NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks

Wenchao Yu[1], Wei Cheng[2], Charu C. Aggarwal[3], Kai Zhang[4], Haifeng Chen[2], and Wei Wang[1]
[1]Department of Computer Science, University of California Los Angeles
[2]NEC Laboratories America, Inc. [3]IBM Research AI
[4]Department of Computer and Information Sciences, Temple University
{wenchaoyu,weiwang}@cs.ucla.edu,charu@us.ibm.com,{weicheng,haifeng}@nec-labs.com,zhang.kai@temple.edu

ABSTRACT

Massive and dynamic networks arise in many practical applications such as social media, security and public health. Given an evolutionary network, it is crucial to detect structural anomalies, such as vertices and edges whose "behaviors" deviate from underlying majority of the network, in a real-time fashion. Recently, network embedding has proven a powerful tool in learning the low-dimensional representations of vertices in networks that can capture and preserve the network structure. However, most existing network embedding approaches are designed for static networks, and thus may not be perfectly suited for a dynamic environment in which the network representation has to be constantly updated. In this paper, we propose a novel approach, NetWalk, for anomaly detection in dynamic networks by learning network representations which can be updated dynamically as the network evolves. We first encode the vertices of the dynamic network to vector representations by *clique embedding*, which jointly minimizes the pairwise distance of vertex representations of each walk derived from the dynamic networks, and the deep autoencoder reconstruction error serving as a global regularization. The vector representations can be computed with constant space requirements using *reservoir sampling*. On the basis of the learned low-dimensional vertex representations, a clustering-based technique is employed to incrementally and dynamically detect network anomalies. Compared with existing approaches, NetWalk has several advantages: 1) the network embedding can be updated dynamically, 2) streaming network nodes and edges can be encoded efficiently with constant memory space usage, 3). flexible to be applied on different types of networks, and 4) network anomalies can be detected in real-time. Extensive experiments on four real datasets demonstrate the effectiveness of NetWalk.

CCS CONCEPTS

• **Computing methodologies** → **Anomaly detection**; **Dimensionality reduction and manifold learning**; • **Theory of computation** → **Dynamic graph algorithms**;

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220024

KEYWORDS

Anomaly detection, dynamic network embedding, deep autoencoder, clique embedding

ACM Reference Format:
Wenchao Yu[1], Wei Cheng[2], Charu C. Aggarwal[3], Kai Zhang[4], Haifeng Chen[2], and Wei Wang[1]. 2018. NetWalk: A Flexible Deep Embedding Approach for Anomaly Detection in Dynamic Networks. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220024

1 INTRODUCTION

Anomaly detection (a.k.a outlier detection) in dynamically changing networks is a long-standing problem deeply motivated in a number of application domains, such as social media, security, public health, and computational biology [1, 5, 7, 16, 33]. Identifying time-varying anomalies (such as edges or vertices), which represent significant deviations from "normal" structural patterns in the evolving network, can shed important light on the functional status of the whole system. Many methods have been proposed in the past decade to solve this problem [3, 18, 23, 32, 40]. Some prominent examples of applications are summarized as follows.

- With the popularity of social media, anomalous behaviors can be found in the underlying social network. The malicious activities such as cyber-bullying, terrorist attack planning and fraud information dissemination can be detected as anomalies using anomaly detection models based on social network.
- The advanced persistent threat (APT) detection problem in security can also be cast as real-time anomaly detection in network streams. In an APT scenario, we are given a stream of system logs which can be used to construct information-flow networks. Information flows induced by malicious activities can be quite different from normal system behaviors.
- In clinics, anomaly detection can provide valuable information on managing and diagnosis with the electric patient records. The data typically consist of records from various types of entities (vertices) such as patients, symptoms and treatments, which can be modeled as a multi-partite network representing their complex interactions. Anomalies in such networks can pinpoint important scenarios requiring instant human interventions, such as abnormal patient condition or recording errors.

To detect network anomalies in these applications, a typical approach is to first perform network sketching and then identify

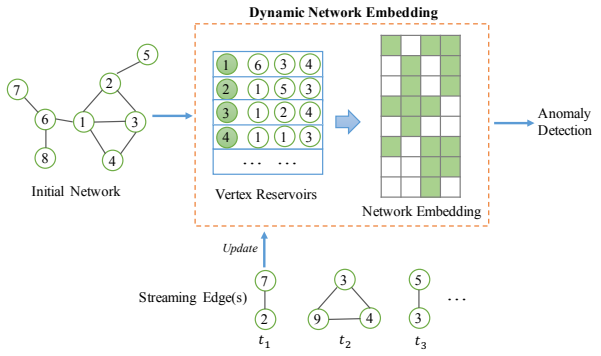

Figure 1: Workflow of NetWalk for anomaly detection in dynamic networks

anomalies in the sketches through clustering and outlier detection, such as in [23, 32]. The network sketches serve as a compact, latent representation of the network and thus allow efficient updates as new network objects arrive in a streaming fashion, without having to maintain the complete details of the full network. In the literature, the network sketches have been learned through locality-sensitive hashing [20] and count-min sketch [11]. However, these approaches are not directly designed to learn the network sketches that can simultaneously preserve important structural relations, such as the local neighborhood composition or proximity landscapes. Thus, the sketches extracted are usually shallow [8, 22], and thereby bottleneck the accuracy of downstream anomaly detection task.

Recently, network embedding through neural networks has attracted significant interest and shown promising results, in particular towards obtaining desired low-dimensional representations of network that best preserve the neighborhood information [8, 22, 25, 26]. The structure preserving property of the network embedding makes it particularly suitable for anomaly detection tasks, by examining the similarity between vertices/edges in the latent representation. For example, vertices staying far away from the majority clusters in the multidimensional latent space will very likely indicate certain types of anomalies, which can be detected conveniently through dynamic clustering algorithms. However, existing methods for network embedding can not update the representation dynamically as new vertices or edges keep feeding, and thus may not be perfectly suitable for anomaly detection in a dynamic environment [8, 22, 25, 26]. In case of a rapidly evolving network, the problem can be even more challenging. It is therefore highly desirable to design an effective and especially efficient embedding algorithm that is capable of fast, real-time detection with bounded memory usage.

To address this problem, in this paper, we propose the NetWalk algorithm to incrementally learn network representations as the network evolves, and detect anomalies in the networks in a real-time fashion. Figure 1 shows an illustrative diagram of the anomaly detection pipeline in dynamic networks. First, we learn the latent network representation by using a number of network walks extracted from the initial network. The representation is obtained not only through maintaining the pairwise vertex-distance in the local walks, but also by hybridizing it with the hidden layer of a deep auto-encoder, such that the resultant embedding is guaranteed to faithfully reconstruct the original network. By doing this, the learned vertex coordinates in the multi-dimensional Euclidean space can achieve both local fitting and global regularization. In addition, the learned representations can be easily updated over dynamic changes by leveraging a *reservoir sampling* strategy. Then, a dynamic clustering model is used to flag anomalous vertices or edges based on the learned vertex or edge representations. We quantitatively validate the effectiveness and efficiency of the proposed framework on real datasets. To summarize, the main contributions are as follows:

- We propose a novel anomaly detection framework, NetWalk, which learns vector representations for vertices and edges, and detects network deviations based on a dynamic clustering algorithm.
- We propose an efficient algorithm for network representation learning based on *deep* neural network embedding and *reservoir sampling*. It can accurately and rapidly encode the evolving network objects.
- The proposed NetWalk is flexible. It is applicable on both directed and undirected networks, either weighted or not, to detect abnormal vertices and edges in a network that may evolve over time by dynamically inserting or deleting vertices and edges.
- We conduct extensive experiments on real-world information networks. Experimental results demonstrate the effectiveness and efficiency of NetWalk.

The rest of this paper is organized as follows. Section 2 formally defines the problem of anomaly detection in dynamic networks. Section 3 introduces the methods used for network representation learning and Section 4 describes the clustering-based anomaly detection algorithm. Section 5 empirically evaluates NetWalk on anomaly detection tasks using various real-world networks. Section 6 briefly surveys related work on anomaly detection in dynamic networks. Finally we conclude in Section 7.

2 PROBLEM FORMULATION

Given a temporal network $\mathcal{G}(t) = (\mathcal{E}(t), \mathcal{V}(t))$, we assume that the incoming stream of network objects at time-stamp t is typically a small number of network objects denoted by an edge set[1] $E^{(t)}$ where $|E^{(t)}| \geq 1$. All vertices in the edge set $E^{(t)}$ at time-stamp t are denoted by $V^{(t)}$. The vertex set $\mathcal{V}(t)$ denotes the union of the vertex sets across all time-stamps from 1 to t, that is, $\mathcal{V}(t) = \cup\{V^{(i)}\}_{i=1}^{t}$. Similarly, we have $\mathcal{E}(t) = \cup\{E^{(i)}\}_{i=1}^{t}$. Note that the complete set of vertices may not be known at time-stamp t, since new vertices may keep arriving at time-stamp t' for any $t' > t$. The network $\mathcal{G}(t)$ includes all edges received from time-stamps 1 to t.

Our goal is to detect anomalous vertices, edges and communities (group of vertices) at any given time-stamp t, i.e., in real time as $E^{(t)}$ occurs. To achieve this goal, we encode the network $\mathcal{G}(t)$ as a feature matrix, where each row is the vector representation of a vertex (Section 3). The main challenges are, i) we need a cohesive way to encode the dynamic network, ii) incoming network objects

[1]Note that the case of incoming stream of edges includes the case of new vertices since an edge contains both the edge itself and the connected vertices. The case of incoming stream of singleton vertices is trivial because they are obviously anomalies.

Table 1: Notation Description

Notation	Description
$\mathcal{E}(t)$	streaming edges received from time-stamps 1 to t
$\mathcal{V}(t)$	vertex set across time-stamp 1 to time-stamp t
$\mathcal{G}(t)$	the network at time-stamp t with $\mathcal{E}(t)$ and $\mathcal{V}(t)$
$\Omega(t)$	network walk set of $\mathcal{G}(t)$
n	number of vertices, $\lvert\mathcal{V}(t)\rvert$
m	number of edges, $\lvert\mathcal{E}(t)\rvert$
l	walk length
ψ	number of network walks per vertex
$\lvert\Omega\rvert$	total number of network walks, $\lvert\Omega\rvert = n \times \psi$
d	latent dimension of vertex representation
ρ	sparsity parameter
n_l	total number of layers of the autoencoder network
$\mathbf{x}_p^{(i)} \in \mathbb{R}^n$	input vector of vertex $p \in [1, l]$ in walk $i \in [1, \lvert\Omega\rvert]$
$\mathbf{W}^{(\ell)} \in \mathbb{R}^{n \times d}$	weight matrix at layer ℓ
$\mathbf{b}^{(\ell)} \in \mathbb{R}^d$	bias vector at layer ℓ
$\mathbf{D} \in \mathbb{R}^{n \times n}$	diagonal degree matrix
$f^{(\ell)}(\mathbf{x})$	network output of layer ℓ

should be easily coded with the learned network representations, iii) network representations need to be efficiently updated as new network objects arrive. We then follow a clustering-based approach to detect the anomalies in the dynamic network (Section 4). The clusters are generated after calculating the distances between the embedded vertices. Whether the incoming network objects are anomalies or not can be determined from the distance between their respective representations and existing clusters. The clustering results are updated efficiently as new network objects arrive. The notations used in this paper are summarized in Table 1.

3 ENCODING NETWORK STREAMS

In order to detect anomalies in dynamic networks in real time, our method needs to learn network representations and perform online updates efficiently as network evolves. For clarity, now we only discuss the case that new edges stream in on unweighed network. The cases of decreasing edges or tackling weighted networks are similar and will be discussed in Section 3.2. In this section, we present the network encoding and the updating phase in NetWalk which does not require to store the entire network.

3.1 Network Walk Generation

Analogous to word embedding techniques [25, 26] in constructing vector representations, we decompose the network into a set of network walks each of which contains a list of vertices selected by a random walk[2]. We formally define the network walk as follows.

Definition 3.1 (Network Walk). For a given vertex $v_1 \in \mathcal{V}$ in a network $\mathcal{G}(\mathcal{E}, \mathcal{V})$, its network walk set is defined as $\Omega_{v_1} = \{(v_1, v_2, ..., v_l) | \ (v_i, v_{i+1}) \in \mathcal{E} \wedge p(v_i, v_{i+1}) = \frac{1}{D_{v_i, v_i}}\}$, which is a collection of l-hop walks starting from vertex v_1. The transition probability $p(v_i, v_{i+1})$ from v_i to v_{i+1} is proportional to the degree D_{v_i, v_i} of vertex v_i. We call Ω_v a network walk set starting from v, and $\Omega = \{\Omega_v\}_{v \in \mathcal{V}}$ as the union of all walks.

[2]Note that this works for both directed and undirected network

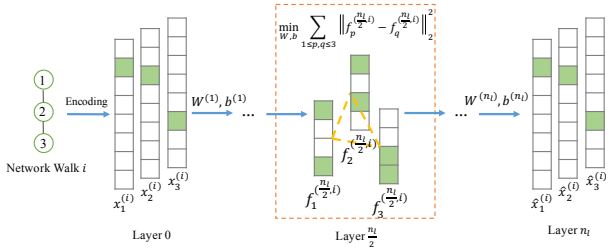

Figure 2: Illustration of clique embedding for one network walk of length 3

Similar to the word frequency which typically follows a power law distribution in natural language, we observe that if the degree distribution of the network follows a power law distribution, the frequency distribution of vertices occurring in the network walks also follows a power law distribution (or Zipf's law) [31]. Therefore we will use a stream of network walks as our basic tool for extracting information from a network. We then learn the vertex representations of the network using a novel embedding method introduced in Section 3.2.

3.2 Learning Network Representations

We formulate the network representation learning problem as an optimization problem. Our goal is to learn a mapping function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ such that each $v \in \mathcal{V}$ is represented as a d-dimensional vector, where d is the latent-space dimension. The mapping function f applies to any (un)directed, (un)weighted network.

Inspired by skip-gram architecture [26], we propose a network embedding algorithm, *clique embedding*, that utilizes an deep auto-encoder neural network to learn the vector representation of vertices through a stream of network walks while minimizing the pairwise distance among all vertices in each walk. Figure 2 depicts the clique embedding model used in NetWalk. The inputs and outputs are one-hot encoded vectors, that is, for a given vertex input $\mathbf{x}_p^{(i)} \in \mathbb{R}^n$, only one out of n elements will be 1, and all others are 0's. Our goal is to learn a latent representation for each input network walk $\{\mathbf{x}_p^{(i)}\}_{p=1}^{l}$. Here l is the walk length, $\{\mathbf{W}^{(\ell)}\}_{\ell=1}^{n_l}$ are the weight matrices, $\{\mathbf{b}^{(\ell)}\}_{\ell=1}^{n_l}$ are the bias vectors, and $f^{(\ell)}(\cdot)$ denotes the output of each layer.

Formally, given a one-hot encoded network walk $\{\mathbf{x}_p^{(i)}\}_{p=1}^{l}$, $i = 1, ..., |\Omega|$, we want to learn the following representations in a n_l-layer autoencoder network,

$$f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) = \sigma(\mathbf{W}^{(\frac{n_l}{2})^\top} h^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) + \mathbf{b}^{(\frac{n_l}{2})}), \tag{1}$$

where

$$h^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) = \mathbf{W}^{(\frac{n_l}{2}-1)} f^{(\frac{n_l}{2}-1)}(\mathbf{x}_p^{(i)}) + \mathbf{b}^{(\frac{n_l}{2}-1)}. \tag{2}$$

Here, $\sigma(z) = \frac{1}{1+\exp(z)}$ is the sigmoid function; $n_l \geq 2$; $f^{(0)}(\mathbf{x}_p^{(i)}) = \mathbf{x}_p^{(i)}$. In an auto-encoder network, the output hypotheses $f^{(n_l)}(\mathbf{x}_p^{(i)})$ is approximately equal to $\mathbf{x}_p^{(i)}$. Therefore, if we use ℓ_2 norm to minimize the reconstruction error, the objective function becomes

$$J_{AE} = \frac{1}{2} \sum_{i=1}^{|\Omega|} \sum_{p=1}^{l} \left\| f^{(n_l)}(\mathbf{x}_p^{(i)}) - \mathbf{x}_p^{(i)} \right\|_2^2. \tag{3}$$

We also seek to minimize the pairwise distance among all vertices[3] of each network walk in the embedding space at layer $\frac{n_l}{2}$, which can be formally described as follows,

$$J_{Clique} = \sum_{i=1}^{|\Omega|} \sum_{1 \le p,q \le l} \left\| f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)}) \right\|_2^2. \tag{4}$$

Due to the sparsity of the input and output vectors, we consider a sparse auto-encoder with sparsity parameter ρ and penalize it with the Kullback-Leibler divergence [27],

$$\mathrm{KL}(\rho \| \hat{\boldsymbol{\rho}}^{(\ell)}) = \sum_{j=1}^{d} \mathrm{KL}(\rho \| \hat{\rho}_j^{(\ell)}) = \sum_{j=1}^{d} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}, \tag{5}$$

where $\hat{\boldsymbol{\rho}}^{(\ell)} = \frac{1}{|\Omega| \times l} \sum_{i=1}^{|\Omega|} \sum_{p=1}^{l} f^{(\ell)}(\mathbf{x}_p^{(i)})$ is the average activation of the units in the hidden layer. This sparsity constraint penalizes large deviation of $\hat{\rho}_j^{(\ell)}$ from ρ. Given a training set of network walks Ω, we then define the overall cost function to be:

$$\begin{aligned} J(\mathbf{W}, \mathbf{b}) = & \underbrace{\sum_{i=1}^{|\Omega|} \sum_{1 \le p,q \le l} \left\| f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)}) \right\|_2^2}_{\text{Clique Embedding Loss}} + \underbrace{\frac{\gamma}{2} \sum_{i=1}^{|\Omega|} \sum_{p=1}^{l} \left\| f^{(n_l)}(\mathbf{x}_p^{(i)}) - \mathbf{x}_p^{(i)} \right\|_2^2}_{\text{Reconstruction Error}} \\ & + \underbrace{\beta \sum_{\ell=1}^{n_l - 1} \sum_{j} \mathrm{KL}\left(\rho \| \hat{\rho}_j^{(\ell)}\right)}_{\text{Sparsity Constraint}} + \underbrace{\frac{\lambda}{2} \sum_{\ell=1}^{n_l} \left\| \mathbf{W}^{(\ell)} \right\|_F^2}_{\text{Weight Decay}}, \end{aligned} \tag{6}$$

where $|\Omega|$ is the number of network walks, l is the walk length. The weight decay term decreases the magnitude of the weights, and helps prevent overfitting. γ, β and λ control the weight of the corresponding penalty terms. The loss function $J(\mathbf{W}, \mathbf{b})$ can also be written in a matrix form,

$$\begin{aligned} J(\mathbf{W}, \mathbf{b}) = & \sum_{i=1}^{|\Omega|} \mathrm{Tr}(\boldsymbol{\mathcal{F}}^{(i)} \mathbf{L} \boldsymbol{\mathcal{F}}^{(i)\top}) + \frac{\gamma}{2} \left\| \boldsymbol{\mathcal{H}}^{(n_l)}(\mathbf{X}) - \mathbf{X} \right\|_F^2 \\ & + \beta \sum_{\ell=1}^{n_l - 1} \mathrm{KL}(\rho \| \hat{\boldsymbol{\rho}}^{(\ell)}) + \frac{\lambda}{2} \left\| \mathbf{W}^{(1)} \right\|_F^2 + \frac{\lambda}{2} \sum_{\ell=1}^{n_l} \left\| \mathbf{W}^{(\ell)} \right\|_F^2, \end{aligned} \tag{7}$$

where $\boldsymbol{\mathcal{F}}^{(i)} = [f_1^{(i)}, f_2^{(i)}, ..., f_l^{(i)}]$, $f_l^{(i)} = f^{(\frac{n_l}{2})}(\mathbf{x}_l^{(i)})$; $\mathbf{L}$ is the Laplacian matrix of the clique with l vertices, thus we have $\mathbf{L} = \mathbf{I}_l \times (l-1) - \boldsymbol{\Phi}$, and $\boldsymbol{\Phi}_{i,j} = 1, \forall i \ne j$. $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(|\Omega|)}]$, $\mathbf{x}^{(i)} = [\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, ..., \mathbf{x}_l^{(i)}]$; $\boldsymbol{\mathcal{H}}^{(n_l)}(\mathbf{X}) = [g^{(1)}, g^{(2)}, ..., g^{(|\Omega|)}]$, $g^{(i)} = [f^{(n_l)}(\mathbf{x}_1^{(i)}), f^{(n_l)}(\mathbf{x}_2^{(i)}), ... f^{(n_l)}(\mathbf{x}_l^{(i)})]$.

Our goal is to minimize $J(\mathbf{W}, \mathbf{b})$ as a function of $\mathbf{W}$ and $\mathbf{b}$. The key step is to compute the partial derivatives of objective function Eq.(7), with respect to $\mathbf{W}$ and $\mathbf{b}$ to derive updates. Inspired by back-propagation algorithm [34], we introduce "error terms" for $\delta^{(\ell)}$ and $\delta^{(n_l)}$ for the hidden layer and output layer respectively. These "error terms" can be computed as follows:

$$\begin{aligned} \delta^{(n_l)} &= -\gamma \nabla_{f^{(n_l)}(\mathbf{X})} J_{AE} \circ \sigma'\left(h^{(n_l)}(\mathbf{X})\right) \\ &= -\gamma \left(\boldsymbol{\mathcal{H}}^{(n_l)}(\mathbf{X}) - \mathbf{X}\right) \circ f^{(n_l)}(\mathbf{X}) \circ \left(1 - f^{(n_l)}(\mathbf{X})\right), \end{aligned} \tag{8}$$

$$\begin{aligned} \delta^{(\ell)} &= \left(\mathbf{W}^{(\ell)\top} \delta^{(\ell+1)} + \beta \frac{\hat{\boldsymbol{\rho}}^{(\ell)} - \boldsymbol{\rho}_0}{\hat{\boldsymbol{\rho}}^{(\ell)}(1 - \boldsymbol{\rho}_0)}\right) \circ \sigma'\left(h^{(\ell)}(\mathbf{X})\right) \\ &= \left(\mathbf{W}^{(\ell)\top} \delta^{(\ell+1)} + \beta \frac{\hat{\boldsymbol{\rho}}^{(\ell)} - \boldsymbol{\rho}_0}{\hat{\boldsymbol{\rho}}^{(\ell)}(1 - \boldsymbol{\rho}_0)}\right) \circ f^{(\ell)}(\mathbf{X}) \circ \left(1 - f^{(\ell)}(\mathbf{X})\right), \end{aligned} \tag{9}$$

where $\boldsymbol{\rho}_0$ is a vector with all entries ρ; "$\circ$" denotes the element-wise product. Since the *clique embedding loss* only depends on $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{\frac{n_l}{2}}$, then the derivatives for $\ell > \frac{n_l}{2}$ are

$$\nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = \delta^{(\ell)} \left(f^{(\ell-1)}(\mathbf{X})\right)^\top + \lambda \mathbf{W}^{(\ell)}, \tag{10}$$

$$\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = \sum_{i=1}^{|\Omega|} \delta_i^{(\ell)}. \tag{11}$$

We need to take the clique embedding loss into consideration when computing the derivatives for $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{\frac{n_l}{2}}$.

$$\begin{aligned} \nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = & \sum_{i=1}^{|\Omega|} \boldsymbol{\mathcal{F}}^{(i)}(\mathbf{L} + \mathbf{L}^\top) \circ \boldsymbol{\mathcal{F}}^{(i)} \circ (1 - \boldsymbol{\mathcal{F}}^{(i)}) \left(f^{(\ell-1)}(\mathbf{X})\right)^\top \\ & + \delta^{(\ell)} \left(f^{(\ell-1)}(\mathbf{X})\right)^\top + \lambda \mathbf{W}^{(\ell)}, \end{aligned} \tag{12}$$

$$\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b}) = \sum_{i=1}^{|\Omega|} \boldsymbol{\mathcal{F}}^{(i)}(\mathbf{L} + \mathbf{L}^\top) \circ \boldsymbol{\mathcal{F}}^{(i)} \circ (1 - \boldsymbol{\mathcal{F}}^{(i)}) + \delta_i^{(\ell)}. \tag{13}$$

Starting from every vertex $v \in V$, we generate all network walks via random walk. Then network representations are learned by optimizing the aforementioned loss function $J(\mathbf{W}, \mathbf{b})$. The pseudocode for network encoding is given in Algorithm 1.

Algorithm 1: Clique Embedding of NetWalk

Input: Network walk set Ω.
Output: Network representations $f^{\frac{n_l}{2}}(\mathbf{x}_p^{(i)})$
Set latent dimension d, sparsity ρ, weight control parameters γ, β and λ.
Randomly initialize $\{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{n_l}$.
Construct input vector $\mathbf{x}_p^{(i)} \in \mathbb{R}^n$ for vertex p in walk i, $1 \le p \le l$, $1 \le i \le |\Omega|$
while *not stopping criterion* **do**
  Perform a feedforward pass to compute $f^{(\ell)}(\mathbf{x}_p^{(i)})$.
  For the output layer n_l, set $\delta^{(n_l)}$ using Eq.(8)
  for $\ell = n_l - 1, n_l - 2, n_l - 3, \ldots, 1$ **do**
    Compute "error terms" $\delta(\ell)$ using Eq.(9).
    if $\ell > \frac{n_l}{2}$ **then**
      Compute partial derivatives $\nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ and $\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ using Eq.(10)-(11).
    else
      Compute partial derivatives $\nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ and $\nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$ using Eq.(12)-(13).
    Determine the step size ξ by line search.
    Update $\mathbf{W}^{(\ell)} = \mathbf{W}^{(\ell)} - \xi \nabla_{\mathbf{W}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$.
    Update $\mathbf{b}^{(\ell)} = \mathbf{b}^{(\ell)} - \xi \nabla_{\mathbf{b}^{(\ell)}} J(\mathbf{W}, \mathbf{b})$.
Compute embedding results $f^{\frac{n_l}{2}}(\mathbf{x}_p^{(i)})$.

In a fully streaming setting, the entire vertex set $\mathcal{V}$ will change over time, and hence is the number of vertices n. In this case, we need to pre-allocate a fixed length for one-hot encoding technique to encode the input vectors $\mathbf{x}_p^{(i)}$'s.

Visualization. In the network representation learning phase, NetWalk takes an initial network as input and learns a latent representation for every vertex. Here, we show the encoding capability of clique embedding in NetWalk by applying our method to the Zachary's karate network [42]. Figure 3 (a) shows the original network in which the vertex color indicates the community to which

[3] This is inspired by skip-gram architecture that considers all word pairs within a distance window. It is effective for extracting local proximity information [26].

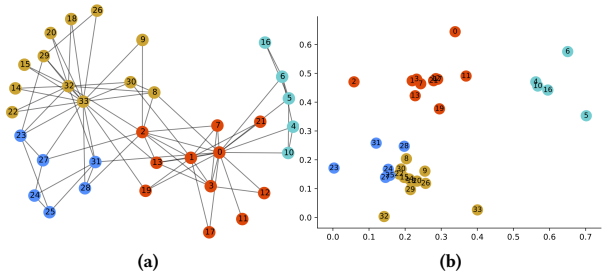

Figure 3: Embedding results on Zachary's karate network. Vertex colors represent a modularity-based clustering on the input graph. (a) the Zachary's karate network, (b) Embedding results of NetWalk.

each vertex belongs. Figure 3 (b) presents the 2-dimensional representations learned by NetWalk. Notably, the linearly separable clusters can be found in the vector representation space learned by our method.

3.3 Edge Encoding

NetWalk learns vector representations for vertices, which allows us to detect vertex anomalies based on clustering. In addition, we are also interested in edge anomaly detection. Therefore, in order to determine whether an incoming edge is an anomaly, we build a lookup table to encode new edge(s) in real-time based on the network representations we have learned. For undirected networks, the operator has to be symmetric. That is, for any edge (u, v) or (v, u), the edge representation should be the same. In this paper, we use the Hadamard operator which has shown good performance in edge encoding [15]. Assume that the d-dimensional representation learned by Algorithm 1 for vertex v is $f(v)$, then the representation of each edge (v, u) under Hadamard operator is $[f(v) \circ f(u)]_i = f_i(v) \times f_i(u)$. It is worth mentioning that the way to encode edges is very flexible. We can add any additional edge-specific features to augment the edge vector.

3.4 Maintaining Network Representations Incrementally

To cope with the fast evolving nature of dynamic networks, we prefer to update the network representations without having to maintain explicitly the complete details of network structures. This section describes how the network representations learned by NetWalk are dynamically updated upon changes of the network. Each added/deleted edge affects in a number of network walks which will be used to update the current network representation. In our model, we design a reservoir-based algorithm to maintain a compact record which consists of a set of "neighbors" for each vertex, and the walks are updated based on the reservoir for each vertex.

Definition 3.2 (Vertex Reservoir). For each vertex $v \in \mathcal{V}$, the corresponding vertex reservoir S_v is a set of vertex with ψ items, which are sampled with replacement from v's neighbors $ne_v = \{u|(u, v) \in \mathcal{E}, u \neq v\}$.

Given a stream of edges, NetWalk maintains a reservoir for each vertex v such that each single item in the reservoir is selected at random from v's neighbors. Thus the reservoir needs to be updated as new edges arrive. The updating rules are described as follows for each newly added edge (u, v):

(1) update the degree of vertices u and v: $D_{u,u} = D_{u,u} + 1$, $D_{v,v} = D_{v,v} + 1$;
(2) for each item in the reservoir S_u, with probability $\frac{1}{D_{u,u}}$, replace the old item with the new item v; and with probability $1 - \frac{1}{D_{u,u}}$, keep the old item;
(3) for each item in the reservoir S_v, with probability $\frac{1}{D_{v,v}}$, replace the old item with the new item u; and with probability $1 - \frac{1}{D_{v,v}}$, keep the old item.

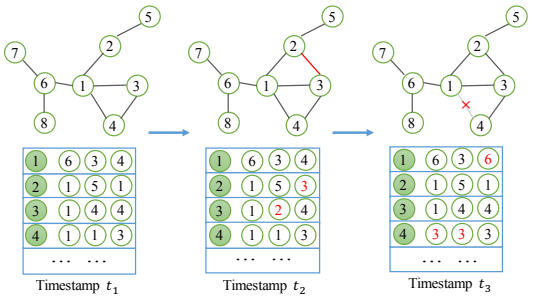

Figure 4: Illustration of updating the reservoirs. Initially we build the reservoir of each vertex based on the network at t_1. When (v_2, v_3) is added at timestamp t_2, the corresponding reservoirs of v_2 and v_3 will be updated. Similarly, when (v_1, v_4) is deleted at timestamp t_3, we replace the deleted items with the remaining neighbors of the corresponding vertex.

Lemma 3.3. *For each i, the i^{th} neighbor of vertex v is chosen to be included in the reservoir S_v with probability $\frac{\psi}{D_{v,v}}$.*

Proof. We will prove it by induction. After the $(i-1)^{\text{th}}$ round, let us assume that the probability of an item being in the reservoir S_v is $\frac{\psi}{D_{v,v}}$. Since the probability of the item being replaced in the i^{th} round is $\frac{1}{D_{v,v}+1}$, the probability that a given item is in the reservoir after the $(i-1)^{\text{th}}$ round will be $\frac{\psi}{D_{v,v}} \times (1 - \frac{1}{D_{v,v}+1}) = \frac{\psi}{D_{v,v}+1}$. We update $D_{v,v} \leftarrow D_{v,v} + 1$. Hence, the result holds for i, and is therefore true by induction. □

In case where edges are deleted, the reservoir is chosen similarly to aforementioned rules. In this case, one needs to update the degree matrix first, and then replace the deleted items with the remaining neighbors of the corresponding vertex. As illustrated in Figure 4, when (v_2, v_3) is added at timestamp t_2, the corresponding reservoirs of v_2 and v_3 will be updated by adding v_3 with a probability of $\frac{1}{3}$, and v_2 with a probability of $\frac{1}{3}$, respectively. Similarly, when (v_1, v_4) is deleted at timestamp t_3, we replace the deleted item v_1 with v_3 with a probability of 1 (there is only one remaining neighbor of the corresponding vertex v_4), and replace the deleted item v_4 with v_3 or v_6 with probability $\frac{1}{2}$.

After updating the reservoir of the corresponding vertices as edge (u, v) arrives, we will generate the network walks that need to

be updated accordingly. For each newly added edge (u, v), the walks need to be added are defined as $\Omega^+ = \{(u_1, u_2, ..., u_i, u, v, v_1, v_2, ..., v_j) \vee (v_1, v_2, ..., v_i, v, u, u_1, u_2, ..., u_j) | i + j = l - 2\}$, which is a collection of network walks with length l including the new edge (u, v). The transition probability of each connected vertex pair (u_m, u_n) is $p(u_m, u_n) = \frac{1}{D_{u_m,u_m}}$. For each edge (u', v') that needs to be removed, the dynamic walk are defined as $\Omega^- = \{\omega | \forall \omega \in \Omega \wedge ((u', v') \in \omega \vee (v', u') \in \omega)\}$. We will then continue to train the model with the updated network walk set in a warm-start fashion. The pseudocode of updating network representations is shown in Algorithm 2.

Algorithm 2: Network Representation Maintenance

Input: Network walk set Ω, a streaming edge set $E^{(t)}$; saved clique embedding model
Output: The updated Ω, the updated embedding clique model
// dynamic walks generation
for (u, v) *in the streaming edge set* $E^{(t)}$ **do**
  Update vertex set $\mathcal{V}$.
  Update degree matrix $\mathbf{D}$.
  Update the reservoirs S_u and S_v using the rules described in Section 3.4.
  Generate the network walk sets Ω^+ for new edges and Ω^- for deleted edges, respectively.
// model update
Load the saved embedding model.
Train the model with the dynamic network walk set Ω^+ with a small sample set of walks from Ω, or with the updated walk set $\Omega - \Omega^-$ if with edge deletion.
Update network representations $f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)})$.
Save the updated clique embedding model.

Discussion. 1). *On the incremental online training.* When new edges come, ideally, we need to retrain the embedding model on the whole walk set $\Omega \cup \Omega^+$. However, this is usually time-consuming. Many online gradient decent methods have been discussed for this problem. For example, we can sample a small set of walks from Ω based on their gradients [12] and add them to Ω^+ for training. For the edge deletion case, we retrain the model with the updated walk set $\Omega - \Omega^-$ for edge deletion. This is time consuming if the original walk set Ω is very large. We can also incorporate the edge deletion part into the objective function Eq.(7),

$$J(\mathbf{W}, \mathbf{b}) = \sum_{i=1}^{|\Omega^+|} \sum_{1 \le p, q \le l} \left\| f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)}) \right\|_2^2 + \sum_{i=1}^{|\Omega^-|} \sum_{1 \le p, q \le l} \max\left(0, \alpha - \left\| f^{(\frac{n_l}{2})}(\mathbf{x}_p^{(i)}) - f^{(\frac{n_l}{2})}(\mathbf{x}_q^{(i)}) \right\|_2^2\right) + \frac{\gamma}{2} J_{AE} + \beta J_{Sparsity} + \frac{\lambda}{2} J_{Weight\ Decay}, \quad (14)$$

which ensures that the deleted edges in the embedding space have a distance of at least α from each other. In the following evaluation section, we focus on the edge addition scenario which is more common in real world. 2). *On the weighted networks.* Only minor modification on current algorithm is needed to accommodate weighted network anomaly detection. First, since the walks generating step adopts random walker technique, it is easy to consider the weights of edges into the transition probability. Accordingly, in Eq.(4), additional weights should be put to the pairwise loss of two vertices.

Visualization. In this subsection, we show the dynamic encoding capability of NetWalk by applying our method to the Email network[4]. Figure 5 (a) shows the embedding results with 50% edges, in which the vertex color indicates the community to which each vertex belongs. Figure 5 (b) presents the online embedding results with additional 25% edges, and Figure 5 (c) updates the embeddings with the remaining 25% edges. Notably, more and more linearly separable clusters can be found in the 2-dimensional representation space in Figure 5 (b) and (c).

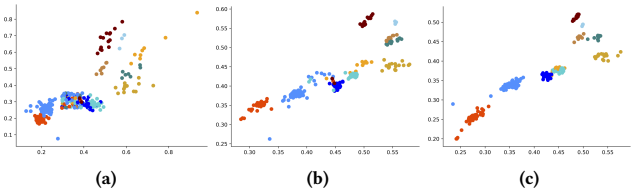

Figure 5: Embedding results on Email network. Vertex colors represent a modularity-based clustering on the input graph. (a) initial embedding with 50% edges, (b) online embedding with additional 25% edges (75% edges in total), (b) online embedding with additional 25% edges (100% edges in total).

Computational Analysis. To help analyzing the complexity of maintaining network representations, a summary of the notations is given in Table 1. In network walk generation section, the time complexity to generate $|\Omega|$ walks with length l in a network with n vertices is $O(nl|\Omega|)$. The edge encoding step takes $O(md)$ time to encode m edges with vertex dimension d. For each newly added edge, it takes $O(\psi)$ time to update the corresponding reservoirs, and $O(\psi l)$ time to generate the walks that need to be retrained.

4 ANOMALY DETECTION

The network representations learned by NetWalk can be beneficial for lots of downstream applications, such as link prediction, anomaly detection and community detection. In this paper, we focus on the anomaly detection problem based on the learned network representations. We define the anomaly detection problem in dynamic network as follows: given the vertex representations $f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_p^{(i)}) \in \mathbb{R}^d$ or corresponding edge representations, group existing representations into k clusters, and detect any newly arriving vertices or edges that do not naturally belong to any existing cluster. This may include the following scenarios: 1) the vertex or edge corresponds to an anomaly; 2) the vertex or edge marks the start of a new cluster in the network stream. It is difficult to distinguish these two cases unless we receive more streaming data afterwards. So in our model, we find the closest cluster to each point. We use the Euclidean distance as the similarity measure, given by $||\boldsymbol{c} - f(\cdot)||_2$, where $\boldsymbol{c}$ is the cluster center and $f(\cdot)$ is the learned representation for each vertex or edge. The anomaly score for each point is reported as its closest distance to any cluster centers.

When new edges stream in, we need to update cluster centers accordingly. In this paper, we leverage the streaming k-means clustering [4] which uses parameters to control the decay of estimates. Here, we introduce a decay factor α when calculating the new cluster centers after absorbing new point(s). We use the parameter α to control the importance of "older" data points in existing clusters.

[4] https://snap.stanford.edu/data/email-Eu-core-temporal.html

Table 2: Anomaly detection performance comparison

Methods	UCI Messages			arXiv hep-th			Digg			DBLP		
	1%	5%	10%	1%	5%	10%	1%	5%	10%	1%	5%	10%
GOutlier	0.7181	0.7053	0.6707	0.6964	0.6813	0.6322	0.6963	0.6763	0.6353	0.7172	0.6891	0.6460
CM-Sketch	0.7270	0.7086	0.6861	0.7030	0.6709	0.6386	0.6871	0.6581	0.6179	0.7097	0.6892	0.6332
Spectral Clustering	0.6324	0.6104	0.5794	0.6114	0.6034	0.5593	0.5949	0.5823	0.5591	0.6141	0.6245	0.5915
DeepWalk	0.7514	0.7391	0.6979	0.7312	0.7000	0.6644	0.7080	0.6881	0.6396	0.7413	0.7202	0.6657
node2vec	0.7371	0.7433	0.6960	0.7374	0.7137	0.6748	0.7364	0.7081	0.6508	0.7368	0.7193	0.6786
SDNE	0.7307	0.7144	0.6868	0.7221	0.7041	0.6609	0.7160	0.6804	0.6340	0.7342	0.7160	0.6565
NetWalk	**0.7758**	**0.7647**	**0.7226**	**0.7489**	**0.7293**	**0.6939**	**0.7563**	**0.7176**	**0.6837**	**0.7654**	**0.7388**	**0.6858**

Table 3: Temporal network dataset description

Dataset	#Vertex	#Edge	Max. Degree	Avg. Degree
UCI Messages (directed)	1,899	13,838	255	14.57
arXiv hep-th (undirected)	6,798	214,693	1,590	63.16
Digg (directed)	30,360	85,155	283	5.61
DBLP (undirected)	315,159	743,709	216	4.72

Assuming that there are n_0 points $\{\boldsymbol{x}_i\}_{i=1}^{n_0}$ in an existing cluster and n' new points $\{\boldsymbol{x}'_i\}_{i=1}^{n'}$ at time-stamp T' to be absorbed by this cluster, the centroid $\boldsymbol{c}$ can be updated in the following way

$$\boldsymbol{c} = \frac{\alpha \boldsymbol{c}_0 n_0 + (1-\alpha)\sum_{i=1}^{n'} \boldsymbol{x}'_i}{\alpha n_0 + (1-\alpha)n'}, \tag{15}$$

where $\boldsymbol{c}_0$ is the previous cluster center. The decay factor α is chosen as 0.5 and used to ignore older instances, which is analogous to an exponentially-weighted moving average.

Computational Analysis. With k clusters signified by k center vectors, finding the nearest cluster takes only $O(kd)$ time. It takes $O(d)$ time to compute the anomaly score for each data point. Updating the centers takes $O(d)$ time with respect to the dimension of vertex/edge representations. Thus the total time complexity of anomaly detection is $O(kd)$ for each incoming data point.

5 EVALUATION

Datasets. To verify the performance of the proposed NetWalk model, we conduct experiments on a variety of dynamic networks from different domains as shown in Table 3. The `UCI Messages` [28] network is based on an online community of students at the University of California, Irvine. Each vertex represents a user, and an edge represents a message interaction. `Digg`[5] is the reply network of the news aggregator website *digg.com*. Each node is a website user, and each edge denotes the reply between two users. The `arXiv hep-th` data [21] is a collaboration network from High Energy Physics - Theory category (hep-th). Each Node represents an author, and edges represent collaborations. The `DBLP` data is also a collaboration network of authors from the DBLP computer science bibliography. Similar to `arXiv hep-th`, the nodes in this network represent the authors, and edges represent co-authorship among authors.

Baselines. The competing methods used in this paper are summarized as follows. We include four network embedding baselines and two streaming anomaly detection baselines.

- Spectral Clustering [38]: Spectral Clustering learns latent vertex features in the network by generating representations in $\mathbb{R}^d$ from the d-smallest eigenvectors of the normalized Laplacian matrix.
- DeepWalk [31]: DeepWalk is an approach for learning latent representations of vertices in a network by treating graph random walks as the equivalent of sentences.
- node2vec [15]: This approach combines the advantage of breadth-first traversal and depth-first traversal algorithms. The random walks generated by node2vec can better represent the structural equivalence.
- Structural Deep Network Embedding (SDNE) [39]: SDNE is a deep learning based network embedding model which uses autoencoder and locality-preserving constraint to learn vertex representations that capture the highly non-linear network structure.
- GOutlier [3]: GOutlier uses a structural connectivity model in order to define outliers in dynamic network. It designs a sampling method to maintain structural summaries of the underlying network.
- CM-Sketch [32]: CM-Sketch is an outlier detection model based on global and local structural properties of an edge stream. It utilizes Count-Min sketch for approximating these properties.

5.1 Identifying Anomalies

In this section, we evaluate NetWalk in two settings: static and streaming. In the static setting, the first 50% edges of the network is used for training, and the rest incoming edges are used for testing. The vertex representations are learned offline. We then use the representations to encode and cluster the training edges. The test edges are scored and ranked based on their distances to the closest cluster centers. The goal is to quantify the effectiveness of network representations of NetWalk in the anomaly detection task. Due to the challenges in collecting data with ground-truth anomalies, we use anomaly injection method to create the anomalous edges [7].

The area under curve (AUC) score is used to measure the predictive power of all methods. We rank and score all encoded testing edges by calculating the distance to the closest center in the cluster generated from the training edges based on their representations, as presented in Table 2. The parameters of NetWalk are tuned by 5-fold cross-validation on a rolling basis using the initial network. Here we set n_l, the layers of the autoencoder network, to 6. The latent dimension of vertex representation is set to 200, 200, 20 for each encoding layer. The walk length l is set to 3. The number of samples per vertex is $\psi = 20$. The other parameters are chosen as

[5] http://konect.uni-koblenz.de/networks

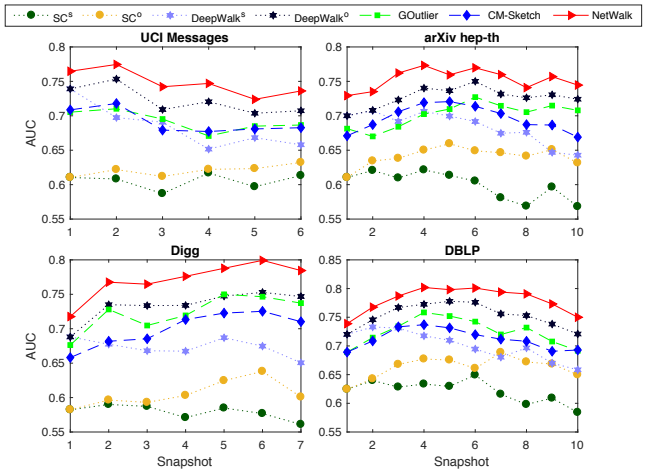

Figure 6: Accuracy of anomaly detection on dynamic network with 5% anomalies

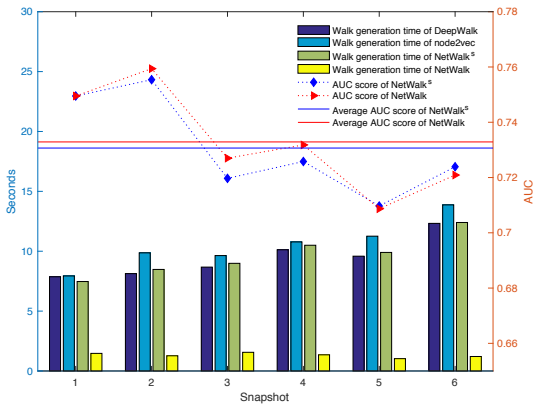

Figure 7: Dynamic maintenance performance evaluation on `UCI Messages` dataset

follows: the weight of reconstruction constraint $\gamma = 5$, sparsity ratio $\rho = 0.1$, the weight of sparsity $\beta = 0.2$, weight decay term $\lambda = 5e{-}4$, number of clusters $k = 10$. The maximum iteration of NetWalk is set to 500. For the first four network embedding methods Spectral Clustering, DeepWalk, node2vec and SDNE, we use the same clustering and ranking method for anomaly detection based on the learned representations. The testing edges of all datasets are injected 1%, 5% and 10% anomalies, respectively. It is evident from Table 2 that, 1) network embedding-based approaches (e.g. DeepWalk, node2vec and SDNE) outperform traditional sketch-based models (GOutlier and CM-Sketch), 2) NetWalk obtains a higher AUC than other baselines on all datasets. And even if 10% anomalies are injected, the performance of NetWalk is still acceptable.

In the streaming setting, we again use the first 50% edges for training to build the initial network representations and clusters. The testing edges arrive sequentially and are processed online. In other words, all testing edges are only partially visible at any given time. For convenience of comparison, we split the streaming edges into several snapshots. The number of edges for each snapshot are set to $1k$, $10k$, $6k$, $30k$ respectively for different dataset based on their test set sizes. For each arriving snapshot, NetWalk updates the corresponding network representations, clusters, and anomaly scores. For the network embedding baselines, we only include Spectral Clustering (SC) and DeepWalk since the performance of node2vec and SDNE are close to DeepWalk. These two embedding baselines are designed for static network (DeepWalk has to generate the new random walks based on the entire network), thus we adopt two versions in the evaluation. 1)The static version SC^s and DeepWalk^s: the latent vertex representations are learned only based on the initial network, and there are no updates upon receiving new edges; 2) The online version SC^o and DeepWalk^o: the algorithm is repeated using all previous $t-1$ snapshots and tested with the t^{th} snapshot. The anomaly percentage for all datasets is set to 5%. Other parameters are chosen similarly as mentioned above. The accuracies are reported in Figure 6. We observe that 1) the online versions SC^o and DeepWalk^o achieve better accuracy than the corresponding static ones, 2) NetWalk outperforms other baselines by a large margin. Note that the online version of DeepWalk needs to store the entire network in memory and repeats the walk generation at each snapshot. So our method is much more efficient on this aspect (see Section 5.2).

5.2 Dynamic Maintenance Performance

We will show that the proposed NetWalk delivers both accurate and efficient solution in a streaming setting on `UCI Messages`. Note that NetWalk leverages the reservoir sampling technique to maintain network representations incrementally. Specifically, NetWalk creates a reservoir for each vertex which contains its neighbors, and new network walks are generated based on these reservoirs without storing the entire network in memory. In this part, we test the performance of dynamic representation maintenance by comparing it with a version of NetWalk^s which needs to keep the network in memory and generate walks based the entire network at the current timestamp on `UCI Messages` with 5% anomalies.

It can be seen from Figure 7 that the average AUC score of NetWalk is 0.7329 which is higher than NetWalk^s (0.7307). However, the walk generation time of NetWalk^s is 5.0 to 10.2 times longer than NetWalk. Similar to NetWalk^s, DeepWalk and node2vec need to keep the entire network in memory to update the walks. Therefore, DeepWalk and node2vec take longer time (5× to 11×) to generate walks than our model. As for SDNE, the calculation of the first-order proximity on the network level is very time consuming compared with other baselines.

5.3 Parameter Sensitivity

The NetWalk framework involves a number of parameters that may affect its performance. We examine such changes in performance on two anomaly detection tasks (`UCI Messages` and `Digg`). We vary the number of samples per vertex (ψ), the dimensions of vertex representation (d), the training data percentage and the walk length (l) to determine their impacts on anomaly detection. Except for the parameters being tested, all other parameters assume default values.

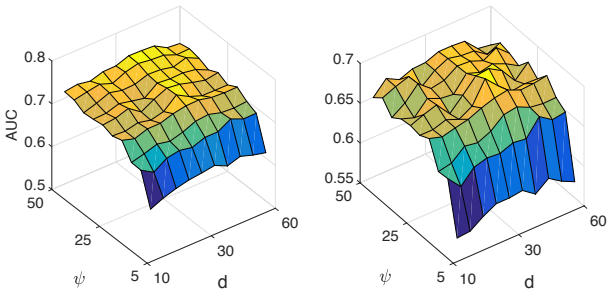

Figure 8: Anomaly detection AUC of `UCI Messages` (left) and `Digg` (right) with different parameter pairs

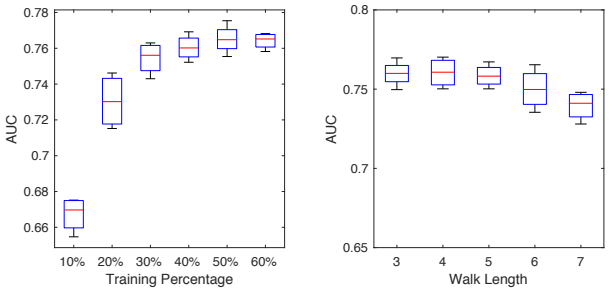

Figure 9: Stability over the training percentage of the initial network (left), and the length of network walk (right) on `UCI Messages` with 5% anomalies

We first examine different choices of parameters ψ and d. We choose values of ψ from 5 to 50, and let d vary from 10 to 60 with an interval 10. The results are summarized in Figure 8. It is evident that AUC initially improves with ψ but further improvements are slower beyond a certain threshold ($\psi > 20$). It indicates that NetWalk is able to learn meaningful vertex representations with a small number of network walks. The performance is relatively stable across different values of d. For both datasets, the AUC increases slightly as d increases, and then drops after d reaches 40. It is because, when d is small, information from the input data may be partially missing in the representation learning phase; while when d is too high, the performance of the clustering phase will be weakened.

The training percentage of the initial network and the length of network walk are also important in the anomaly detection task. Figure 9 examines the effects of varying the training percentage and the length of network walk. We observe from Figure 9(a) that the AUC increases sharply when the training percentage of network goes from 10% to 30%, and then the performance stays relatively stable. It demonstrates that our model can learn a better representation even trained with a small number of data instances. The performance slightly increases when l goes from 3 to 4; after that, the AUC decreases. This is because the clique constraint in Eq. (7) forces all the vertices in the same walk to have similar representations, which is too restrictive for longer walks. Taking both prediction performance and computational time into consideration, we will choose a relative small walk length.

6 RELATED WORK

Anomaly detection has been extensively studied in the context of multi-dimensional data [1, 10, 16] and structured networks [6, 7, 13, 14, 19, 33], including attributed networks [29, 30, 35]. Massive networks arise in many applications such as social media and public health, thus numerous algorithms have been developed for processing networks in the data stream model [2, 3, 32]. In this section, we briefly review anomaly detection algorithms in dynamic networks, as well as network embedding techniques.

Anomaly detection on streaming networks. In streaming networks, a number of methods perform anomaly detection in the context of edge streams [3, 16, 32, 33]. For instance, GOutlier introduced a structural connectivity model to define anomalies, and proposed a reservoir sampling method to maintain structural summaries of the underlying graph streams. The anomalies can then be identified as those graph objects which contain unusual bridging edges. The recent work [32] proposed an anomaly detection method based on edge scoring. The score of an incoming edge was based on historical evidence and vertex neighborhood. There is a new type of anomalies in graph streams: anomalous graph snapshots [16, 23, 33]. The StreamSpot [23] introduced a new similarity function to compare two heterogeneous graphs based on their relative frequency of local substructures, and leveraged a centroid-based clustering methods to capture the normal behaviors. Graph stream clustering algorithms like GMicro [2] created sketch-based micro clusters which using a hash-based compression of the edges to a lower-dimensional domain space in order to reduce the size of representation. Variants include graph streams with attributes [24, 43]. The communities evolve across snapshots. The evolutionary community anomalies can be defined as those objects which evolve in a very different way rather than following the community change trends [17, 18].

Network embedding. Recent advances in word embedding [8, 9, 22, 25, 26, 39, 41] open a new way to learn representations for words. In particular, the Skip-gram model learns word features by preserving the neighborhood structure extracted from words in each sentence [25, 26]. Inspired by this feature learning strategy, recent developments such as DeepWalk [31], LINE [36] and Node2Vec [15] learn vertex representations using the same way as word embedding. These methods extract "walks" which are sequences of vertices from the graph, and then learn the vertex representations by maximizing the likelihood of preserving network neighborhoods of vertices. The main difference among these methods is in the way they generate the "walks". The DeepWalk [31] created the local "walks" by truncated random walks similar to a depth-first search. The LINE [36] preserved both first-order (observed tie strength) and second-order proximities (shared neighborhood structures of the vertices). The node2vec [15] argued that different sampling strategies for vertices will result in different feature representations. It defined a flexible notion of a vertex's network neighborhood by interpolating between two extreme sampling strategies: breadth-first sampling and depth-first sampling. Some work also exists for representation learning based on novel graph-specific network architectures [9, 37, 39, 44]. Finally, work on

graph sketches and compact representations focused on constructing approximate graph descriptions in order to detect graph-based anomalies [23, 32].

7 CONCLUSION

We have presented NetWalk to detect anomalies in dynamic networks, by learning faithful network representations which can be updated dynamically as the network evolves over time. We first learn the latent network representations by using a number of network walks extracted from the initial network. The representations are obtained by clique embedding, which jointly minimizes the pairwise distance of vertex representations from each network walk, and the auto-encoder reconstruction error that serves as a global regularization. Based on the low-dimensional vertex representations, a clustering-based technique is employed to incrementally and dynamically detect network anomalies. Quantitative validation on anomaly detection task using four read-world datasets shows that NetWalk is computationally efficient and outperforms state-of-the-art techniques in anomaly detection.

ACKNOWLEDGEMENT

The work has been partially supported by NSF IIS-1313606, NIH U01HG008488, NIH R01GM115833 and NIH U54GM114833. Research of the third author was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. We thank the anonymous reviewers for their careful reading and insightful comments on our manuscript.

REFERENCES

[1] Charu C Aggarwal. 2013. *Outlier Analysis*. Springer.
[2] Charu C Aggarwal, Yuchen Zhao, and S Yu Philip. 2010. On Clustering Graph Streams.. In *SDM*. SIAM, 478–489.
[3] Charu C Aggarwal, Yuchen Zhao, and S Yu Philip. 2011. Outlier detection in graph streams. In *ICDE*. IEEE, 399–409.
[4] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. 2009. Streaming k-means approximation. In *NIPS*. 10–18.
[5] Leman Akoglu and Christos Faloutsos. 2013. Anomaly, event, and fraud detection in large network datasets. In *WSDM*. ACM, 773–774.
[6] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *PAKDD*. Springer, 410–421.
[7] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery* 29, 3 (2015), 626–688.
[8] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *JMLR* 3, Feb (2003), 1137–1155.
[9] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *SIGKDD*. ACM, 119–128.
[10] Wei Cheng, Kai Zhang, Haifeng Chen, Guofei Jiang, and Wei Wang. 2016. Ranking Causal Anomalies via Temporal and Dynamical Analysis on Vanishing Correlations. In *SIGKDD*.
[11] Graham Cormode and S Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.
[12] Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. 2014. SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives. In *NIPS*. 1646–1654.
[13] William Eberle and Lawrence Holder. 2007. Anomaly detection in data represented as graphs. *Intelligent Data Analysis* 11, 6 (2007), 663–689.
[14] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. 2010. On community outliers and their efficient detection in information networks. In *SIGKDD*. ACM, 813–822.
[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. (2016).
[16] Manish Gupta, Jing Gao, Charu C Aggarwal, and Jiawei Han. 2014. Outlier Detection for Temporal Data: A Survey. *TKDE* 9, 26 (2014), 2250–2267.
[17] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. 2012. Community trend outlier detection using soft temporal pattern mining. In *ECML/PKDD*. Springer, 692–708.
[18] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. 2012. Integrating community matching and outlier detection for mining evolutionary community outliers. In *SIGKDD*. ACM, 859–867.
[19] Manish Gupta, Arun Mallya, Subhro Roy, Jason HD Cho, and Jiawei Han. 2014. Local Learning for Mining Outlier Subgraphs from Network Datasets. In *SDM*. 73–81.
[20] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *ACM Symposium on Theory of Computing*. ACM, 604–613.
[21] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *TKDD* 1, 1 (2007), 2.
[22] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*. 2177–2185.
[23] Emaad A Manzoor, Sadegh Momeni, Venkat N Venkatakrishnan, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *KDD*.
[24] Ryan McConville, Weiru Liu, and Paul Miller. 2015. Vertex clustering of augmented graph streams. *SDM* (2015).
[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.
[27] Andrew Ng. 2011. Sparse autoencoder. *CS294A Lecture notes* 72, 2011 (2011), 1–19.
[28] Tore Opsahl and Pietro Panzarasa. 2009. Clustering in weighted networks. *Social networks* (2009), 155–163.
[29] Bryan Perozzi and Leman Akoglu. 2016. Scalable anomaly ranking of attributed neighborhoods. In *SDM*. SIAM, 207–215.
[30] Bryan Perozzi, Leman Akoglu, Patricia Iglesias Sánchez, and Emmanuel Müller. 2014. Focused Clustering and Outlier Detection in Large Attributed Graphs. In *SIGKDD*. 1346–1355.
[31] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. ACM, 701–710.
[32] Stephen Ranshous, Steve Harenberg, Kshitij Sharma, Nagiza F Samatova, et al. 2016. A Scalable Approach for Outlier Detection in Edge Streams Using Sketch-based Approximations. In *SDM*.
[33] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova. 2015. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics* 7, 3 (2015), 223–247.
[34] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.
[35] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. 2005. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*. IEEE, 8–17.
[36] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. ACM, 1067–1077.
[37] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning Deep Representations for Graph Clustering.. In *AAAI*. 1293–1299.
[38] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and computing* 17, 4 (2007), 395–416.
[39] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *SIGKDD*. ACM, 1225–1234.
[40] Wenchao Yu, Charu C Aggarwal, and Wei Wang. 2017. Temporally factorized network modeling for evolutionary network analysis. In *WSDM*. ACM, 455–464.
[41] Wenchao Yu, Guangxiang Zeng, Ping Luo, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. 2013. Embedding with autoencoder regularization. In *ECML/PKDD*. Springer, 208–223.
[42] Wayne W Zachary. 1977. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research* (1977), 452–473.
[43] Yuchen Zhao and Philip Yu. 2013. On graph stream clustering with side information. In *SDM*. SIAM, 139–150.
[44] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. (2018).