, Liu Y-Y, Wang L, Barabási A-L (2017) The fundamental advantages of temporal networks. Science 358(6366):1042–1046
56. Nguyen GH, Lee JB, Rossi RA, Ahmed NK, Koh E, Kim S (2018) Dynamic network embeddings: From random walks to temporal random walks. In: 2018 IEEE International Conference on Big Data (Big Data). IEEE, pp 1085–1092
57. Starnini M, Baronchelli A, Barrat A, Pastor-Satorras R (2012) Random walks on temporal networks. Phys Rev E 85(5):056115
58. Levy O, Goldberg Y (2014) Neural word embedding as implicit matrix factorization. In: Advances in neural information processing systems, pp 2177–2185
59. Ma X, Sun P, Qin G (2017) Nonnegative matrix factorization algorithms for link prediction in temporal networks using graph communicability. Pattern Recogn 71:361–374
60. Ahmed NM, Chen L, Wang Y, Li B, Li Y, Liu W (2018) Deepeye: link prediction in dynamic networks based on non-negative matrix factorization. Big Data Min Anal 1(1):19–33
61. Yasami Y, Safaei F (2018) A novel multilayer model for missing link prediction and future link forecasting in dynamic complex networks. Phys A 492:2166–2197
62. Zhang M, Chen Y (2018) Link prediction based on graph neural networks. In: Advances in neural information processing systems, pp 5165–5175
63. Li T, Zhang J, Philip SY, Zhang Y, Yan Y (2018) Deep dynamic network embedding for link prediction. IEEE Access 6:29219–29230
64. Kipf TN, Welling M (2016) Variational graph auto-encoders. arXiv:1611.07308

65. Kunegis J (2013) Konect: the koblenz network collection. In: Proceedings of the 22nd International Conference on World Wide Web, pp 1343–1350
66. Liben-Nowell D, Kleinberg J (2007) The link-prediction problem for social networks. J Am Soc Inform Sci Technol 58(7):1019–1031