Graph Neural Network-Based Anomaly Detection in Multivariate Time Series

Ailin Deng, Bryan Hooi
National University of Singapore
ailin@comp.nus.edu.sg, bhooi@comp.nus.edu.sg

Abstract

Given high-dimensional time series data (e.g., sensor data), how can we detect anomalous events, such as system faults and attacks? More challengingly, how can we do this in a way that captures complex inter-sensor relationships, and detects and explains anomalies which deviate from these relationships? Recently, deep learning approaches have enabled improvements in anomaly detection in high-dimensional datasets; however, existing methods do not explicitly learn the structure of existing relationships between variables, or use them to predict the expected behavior of time series. Our approach combines a structure learning approach with graph neural networks, additionally using attention weights to provide explainability for the detected anomalies. Experiments on two real-world sensor datasets with ground truth anomalies show that our method detects anomalies more accurately than baseline approaches, accurately captures correlations between sensors, and allows users to deduce the root cause of a detected anomaly.

1 Introduction

With the rapid growth in interconnected devices and sensors in Cyber-Physical Systems (CPS) such as vehicles, industrial systems and data centres, there is an increasing need to monitor these devices to secure them against attacks. This is particularly the case for critical infrastructures such as power grids, water treatment plants, transportation, and communication networks.

Many such real-world systems involve large numbers of interconnected sensors which generate substantial amounts of time series data. For instance, in a water treatment plant, there can be numerous sensors measuring water level, flow rates, water quality, valve status, and so on, in each of their many components. Data from these sensors can be related in complex, nonlinear ways: for example, opening a valve results in changes in pressure and flow rate, leading to further changes as automated mechanisms respond to the change.

As the complexity and dimensionality of such sensor data grow, humans are increasingly less able to manually monitor this data. This necessitates automated anomaly detection approaches which can rapidly detect anomalies in high-dimensional data, and explain them to human operators to allow them to diagnose and respond to the anomaly as quickly as possible.

Due to the inherent lack of labeled anomalies in historical data, and the unpredictable and highly varied nature of anomalies, the anomaly detection problem is typically treated as an unsupervised learning problem. In past years, many classical unsupervised approaches have been developed, including linear model-based approaches (Shyu et al. 2003), distance-based methods (Angiulli and Pizzuti 2002), and one-class methods based on support vector machines (Schölkopf et al. 2001). However, such approaches generally model inter-relationships between sensors in relatively simple ways: for example, capturing only linear relationships, which is insufficient for complex, highly nonlinear relationships in many real-world settings.

Recently, deep learning-based techniques have enabled improvements in anomaly detection in high-dimensional datasets. For instance, Autoencoders (AE) (Aggarwal 2015) are a popular approach for anomaly detection which uses reconstruction error as an outlier score. More recently, Generative Adversarial Networks (GANs) (Li et al. 2019) and LSTM-based approaches (Qin et al. 2017) have also reported promising performance for multivariate anomaly detection. However, most methods do not explicitly learn which sensors are related to one another, thus facing difficulties in modelling sensor data with many potential inter-relationships. This limits their ability to detect and explain deviations from such relationships when anomalous events occur.

How do we take full advantage of the complex relationships between sensors in multivariate time series? Recently, graph neural networks (GNNs) (Defferrard, Bresson, and Vandergheynst 2016) have shown success in modelling graph-structured data. These include graph convolution networks (GCNs) (Kipf and Welling 2016), graph attention networks (GATs) (Veličković et al. 2017) and multi-relational approaches (Schlichtkrull et al. 2018). However, applying them to time series anomaly detection requires overcoming two main challenges. Firstly, different sensors have very different behaviors: e.g. one may measure water pressure, while another measures flow rate. However, typical GNNs use the same model parameters to model the behavior of each node. Secondly, in our setting, the graph edges (i.e. relationships between sensors) are initially unknown, and have

Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

to be learned along with our model, while GNNs typically treat the graph as an input.

Hence, in this work, we propose our novel Graph Deviation Network (GDN) approach, which learns a graph of relationships between sensors, and detects deviations from these patterns[1]. Our method involves four main components: 1) **Sensor Embedding**, which uses embedding vectors to flexibly capture the unique characteristics of each sensor; 2) **Graph Structure Learning** learns the relationships between pairs of sensors, and encodes them as edges in a graph; 3) **Graph Attention-Based Forecasting** learns to predict the future behavior of a sensor based on an attention function over its neighboring sensors in the graph; 4) **Graph Deviation Scoring** identifies and explains deviations from the learned sensor relationships in the graph.

To summarize, the main contributions of our work are:

- We propose GDN, a novel attention-based graph neural network approach which learns a graph of the dependence relationships between sensors, and identifies and explains deviations from these relationships.
- We conduct experiments on two water treatment plant datasets with ground truth anomalies. Our results demonstrate that GDN detects anomalies more accurately than baseline approaches.
- We show using case studies that GDN provides an explainable model through its embeddings and its learned graph. We show that it helps to explain an anomaly, based on the subgraph over which a deviation is detected, attention weights, and by comparing the predicted and actual behavior on these sensors.

2 Related Work

We first review methods for anomaly detection, and methods for multivariate time series data, including graph-based approaches. Since our approach relies on graph neural networks, we summarize related work in this topic as well.

Anomaly Detection Anomaly detection aims to detect unusual samples which deviate from the majority of the data. Classical methods include density-based approaches (Breunig et al. 2000), linear-model based approaches (Shyu et al. 2003), distance-based methods (Angiulli and Pizzuti 2002), classification models (Schölkopf et al. 2001), detector ensembles (Lazarevic and Kumar 2005) and many others.

More recently, deep learning methods have achieved improvements in anomaly detection in high-dimensional datasets. These include approaches such as autoencoders (AE) (Aggarwal 2015), which use reconstruction error as an anomaly score, and related variants such as variational autoencoders (VAEs) (Kingma and Welling 2013), which develop a probabilistic approach, and autoencoders combining with Gaussian mixture modelling (Zong et al. 2018).

However, our goal is to develop specific approaches for multivariate time series data, explicitly capturing the graph of relationships between sensors.

[1]The code is available at https://github.com/d-ailin/GDN

Multivariate Time Series Modelling These approaches generally model the behavior of a multivariate time series based on its past behavior. A comprehensive summary is given in (Blázquez-García et al. 2020).

Classical methods include auto-regressive models (Hautamaki, Karkkainen, and Franti 2004) and the auto-regressive integrated moving average (ARIMA) models (Zhang et al. 2012; Zhou et al. 2018), based on a linear model given the past values of the series. However, their linearity makes them unable to model complex nonlinear characteristics in time series, which we are interested in.

To learn representations for nonlinear high-dimensional time series and predict time series data, deep learning-based time series methods have attracted interest. These techniques, such as Convolutional Neural Network (CNN) based models (Munir et al. 2018), Long Short Term Memory (LSTM) (Filonov, Lavrentyev, and Vorontsov 2016; Hundman et al. 2018a; Park, Hoshi, and Kemp 2018) and Generative Adversarial Networks (GAN) models (Zhou et al. 2019; Li et al. 2019), have found success in practical time series tasks. However, they do not explicitly learn the relationships between different time series, which are meaningful for anomaly detection: for example, they can be used to diagnose anomalies by identifying deviations from these relationships.

Graph-based methods provide a way to model the relationships between sensors by representing the interdependencies with edges. Such methods include probabilistic graphical models, which encode joint probability distributions, as described in (Bach and Jordan 2004; Tank, Foti, and Fox 2015). However, most existing methods are designed to handle stationary time series, and have difficulty modelling more complex and highly non-stationary time series arising from sensor settings.

Graph Neural Networks In recent years, graph neural networks (GNNs) have emerged as successful approaches for modelling complex patterns in graph-structured data. In general, GNNs assume that the state of a node is influenced by the states of its neighbors. Graph Convolution Networks (GCNs) (Kipf and Welling 2016) model a node's feature representation by aggregating the representations of its one-step neighbors. Building on this approach, graph attention networks (GATs) (Veličković et al. 2017) use an attention function to compute different weights for different neighbors during this aggregation. Related variants have shown success in time-dependent problems: for example, GNN-based models can perform well in traffic prediction tasks (Yu, Yin, and Zhu 2017; Chen et al. 2019). Applications in recommendation systems (Lim et al. 2020; Schlichtkrull et al. 2018) and relative applications (Wang et al. 2020) verify the effectiveness of GNN to model large-scale multi-relational data.

However, these approaches use the same model parameters to model the behavior of each node, and hence face limitations in representing very different behaviors of different sensors. Moreover, GNNs typically require the graph structure as an input, whereas the graph structure is initially unknown in our setting, and needs to be learned from data.

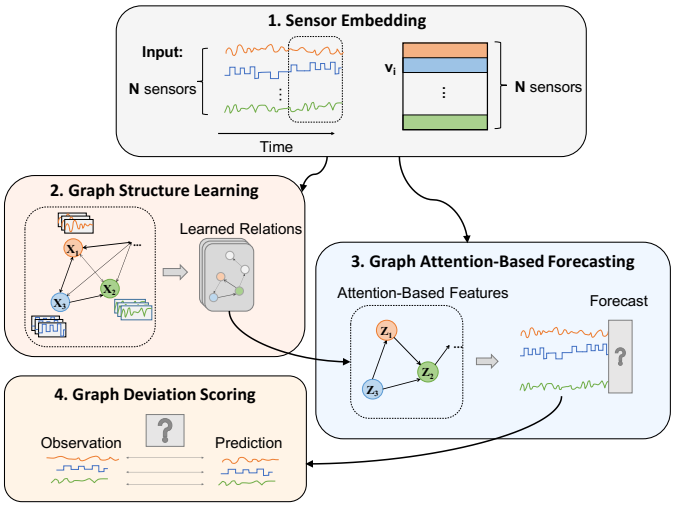

Figure 1: Overview of our proposed framework.

3 Proposed Framework

3.1 Problem Statement

In this paper, our training data consists of sensor (i.e. multivariate time series) data from N sensors over T_{train} time ticks: the sensor data is denoted $\mathbf{s}_{\text{train}} = \left[\mathbf{s}_{\text{train}}^{(1)}, \cdots, \mathbf{s}_{\text{train}}^{(T_{\text{train}})}\right]$, which is used to train our approach. In each time tick t, the sensor values $\mathbf{s}_{\text{train}}^{(t)} \in \mathbb{R}^N$ form an N dimensional vector representing the values of our N sensors. Following the usual unsupervised anomaly detection formulation, the training data is assumed to consist of only normal data.

Our goal is to detect anomalies in testing data, which comes from the same N sensors but over a separate set of T_{test} time ticks: the test data is denoted $\mathbf{s}_{\text{test}} = \left[\mathbf{s}_{\text{test}}^{(1)}, \cdots, \mathbf{s}_{\text{test}}^{(T_{\text{test}})}\right]$.

The output of our algorithm is a set of T_{test} binary labels indicating whether each test time tick is an anomaly or not, i.e. $\mathsf{a}(t) \in \{0, 1\}$, where $\mathsf{a}(t) = 1$ indicates that time t is anomalous.

3.2 Overview

Our GDN method aims to learn relationships between sensors as a graph, and then identifies and explains deviations from the learned patterns. It involves four main components:

1. **Sensor Embedding**: uses embedding vectors to capture the unique characteristics of each sensor;
2. **Graph Structure Learning**: learns a graph structure representing dependence relationships between sensors;
3. **Graph Attention-Based Forecasting**: forecasts future values of each sensor based on a graph attention function over its neighbors;
4. **Graph Deviation Scoring**: identifies deviations from the learned relationships, and localizes and explains these deviations.

Figure 1 provides an overview of our framework.

3.3 Sensor Embedding

In many sensor data settings, different sensors can have very different characteristics, and these characteristics can be related in complex ways. For example, imagine we have two water tanks, each containing a sensor measuring the water level in the tank, and a sensor measuring the water quality in the tank. Then, it is plausible that the two water level sensors would behave similarly, and the two water quality sensors would behave similarly. However, it is equally plausible that sensors within the same tank would exhibit strong correlations. Hence, ideally, we would want to represent each sensor in a flexible way that captures the different 'factors' underlying its behavior in a multidimensional way.

Hence, we do this by introducing an **embedding vector** for each sensor, representing its characteristics:

$$\mathbf{v_i} \in \mathbb{R}^d, \text{ for } i \in \{1, 2, \cdots, N\}$$

These embeddings are initialized randomly and then trained along with the rest of the model.

Similarity between these embeddings $\mathbf{v_i}$ indicates similarity of behaviors: hence, sensors with similar embedding values should have a high tendency to be related to one another. In our model, these embeddings will be used in two ways: 1) for structure learning, to determine which sensors are related to one another, and 2) in our attention mechanism, to perform attention over neighbors in a way that allows heterogeneous effects for different types of sensors.

3.4 Graph Structure Learning

A major goal of our framework is to learn the relationships between sensors in the form of a graph structure. To do this, we will use a **directed graph**, whose nodes represent sensors, and whose edges represent dependency relationships between them. An edge from one sensor to another indicates that the first sensor is used for modelling the behavior of the second sensor. We use a directed graph because the dependency patterns between sensors need not be symmetric. We use an adjacency matrix A to represent this directed graph, where A_{ij} represents the presence of a directed edge from node i to node j.

We design a flexible framework which can be applied either to 1) the usual case where we have no prior information about the graph structure, or 2) the case where we have some prior information about which edges are plausible (e.g. the sensor system may be divided into parts, where sensors in different parts have minimal interaction).

This prior information can be flexibly represented as a set of **candidate relations** $\mathcal{C}_i$ for each sensor i, i.e. the sensors it could be dependent on:

$$\mathcal{C}_i \subseteq \{1, 2, \cdots, N\} \setminus \{i\} \tag{1}$$

In the case without prior information, the candidate relations of sensor i is simply all sensors, other than itself.

To select the dependencies of sensor i among these candidates, we compute the similarity between node i's embedding vector, and the embeddings of its candidates $j \in \mathcal{C}_i$:

$$e_{ji} = \frac{\mathbf{v_i}^\top \mathbf{v_j}}{\|\mathbf{v_i}\| \cdot \|\mathbf{v_j}\|} \text{ for } j \in \mathcal{C}_i \tag{2}$$

$$A_{ji} = \mathbb{1}\{j \in \mathsf{TopK}(\{e_{ki} : k \in \mathcal{C}_i\})\} \tag{3}$$

That is, we first compute e_{ji}, the normalized dot product between the embedding vectors of sensor i, and the candidate relation $j \in \mathcal{C}_i$. Then, we select the top k such normalized dot products: here TopK denotes the indices of top-k values among its input (i.e. the normalized dot products). The value of k can be chosen by the user according to the desired sparsity level. Next, we will define our graph attention-based model which makes use of this learned adjacency matrix A.

3.5 Graph Attention-Based Forecasting

In order to provide useful explanations for anomalies, we would like our model to tell us:

- Which sensors are deviating from normal behavior?
- In what ways are they deviating from normal behavior?

To achieve these goals, we use a **forecasting**-based approach, where we forecast the expected behavior of each sensor at each time based on the past. This allows the user to easily identify the sensors which deviate greatly from their expected behavior. Moreover, the user can compare the expected and observed behavior of each sensor, to understand why the model regards a sensor as anomalous.

Thus, at time t, we define our model input $\mathbf{x}^{(t)} \in \mathbb{R}^{N \times w}$ based on a sliding window of size w over the historical time series data (whether training or testing data):

$$\mathbf{x}^{(t)} := \left[\mathbf{s}^{(\mathbf{t-w})}, \mathbf{s}^{(\mathbf{t-w+1})}, \cdots, \mathbf{s}^{(\mathbf{t-1})}\right] \tag{4}$$

The target output that our model needs to predict is the sensor data at the current time tick, i.e. $\mathbf{s}^{(\mathbf{t})}$.

Feature Extractor To capture the relationships between sensors, we introduce a graph attention-based feature extractor to fuse a node's information with its neighbors based on the learned graph structure. Unlike existing graph attention mechanisms, our feature extractor incorporates the sensor embedding vectors $\mathbf{v}_i$, which characterize the different behaviors of different types of sensors. To do this, we compute node i's aggregated representation $\mathbf{z}_i$ as follows:

$$\mathbf{z}_i^{(t)} = \mathsf{ReLU}\left(\alpha_{i,i}\mathbf{W}\mathbf{x}_i^{(t)} + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\mathbf{W}\mathbf{x}_j^{(t)}\right), \tag{5}$$

where $\mathbf{x}_i^{(t)} \in \mathbb{R}^w$ is node i's input feature, $\mathcal{N}(i) = \{j \mid A_{ji} > 0\}$ is the set of neighbors of node i obtained from the learned adjacency matrix A, $\mathbf{W} \in \mathbb{R}^{d \times w}$ is a trainable weight matrix which applies a shared linear transformation to every node, and the attention coefficients $\alpha_{i,j}$ are computed as:

$$\mathbf{g}_i^{(t)} = \mathbf{v}_i \oplus \mathbf{W}\mathbf{x}_i^{(t)} \tag{6}$$

$$\pi(i,j) = \mathsf{LeakyReLU}\left(\mathbf{a}^\top\left(\mathbf{g}_i^{(t)} \oplus \mathbf{g}_j^{(t)}\right)\right) \tag{7}$$

$$\alpha_{i,j} = \frac{\exp(\pi(i,j))}{\sum_{k \in \mathcal{N}(i) \cup \{i\}} \exp(\pi(i,k))}, \tag{8}$$

where $\oplus$ denotes concatenation; thus $\mathbf{g}_i^{(t)}$ concatenates the sensor embedding $\mathbf{v}_i$ and the corresponding transformed feature $\mathbf{W}\mathbf{x}_i^{(t)}$, and $\mathbf{a}$ is a vector of learned coefficients for the attention mechanism. We use $\mathsf{LeakyReLU}$ as the nonlinear activation to compute the attention coefficient, and normalize the attention coefficents using the softmax function in Eq. (8).

Output Layer From the above feature extractor, we obtain representations for all N nodes, namely $\{\mathbf{z}_1^{(t)}, \cdots, \mathbf{z}_N^{(t)}\}$. For each $\mathbf{z}_i^{(t)}$, we element-wise multiply (denoted $\circ$) it with the corresponding time series embedding $\mathbf{v}_i$, and use the results across all nodes as the input of stacked fully-connected layers with output dimensionality N, to predict the vector of sensor values at time step t, i.e. $\mathbf{s}^{(\mathbf{t})}$:

$$\hat{\mathbf{s}}^{(\mathbf{t})} = f_\theta\left(\left[\mathbf{v}_1 \circ \mathbf{z}_1^{(t)}, \cdots, \mathbf{v}_N \circ \mathbf{z}_N^{(t)}\right]\right) \tag{9}$$

The model's predicted output is denoted as $\hat{\mathbf{s}}^{(\mathbf{t})}$. We use the Mean Squared Error between the predicted output $\hat{\mathbf{s}}^{(\mathbf{t})}$ and the observed data, $\mathbf{s}^{(\mathbf{t})}$, as the loss function for minimization:

$$L_{\text{MSE}} = \frac{1}{T_{\text{train}} - w} \sum_{t=w+1}^{T_{\text{train}}} \left\|\hat{\mathbf{s}}^{(\mathbf{t})} - \mathbf{s}^{(\mathbf{t})}\right\|_2^2 \tag{10}$$

3.6 Graph Deviation Scoring

Given the learned relationships, we want to detect and explain anomalies which deviate from these relationships. To do this, our model computes individual anomalousness scores for each sensor, and also combines them into a single anomalousness score for each time tick, thus allowing the user to localize which sensors are anomalous, as we will show in our experiments.

The anomalousness score compares the expected behavior at time t to the observed behavior, computing an error value Err at time t and sensor i:

$$\mathsf{Err}_i(t) = |\mathbf{s}_\mathbf{i}^{(\mathbf{t})} - \hat{\mathbf{s}}_\mathbf{i}^{(\mathbf{t})}| \tag{11}$$

As different sensors can have very different characteristics, their deviation values may also have very different scales. To prevent the deviations arising from any one sensor from being overly dominant over the other sensors, we perform a robust normalization of the error values of each sensor:

$$a_i(t) = \frac{\mathsf{Err}_i(t) - \widetilde{\mu}_i}{\widetilde{\sigma}_i}, \tag{12}$$

where $\widetilde{\mu}_i$ and $\widetilde{\sigma}_i$ are the median and inter-quartile range (IQR[2]) across time ticks of the $\mathsf{Err}_i(t)$ values respectively. We use median and IQR instead of mean and standard deviation as they are more robust against anomalies.

Then, to compute the overall anomalousness at time tick t, we aggregate over sensors using the max function (we use max as it is plausible for anomalies to affect only a small subset of sensors, or even a single sensor) :

$$A(t) = \max_i a_i(t) \tag{13}$$

[2]IQR is defined as the difference between the 1st and 3rd quartiles of a distribution or set of values, and is a robust measure of the distribution's spread.

To dampen abrupt changes in values are often not perfectly predicted and result in sharp spikes in error values even when this behavior is normal, similar with (Hundman et al. 2018b), we use a simple moving average(SMA) to generate the smoothed scores $A_s(t)$.

Finally, a time tick t is labelled as an anomaly if $A_s(t)$ exceeds a fixed threshold. While different approaches could be employed to set the threshold such as extreme value theory (Siffer et al. 2017), to avoid introducing additional hyperparameters, we use in our experiments a simple approach of setting the threshold as the max of $A_s(t)$ over the validation data.

4 Experiments

In this section, we conduct experiments to answer the following research questions:

- **RQ1 (Accuracy):** Does our method outperform baseline methods in accuracy of anomaly detection in multivariate time series, based on ground truth labelled anomalies?
- **RQ2 (Ablation):** How do the various components of the method contribute to its performance?
- **RQ3 (Interpretability of Model):** How can we understand our model based on its embeddings and its learned graph structure?
- **RQ4 (Localizing Anomalies):** Can our method localize anomalies and help users to identify the affected sensors, as well as to understand how the anomaly deviates from the expected behavior?

4.1 Datasets

As real-world datasets with labeled ground-truth anomalies are scarce, especially for large-scale plants and factories, we use two sensor datasets based on water treatment physical test-bed systems: `SWaT` and `WADI`, where operators have simulated attack scenarios of real-world water treatment plants, recording these as the ground truth anomalies.

The Secure Water Treatment (`SWaT`) dataset comes from a water treatment test-bed coordinated by Singapore's Public Utility Board (Mathur and Tippenhauer 2016). It represents a small-scale version of a realistic modern Cyber-Physical system, integrating digital and physical elements to control and monitor system behaviors. As an extension of `SWaT`, Water Distribution (`WADI`) is a distribution system comprising a larger number of water distribution pipelines (Ahmed, Palleti, and Mathur 2017). Thus `WADI` forms a more complete and realistic water treatment, storage and distribution network. The datasets contain two weeks of data from normal operations, which are used as training data for the respective models. A number of controlled, physical attacks are conducted at different intervals in the following days, which correspond to the anomalies in the test set.

Table 1 summarises the statistics of the two datasets. In order to speed up training, the original data samples are downsampled to one measurement every 10 seconds by taking the median values. The resulting label is the most common label during the 10 seconds. Since the systems took 5-6 hours to reach stabilization when first turned on (Goh et al. 2016), we eliminate the first 2160 samples for both datasets.

Datasets	#Features	#Train	#Test	Anomalies
`SWaT`	51	47515	44986	11.97%
`WADI`	127	118795	17275	5.99%

Table 1: Statistics of the two datasets used in experiments

4.2 Baselines

We compare the performance of our proposed method with five popular anomaly detection methods, including:

- **PCA:** Principal Component Analysis (Shyu et al. 2003) finds a low-dimensional projection that captures most of the variance in the data. The anomaly score is the reconstruction error of this projection.
- **KNN:** K Nearest Neighbors uses each point's distance to its kth nearest neighbor as an anomaly score (Angiulli and Pizzuti 2002).
- **FB:** A Feature Bagging detector is a meta-estimator that fits a number of detectors on various sub-samples of the dataset, then aggregates their scores (Lazarevic and Kumar 2005).
- **AE:** Autoencoders consist of an encoder and decoder which reconstruct data samples (Aggarwal 2015). It uses the reconstruction error as the anomaly score.
- **DAGMM:** Deep Autoencoding Gaussian Model joints deep Autoencoders and Gaussian Mixture Model to generate a low-dimensional representation and reconstruction error for each observation (Zong et al. 2018).
- **LSTM-VAE:** LSTM-VAE (Park, Hoshi, and Kemp 2018) replaces the feed-forward network in a VAE with LSTM to combine LSTM and VAE. It can measure reconstruction error with the anomaly score.
- **MAD-GAN:** A GAN model is trained on normal data, and the LSTM-RNN discriminator along with a reconstruction-based approach is used to compute anomaly scores for each sample (Li et al. 2019).

4.3 Evaluation Metrics

We use precision (Prec), recall (Rec) and F1-Score (F1) over the test dataset and its ground truth values to evaluate the performance of our method and baseline models: $\mathsf{F1} = \frac{2\times\mathsf{Prec}\times\mathsf{Rec}}{\mathsf{Prec}+\mathsf{Rec}}$, where $\mathsf{Prec} = \frac{\mathsf{TP}}{\mathsf{TP}+\mathsf{FP}}$ and $\mathsf{Rec} = \frac{\mathsf{TP}}{\mathsf{TP}+\mathsf{FN}}$, and $\mathsf{TP}, \mathsf{TN}, \mathsf{FP}, \mathsf{FN}$ are the numbers of true positives, true negatives, false positives, and false negatives. Note that our datasets are unbalanced, which justifies the choice of these metrics, which are suitable for unbalanced data. To detect anomalies, we use the maximum anomaly score over the validation dataset to set the threshold. At test time, any time step with an anomaly score over the threshold will be regarded as an anomaly.

4.4 Experimental Setup

We implement our method and its variants in PyTorch (Paszke et al. 2017) version 1.5.1 with CUDA 10.2 and PyTorch Geometric Library (Fey and Lenssen 2019)

Method	SWaT			WADI		
	Prec	Rec	F1	Prec	Rec	F1
PCA	24.92	21.63	0.23	39.53	5.63	0.10
KNN	7.83	7.83	0.08	7.76	7.75	0.08
FB	10.17	10.17	0.10	8.60	8.60	0.09
AE	72.63	52.63	0.61	34.35	34.35	0.34
DAGMM	27.46	69.52	0.39	54.44	26.99	0.36
LSTM-VAE	96.24	59.91	0.74	87.79	14.45	0.25
MAD-GAN	98.97	63.74	0.77	41.44	33.92	0.37
GDN	**99.35**	**68.12**	**0.81**	**97.50**	**40.19**	**0.57**

Table 2: Anomaly detection accuracy in terms of precision(%), recall(%), and F1-score, on two datasets with ground-truth labelled anomalies. Part of the results are from (Li et al. 2019).

version 1.5.0, and train them on a server with Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz and 4 NVIDIA RTX 2080Ti graphics cards. The models are trained using the Adam optimizer with learning rate 1×10^{-3} and $(\beta_1, \beta_2) = (0.9, 0.99)$. We train models for up to 50 epochs and use early stopping with patience of 10. We use embedding vectors with length of $128(64)$, k with $30(15)$ and hidden layers of $128(64)$ neurons for the WADI (SWaT) dataset, corresponding to their difference in input dimensionality. We set the sliding window size w as 5 for both datasets.

4.5 RQ1. Accuracy

In Table 2, we show the anomaly detection accuracy in terms of precision, recall and F1-score, of our GDN method and the baselines, on the SWaT and WADI datasets. The results show that GDN outperforms the baselines in both datasets, with high precision in both datasets of 0.99 on SWaT and 0.98 on WADI. In terms of F-measure, GDN outperforms the baselines on SWaT; on WADI, it has 54% higher F-measure than the next best baseline. WADI is more unbalanced than SWaT and has higher dimensionality than SWaT as shown in Table 1. Thus, our method shows effectiveness even in unbalanced and high-dimensional attack scenarios, which are of high importance in real-world applications.

4.6 RQ2. Ablation

To study the necessity of each component of our method, we gradually exclude the components to observe how the model performance degrades. First, we study the importance of the learned graph by substituting it with a static complete graph, where each node is linked to all the other nodes. Second, to study the importance of the sensor embeddings, we use an attention mechanism without sensor embeddings: that is, $\mathbf{g}_i = \mathbf{W}\mathbf{x}_i$ in Eq. (6). Finally, we disable the attention mechanism, instead aggregating using equal weights assigned to all neighbors. The results are summarized in Table 3 and provide the following findings:

- Replacing the learned graph structure with a complete graph degrades performance in both datasets. The effect on the WADI dataset is more obvious. This indicates that the graph structure learner enhances performance, especially for large-scale datasets.
- The variant which removes the sensor embedding from the attention mechanism underperforms the original model in both datasets. This implies that the embedding feature improves the learning of weight coefficients in the graph attention mechanism.
- Removing the attention mechanism degrades the model's performance most in our experiments. Since sensors have very different behaviors, treating all neighbors equally introduces noise and misleads the model. This verifies the importance of the graph attention mechanism.

These findings suggest that GDN's use of a learned graph structure, sensor embedding, and attention mechanisms all contribute to its accuracy, which provides an explanation for its better performance over the baseline methods.

Method	SWaT			WADI		
	Prec	Rec	F1	Prec	Rec	F1
GDN	**99.35**	**68.12**	**0.81**	**97.50**	**40.19**	**0.57**
- TopK	97.41	64.70	0.78	92.21	35.12	0.51
- Emb	92.31	61.25	0.76	91.86	33.49	0.49
- Att	71.05	65.06	0.68	61.33	38.85	0.48

Table 3: Anomaly detection accuracy in term of percision(%), recall(%), and F1-score of GDN and its variants.

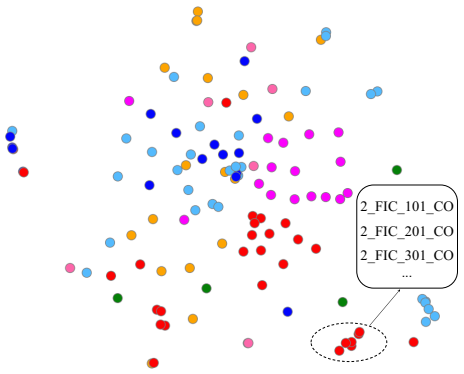

Figure 2: A t-SNE plot of the sensor embeddings of our trained model on the WADI dataset. Node colors denote classes. Specifically, the dashed circled region shows localized clustering of 2_FIC_x01_CO sensors. These sensors are measuring similar indicators in WADI.

4.7 RQ3. Interpretability of Model

Interpretability via Sensor Embeddings To explain the learned model, we can visualize its sensor embedding vectors, e.g. using t-SNE(Maaten and Hinton 2008), shown on the WADI dataset in Figure 2. Similarity in this embedding space indicate similarity between the sensors' behaviors, so inspecting this plot allows the user to deduce groups of sensors which behave in similar ways.

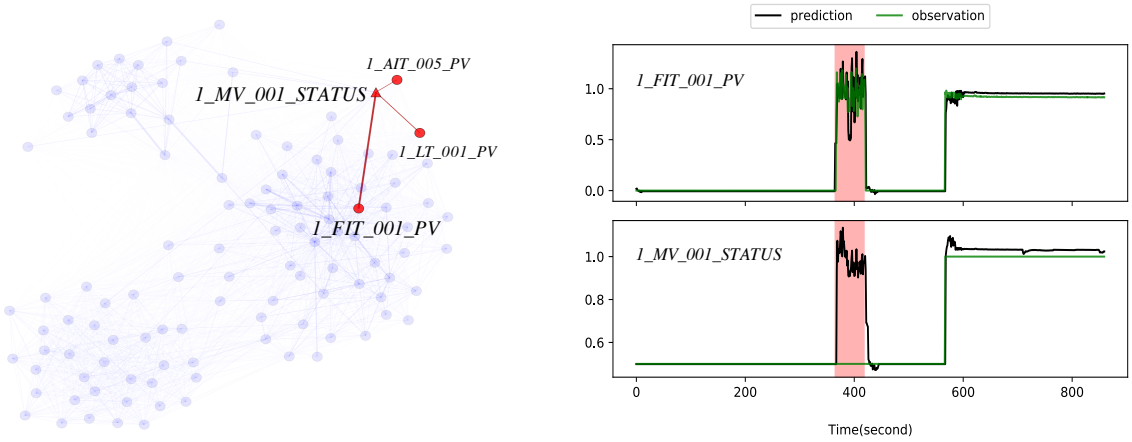

Figure 3: *Left:* Force-directed graph layout with attention weights as edge weights, showing an attack in WADI. The red triangle denotes the central sensor identified by our approach, with highest anomaly score. Red circles indicate nodes with edge weights larger than 0.1 to the central node. *Right:* Comparing expected and observed data helps to explain the anomaly. The attack period is shaded in red.

To validate this, we color the nodes using 7 colors corresponding to 7 classes of sensors in WADI systems. The representation exhibits localized clustering in the projected 2D space, which verifies the effectiveness of the learned feature representations to reflect the localized sensors' behavior similarity. Moreover, we observe a group of sensors forming a localized cluster, shown in the dashed circled region. Inspecting the data, we find that these sensors measure similar indicators in water tanks that perform similar functions in the WADI water distribution network, explaining the similarity between these sensors.

Interpretability via Graph Edges and Attention Weights Edges in our learned graph provide interpretability by indicating which sensors are related to one another. Moreover, the attention weights further indicate the importance of each of a node's neighbors in modelling the node's behavior. Figure 3 (left) shows an example of this learned graph on the WADI dataset. The following subsection further shows a case study of using this graph to localize and understand an anomaly.

4.8 RQ4. Localizing Anomalies

How well can our model help users to localize and understand an anomaly? Figure 3 (left) shows the learned graph of sensors, with edges weighted by their attention weights, and plotted using a force-directed layout(Kobourov 2012).

We conduct a case study involving an anomaly with a known cause: as recorded in the documentation of the WADI dataset, this anomaly arises from a flow sensor, *1_FIT_001_PV*, being attacked via false readings. These false readings are within the normal range of this sensor, so detecting this anomaly is nontrivial.

During this attack period, GDN identifies *1_MV_001_STATUS* as the deviating sensor with the highest anomaly score, as indicated by the red triangle in Figure 3 (left). The large deviation at this sensor indicates that *1_MV_001_STATUS* could be the attacked sensor, or closely related to the attacked sensor.

GDN indicates (in red circles) the sensors with highest attention weights to the deviating sensor. Indeed, these neighbors are closely related sensors: the *1_FIT_001_PV* neighbor is normally highly correlated with *1_MV_001_STATUS*, as the latter shows the valve status for a valve which controls the flow measured by the former. However, the attack caused a deviation from this relationship, as the attack gave false readings only to *1_FIT_001_PV*. GDN further allows understanding of this anomaly by comparing the predicted and observed sensor values in Figure 3 (right): for *1_MV_001_STATUS*, our model predicted an increase (as *1_FIT_001_PV* increased, and our model has learned that the sensors increase together). Due to the attack, however, no change was observed in *1_MV_001_STATUS*, leading to a large error which was detected as an anomaly by GDN.

In summary: 1) our model's individual anomaly scores help to localize anomalies; 2) its attention weights help to find closely related sensors; 3) its predictions of expected behavior of each sensor allows us to understand how anomalies deviate from expectations.

5 Conclusion

In this work, we proposed our Graph Deviation Network (GDN) approach, which learns a graph of relationships between sensors, and detects deviations from these patterns, while incorporating sensor embeddings. Experiments on two real-world sensor datasets showed that GDN outperformed baselines in accuracy, provides an interpretable model, and helps users to localize and understand anomalies. Future work can consider additional architectures and online training methods, to further improve the practicality of the approach.

Acknowledgments

This work was supported in part by NUS ODPRT Grant R252-000-A81-133. The datasets are provided by iTrust, Centre for Research in Cyber Security, Singapore University of Technology and Design.

References

Aggarwal, C. C. 2015. Outlier analysis. In *Data mining*, 237–263. Springer.

Ahmed, C. M.; Palleti, V. R.; and Mathur, A. P. 2017. WADI: a water distribution testbed for research in the design of secure cyber physical systems. In *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, 25–28.

Angiulli, F.; and Pizzuti, C. 2002. Fast outlier detection in high dimensional spaces. In *European conference on principles of data mining and knowledge discovery*, 15–27. Springer.

Bach, F. R.; and Jordan, M. I. 2004. Learning graphical models for stationary time series. *IEEE transactions on signal processing* 52(8): 2189–2199.

Blázquez-García, A.; Conde, A.; Mori, U.; and Lozano, J. A. 2020. A review on outlier/anomaly detection in time series data. *arXiv preprint arXiv:2002.04236* .

Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; and Sander, J. 2000. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 93–104.

Chen, W.; Chen, L.; Xie, Y.; Cao, W.; Gao, Y.; and Feng, X. 2019. Multi-range attentive bicomponent graph convolutional network for traffic forecasting. *arXiv preprint arXiv:1911.12093* .

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, 3844–3852.

Fey, M.; and Lenssen, J. E. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Filonov, P.; Lavrentyev, A.; and Vorontsov, A. 2016. Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model. *arXiv preprint arXiv:1612.06676* .

Goh, J.; Adepu, S.; Junejo, K. N.; and Mathur, A. 2016. A dataset to support research in the design of secure water treatment systems. In *International conference on critical information infrastructures security*, 88–99. Springer.

Hautamaki, V.; Karkkainen, I.; and Franti, P. 2004. Outlier detection using k-nearest neighbour graph. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, 430–433. IEEE.

Hundman, K.; Constantinou, V.; Laporte, C.; Colwell, I.; and Soderstrom, T. 2018a. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 387–395.

Hundman, K.; Constantinou, V.; Laporte, C.; Colwell, I.; and Soderstrom, T. 2018b. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 387–395.

Kingma, D. P.; and Welling, M. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* .

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

Kobourov, S. G. 2012. Spring embedders and force directed graph drawing algorithms. *arXiv preprint arXiv:1201.3011* .

Lazarevic, A.; and Kumar, V. 2005. Feature bagging for outlier detection. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, 157–166.

Li, D.; Chen, D.; Jin, B.; Shi, L.; Goh, J.; and Ng, S.-K. 2019. MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. In *International Conference on Artificial Neural Networks*, 703–716. Springer.

Lim, N.; Hooi, B.; Ng, S.-K.; Wang, X.; Goh, Y. L.; Weng, R.; and Varadarajan, J. 2020. STP-UDGAT: Spatial-Temporal-Preference User Dimensional Graph Attention Network for Next POI Recommendation. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 845–854.

Maaten, L. v. d.; and Hinton, G. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9(Nov): 2579–2605.

Mathur, A. P.; and Tippenhauer, N. O. 2016. SWaT: a water treatment testbed for research and training on ICS security. In *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, 31–36. IEEE.

Munir, M.; Siddiqui, S. A.; Dengel, A.; and Ahmed, S. 2018. DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access* 7: 1991–2005.

Park, D.; Hoshi, Y.; and Kemp, C. C. 2018. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters* 3(3): 1544–1551.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

Qin, Y.; Song, D.; Chen, H.; Cheng, W.; Jiang, G.; and Cottrell, G. 2017. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971* .

Schlichtkrull, M.; Kipf, T. N.; Bloem, P.; Van Den Berg, R.; Titov, I.; and Welling, M. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, 593–607. Springer.

Schölkopf, B.; Platt, J. C.; Shawe-Taylor, J.; Smola, A. J.; and Williamson, R. C. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13(7): 1443–1471.

Shyu, M.-L.; Chen, S.-C.; Sarinnapakorn, K.; and Chang, L. 2003. A novel anomaly detection scheme based on principal component classifier. Technical report, MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING.

Siffer, A.; Fouque, P.-A.; Termier, A.; and Largouet, C. 2017. Anomaly detection in streams with extreme value theory. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1067–1075.

Tank, A.; Foti, N.; and Fox, E. 2015. Bayesian structure learning for stationary time series. *arXiv preprint arXiv:1505.03131* .

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* .

Wang, Y.; Wang, W.; Ca, Y.; Hooi, B.; and Ooi, B. C. 2020. Detecting Implementation Bugs in Graph Convolutional Network based Node Classifiers. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 313–324. IEEE.

Yu, B.; Yin, H.; and Zhu, Z. 2017. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875* .

Zhang, Y.; Hamm, N. A.; Meratnia, N.; Stein, A.; Van De Voort, M.; and Havinga, P. J. 2012. Statistics-based outlier detection for wireless sensor networks. *International Journal of Geographical Information Science* 26(8): 1373–1392.

Zhou, B.; Liu, S.; Hooi, B.; Cheng, X.; and Ye, J. 2019. BeatGAN: Anomalous Rhythm Detection using Adversarially Generated Time Series. In *IJCAI*, 4433–4439.

Zhou, Y.; Qin, R.; Xu, H.; Sadiq, S.; and Yu, Y. 2018. A data quality control method for seafloor observatories: the application of observed time series data in the East China Sea. *Sensors* 18(8): 2628.

Zong, B.; Song, Q.; Min, M. R.; Cheng, W.; Lumezanu, C.; Cho, D.; and Chen, H. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*.