Structural Temporal Graph Neural Networks for Anomaly Detection in Dynamic Graphs

Lei Cai\textsuperscript{1,2,*}, Zhengzhang Chen\textsuperscript{2,†}, Chen Luo\textsuperscript{3}, Jiaping Gui\textsuperscript{2}, Jingchao Ni\textsuperscript{2}, Ding Li\textsuperscript{2}, Haifeng Chen\textsuperscript{2}

\textsuperscript{1}Washington State University, USA
\textsuperscript{2}NEC Laboratories America, USA
\textsuperscript{3}Rice University, USA

Abstract

Detecting anomalies in dynamic graphs is a vital task, with numerous practical applications in areas such as security, finance, and social media. Previous network embedding based methods have been mostly focusing on learning good node representations, whereas largely ignoring the subgraph structural changes related to the target nodes in dynamic graphs. In this paper, we propose StrGNN, an end-to-end structural temporal Graph Neural Network model for detecting anomalous edges in dynamic graphs. In particular, we first extract the h-hop enclosing subgraph centered on the target edge and propose the node labeling function to identify the role of each node in the subgraph. Then, we leverage graph convolution operation and Sort-pooling layer to extract the fixed-size feature from each snapshot/timestamp. Based on the extracted features, we utilize Gated recurrent units (GRUs) to capture the temporal information for anomaly detection. Extensive experiments on six benchmark datasets and a real enterprise security system demonstrate the effectiveness of StrGNN.

1 Introduction

Recent studies of dynamic graphs/networks have witnessed a growing interest. Such dynamic graphs model a variety of systems including societies, ecosystems, the Internet, and others. For example, in enterprise dynamic network [Luo et al., 2018], the node represents a system entity (such as process, file, and Internet sockets) and an edge indicates the corresponding interaction between two system entities. These dynamic networks, unlike static networks, are constantly changing. Possible changes include graph structure change or modification of node attributes.

A fundamental task on dynamic graph analysis is anomaly detection—identifying objects, relationships, or subgraphs, whose “behaviors” significantly deviate from underlying majority of the network [Aggarwal et al., 2011; Ranshous et al., 2015]. In this work, we focus on the anomalous edge detection in dynamic graphs. Detecting anomalous edges can help understand the system status and diagnose system fault [Ranshous et al., 2015; Akoglu et al., 2015]. For example, in an enterprise dynamic network, some system entity pairs, such as a user software and system-specific internet socket ports (e.g., port number \( \leq 1024 \)), never form an edge (interaction/connection) in-between in normal system environments. Once occurring, these suspicious interactions/activities may indicate some serious cyber-attack happened and could significantly damage the enterprise system [Cheng et al., 2016].

Recently, graph embedding has shown to be a powerful tool in learning the low-dimensional representations in networks that can capture and preserve the graph structure. However, most existing graph embedding approaches are designed for static graphs, and thus may not be suitable for a dynamic environment in which the network representation has to be constantly updated. Only a few advanced embedding-based methods (such as NetWalk [Yu et al., 2018]) are suitable for updating the representation dynamically as the network evolves. However, these methods require the knowledge of the nodes over the whole time span and thus can hardly promise the performance on new nodes in the future. More importantly, these methods neglect a notable characteristic of the dynamic networks—the subgraph structural changes related to the target nodes. These structural temporal dynamics are key to understanding system behavior. For example, in Figure 1, the target edge at timestamp \( t \) is marked as a double red line, and the 1-hop subgraph centered on the target edge is marked with gray. It can be seen from Figure 1 (A) that the interactions between nodes of the subgraph (i.e., gray nodes) become more frequent. Therefore, the target edge in Figure 1 (A) is reasonable to be a normal edge. In contrast, in Figure 1 (B), there is no interactions between the neighbors of the subgraph from timestamp \( t - 3 \) to \( t - 1 \). Therefore, the target edge at timestamp \( t \) is more likely to be an anomalous edge. Thus, it is critical to model and detect the structural changes over time for the anomaly detection task.

To address the aforementioned issues, we propose StrGNN, a structural graph neural network to identify anomalous edges in dynamic graphs. StrGNN is designed to detect unusual subgraph structures centered on the target edge in a given time window while considering the temporal dependency. StrGNN consists of three sub-models: ESG (En-
closing Subgraph Generation), GSFE (Graph Structural Feature Extraction), and TDN (Temporal Detection Network). First, ESG extracts a $h$-hop enclosing subgraph centered on the target edge from each graph snapshot. Subgraphs extracted based on different edges can result in the same topology structure. Thus, a node labeling function is proposed to indicate the role of each node in the subgraph. Then, GSFE module leverages Graph Convolution Neural Network and pooling technologies to extract fixed-size feature from each subgraph. Based on the extracted features, TDN employs the Gated recurrent units (GRUs) to capture the temporal dependency for anomaly detection. Different from the previous embedding based methods, the whole process of StrGNN can be trained end-to-end, i.e., StrGNN takes the test edges along with the original dynamic graphs as input and directly outputs the category (i.e., anomaly or normal) for each test edge. Moreover, our proposed StrGNN framework focuses on mining the structural temporal patterns in a given time window. Therefore, node embedding is not required to learn and StrGNN is not sensitive to the edge and vertex changes (such as new nodes) in the dynamic graphs.

We conduct extensive experiments on six benchmark datasets to evaluate the performance of StrGNN. The results demonstrate the effectiveness of our proposed algorithm. We also apply StrGNN to a real enterprise security system for intrusion detection. By using StrGNN, we can reduce false positives of the state-of-the-art methods by at least 50%, while keeping zero false negatives.

2 Related Work

In this section, we briefly introduce previous work on embedding based anomaly detection in graphs.

2.1 Anomaly Detection on Static Graphs

Inspired by word embedding methods [Mikolov et al., 2013] in natural language processing tasks, recent advances such as DeepWalk [Perozzi et al., 2014], LINE [Tang et al., 2015], and Node2Vec [Grover and Leskovec, 2016] have been proposed to learn node embedding via the skip-gram technology. The DeepWalk generates random walks for each vertex with a given length and picks the next step uniformly from the neighbors. Different from DeepWalk, the LINE [Tang et al., 2015] preserves not only the first-order (observed tie strength) relations but also the second-order proximities (shared neighborhood structures of the vertices). Node2Vec [Grover and Leskovec, 2016] uses two different sampling strategies (breadth-first sampling and depth-first sampling) for vertices that result in different feature representations. Through the network embedding technology, both anomalous node and edge detection tasks can be performed with traditional anomaly detection methods.

2.2 Anomaly Detection on Dynamic Graphs

Dynamic graphs are more complex due to the variation of the graph structure. That is, the vertices and edges are changing along the time dimension. To capture the dependency between different graphs along the time dimension, recently few network embedding based methods have been proposed [Zhou et al., 2018]. Dyngem [Goyal et al., 2018] employs the auto-encoder method to learn the embedding for each graph, and a constraint loss function is employed to minimize the difference between all graphs. Dyngem [Goyal et al., 2019] uses the Recurrent Neural Network to capture the temporal information and learn the embedding using auto-encoder technology. Recently, NetWalk [Yu et al., 2018], one of the state-of-the-art methods for anomaly detection in dynamic networks, is proposed to learn the embedding while considering the temporal dependency and detect the anomaly using the density-based method. The NetWalk generates several random walks for each vertex and learns a unified embedding for each node using auto-encoder technology. The embedding representation is updated along the time dimension.

3 Method

In this section, we introduce our method in detail. We start with the overall framework of our proposed Structural Temporal Graph Neural Networks for anomaly detection in dynamic graphs. The details of each component in our proposed method are introduced afterwards.

3.1 Overall Framework

Compared with the anomaly detection in a static graph, dynamic graphs are more complex and challenging in two perspectives: (1) The anomalous edges cannot be determined by the graph from a single timestamp. The detection procedure must take the previous graphs into consideration; (2) Both the vertex and edge sets are changing over time. To tackle these challenges, we propose StrGNN, a structural temporal Graph Neural Network framework. The key idea of our proposed method is to capture structural changes centered on the target edge in a given time window and determine the category (i.e., anomaly or normal) of the target edge based on the structural changes. Our proposed StrGNN framework consists of three key components: ESG (Enclosing Subgraph Generation), GSFE (Graph Structural Feature Extraction), and TDN (Temporal Detection Network), as illustrated in Figure 2.

3.2 ESG: Enclosing Subgraph Generation

For the first module, Enclosing Subgraph Generation, our goal is to generate enclosing subgraph structure related to
the target edge so as to detect the anomalies more efficiently. Directly employing the whole graph for analysis can be highly computational expensive, especially considering the real-world networks with thousands or even millions of nodes and edges. Recent work [Xu et al., 2018] also proved that in Graph Neural Networks, each node is most influenced by its neighbors.

Definition 1. (Enclosing subgraph in static graphs) For a static network $G = (V, E)$, given a target edge $e$ with source node $x$ and destination node $y$, the $h$-hop enclosing subgraph $G^h_{x,y}$ centered on edge $e$ can be obtained by $\{i | d(i, x) \leq h \lor d(i, y) \leq h\}$, where $d(i, x)$ is the shortest path distance between node $i$ and node $x$.

Definition 2. (Enclosing subgraph in dynamic graphs) For a temporal network $\{G(i) = (V(i), E(i))\}_{t=1}^T$ with window size $w$, given a target edge $e^t$ with source node $x^t$ and destination node $y^t$, the $h$-hop enclosing subgraph $G^h_{x^t,y^t}$ centered on edge $e^t$ is a collection of all subgraph centered on $e^t$ in the temporal network $\{G(i)_{x^t,y^t} | t-w \leq i \leq t\}$.

For a target edge $e^t$, we extract the enclosing subgraph in dynamic graphs based on Definition 2. However, the extracted subgraph only contains topological information. Subgraphs extracted based on different edges can result in the same topological structure. To distinguish the role of each node in the subgraph, in this work, we propose to annotate the nodes in the subgraph with different labels. A good node labeling function should convey the following information: 1) which edge is the target edge in the current subgraph, and 2) the contribution of each node in identifying the category of each edge. More specifically, given the edge $e^t$ and the corresponding source and destination node $x^t$ and $y^t$, our node labeling function for the enclosing subgraph $G(i)^h_{x^t,y^t}$ is defined as follows:

\[
f(i, x^t, y^t) = 1 + \min(d(i, x^t), d(i, y^t)) + (d_{sum}/2)\left((d_{sum}/2) + (d_{sum}\%2) - 1\right),
\]

where $d(i, x^t)$ is the shortest path distance between node $i$ and node $x^t$, and $d_{sum} = d(i, x^t) + d(i, y^t)$. In addition, the two center nodes are labeled with 1. If a node $i$ satisfies $d(i, x^t) = \infty$ or $d(i, y^t) = \infty$, it will be labeled as 0. The label will be converted into a one-hot vector as the attribute $X$ for each node. By employing the node labeling function, we can generate the label for each node, which can represent structure information for the given subgraph. The category of the target edge $e^t$ at timestamp $t$ can be predicted by analyzing the labeled subgraph in the given time window.

3.3 GSFE: Graph Structural Feature Extraction

To analyze the structure of each enclosing subgraph from the given time period, the Graph Convolution Neural Network (GCN) [Kipf and Welling, 2016] can be employed to project the subgraph into an embedding space. In GCN, the graph convolution layer was proposed to learn the embedding of each node in the graph and aggregate the embedding from its neighbors. The layer-wise forward operation of graph convolution layer can be described as follows:

\[
G(X, A) = \sigma(\hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}XW),
\]

where $\hat{A} = A + I$ is the summation of the adjacency matrix and identity matrix, $\sigma(\cdot)$ denotes an activation function, such as the ReLU ($\cdot$) = max(0, $\cdot$), and $W$ is the trainable weight matrix. By employing the graph convolution layer, each node can aggregate the embedding from its neighbors. By stacking the graph convolution layer in the neural network, each node can obtain more information from other nodes. For example, each node can obtain information from its 2-hop neighbors by stacking two graph convolution layers.

GCN can generate node embedding for detecting anomalous edges in a single graph. However, in our dynamic graph setting, the anomalies should be determined in the context of $\{G(i)^h_{x^t,y^t} | t-w \leq i \leq t\}$. The number of nodes in different enclosing subgraphs is commonly different, thus results
in different sizes of the feature vector in different subgraphs. Therefore, it is challenging to analyze the dynamic graphs using Graph Neural Networks due to the various sizes of the input.

To tackle this problem, we leverage the graph pooling technology to extract the fixed-size feature for each enclosing subgraph. Any graph pooling method can be employed in our proposed StrGNN framework to extract the fixed-size feature for further analysis. In this work, we employ the Sortpooling layer proposed by [Zhang et al., 2018], which can sort the nodes in the enclosing subgraph based on their importance and select the feature from the top K nodes.

Given the node embedding $H_i$ corresponding to graph $G(i)_{x^i_t, y^i_t}$, the importance score for each node in the Sortpooling layer is defined as follows:

$$ S(H_i, A) = \sigma(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H_i W^1), $$

where $A$ is the adjacency matrix of graph $G(i)_{x^i_t, y^i_t}$, and $W^1$ is the projection matrix with output channel 1. Each node can obtain the importance score by using Equation 3. All nodes in the enclosing subgraph will be sorted in order of the importance score. And only the top K nodes will be selected for further analysis. For the subgraphs that contain less than K nodes, the zero-padding will be employed to guarantee that each subgraph contains the same fixed-size feature.

### 3.4 TDN: Temporal Detection Network

The Graph Structural Feature Extraction module can generate low-dimensional features for anomaly detection. However, it does not consider the temporal information, which is of great importance for determining the category (i.e., anomaly or normal) of an edge in the dynamic setting.

Given the extracted structural feature $\{H_i\}_{i=t-w}^t \in R^{K \times d}$, where $K$ is the number of selected nodes in each graph, and $d$ is the dimension of feature for each node, in this work, we employ the Gated recurrent units (GRUs) [Chung et al., 2014], which can alleviate the vanishing and exploding gradient problems [Goodfellow et al., 2016], to capture the temporal information as:

$$ z_t = \sigma(W_z \tilde{H}_t + U_z h_{t-1} + b_z) $$

$$ r_t = \sigma(W_r \tilde{H}_t + U_r h_{t-1} + b_r) $$

$$ h'_t = \tanh(W_h \tilde{H}_t + U_h (r_t \circ h_{t-1}) + b_h) $$

$$ h_t = z_t \circ h_{t-1} + (1 - z_t) \circ h'_t, $$

where $\circ$ represents the element-wise product operation, $W$, $U$, and $b$ are parameters. The GRU network takes the feature at each timestamp as input, and feeds the output of current timestamp into the next timestamp. Therefore, the temporal information can be modeled by the GRU network. The output of last timestamp $h_t$ is employed to analyze the category of the target edge $e^i_t$. The anomalous edge detection problem can be formulated as follows:

$$ L = -(y^i \log(g(h_t)) + (1 - y^i) \log(1 - g(h_t))), $$

where $g(\cdot)$ is a fully connected network, and $y^i$ is the category of edge $e^i_t$.

For the anomaly detection task, in many real-world cases, the dataset does not contain any anomalous samples or only contain a small number of anomalous samples.

One straightforward way of generating negative samples is to draw samples from a “context-independent” noise distribution (such as Random sampling or injected sampling [Akoglu et al., 2015]), where a negative sample is independently and does not depend on the observed samples. However, due to the large anomalous edge space, this noise distribution would be very different from the data distribution, which would lead to poor model learning. Thus, in this work, we propose “context-dependent” negative sampling strategy.

The intuition behind our strategy is to generate negative samples from “context-dependent” noise distribution. Here, the “context-dependent” noise distribution for the sampled data $E'$ is defined as: $P_{E'} \sim P(E) \ast (\frac{1}{N - |E|})$, where $P(E)$ denotes the observed data distribution, $|E|$ is the number of edges in the graph, and $N$ is the number of nodes in the graph. Specifically, we first randomly sample a normal vertex pair $e = \{x_a, x_b\}$ in the graph. Then, we replace one of the nodes, say $x_a$ with a randomly sampled node $x'$ in the graph and form a new negative sample $e' = \{x', x_b\}$. If $e'$ is not belongs to the normal graph, we retain the sample, otherwise, we delete it.

The proposed StrGNN framework is quite flexible and easy to be customized. Any network that can capture the temporal information can be used in our proposed framework, such as Convolution Neural Network (CNN) and Vanilla Recurrent Neural Network (RNN).

### 4 Experiments

In this section, we evaluate StrGNN on six benchmark datasets and a real enterprise network.

#### 4.1 Datasets

We conduct experiments on six public datasets from different domains. The UCI Messages dataset [Opsahl and Panzarasa, 2009] is collected from an online community platform of students at the University of California, Irvine. Each node in the constructed graph represents a user in the platform. And the edge indicates that there is a message interaction between two users. The Digg dataset [De Choudhury et al., 2009] is collected from a news website digg.com. Each node represents a user of the website, and each edge represents a reply between two users. The Email dataset is a dump of emails of Democratic National Committee. Each node corresponds to a person. And the edge indicates an email communication between two persons. The Topology [Zhang et al., 2005] dataset is the network connections between autonomous systems of the Internet. Nodes are autonomous systems, and edges are connections between autonomous systems. The Bitcoin-alpha and Bitcoin-otc [Kumar et al., 2016; Kumar et al., 2018] datasets are collected from two Bitcoin platform named Alpha and OTC, respectively. Nodes represent users from the platform. If one user rates another user on the platform, there is an edge between them.
4.2 Baselines

We compare StrGNN with four network embedding based baselines.

**DeepWalk** [Perozzi et al., 2014]: DeepWalk generates the random walks with given length starting from a node and learns the embedding using Skip-gram.

**Node2Vec** [Grover and Leskovec, 2016]: Node2Vec combines breadth-first traversal and depth-first traversal in the random walks generation procedure. The embedding is learned using Skip-gram technology.

**Spectral Clustering** [von Luxburg, 2007]: To preserve the local connection relationship, the spectral embedding generates the node embedding by maximizing the similarity between nodes in the neighborhood.

**NetWalk** [Yu et al., 2018]: NetWalk generates several random walks for each vertex and learns a unified embedding for each node using auto-encoder technology. The embedding representation will be updated along the time dimension. The framework, using an extended temporal GCN with an attention-based GRU.

For the first three baselines, after representation learning, the same K-means clustering based method [Yu et al., 2018] (as in NetWalk) is used for anomaly detection.

4.3 Experiment Setup

The parameters of StrGNN can be tuned by 5-fold cross-validation on a rolling basis. Here, by default, we set the window size $w$ to 5 and the number of hops $h$ in enclosing subgraph to 1. We evaluate the influence of each parameter. The AUC results of StrGNN with different $h$ on UCI Messages are shown in Table 1. StrGNN with 2-hop or 3-hop subgraph achieves similar performance as 1-hop but requiring way more computational cost. The parameter $w$ (with $w \geq 5$) shares similar influence as $h$ on the performance of StrGNN.

We employ a Graph Neural Network with three graph convolution layers to extract graph features. The size of the output feature map is set to 32 for all three layers. The outputs of all three layers are concatenated as the embedding feature. The selected rate in the Sortpooling layer is set to 0.6. In terms of the temporal neural network, the hidden size of GRU is set to 256. We employ Adam method [Kingma and Ba, 2014] to train the network. The learning rate of Adam is set to $1e^{-4}$. We employ batch training in the experiments and the batch size is set to 32 for our proposed StrGNN method. StrGNN is end-to-end trained for 50 epochs. We use the first 50% edges as the training dataset, and the rest as the test dataset. Since the anomalous edges do not exist in the six benchmark datasets, we follow the approach used in [Yu et al., 2018] to inject 1%, 5%, 10% anomalous edges in the test dataset to evaluate the performance of each model. The metric used to compare the performance of different methods is AUC (the area under the ROC curve). The higher AUC value indicates the high quality of the method.

4.4 Results on Benchmark Datasets

We first compare StrGNN with the baseline methods on six benchmark datasets. The experimental results in Table 2 show that StrGNN outperforms all four baseline methods on all the benchmark datasets. And even if 10% anomalies are injected, the performance of StrGNN is still acceptable. This outstanding effect proves that StrGNN can exploit the structural and temporal features effectively and the learned representation of the dynamic graph structure is well suited for the anomaly detection task.

To further demonstrate the effectiveness of our
Table 2: AUC comparison on benchmark datasets.

| Methods   | UCI          | Digg          | Email          |
|-----------|--------------|---------------|----------------|
|           | 1% 5% 10%    | 1% 5% 10%     | 1% 5% 10%      |
| Node2Vec  | 0.7371 0.7433 0.6960 | 0.7364 0.7081 0.6508 | 0.7391 0.7284 0.7103 |
| Spectral Clustering | 0.6324 0.6104 0.5794 | 0.5949 0.5823 0.5591 | 0.8096 0.7857 0.7759 |
| DeepWalk  | 0.7514 0.7391 0.6979 | 0.7080 0.6881 0.6396 | 0.7481 0.7303 0.7197 |
| NetWalk   | 0.7758 0.7647 0.7226 | 0.7563 0.7176 0.6837 | 0.8105 0.8371 0.8305 |
| StrGNN    | **0.8179 0.8252 0.7959** | **0.8162 0.8254 0.8272** | **0.8775 0.9103 0.9080** |

Table 3: Results on intrusion detection.

| Method    | AUC  |
|-----------|------|
| Node2Vec  | 0.71 |
| DeepWalk  | 0.76 |
| Spectral Clustering | 0.75 |
| NetWalk   | 0.90 |
| StrGNN    | **0.99** |

5 Conclusion

In this paper, we investigated an important and challenging problem of anomaly detection in dynamic graphs. Different from network embedding based methods that focus on learning good node representations, we proposed StrGNN, a structural temporal Graph Neural Network to detect anomalous edges by mining the unusual temporal subgraph structures. StrGNN can be trained end-to-end and it is not sensitive to the percentage of anomalies. We evaluated the proposed framework using extensive experiments on six benchmark datasets. The experimental results convince us of the effectiveness of our approach. We also applied StrGNN to a real enterprise security system for intrusion detection. Our method achieved superior detection performance with zero false negatives.

References

[Aggarwal et al., 2011] Charu C. Aggarwal, Yuchen Zhao, and Philip S. Yu. Outlier detection in graph streams. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering, page 399–409, USA, 2011.

[Akoglu et al., 2015] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. Data Mining and Knowledge Discovery, 29(3):626–688, 2015.

[Cheng et al., 2016] Wei Cheng, Kai Zhang, Haifeng Chen, Guofei Jiang, Zhengzhang Chen, and Wei Wang. Ranking causal anomalies via temporal and dynamical analysis...
on vanishing correlations. In SIGKDD, pages 805–814. ACM, 2016.

[Chung et al., 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.

[De Choudhury et al., 2009] Munmun De Choudhury, Hari Sundaram, Ajita John, and Dorée Duncan Seligmann. Social synchrony: Predicting mimicry of user actions in online social media. In CSE, volume 4, pages 151–158. IEEE, 2009.

[Goodfellow et al., 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[Goyal et al., 2018] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. arXiv preprint arXiv:1805.11273, 2018.

[Chung et al., 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.

[Goyal et al., 2019] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowledge-Based Systems, 2019.

[Aditya Grover and Jure Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In SIGKDD, pages 855–864. ACM, 2016.

[Diederek P Kingma and Jimmy Ba, 2014] Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos, 2016] Edge weight prediction in weighted signed networks. In ICDM, pages 221–230. IEEE, 2016.

[Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian, 2018] Rev2: Fraudulent user prediction in rating platforms. In WSDM, pages 333–341. ACM, 2018.

[Chen Luo, Zhengzhang Chen, Lu-An Tang, Anshumali Shrivastava, Zhichun Li, Haifeng Chen, and Jieping Ye, 2018] Tinet: Learning invariant networks via knowledge transfer. In SIGKDD, pages 1890–1899. ACM, 2018.

[Mikolov et al., 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In NIPS, pages 3111–3119, 2013.

[Michael Tore Opsahl and Pietro Panzarasa, 2009] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. Social Networks, 31(2):155–163, 2009.

[Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, 2014] Deepwalk: Online learning of social representations. In SIGKDD, pages 701–710. ACM, 2014.

[Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F. Samatova, 2018] Anomaly detection in dynamic networks: A survey. WIREs Comput. Stat., 7(3):223–247, May 2015.

[Stephen Ranshous, Shitian Shen, Christos Faloutsos, and Nagiza F. Samatova, 2018] Anomaly detection in dynamic networks: A survey. WIREs Comput. Stat., 7(3):223–247, May 2015.

[Chung et al., 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555, 2014.

[Munmun De Choudhury, Hari Sundaram, Ajita John, and Dorée Duncan Seligmann, 2009] Social synchrony: Predicting mimicry of user actions in online social media. In CSE, volume 4, pages 151–158. IEEE, 2009.

[Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu, 2018] Dyngem: Deep embedding method for dynamic graphs. arXiv preprint arXiv:1805.11273, 2018.

[Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo, 2019] dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. Knowledge-Based Systems, 2019.

[Aditya Grover and Jure Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In SIGKDD, pages 855–864. ACM, 2016.

[Diederek P Kingma and Jimmy Ba, 2014] Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos, 2016] Edge weight prediction in weighted signed networks. In ICDM, pages 221–230. IEEE, 2016.

[Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian, 2018] Rev2: Fraudulent user prediction in rating platforms. In WSDM, pages 333–341. ACM, 2018.

[Chen Luo, Zhengzhang Chen, Lu-An Tang, Anshumali Shrivastava, Zhichun Li, Haifeng Chen, and Jieping Ye, 2018] Tinet: Learning invariant networks via knowledge transfer. In SIGKDD, pages 1890–1899. ACM, 2018.

[Mikolov et al., 2013] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In NIPS, pages 3111–3119, 2013.

[Michael Tore Opsahl and Pietro Panzarasa, 2009] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. Social Networks, 31(2):155–163, 2009.

[Bryan Perozzi, Rami Al-Rfou, and Steven Skiena, 2014] Deepwalk: Online learning of social representations. In SIGKDD, pages 701–710. ACM, 2014.