Chemical Engineering Science 254 (2022) 117637

Contents lists available at ScienceDirect

Chemical Engineering Science

journal homepage: www.elsevier.com/locate/ces

Graph dynamic autoencoder for fault detection

Lu Liu, Haitao Zhao *, Zhengwei Hu

Automation Department, School of Information Science and Engineering, East China University of Science and Technology

HIGHLIGHTS

- A novel dynamic method called graph dynamic autoencoder for fault detection is proposed.
- The use of graph convolution avoids the dimensionality increase problem of classic dynamic methods.
- A weighted adjacency matrix is designed to adaptively assign weights to the temporal samples.
- Experiments are carried out to demonstrate the effectiveness and merits.

ARTICLE INFO

Article history:
Received 6 July 2021
Received in revised form 11 January 2022
Accepted 23 March 2022
Available online 28 March 2022

Keywords:
Process monitoring
Dynamic fault detection
Graph convolution
Autoencoder

ABSTRACT

Dynamic information is a non-negligible part of time-correlated process data, and its full utilization can improve the performance of fault detection. Traditional dynamic methods concatenate the current process data with a certain number of previous process data into an extended vector before performing feature extraction. However, this simple way of using dynamic information inevitably increases the input dimensionality and it is inappropriate to treat previous process data as equally important. To address these problems, this paper proposes a novel nonlinear dynamic method, called graph dynamic autoencoder (GDAE), for fault detection. GDAE utilizes a graph structure to model the dynamic information between different data points. GDAE firstly embeds the current data point and previous data points as the features of the central node and its neighbors, respectively, then convolves the feature of the central node with the features of its neighbors to derive the updated feature for the central node, and finally, an encoder-decoder structure is adopted to extract the key low-dimensional feature. Due to the utilization of the graph structure, the extended high-dimensional vectors utilized by traditional dynamic fault detection methods are avoided in GDAE. Furthermore, with the dynamically constructed graph, GDAE is able to adaptively assign different weights to its neighbors by updating the adjacency matrix of the graph. Experimental results obtained from a numerical simulation and the Tennessee Eastman process illustrate the superiority of GDAE in terms of missed detection rate (MDR) and false alarm rate (FAR). The source code of GDAE can be found in https://github.com/luliu-fighting/Graph-Dynamic-Autoencoder.

1. Introduction

Process monitoring is essential for normal operation and safe production in chemical process (Tanjin Amin et al., 2018; Joe Qin, 2012). In recent years, the adoption of distributed control systems and rapid advances in data collection technology has enabled large-scale process data to be collected and stored for process monitoring (Ge, 2017; Ge et al., 2013). In order to mine the most significant information from these complex data, data-driven methods, especially multivariate statistical process monitoring (MSPM) methods, are increasingly gaining the attention of researchers (Joe Qin, 2012; Yin et al., 2014).

The key idea of MSPM methods is to project the high-dimension data into low-dimension spaces by a certain multivariate analysis method, and then compute monitoring statistics in the low-dimension spaces for fault assessment (Jiang et al., 2019; Jiang et al., 2019). As a typical MSPM method, principal component analysis (PCA) utilizes linear transformation to capture key information from high-dimensional process data (Jiang et al., 2019; Ammiche et al., 2018). However, PCA does not take the dynamic information of the process data into account, which may lead to some unnecessarily high missed detection rates (Li et al., 2014). Since most process data is generated sequentially over time, i.e. the process data is inherently dynamic, it is essential to incorporate dynamic information into the monitoring model.

* Corresponding author.
E-mail address: htz.ecust@gmail.com (H. Zhao).

https://doi.org/10.1016/j.ces.2022.117637
0009-2509/

Data augmentation is a widely used technique for dealing with temporally correlated process data. The data augmentation technique is also called the time lag shift technique, which is to augment each sample vector with a certain number of previous samples (Hu et al., 2021). Ku et al. (1995) used the augmented data to model the dynamic information between different samples and developed the dynamic principal component analysis (DPCA) method. Although the dynamic information among data is variable and difficult to interpret in the data augmentation technique, DPCA does improve fault detection performance to some extent compared to conventional PCA (Shang et al., 2018). Note that DPCA is a linear method, which assumes a linear correlation between variables, and this assumption is usually violated in real-world industrial processes.

To monitor industrial processes with both nonlinear and dynamic characteristics, some researchers combined the data augmentation technique with the kernel method (Choi and Lee, 2004; Zhang et al., 2020; Bounoua and Bakdi, 2021). Choi and Lee (2004) proposed dynamic kernel principal component analysis (DKPCA), which has been experimentally demonstrated better detection performance. Zhang et al. (2020) further enhanced the performance of DKPCA by using feature vector selection. However, the kernel function of the kernel method and its associated parameter are prefixed, so DKPCA may not be suitable to deal with complicated and variable dynamic information in a process system. Autoencoder (AE) and its extensions recently are widely studied for nonlinear process monitoring (Fan et al., 2017; Zheng and Zhao, 2020; Zhang et al., 2018; Lv et al., 2017). Classical AE consists of two parts, an encoder for low-dimensional feature extractions and a decoder for data reconstruction, which can be optimized by the backpropagation algorithm (LeCun et al., 2015). To tackle the nonlinear dynamic fault detection problem, Jiang et al. (2017) proposed the dynamic stacked sparse autoencoders (DSSAE) which combined dynamic data window into stacked sparse autoencoders (SSAE). Furthermore, based on DSSAE, (Yin and Yan, 2019) proposed MI-DSSAE, which adopted mutual information to select variables with relatively strong autocorrelation.

All the above methods capture dynamic information by the data augmentation technique which concatenates the current sample with l past samples into an extended vector. However, assuming that the original data dimensionality is d, this technique increases the data dimensionality from d to $d \times (l+1)$. If d and l are large, the data augmentation technique may cause the curse of dimensionality problem. To solve this problem, Zhao (2018) proposed dynamic graph embedding (DGE), which extended each sample into a matrix instead of a vector and thus avoided the dimensionality increase. But DGE assumes that the past l samples are equally weighted for the current sample. Lv et al. (2017) assigned different weight to each sample by the principles $\omega_1 + \omega_2 + \ldots + \omega_l = 1$ and $\omega_1 < \omega_2 < \ldots < \omega_l$, where ω_i is the weight of the ith previous sample. However, these weights are handcrafted and may not be suitable to obtain variable dynamic information for all samples. In this paper, we put our attention on graph structure and graph convolutional neural networks (GCN). In the past three years, GCN has been successfully applied to many areas, such as traffic flow prediction and recommendation systems (Yin et al., 2021; Ying et al., 2018; He et al., 2020). GCN utilizes graph convolution to aggregate information from neighbors (Wu et al., 2021), which can avoid the dimensionality increase problem. However, in the classical GCN, the adjacency matrix are fixed in the training.

Based on the above analysis, we propose a novel method, called graph dynamic autoencoder (GDAE), for nonlinear dynamic process monitoring. GDAE utilizes a graph structure and adopts the Euclidean distance normalized by softmax as the adjacency matrix to model variable dynamic information. An encoder-decoder structure is adopted in GDAE to extract low-dimensional key features. Firstly, GDAE uses the normalized data to construct the graph as the model input. The current sample and past samples are embedded as the features of the central node and its neighbors. Then, an encoder consisting of graph convolution layers is used to aggregate the dynamic information and extract the key low-dimensional features, followed by a decoder to reconstruct the original data. Finally, the network parameters are automatically optimized by minimizing the reconstruction errors. It is worth noting that GDAE does not change the dimensionality of the data. In addition, the adjacency matrix can be dynamically updated, which is equivalent to adaptively assign different weights to the samples within the lag time.

The remainder of this paper is organized as follows: Section 2 introduces the data augmentation technique used in traditional dynamic methods and the concepts associated with the proposed model. Section 3 describes the structure and the advantages of the proposed GDAE. Section 4 presents the fault detection method based on GDAE and describes the offline training and online monitoring procedures. Section 5 uses a numerical example and the Tennessee Eastman (TE) benchmark process to demonstrate the superiority of the GDAE. Section 6 summarizes the main work of this paper.

2. Preliminaries

2.1. Data augmentation technique

Data augmentation technique is widely used in dynamic process monitoring. Let the original training set input $\mathrm{X} \in \mathbb{R}^{n\times m}$:

$$\mathrm{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_n]^T = \begin{bmatrix} x_{11} & x_{21} & \ldots & x_{m1} \\ x_{12} & x_{22} & \ldots & x_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ x_{1n} & x_{2n} & \ldots & x_{mn} \end{bmatrix}$$

where n and m denote the number of samples and data dimensionality, respectively. $\boldsymbol{x}_i = [x_{1i}, x_{2i}, x_{3i}, \ldots, x_{mi}]^T \in \mathbb{R}^m (i = 1, 2, \ldots, n)$ is the ith sample. Assume that the lag time is l, then the extended vector is $\widetilde{\boldsymbol{x}}_i = \left[\boldsymbol{x}_{i-l}^T, \boldsymbol{x}_{i-l+1}^T, \ldots, \boldsymbol{x}_i^T\right]^T \in \mathbb{R}^{m(l+1)}$. The procedure of data augmentation is shown in Fig. 1, the original vector $\boldsymbol{x}_i$ is first augmented to $\widetilde{\boldsymbol{x}}_i$ and then through a specific model to extract feature. It can be found that after data augmentation, the dimensionality of the data has changed from m to $m \times (l+1)$, which may cause the so-called "curse of dimensionality" problem. Furthermore, this technique treats samples within the lag time as equally important.

2.2. Graph convolution

A graph can be represented as $G = (\mathbb{V}, \mathbb{E}, \mathrm{A})$. $\mathbb{V}$ and $\mathbb{E}$ denote the set of nodes and edges, respectively. Let $v_i \in \mathbb{V}$ denotes a node, and $e_{ij} = (v_i, v_j) \in \mathbb{E}$ denotes an edge connecting v_i and v_j. If v_i and v_j are connected by an edge ($i \neq j$), then v_i and v_j are neighbors to each other. $\mathrm{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix, where N denotes the number of nodes. $\mathrm{D} \in \mathbb{R}^{N\times N}$ is the degree matrix which is a diagonal matrix with $D_{ii} = \sum_j A_{ij}$. $\mathrm{X}_v \in \mathbb{R}^{N\times d}$ represents the node feature matrix of G. Each row of X_v represents a sample, i.e. the node feature of a single node and d is the dimensionality of the sample.

Explained in terms of the spatial domain, graph convolution is the aggregation of information from neighbors to update the state information of the current node. The propagation rules for multilayer graph convolutional neural networks are as follows:

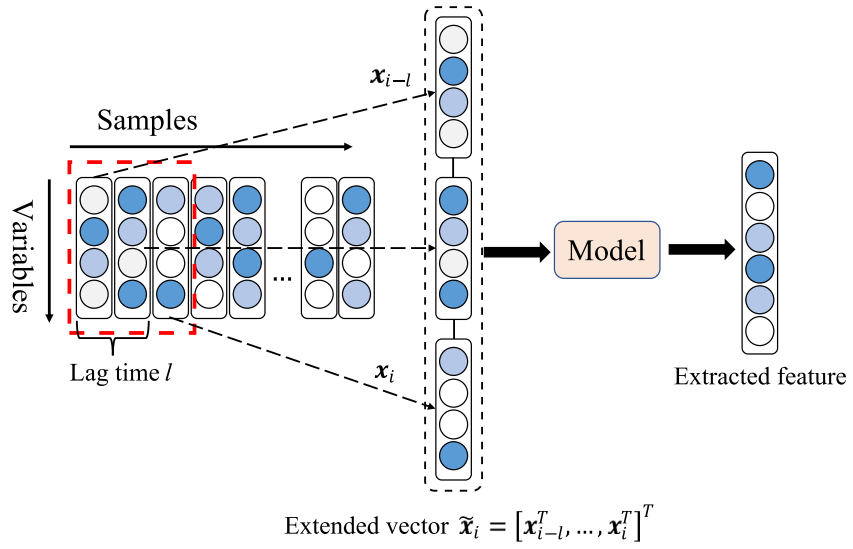


Fig. 1. Diagram of the data augmentation technique.

$$X^{p+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X^pW^p\right) \tag{1}$$

where $X^p \in \mathbb{R}^{N\times d_1}$ is the node feature matrix of the graph in pth layer, $X^{p+1} \in \mathbb{R}^{N\times d_2}$ is the node feature matrix of the graph in $(p+1)$th layer. When $p=0$, X^0 represents the node feature matrix of the graph in the input layer, and X^0 is composed of the original training or testing samples. $\tilde{D}$ and $\tilde{A}$ are the degree matrix and the adjacency matrix after adding the self-loop, respectively, i.e.:

$$\tilde{A} = A + I_N \tag{2}$$

$$\tilde{D} = \sum_j \tilde{A}_{ij} \tag{3}$$

where I_N is the identity matrix. $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ is a normalized adjacency matrix. W^p is the weight matrix of the pth layer. $\sigma(\cdot)$ is the activation function and it can be sigmoid, hyperbolic tangent, and rectified linear unit (ReLU). In this paper, we choose ReLU as our activation function. To illustrate the principle of graph convolution more intuitively, a specific example is given in Fig. 2. Fig. 2a shows a graph containing 6 nodes and 7 edges. $\boldsymbol{x}_i (i = 1, 2, \ldots, 6)$ represents the node feature, i.e. a sample. The original and the normalized adjacency matrix of this graph as follows:

$$A^{\sim} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}} = \begin{bmatrix} \frac{1}{3} & \frac{1}{2\sqrt{3}} & \frac{1}{\sqrt{15}} & 0 & 0 & 0 \\ \frac{1}{2\sqrt{3}} & \frac{1}{4} & \frac{1}{2\sqrt{15}} & \frac{1}{2\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{15}} & \frac{1}{2\sqrt{15}} & \frac{1}{5} & 0 & \frac{1}{\sqrt{15}} & \frac{1}{\sqrt{15}} \\ 0 & \frac{1}{2\sqrt{2}} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{15}} & 0 & \frac{1}{3} & \frac{1}{3} \\ 0 & 0 & \frac{1}{\sqrt{15}} & 0 & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

The procedure of its graph convolution is given in Fig. 2b. The procedure of graph convolution can be divided into two steps: message passing and updating. The mathematical representation of these two steps are as follows:

$$\text{message passing}: Y^p = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X^p \tag{4}$$

$$\text{updating}: X^{p+1} = \sigma(Y^pW^p) \tag{5}$$

where Y^p represents the intermediate state. In the message passing step, each node aggregates the information from its neighbors to obtain the corresponding intermediate state. Taking sample $\boldsymbol{x}_1$ as an example, it has two neighbors $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$ and the aggregation process can be presented as follows:

$$\boldsymbol{y}_1^p = \frac{1}{3}\boldsymbol{x}_1^p + \frac{1}{2\sqrt{3}}\boldsymbol{x}_2^p + \frac{1}{\sqrt{15}}\boldsymbol{x}_3^p \tag{6}$$

It can be found that samples can be given different weights by the normalized adjacency matrix. In Section 3, we redesign another weighted adjacency matrix and it can be adaptively updated during the training process. After message passing, the updating step performs the update of node features through the weight matrix W^p and the nonlinear activation function $\sigma(\cdot)$, it can be presented as follows:

$$\boldsymbol{x}_1^{p+1} = \sigma(\boldsymbol{y}_1^pW^p) \tag{7}$$

After this step, the node feature has been transformed from $\boldsymbol{x}_1^p$ to $\boldsymbol{x}_1^{p+1}$ and the dimensionality of the node feature has changed from d_1 to d_2, so the updating step can implement feature extraction to obtain the key features. The above process is just the graph convolution process for $\boldsymbol{x}_1$, and the graph convolution process for the entire graph is to traverse each node and perform the above process.

From the above explanation of the graph convolution process, it can be found that the graph convolution can exploit dynamic information through the message passing step and perform feature extraction through the updating step. Compared to the concatenation of the data augmentation technique, the weighted summation utilized by the graph convolution as shown in Eq. (6) not only avoids the dimensionality increase problem but can also assigns different weights to different samples by designing the adjacency matrix.

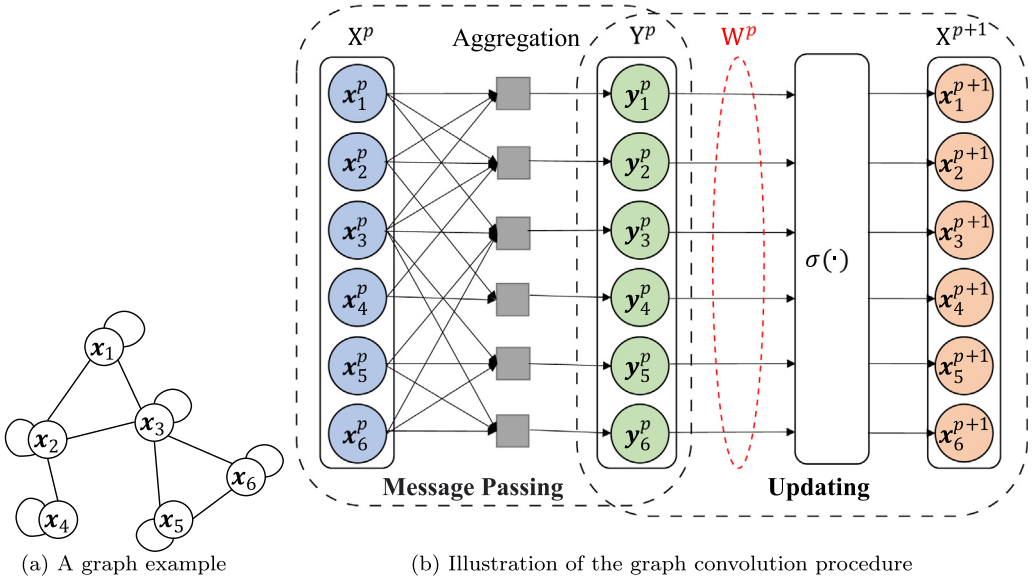


Fig. 2. A graph example and its the graph convolution procedure.

2.3. Stacked Autoencoder (SAE)

AE is an unsupervised feature extraction algorithm, which consists of two parts: an encoder and a decoder. The basic AE includes three layers: input, hidden and output layers. The output layer has the same number of units as the input layer. In general, the number of hidden units is less than the number of input layer units, which is used for obtaining the key feature of the input. The encoding step maps the input data $\mathrm{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^T \in \mathbb{R}^{n \times d_1}$ to the hidden feature $\mathrm{Z} = [\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_n]^T \in \mathbb{R}^{n \times d_2}$, and the decoding step reconstructs the output data $\hat{\mathrm{X}} = [\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_n]^T \in \mathbb{R}^{n \times d_1}$ from Z. The mathematical representation of these two steps are as follows:

$$\boldsymbol{z}_i = f(\mathrm{W}_1 \boldsymbol{x}_i + b_1) \tag{8}$$

$$\hat{\boldsymbol{x}}_i = g(\mathrm{W}_2 \boldsymbol{z}_i + b_2) \tag{9}$$

where $f(\cdot)$ and $g(\cdot)$ denote the activation function of encoder and decoder, respectively, which lead to the nonlinear characteristics of AE, $\theta_1 = \{\mathrm{W}_1 \in \mathbb{R}^{d_2 \times d_1}, \boldsymbol{b}_1 \in \mathbb{R}^{d_2}\}$ and $\theta_2 = \{\mathrm{W}_2 \in \mathbb{R}^{d_1 \times d_2}, \boldsymbol{b}_2 \in \mathbb{R}^{d_1}\}$ are the parameters of encoder and decoder, respectively. The objective function of AE can be expressed as follows:

$$\{\theta_1, \theta_2\} = \arg\min_{\theta_1, \theta_2} \sum_{i=1}^{n} ||\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i||_2^2 \tag{10}$$

and AE is trained by the backpropagation algorithm.

SAE is the stacking of basic AEs, which can generate deeper and higher-level representations (Jiang et al., 2019). In the SAE, the features extracted by the previous AE are taken as input to the subsequent AE. Fig. 3a is the architecture of SAE, SAE still has an encoder-decoder structure, and only the number of hidden layers is different from the basic AE. Fig. 3b is the diagram of SAE training, we take an SAE with 5 hidden layers as an example to explain the training process of SAE. The training process of SAE consists of two phases: pre-training and fine-tuning (Bengio et al., 2007). The pre-training phase takes a greedy layer-wise training approach to optimize each AE. This phase trains each AE by gradient descent and backpropagation algorithm so that the parameters of each AE reach their optimal values. However, the optimal parameters of each AE are not certainly the optimal value of the whole SAE, so after pre-training, fine-tuning of the entire network is required to make the network parameters globally optimal. The parameters obtained in the pre-training phase are treated as initialization parameters for this phase. The GDAE proposed in this paper adopts the same training strategy as SAE.

3. Graph Dynamic Autoencoder (GDAE)

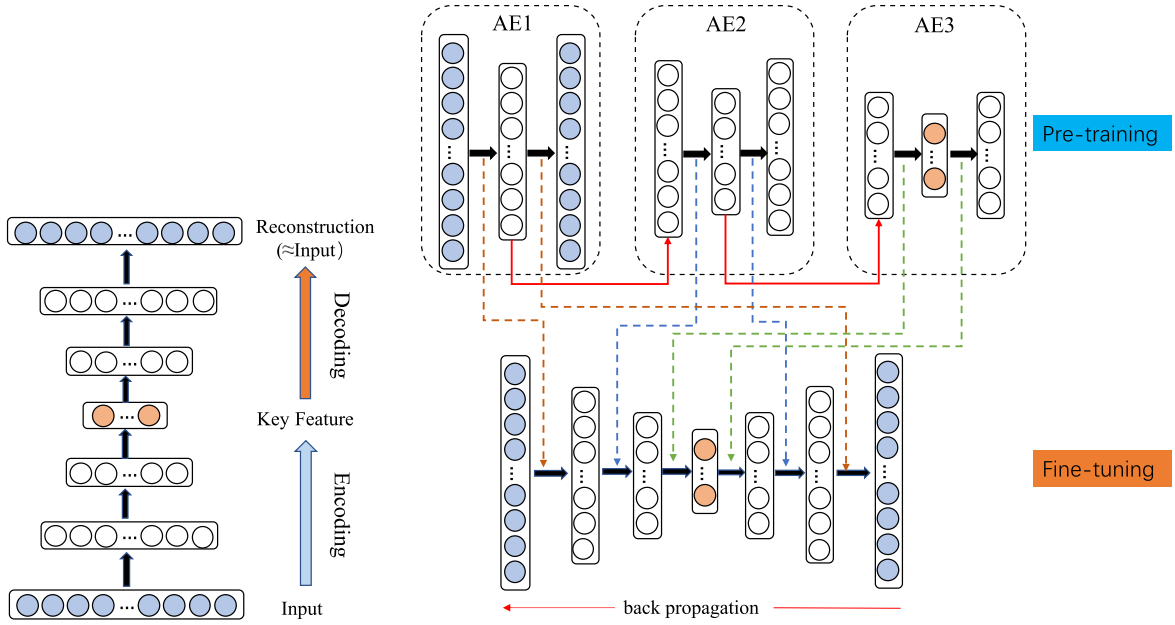
In processing dynamic data with graph convolution, the current sample and the previous samples within the lag time are embedded as the features of the central node and its neighbors, respectively. If the lag time l is given, then the number of neighbors of each node in the graph is l. It results in all the elements on the diagonal of the degree matrix being l, i.e., the degree of each node is the same. Moreover, $\frac{A_{ij}}{\sqrt{D_{ii}} \times \sqrt{D_{jj}}}$ in Eq. (4) will become the same, which means that the same weight is given to all the past l samples. However, it is inappropriate to treat previous process data as equally important. To address this issue, GDAE adopts the Euclidean distance normalized by softmax as the elements to obtain a weighted adjacency matrix, which we denote as $\bar{\mathrm{A}}$. $\bar{\mathrm{A}}$ can be updated adaptively according to the node feature matrix of each layer. Let $\boldsymbol{x}_i^p$ and $\boldsymbol{x}_j^p$ represent the node features at pth layer. We define the Euclidean distance between nodes as:

$$dist_{ij}^p = \begin{cases} \left|\left|\boldsymbol{x}_i^p - \boldsymbol{x}_j^p\right|\right|_2^2 & \text{if } \boldsymbol{x}_j^p \text{is a neighbor of } \boldsymbol{x}_i^p \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

Then softmax is used to normalize this Euclidean distance to obtain the weighted adjacency matrix $\bar{\mathrm{A}}^p$ at pth layer:

$$\bar{A}_{ij}^p = \begin{cases} 1 & \text{if } i = j \\ \mathrm{softmax}\left(-\frac{dist_{ij}^p}{\beta_i^p}\right) = \frac{\exp\left(-\frac{dist_{ij}^p}{\beta_i^p}\right)}{\sum_{j=1}^{l} \exp\left(-\frac{dist_{ij}^p}{\beta_i^p}\right)} & \text{if } i \neq j \text{ and } \boldsymbol{x}_j^p \text{ is a neighbor of } \boldsymbol{x}_i^p \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where l represents the number of neighbors and $\beta_i^p = \frac{1}{l}\sum_{j=1}^{l} dist_{ij}^p$. Then the propagation rule for the multi-layer graph convolution of Section 2.2 can be rewritten as:



(a) Structure of SAE (b) training stages of SAE

Fig. 3. Structure and training process of SAE.

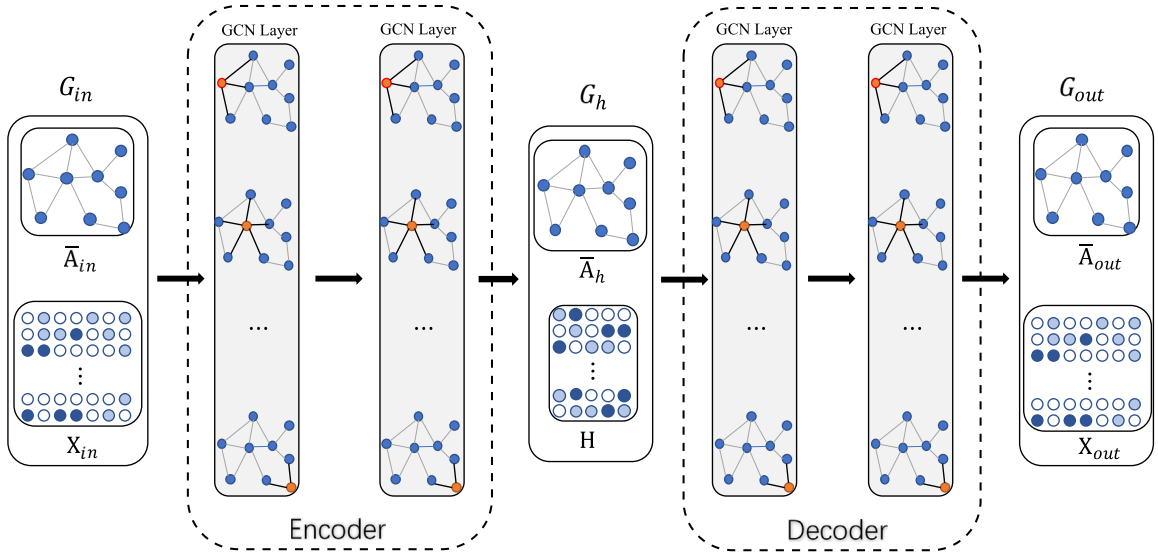


Fig. 4. Diagram of the proposed GDAE.

$$X^{p+1} = \sigma(\bar{A}^p X^p W^p) \tag{13}$$

The structure of the GDAE is given in Fig. 4. GDAE employs the same symmetrical structure as SAE, the difference is that the input of GDAE is a graph and the graph convolution layer is used. G_{in}, G_h and G_{out} are the graphs of the input layer, the hidden layer and the output layer, respectively. $\bar{A}_{in}$, $\bar{A}_h$ and $\bar{A}_{out}$ are their corresponding weighted adjacency matrices. X_{in}, X_h and X_{out} are their corresponding node feature matrices. Each row of X_{in} represents a sample, i.e., the node feature of a single node. G_{in} is a graph constructed using the current sample and the previous samples within the lag time l. The current sample is embedded as the feature of the central node and the previous samples within the lag time l are embedded as the features of its neighbors. G_{in} has $(l+1)$ nodes and all nodes are connected to each other by edges. Note that the structure of G_{in}, G_h and G_{out} is identical, the difference lies in the node features and the element values of adjacency matrices. The optimal parameters $W^p (p = 0, 1, 2, 3)$ of GDAE are obtained by minimizing the errors between X_{out} and X_{in}. From Section 2.2, instead of concatenating the current sample with several past samples, the graph convolution operation processes dynamic data by assigning a certain weight to each past sample and then adding them to update the current sample. Therefore, GDAE does not change the dimensionality of the original data.

It can be found that the elements of $\bar{A}^p$ are computed from the node features of pth layer, thus assigning different weights to the neighbors and the elements of $\bar{A}^p$ are updated as the node features change during the training process, which can better incorporate variable dynamic information. In addition, GDAE uses an encoder-decoder structure to efficiently extract the low-

dimensional features of the data. It is considered that GDAE absorbs the advantages of graph structure and stacked autoencoder.

4. Fault Detection based on GDAE

In this section, a new dynamic fault detection method based on GDAE is developed. The implementation is as follows. Firstly, the process data under normal condition $\mathrm{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^T \in \mathbb{R}^{n \times d_1}$ (n is the number of samples) is collected as training data. Secondly, the collected data is utilized to construct the input graph. Thirdly, GDAE is used to extract the key low-dimensional features $\mathrm{H} = [\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n]^T \in \mathbb{R}^{n \times d_2}$. The reconstructed data $\hat{\mathrm{X}} = [\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_n]^T \in \mathbb{R}^{n \times d_1}$ is also obtained in this step. Finally, Hotelling T^2 statistic and *SPE* statistic are calculated for fault detection.

Let $\Sigma_H \in \mathbb{R}^{d_2 \times d_2}$ is the covariance matrix of potential features H. The T^2 statistic is calculated as follows:

$$T_i^2 = \boldsymbol{h}_i^T \Sigma_H^{-1} \boldsymbol{h}_i \tag{14}$$

Let $\mathrm{R} = \mathrm{X} - \hat{\mathrm{X}} = [\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_n]^T \in \mathbb{R}^{n \times d_1}$ is the residual matrix. The *SPE* statistic is calculated as follows:

$$SPE_i = \boldsymbol{r}_i^T \boldsymbol{r}_i \tag{15}$$

Since we do not have any prior knowledge of the distribution of the features H extracted by neural networks, the control limits of T^2 statistic and *SPE* statistic can be determined based on the kernel density estimation (KDE) (Samuel and Cao, 2016). Assuming that the T^2 statistic of the n training samples have been calculated, the corresponding kernel density estimator can be expressed as follows:

$$\hat{\rho}\left(T^2\right) = \frac{1}{\tau(n)} \sum_{i=1}^{n} K\left(\frac{T^2 - T_i^2}{\tau}\right) \tag{16}$$

where $\tau > 0$ is a bandwidth parameter. If the Gaussian kernel function is adopted, Eq. (16) can be rewritten as:

$$\hat{\rho}\left(T^2\right) = \frac{1}{\sqrt{2\pi}\tau n} \sum_{i=1}^{n} \exp\left(\frac{\left(T^2 - T_i^2\right)^2}{2\tau^2}\right) \tag{17}$$

The control limit T_{limit}^2 is generally taken as the point at which the T^2 value presents a 99% confidence level.For a new sample $\boldsymbol{x}_{new}$, if $T_{new}^2 < T_{limit}^2$ then $\boldsymbol{x}_{new}$ is normal, otherwise $\boldsymbol{x}_{new}$ is abnormal.

Similarly, assuming that the *SPE* statistic of the n training samples have been calculated, we can also obtain:

$$\hat{\rho}(SPE) = \frac{1}{\sqrt{2\pi}\tau n} \sum_{i=1}^{n} \exp\left(\frac{\left(SPE^2 - SPE_i^2\right)^2}{2\tau^2}\right) \tag{18}$$

The control limit SPE_{limit} is also taken as the point at which the *SPE* value presents a 99% confidence level. For a new sample $\boldsymbol{x}_{new}$, if $SPE_{new} < SPE_{limit}$ then $\boldsymbol{x}_{new}$ is normal, otherwise $\boldsymbol{x}_{new}$ is abnormal.

The offline modeling and online monitoring flowchart are shown in Fig. 5. The offline modeling and online monitoring procedures are as follows:

Offline modeling:

1. Collect samples from normal process as training data.
2. Normalize training data to zero mean and unit variance.
3. Construct graph. Each sample is embedded as a node feature into the graph, the weighted adjacency matrix is calculated based on Eqs. (11) and and (12), and the number of neighbors is identical to the lag time l.
4. Train the GDAE model and obtain the optimal parameter W^p in Eq. (13).
5. Obtain the low-dimensional features of training data.
6. Calculate the T^2 statistic and *SPE* statistic for each sample according to Eqs. (14) and (15).
7. Determine the control limit of T^2 and *SPE* by KDE.

Online monitoring:

1. Collect new sample $\boldsymbol{x}_{new}$. Normalize $\boldsymbol{x}_{new}$ according to the parameters of the training data.
2. Construct graph.
3. Obtain the low-dimensional feature of $\boldsymbol{x}_{new}$ using the trained GDAE model.
4. Calculate the T^2 statistic and *SPE* statistic for $\boldsymbol{x}_{new}$.
5. Alarm if T^2 statistic or *SPE* statistic of $\boldsymbol{x}_{new}$ exceed the control limits; Otherwise, view $\boldsymbol{x}_{new}$ as normal data.

5. Experiments and discussion

In this section, we employ two cases, a nonlinear dynamic numerical case and the Tennessee Eastman process (TEP), to demonstrate the effectiveness of GDAE.

5.1. Numerical example

This example is modified according to the literature (Zhu et al., 2020). The sample $\check{\boldsymbol{x}}_i = [\check{x}_{1i}, \check{x}_{2i}, \check{x}_{3i}]$ is generated as follows:

$$\begin{cases} \check{x}_{1i} = u_i + e_1 \\ \check{x}_{2i} = u_{i-1}^2 - 2u_{i-3} + e_2 \\ \check{x}_{3i} = \sin\left(u_{i-2}\right) + e_3 \end{cases} \tag{19}$$

where i denotes the sampling moment. e_1, e_2 and e_3 are independently sampled from the normal distribution $\mathcal{N}(0, 0.1)$ and represent the noises. To ensure that the data is dynamic, we define u_i as:

$$u_i = \varphi(u_{i-1}) + q_i \tag{20}$$

$$\varphi(u_{i-1}) = \begin{cases} u_{i-1} - 1 & \text{if } u_{i-1} \geqslant 1 \\ 0.5u_{i-1} + 1 & \text{if } u_{i-1} \leqslant -1 \\ -0.8u_{i-1} + 0.5 & \text{otherwise} \end{cases} \tag{21}$$

where q_i is generated by $\mathcal{N}(0,\ 1)$. $\varphi(\cdot)$ is a nonlinear function that is used to achieve a dynamic relationship between samples.

Training data and test data are generated according to Eqs. (19)–(21). Training data and test data include 500 samples, respectively. The test data simulated with a step fault: a step change of 2 is added to the variable $\check{x}_{2i}$ from the 201st sample, and a step change of 1.5 is added to the variable $\check{x}_{3i}$ from the 201st sample. In other words, the first 200 samples are normal, and the 201st to the 500th samples are fault samples.

5.1.1. Autocorrelation of numerical example

Autocorrelation is a mathematical tool for finding the cross-correlation of a variable with itself at different time points. We use autocorrelation to determine the lag time l. Let γ_t denotes the sample at t moment. The correlation of γ_t and $\gamma_{t+k} (k = 1, 2, \cdots, K)$ is calculated as follows (Härdle and Simar, 2015):

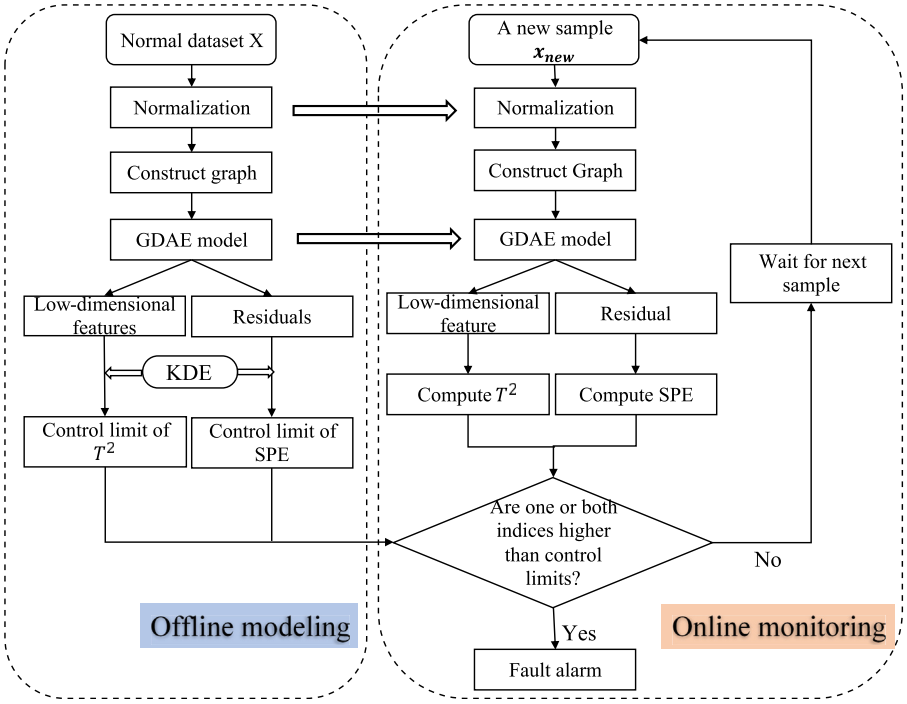


Fig. 5. Flowchart of GDAE for fault detection.

$$\hat{r}_k = \frac{c_k}{c_0} \tag{22}$$

$$c_k = \frac{1}{T-1}\sum_{t=1}^{T-k}(\gamma_t - \bar{\gamma})(\gamma_t - \bar{\gamma}) \tag{23}$$

where c_0 is the sample variance of the time sequence and $\bar{\gamma}$ is the mean of γ_t (t = 1, 2, ⋯T). The standard error for testing the significance of a single lag-f autocorrelation, $\hat{r}_f$ is approximate:

$$SE_r = \sqrt{\left(1 + 2\sum_{i=1}^{f-1}\hat{r}_i^2\right)/N} \tag{24}$$

Fig. 6 shows the number of time lags for past and future data points determined from autocorrelation function (ACF) of measurements of $\check{\boldsymbol{x}}_1$, $\check{\boldsymbol{x}}_2$, $\check{\boldsymbol{x}}_3$, and the sum of these three variables, respectively. As shown with the blue lines in the figure, approximate 95% confidence intervals are drawn at $\pm 2SE_r$. It can be found that the autocorrelation of each variable is various. In order to determine the number of time lags, we adopted the autocorrelation of the sum of three variables, as is shown in Fig. 6d. According to Fig. 6d, We set the number of time lags for numerical example to 8.

5.1.2. Parameter setting for numerical example

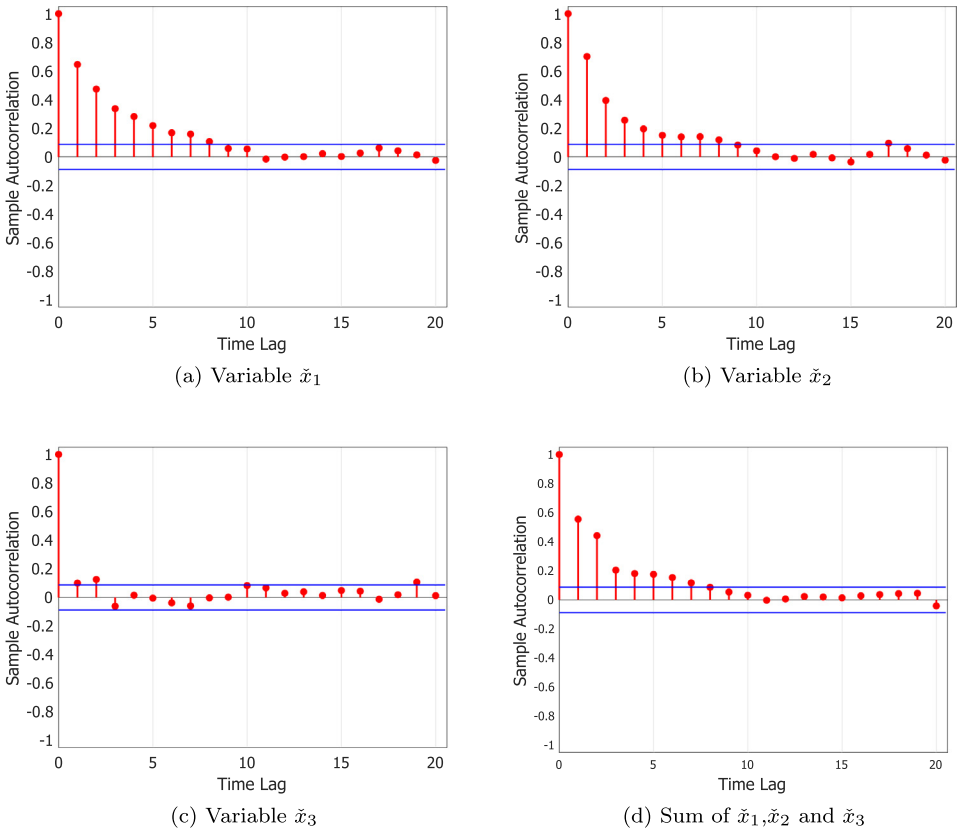
The proposed GDAE are compared with DPCA, DKPCA, and DSAE. DSAE denotes dynamic SAE. Since there is no literature related to dynamic SAE, here we combine the data augmentation technique with SAE to form DSAE. According to Section 5.1.1, we select time lag $l = 8$ for numerical example. For DPCA, DKPCA and DSAE, the dimensionality of input will increase from 3 to 27 after using the data augmentation technique. To retain 85% of the energy in the eigenspectrum (computed as the sum of eigenvalues), the number of principal components, i.e. the reduced dimensionality ξ was determined as 12 for DPCA based on the cumulative percentage variance (CPV) (Malinowski and Howery, 1980) rule. For fairness, DKPCA and DSAE use the same value $\xi = 12$, we set the structure of DSAE to 27–27-12-27–27. As for our proposed GDAE, the input dimensionality has not changed, so the reduced dimensionality of GDAE is set to 2. The structure of the proposed GDAE is set to 3–3-2-3–3.

5.1.3. Implementation of GDAE

In this subsection, to better interpret the proposed GDAE in Fig. 4, we use the numerical example to demonstrate how GDAE is implemented.

In the training phase, GDAE uses the same greedy training strategy as SAE and here we mainly discuss the fine-tuning step of GDAE. Firstly, construct the graph G_{in} using the current sample and the previous samples within the lag time. For this numerical example, G_{in} has 9 nodes and all nodes are connected to each other by edges. The weighted adjacency matrix $\bar{A}_{in}$ has a size of 9×9 and the node feature matrix X_{in} has a size of 9×3. Each element of $\bar{A}_{in}$ is obtained by Eq. (12) and each row of X_{in} represents a sample, i.e. the node feature of a single node. Secondly, G_{in} is input to the first graph convolution (GCN) layer. Each node of G_{in} performs the graph convolution operation as shown in Eq. (13) and Fig. (2). After the first GCN layer we can obtain X^1. Then the updated adjacency matrix $\bar{A}^1$ can be computed by X^1 and Eq. (12). The graph convolution process of the second GCN layer is performed by $\bar{A}^1$. The key feature matrix H can be obtained by the second GCN layer, and further used to compute $\bar{A}_h$ to obtain G_h. Passing the third and fourth GCN layer and continuing to update the adjacency matrix according to the node feature matrix of each layer, we can get $\bar{A}_{out}$ and the reconstructed X_{out}. Through continuously reducing the loss between X_{in} and X_{out} to obtain the optimal parameters $W^p(p = 0, 1, 2, 3)$ in Eq. (13).

In the testing phase, G_{in} is constructed in the same way as the training phase. Table 1 shows a portion of the test data, which we then use to illustrate the formulation of the adjacency matrix and the implementation of the GDAE. $\boldsymbol{x}_{current} \in \mathbb{R}^{1\times 3}$ represents the current test sample and $\boldsymbol{x}_{lagi} \in \mathbb{R}^{1\times 3}(i = 1, 2, \ldots, 8)$ denotes the



(a) Variable $\check{x}_1$ (b) Variable $\check{x}_2$

(c) Variable $\check{x}_3$ (d) Sum of $\check{x}_1$,$\check{x}_2$ and $\check{x}_3$

Fig. 6. Number of time lags for numerical example determined from autocorrelation function of different measurements.

Table 1
A portion of the test data in numerical example.

$\boldsymbol{x}_{current}$	$\boldsymbol{x}_{lag1}$	$\boldsymbol{x}_{lag2}$	$\boldsymbol{x}_{lag3}$	$\boldsymbol{x}_{lag4}$	$\boldsymbol{x}_{lag5}$	$\boldsymbol{x}_{lag6}$	$\boldsymbol{x}_{lag7}$	$\boldsymbol{x}_{lag8}$
0.166284	0.409495	−0.05372	0.165897	0.889122	1.285879	1.225126	1.637312	2.348802
-0.05864	−0.78876	−0.69878	0.088853	0.569423	0.339464	0.920864	3.572709	3.853921
1.065502	0.63154	−1.4992	−1.78774	−1.71091	−1.16574	0.955043	1.07167	0.107119

ith previous sample. The Euclidean distance matrix is first calculated according to Eq. (11):

$$\text{dist} = \begin{bmatrix} 0 & 1.506 & 1.409 & 1.427 & 1.742 & 2.220 & 2.146 & 2.446 & 4.601 \\ 1.506 & 0 & 0.883 & 2.653 & 2.857 & 2.937 & 2.528 & 1.447 & 3.918 \\ 1.409 & 0.883 & 0 & 2.182 & 2.585 & 2.750 & 2.296 & 1.922 & 4.552 \\ 1.427 & 2.653 & 2.182 & 0 & 0.867 & 1.594 & 1.727 & 3.207 & 5.264 \\ 1.742 & 2.857 & 2.585 & 0.867 & 0 & 0.872 & 1.305 & 3.056 & 4.741 \\ 2.220 & 2.937 & 2.750 & 1.594 & 0.872 & 0 & 0.712 & 2.710 & 4.162 \\ 2.146 & 2.528 & 2.296 & 1.727 & 1.305 & 0.712 & 0 & 2.200 & 3.948 \\ 2.446 & 1.447 & 1.922 & 3.207 & 3.056 & 2.710 & 2.200 & 0 & 2.686 \\ 4.601 & 3.918 & 4.552 & 5.264 & 4.741 & 4.162 & 3.948 & 2.686 & 0 \end{bmatrix}$$

Then according to Eq. (12), the weighted adjacency matrix of G_{in} can be obtained:

$$\bar{\text{A}}_{in} = \begin{bmatrix} 1 & 0.162 & 0.169 & 0.167 & 0.143 & 0.112 & 0.116 & 0.099 & 0.032 \\ 0.169 & 1 & 0.229 & 0.098 & 0.088 & 0.085 & 0.104 & 0.174 & 0.053 \\ 0.175 & 0.226 & 1 & 0.120 & 0.099 & 0.091 & 0.114 & 0.137 & 0.038 \\ 0.168 & 0.094 & 0.118 & 1 & 0.220 & 0.155 & 0.144 & 0.0721 & 0.0270 \\ 0.137 & 0.078 & 0.090 & 0.212 & 1 & 0.211 & 0.170 & 0.071 & 0.031 \\ 0.110 & 0.077 & 0.084 & 0.151 & 0.216 & 1 & 0.234 & 0.086 & 0.042 \\ 0.110 & 0.091 & 0.102 & 0.138 & 0.173 & 0.237 & 1 & 0.107 & 0.042 \\ 0.122 & 0.192 & 0.155 & 0.086 & 0.092 & 0.108 & 0.136 & 1 & 0.109 \\ 0.111 & 0.133 & 0.113 & 0.093 & 0.107 & 0.125 & 0.132 & 0.186 & 1 \end{bmatrix}$$

After passing the first GCN layer of the GDAE, we can obtained $\boldsymbol{x}^1_{current} \in \mathbb{R}^{1\times 3}$ according to Eq. (13):

$$\begin{aligned} \boldsymbol{x}^1_{current} = \sigma\big((\boldsymbol{x}_{current} &+ 0.162\boldsymbol{x}_{lag1} + 0.169\boldsymbol{x}_{lag2} + 0.167\boldsymbol{x}_{lag3} \\ &+0.143\boldsymbol{x}_{lag4} + 0.112\boldsymbol{x}_{lag5} + 0.116\boldsymbol{x}_{lag6} + 0.099\boldsymbol{x}_{lag7} \\ &+0.032\boldsymbol{x}_{lag8})\text{W}^0\big) \end{aligned} \tag{25}$$

where $\text{W}^0 \in \mathbb{R}^{3\times 3}$ is the learned parameter of the first GCN layer from the training phase. It can be found that with the designed $\bar{\text{A}}_{in}$, samples closer to the current sample are given a larger weight, while those further away are given a smaller weight. The same steps are performed for $\boldsymbol{x}_{lagi}(i = 1, 2, \ldots, 8)$ to obtain the corresponding $\boldsymbol{x}^1_{lagi} \in \mathbb{R}^{1\times 3}$. Then the updated adjacency matrix $\bar{\text{A}}^1$ can be obtained by $\boldsymbol{x}^1_{current}$ and $\boldsymbol{x}^1_{lagi}$ according to Eq. (11) and Eq. (12). Finally passing the second GCN layer, the key feature $\boldsymbol{h}_{current} \in \mathbb{R}^{1\times 2}$ can be computed by $\bar{\text{A}}^1$ and $\text{W}^1 \in \mathbb{R}^{3\times 2}$:

$$\begin{aligned} \boldsymbol{h}_{current} = \sigma\Big(\big(&\bar{A}^1_{11}\boldsymbol{x}^1_{current} + \bar{A}^1_{12}\boldsymbol{x}^1_{lag1} + \bar{A}^1_{13}\boldsymbol{x}^1_{lag2} + \bar{A}^1_{14}\boldsymbol{x}^1_{lag3} + \bar{A}^1_{15}\boldsymbol{x}^1_{lag4} \\ &+\bar{A}^1_{16}\boldsymbol{x}^1_{lag5} + \bar{A}^1_{17}\boldsymbol{x}^1_{lag6} + \bar{A}^1_{18}\boldsymbol{x}^1_{lag7} + \bar{A}^1_{19}\boldsymbol{x}^1_{lag8}\big)\text{W}^1\Big) \end{aligned} \tag{26}$$

5.1.4. Monitoring results for Numerical example

We visualized the first two dimensions with the largest variance of the features extracted by DPCA, DKPCA, DSAE and the proposed GDAE, the visualization results are shown in Fig. 7. In Fig. 7(a-c), normal samples and fault samples are largely mixed

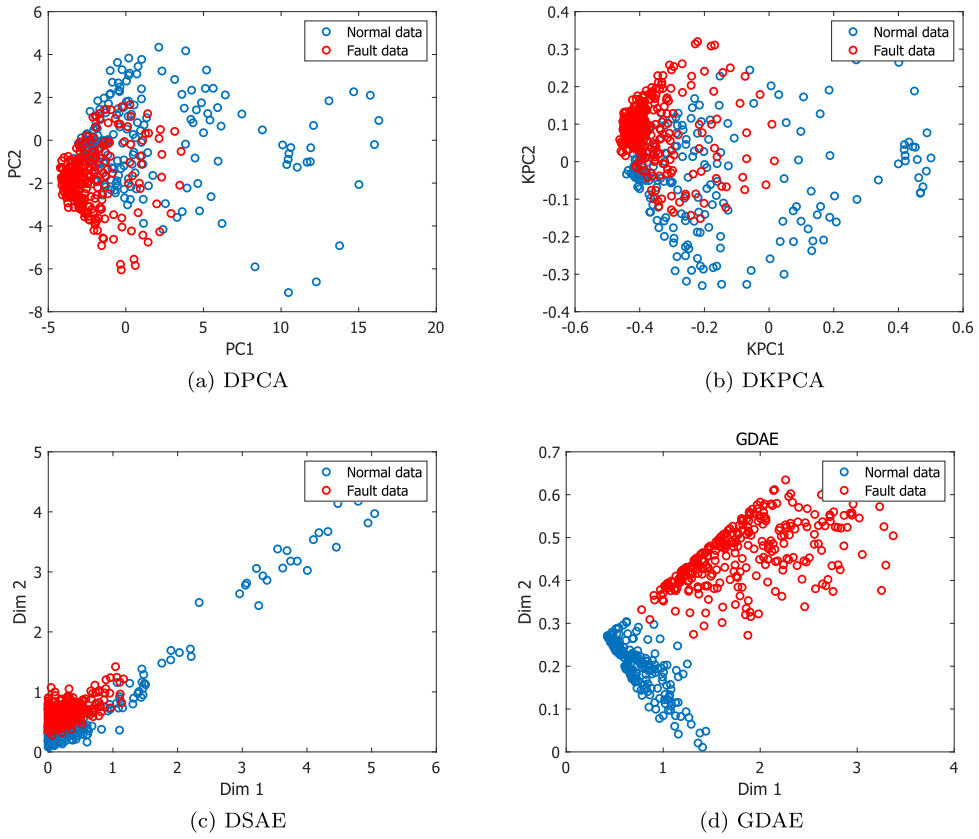


Fig. 7. Visualization of the normal and the fault samples of the numerical example in two-dimensional spaces of DPCA, DKPCA, DSAE, and GDAE. Red dots and blue dots indicate fault data and normal data, respectively.

up. However, Fig. 7d shows that our proposed GDAE can almost distinguish normal samples and fault samples.

We also give the monitoring performance of the four methods for this numerical example using T^2 statistic and *SPE* statistic, as is shown in Table 2. We use the missed detection rate (MDR) and false alarm rate (FAR) (shown in parentheses) to measure the monitoring performance. MDR refers to the ratio of fault samples being misidentified as normal samples during monitoring. FAR refers to the ratio of normal samples being misidentified as abnormal samples. In this paper, when MDR < 50% and FAR < 5%, we consider that the fault is successfully detected. Please note that if MDR is larger than 50%, the detection performance is even worse than random guess. When FAR < 5%, MDR with smaller values indicates better performance. From Table 2, it can be found that none of the four methods of the *SPE* statistic detects this fault, so the following analysis will only focus on the results of the T^2 statistic.

Fig. 8 presents the detailed monitoring results for this numerical example with T^2 statistic. In this figure, the blue dots represent the first 200 normal samples; the red dots represent the remaining fault samples; the black dotted line is the control limit. Blue dots above the control limit cause false alarm, while red dots below the control limit lead to missed detection. It is clearly observed that DPCA, DKPCA and DSAE all have high FARs, which are larger than 5% as can be seen in Table 2. However, the proposed GDAE can achieve 27.0% MDR with no false alarm. This means that only the proposed GDAE correctly identifies the fault in the numerical example, which is consistent with the results of the two-dimensional scatter plot shown earlier.

From the two-dimensional visualization results and the monitoring performance using the monitoring statistics above, we can conclude that dynamic information is the key to detect the fault in this numerical example and the proposed GDAE can better model dynamic information than DPCA, DKPCA and DSAE so detect this fault correctly.

5.2. Experiments on Tennessee Eastman process

Tennessee Eastman process (TEP), proposed by Downs and Vogel (1993), is a benchmark in process monitoring. TEP is a simulation of an actual chemical process and has been widely used as a source data for evaluating different fault detection methods. The MATLAB code for generating TEP data can be downloaded from http://depts.washington.edu/control/LARRY/TE/download.html. The data generated by TEP are nonlinear, dynamic, and strong coupling. Fig. 9 is the control structure of TEP. TEP is composed of five operating units, including reactor, condenser, compressor, stripper, and vapor liquid separator.

In our experiment, 52 variables are selected as monitoring variables, including 22 continuous process variables, 19 combinations, and 11 manipulated variables. There are 21 fault modes simulated by TEP, which are shown in Table 3. The training data contains 500 normal samples. The testing data contains 960 samples and the fault occurred from the 161st sample, in other words, the first 160 samples are normal, and the 161st to the 960th samples are fault samples.

5.2.1. Autocorrelation of TEP

We also determine the number of time lags for TEP by autocorrelation function. Fig. 10 shows the number of time lags determined from autocorrelation function of reactor pressure, reactor temperature, compressor recycle flow, and the sum of 52 variables, respectively. Fig. 10a illustrates that there is a strong autocorrela-

Table 2
Missed detection rate (%) and false alarm rate (%) (shown in parentheses) of DPCA, DKPCA, DSAE, and GDAE in numerical example.

DPCA		DKPCA		DSAE		GDAE	
T^2	*SPE*	T^2	*SPE*	T^2	*SPE*	T^2	*SPE*
48.5(10.9)	100(4.66)	44.8(7.25)	100(0.00)	47.7(6.77)	58.7(8.85)	**27.0(0.00)**	78.7(1.00)

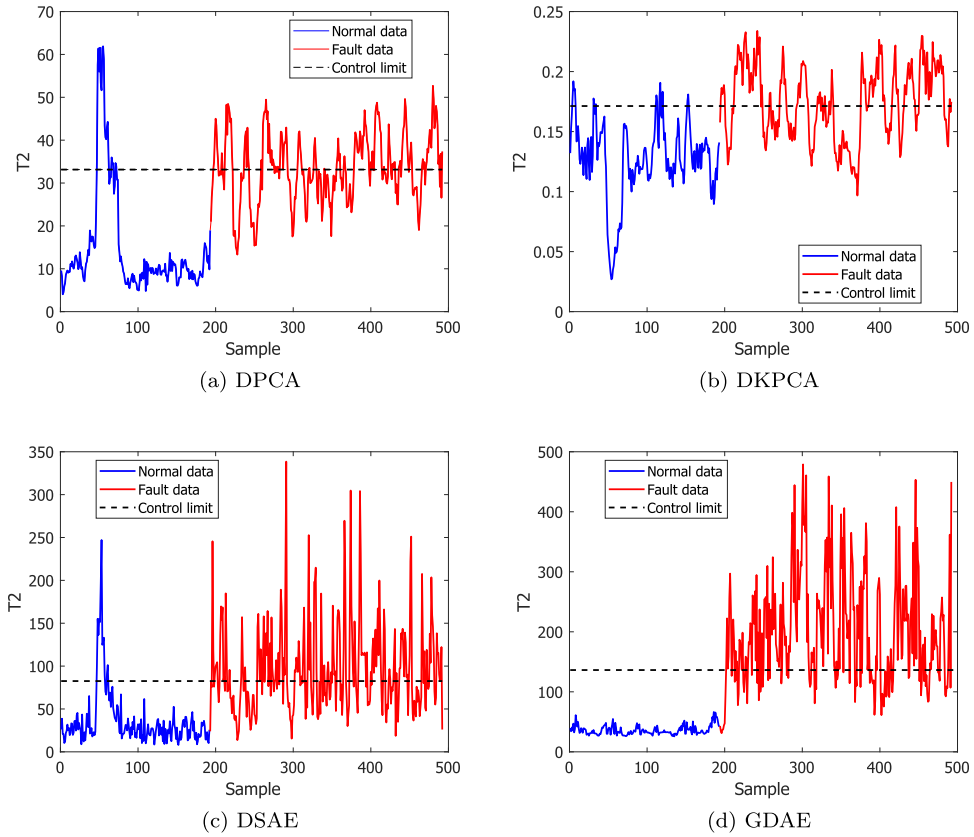


Fig. 8. Monitoring results of 4 different methods for Numerical example.

tion of reactor pressure. In Fig. 10b, it can be found that the autocorrelation of reactor temperature is relatively weaker compared to reactor pressure. Fig. 10c shows that the ACF cuts off with no lag, which indicates that there is no serial correlation in compressor recycle flow, thus compressor recycle flow can be considered as an independent sampled variable. It can be found that the autocorrelation of each variable in TEP is different. We adopted the autocorrelation of the sum of 52 variables, as is shown in Fig. 10d, to determine the number of time lags l. According to Fig. 10d, we select $l = 10$ for TEP.

Noted that when the time lag is 10, the dimensionality of input data will increase from 52 to 572 if using the data augmentation technique. However, GDAE does not have this problem. In the following experiments, the same time lag ($l = 10$) is used both in offline training and online monitoring.

5.2.2. Parameter setting for TEP

According to Section 5.2.1, we select time lag $l = 10$ for TEP. Based on the CPV rule, the reduced dimensionality ξ was determined as 128 for DPCA to retain 85% of the energy in the eigenspectrum. The reduced dimensionality for DKPCA and DSAE is also set to 128 to ensure fairness. Since the input dimensionality of the proposed GDAE has not changed, so the reduced dimensionality of GDAE is set to 27, which is consistent with the traditional PCA.

The kernel of DKPCA is Gaussian kernel, and the mathematical formula is as follows:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{\eta}\right)$$

the kernel parameter η is $15((l+1)d)\bar{\delta}$, where $\bar{\delta}$ denotes the average of the standard deviations of different variables (Zhao et al., 2006) and $(l+1)d$ denotes the data dimension after using the data augmentation technique. The structure of our proposed GDAE is set to 52–52-27-52–52. To verify the superiority of our proposed GDAE for handling dynamic information, we set the structure of DSAE to 572–572-128-572–572. The training parameters of GDAE are as follows: for offline modeling, the learning rate is set to 0.005 and the number of iterations is set to 80 in the pre-training stage; In the fine-tuning stage, the learning rate is set to 0.0001 and the number of iterations is set to 20. In the whole training process, we use the Adam algorithm (Kingma and Ba, 2015) to optimize the parameters of GDAE.

In Section 3, we present a weighted adjacency matrix $\bar{\mathrm{A}}$ to assign different weights to samples within the lag time l. The elements of $\bar{\mathrm{A}}$ are updated with the node features during the training process, while the data augmentation technique fixes the weight of each sample to $\frac{1}{l}$. Fig. 11 displays the heatmaps of the initial weighted adjacency matrix $\bar{\mathrm{A}}$ for Fault 1. Fig. 11a and b show the

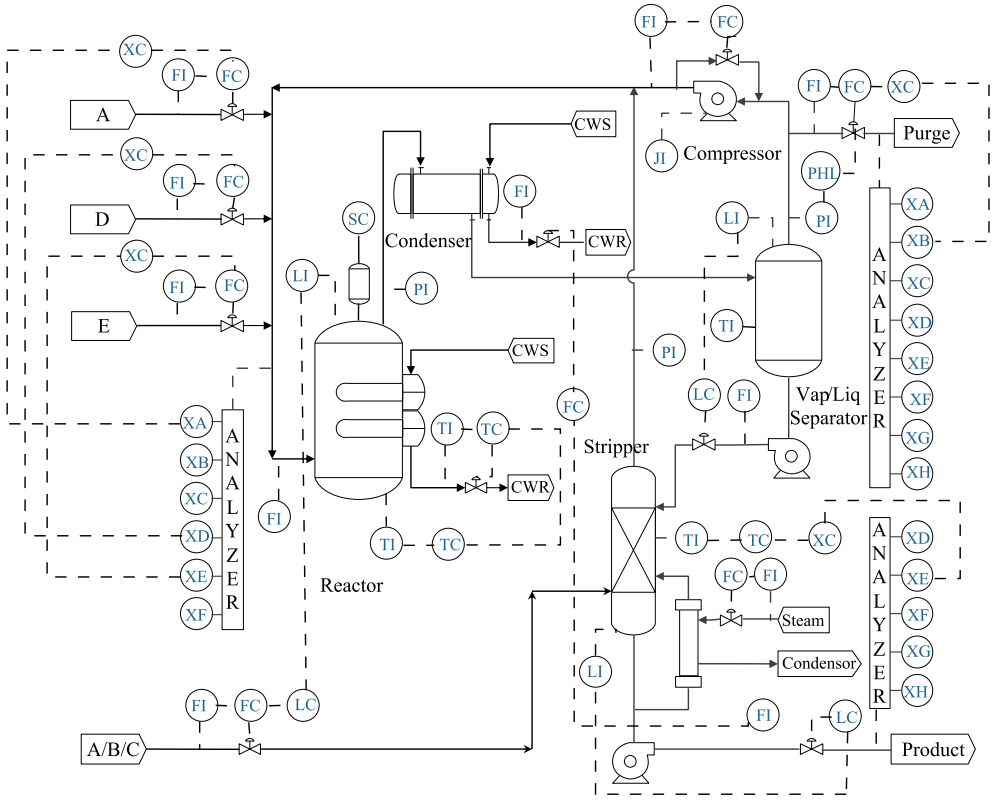


Fig. 9. A diagram of the TEP simulator.

Table 3
TEP fault modes.

Fault No.	Description	Type
1	A/C feed ratio, B composition constant (Stream 4)	Step
2	B composition, A/C ratio constant (Stream 4)	Step
3	D feed temperature (Stream 2)	Step
4	Reactor cooling water inlet temperature	Step
5	Condenser cooling water inlet temperature	Step
6	A feed loss (Stream 1)	Step
7	C header pressure loss (Stream 4)	Step
8	A, B, C feed composition (Stream 4)	Random variation
9	D feed temperature (Stream 2)	Random variation
10	C feed temperature (Stream 4)	Random variation
11	Reactor cooling water inlet temperature	Random variation
12	Condenser cooling water inlet temperature	Random variation
13	Reaction kinetics	Slow drift
14	Reactor cooling water valve	Sticking
15	Condenser cooling water valve	Sticking
16	Unknown	Unknown
17	Unknown	Unknown
18	Unknown	Unknown
19	Unknown	Unknown
20	Unknown	Unknown
21	Valve (Stream 4)	Constant position

average of the initial weighted adjacency matrix of the 50 normal graphs and 50 faulty graphs, respectively. Note that in order to make the non-diagonal elements more distinct, we set the diagonal elements to the maximum value of each line instead of 1. It can be found that under the normal condition, only the sample in the adjacent moment gets larger weight, while the weights of the samples in other moments differ little. In the fault condition, more samples were assigned larger weights. In other words, there is a difference between the dynamic information of normal samples and that of faulty samples.

The experiments were performed on a computer with the following specifications:

CPU: AMD Ryzen 5 2600x; RAM: 8.0 GB;

GPU: NVIDIA GeForce GTX 970;

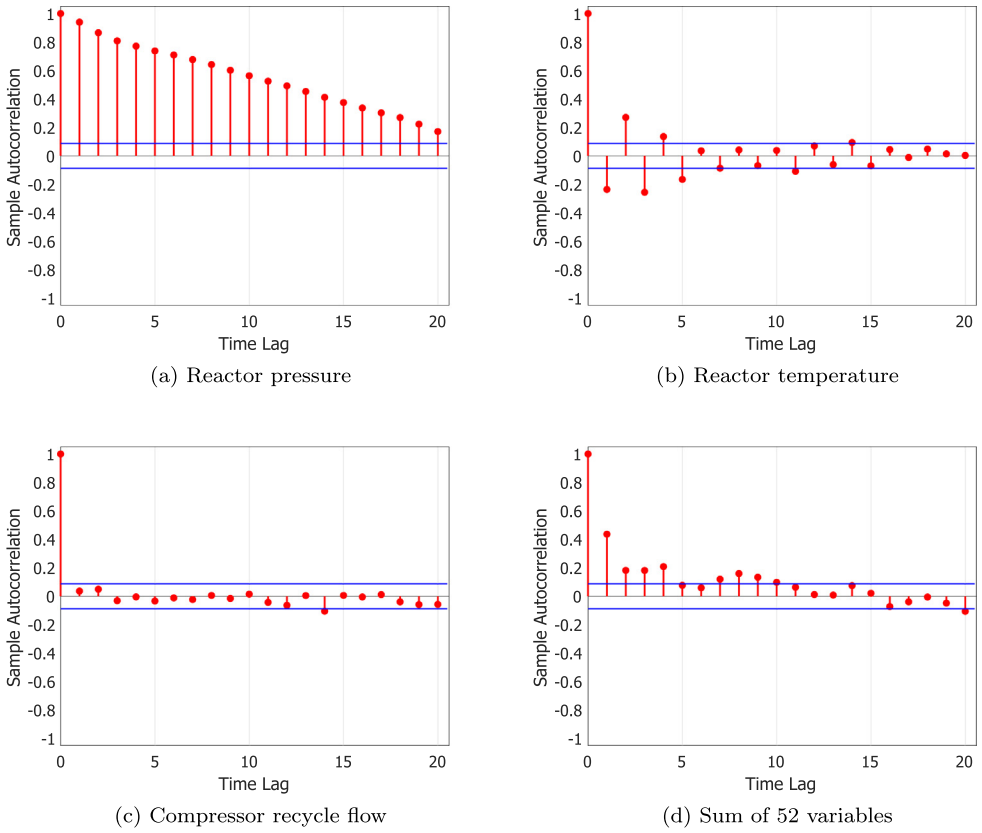
Operating System: Ubuntu 18.04 LTS 64-bit computing (DPCA and DKPCA: MATLAB 2019b; DSAE and GDAE: Python 3.8).

Fig. 12 plots the reconstruction loss of the second graph convolution layer in the pre-training phase. It can be found that GDAE is convergent. The offline training time of the proposed algorithm requires 60.08 s. The testing time of GDAE for one sample is 0.024 s, which is completely acceptable in industrial processes.

5.2.3. Case studies

In this section, the performance in monitoring TEP of the proposed GDAE is compared with DPCA, DKPCA, and DSAE. For testing all the 21 faults, MDR and FAR(shown in parentheses) are recorded together in Table 4. In Table 4, the results are obtained by averaging over 10 times of experiments and the best performance for each fault is highlighted in bold. The source code of GDAE can be found in https://github.com/luliu-fighting/Graph-Dynamic-Autoencoder.

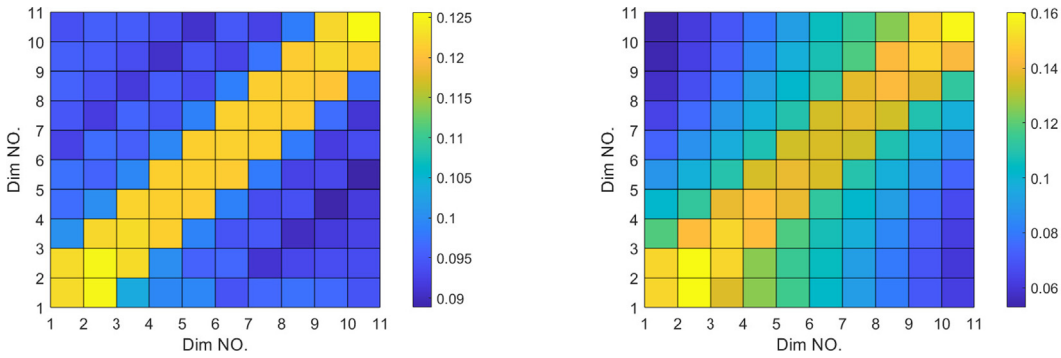
From Table 4, it is easy to find that although both T^2 and *SPE* statistics can detect certain faults in these methods, the performances of using T^2 statistic for fault detection in TEP are better than those of using *SPE* statistic and the use of SPE statistic had little effect on the number of best cases for DPCA, DKPCA, DSAE and the proposed GDAE. Therefore, in the following, the comparison and analysis will only focus on the results of the T^2 statistic.



(a) Reactor pressure (b) Reactor temperature

(c) Compressor recycle flow (d) Sum of 52 variables

Fig. 10. Number of time lags for TEP determined from autocorrelation function of different measurements.



(a) Average weighted adjacency matrix of normal graph (b) Average weighted adjacency matrix of fault graph

Fig. 11. Illustrations of average weighted adjacency matrix under normal and fault conditions for Fault 1.

As can be seen from Table 4, none of the four methods can correctly detect all faults. However, with FAR < 5%, GDAE obtains the lowest MDRs of the T^2 statistic in 10 cases. DPCA gives 2 best cases, DKPCA gives the best performances with 3 cases, and DSAE has 4 best cases. DKPCA, DSAE, and GDAE have better performance than DPCA because they are all nonlinear methods that can better represent the nonlinear TEP data. Although DKPCA, DSAE, and GDAE are all nonlinear methods, DSAE and GDAE perform better than DKPCA. The reason is that DKPCA uses prefixed kernel function and parameter to extract nonlinear features, while DSAE and GDAE learn the parameters of the neural network adaptively by back-propagation and thus more suitable for complex multivariate nonlinear data. DSAE and GDAE are both neural network-based methods, but the performances of GDAE are much better than those of DSAE. The reason is that GDAE uses graph convolution to model the dynamic information between samples and assigns different weights to the samples, while DSAE uses the data augmentation techniques to simply concatenate the samples. The result also indicates that GDAE can more sufficiently exploit the dynamic information of the data.

Figs. 13–15 present the detailed test results for Fault 4, 10, and 17. In these figures, the blue dots represent the first 160 normal samples; the red dots represent the remaining fault samples; the black dotted line is the control limit.

Fig. 13 shows the results of the four methods for Fault 4. We can clearly observe that DPCA, DKPCA, and DSAE cannot detect this

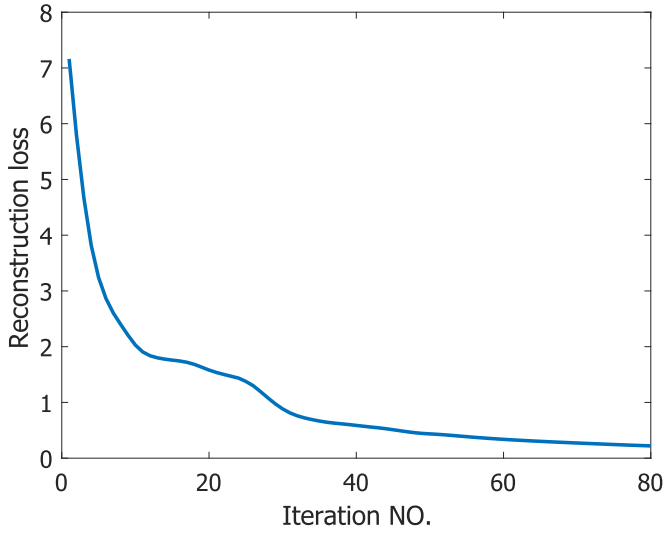


Fig. 12. Convergence plot of GDAE.

fault mode correctly, while GDAE can perfectly separate the faulty samples from the normal ones. This result can also be found in Table 4. The MDRs of DPCA, DKPCA, and DSAE are 74.94%, 96.96%, and 75.19%, respectively. The MDRs of all these three methods are much greater than 50%. For GDAE, it can achieve 0.00% MDR with only 0.63% FAR, which is much better than the other three methods. From Table 3, it can be found that Fault 4 is related to the reactor temperature. There is a strong autocorrelation of reactor temperature in Fig. 10b, b, and this autocorrelation is not a simple linear relationship. Therefore, making full use of dynamic information is crucial to detecting Fault 4. This experimental result indicates that graph convolution is more appropriate for handling dynamic data than the data augmentation technique that simply concatenate the samples.

Fig. 14 illustrates the results of Fault 10. The experimental results of Fault 10 show that all the four methods can achieve no false alarm on this fault mode. However, the MDRs of T^2 statistic of DPCA and DKPCA are 56.84%, 76.96%, respectively, which are larger than 50% and means that these two methods cannot correctly detect Fault 10. For DSAE and GDAE, the MDRs of these two methods are 42.41%, 39.88%, respectively. According to Fig. 14 and the experimental results, we can find that although DKPCA, DSAE, and GDAE are all nonlinear methods, DSAE and GDAE can successfully detect Fault 10 while DKPCA cannot do so. The reason is that the prefixed kernel function and parameter of DKPCA may not suitable for all variables in complex multivariate data, while DSAE and GDAE are based on neural network that uses gradient descent and backpropagation algorithm to make the model parameters reach the optimal value for each variable.

From Table 4, it can be found that DPCA, DKPCA, DSAE, and GDAE can all successfully detect Fault 17. For the linear dimensionality reduction method DPCA, it can achieve no false alarm rate and the MDR is 9.49%. For nonlinear methods, with FAR < 5%, the MDR of the T^2 statistic of DKPCA, DSAE, GDAE, is 7.47%, 6.96%, 5.75%, respectively. Fig. 15 plots the results of these four methods on Fault 17 and the small overlay plot is a partial magnification of the statistical results for the first 160 samples. Based on Table 4 and Fig. 15, for Fault 17, it can be found that DKPCA, DSAE, and GDAE have much better performance than DPCA, which indicates that nonlinear method is more suitable for monitoring nonlinear data process data.

5.2.4. Ablation study

To verify the effectiveness of the graph structure and the weighted adjacency matrix, we conducted ablation experiments. We consider DSAE as the baseline. We replace the data augmentation technique used by DSAE with a graph structure and keep the weight of each sample as $\frac{1}{I}$ to obtain GDAE with same weights (GDAE-SW). Then we compare them with GDAE. Table 5 presents the experimental results with T^2 statistic. The up arrow and down arrow indicate higher and lower MDR or FAR than the baseline, respectively, and the horizontal line indicates the result is equal.

Compared with the baseline (DSAE), GDAE-SW has a lower MDR than DSAE in 7 cases, a higher MDR in 4 cases, and the same MDR with DSAE in 1 case, which means GDAE-SW has better performance than DSAE. This demonstrates that using graph structure

Table 4
Missed detection rate (%) and false alarm rate (%) (shown in parentheses) of DPCA, DKPCA, DSAE, and GDAE in TEP.

Fault No.	DPCA		DKPCA		DSAE		GDAE	
	T^2	*SPE*	T^2	*SPE*	T^2	*SPE*	T^2	*SPE*
1	**0.00(2.50)**	0.00(23.1)	**0.00(2.50)**	**0.00(2.50)**	**0.00(3.75)**	**0.00(1.25)**	0.25(0.00)	0.25(0.00)
2	0.76(0.00)	0.13(17.5)	0.51(0.63)	0.76(0.63)	**0.25(1.88)**	0.63(0.00)	1.38(5.63)	1.38(0.00)
3	99.7(0.00)	76.5(31.9)	98.0(0.00)	99.1(0.00)	90.8(0.00)	94.9(1.88)	90.2(22.5)	99.1(0.00)
4	74.9(3.13)	0.00(33.1)	97.0(0.00)	97.9(0.00)	75.2(0.00)	96.1(0.00)	**0.00(0.63)**	83.2(0.00)
5	74.7(5.00)	30.3(33.1)	78.3(5.00)	78.8(5.00)	68.0(3.75)	72.7(0.00)	59.1(0.63)	75.0(0.00)
6	0.25(0.00)	0.00(11.9)	0.63(0.00)	0.76(0.00)	**0.00(0.00)**	**0.00(0.00)**	**0.00(0.00)**	0.38(0.00)
7	0.00(6.25)	0.00(12.5)	0.00(5.00)	0.00(5.00)	0.00(5.63)	21.8(5.63)	**0.00(4.38)**	**0.00(1.88)**
8	1.77(0.00)	0.13(11.3)	2.15(0.00)	3.13(0.00)	1.77(0.63)	2.15(0.00)	**1.63(1.88)**	2.25(0.00)
9	99.7(1.25)	80.5(20.0)	98.1(1.88)	98.1(1.88)	91.8(25.6)	96.2(16.2)	91.1(13.7)	99.1(4.38)
10	56.8(0.00)	18.0(10.6)	77.0(0.00)	77.5(0.00)	42.4(0.00)	52.9(0.00)	**39.9(0.00)**	61.9(0.00)
11	49.7(0.00)	0.38(20.0)	74.3(1.25)	78.3(4.38)	50.6(3.13)	84.7(0.00)	**27.0(0.63)**	61.1(0.00)
12	0.00(5.00)	0.00(10.6)	**1.14(3.75)**	1.38(0.63)	0.00(10.6)	1.14(6.88)	0.00(18.1)	2.75(0.00)
13	**3.80(0.00)**	3.54(10.0)	4.05(4.38)	6.88(1.38)	**3.80(0.63)**	5.06(0.00)	4.38(2.50)	6.88(0.00)
14	0.00(5.00)	0.00(28.1)	**0.00(4.38)**	**0.00(4.38)**	0.00(5.00)	0.00(5.00)	**0.00(1.25)**	2.25(0.00)
15	96.2(0.00)	83.7(20.0)	96.7(1.25)	86.7(3.75)	79.6(0.00)	90.9(0.00)	81.4(0.00)	91.9(0.00)
16	82.7(0.00)	17.1(24.4)	88.7(6.25)	89.1(6.25)	56.7(35.6)	69.0(11.9)	55.9(35.0)	75.4(8.75)
17	9.49(0.00)	1.52(13.8)	7.47(0.63)	9.40(1.25)	6.96(2.50)	16.3(0.00)	**5.75(0.00)**	18.1(0.00)
18	10.6(0.00)	6.08(18.8)	11.4(0.00)	12.6(0.00)	10.4(0.00)	11.0(0.00)	**9.00(0.00)**	10.6(0.00)
19	95.9(0.00)	0.51(11.9)	95.7(0.00)	95.7(0.00)	97.6(1.25)	99.7(0.00)	94.7(8.75)	99.9(1.88)
20	49.6(0.00)	14.2(10.0)	58.6(0.63)	58.9(0.63)	42.7(0.00)	61.0(0.00)	**31.6(0.00)**	53.1(0.00)
21	53.8(0.00)	43.9(20.0)	74.0(2.50)	74.2(0.63)	67.6(1.88)	76.8(0.00)	51.7(3.75)	84.6(0.00)
Best	2		3		4		10	

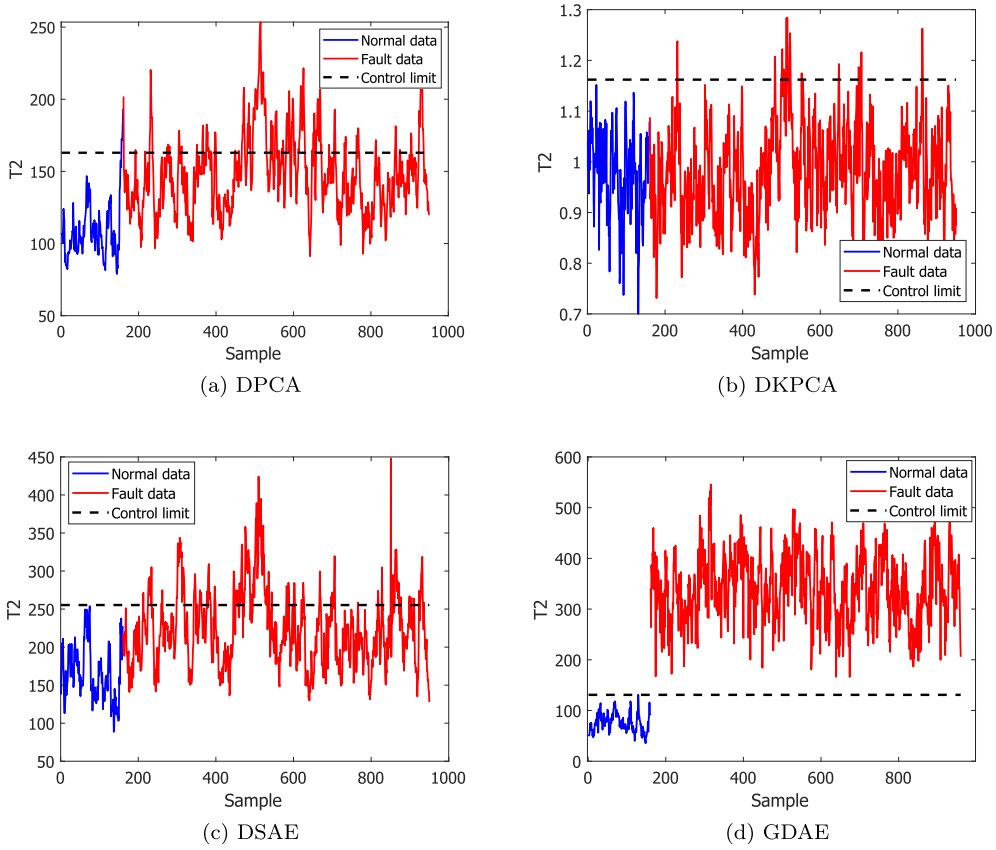


Fig. 13. Monitoring results of 4 different methods for Fault 4.

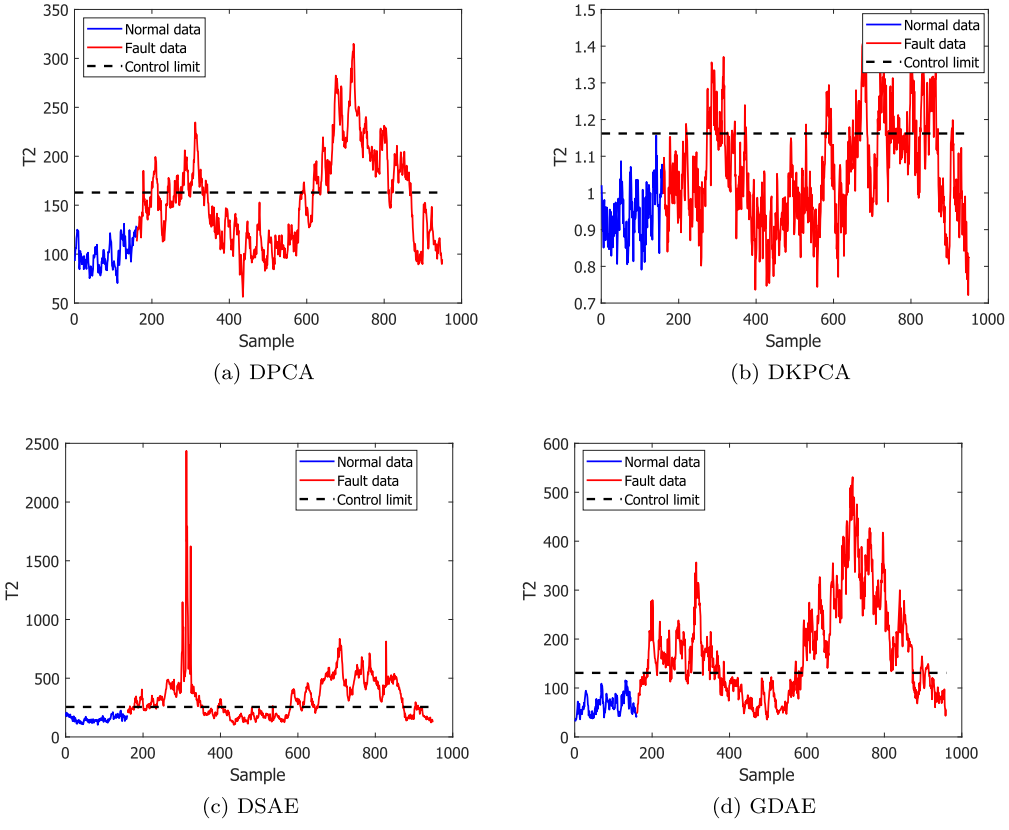


Fig. 14. Monitoring results of 4 different methods for Fault 10.

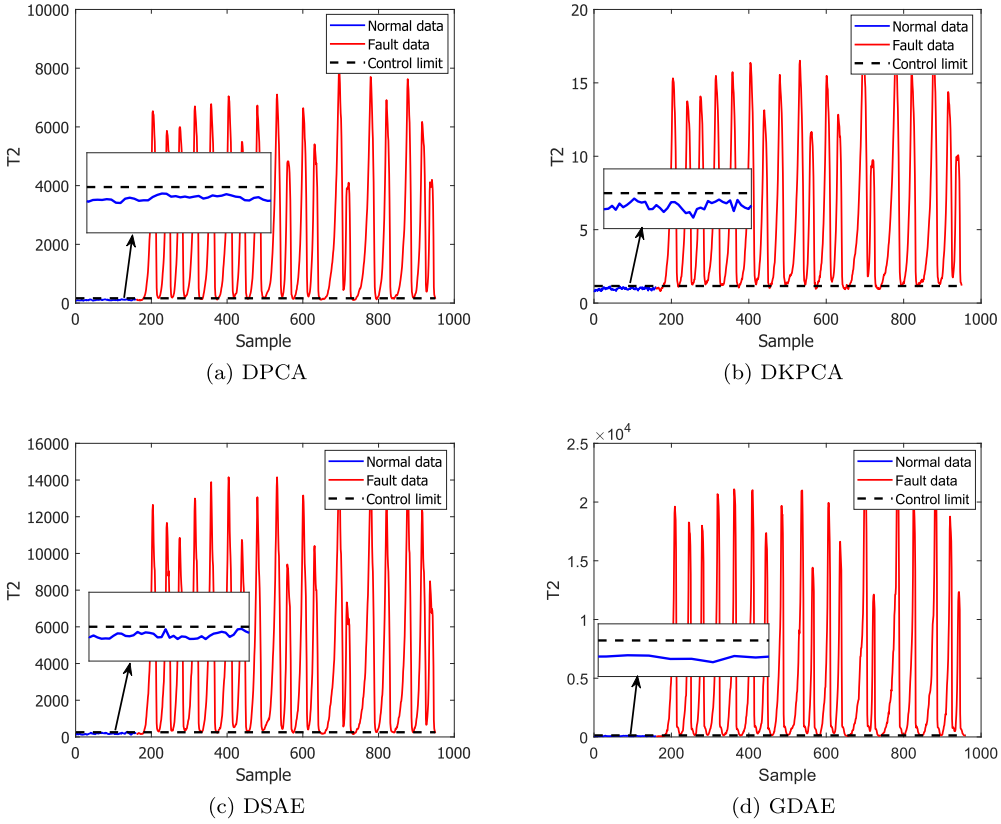


Fig. 15. Monitoring results of 4 different methods for Fault 17.

to process dynamic information is more effective than traditional data augmentation technique. After assigning different weights to the samples using the weighted adjacency matrix, GDAE obtains a lower MDR than DSAE in 9 cases, a higher MDR than DSAE in 3 cases, and the same MDR with DSAE in 1 case. It can be found that the performance has been further improved compared to the GDAE-SW. This indicates that it is essential to assign appropriate weights to the samples.

We summarize all the experiments in this section below:

1. Although no single method can detect all fault modes, GDAE clearly outperforms DPCA, DKPCA, and DSAE in terms of the number of best cases.
2. Different from DPCA, DKPCA, and DSAE which use the data augmentation technique to incorporate the dynamic information of different samples within the lag time, GDAE adopts graph con-

Table 5
Missed detection rate (%) and false alarm rate (%) of DSAE, GDAE with same weights (GDAE-SW), and GDAE in TEP.

Fault NO.	DSAE		GDAE-SW		GDAE	
	MDR	FAR	MDR	FAR	MDR	FAR
1	0.00	3.75	0.00(-)	0.00	0.25(↑)	0.00
2	0.25	1.88	1.13(↑)	5.00	1.38(↑)	5.63
3	90.88	0.00	91.88	10.63	90.25	22.50
4	75.19	0.00	0.00(↓)	0.00	0.00(↓)	0.63
5	67.97	3.75	60.75	1.88	59.13	0.63
6	0.00	0.00	0.13(↑)	0.63	0.00(-)	0.00
7	0.00	5.63	0.00	6.88	0.00(↓)	4.38
8	1.77	0.63	2.25(↑)	8.13	1.63(↓)	1.88
9	91.77	25.63	93.00	13.75	91.13	13.75
10	42.41	0.00	41.75(↓)	0.00	39.88(↓)	0.00
11	50.63	3.13	26.88(↓)	3.13	27(↓)	0.63
12	0.00	10.63	0.25	16.25	0.00	18.13
13	3.80	0.63	4.13(↑)	3.75	4.38(↑)	2.50
14	0.00	5.00	0.00(↓)	3.13	0.00(↓)	1.25
15	79.62	0.00	84.00	0.00	81.38	0.00
16	56.71	35.63	61.63	32.50	55.88	35.00
17	6.96	2.50	6.75(↓)	0.63	5.75(↓)	0.00
18	10.38	0.00	9.13(↓)	0.00	9.00(↓)	0.00
19	97.59	1.25	94.88	8.13	94.75	8.75
20	42.66	0.00	38.25(↓)	1.25	31.63(↓)	0.00
21	67.59	1.88	61.00	2.50	51.75	3.75

volutional neural network to model dynamic information adaptively. Experimental results also show that GDAE is more suitable for handling dynamic data than the other three methods.
3. Adaptively assigning appropriate weights to samples within the lag time can make better use of dynamic information and thus improve the performance of dynamic fault detection.
4. Although GDAE requires more time to train, given the excellent performance of GDAE, the trade-off between training time and performance seems to be reasonable.

6. Conclusion

In this paper, we propose a novel nonlinear dynamic method called graph dynamic autoencoder (GDAE) for fault detection. Traditional dynamic methods that using the data augmentation technique inevitably increase the dimensionality of the data and the samples within the lag time are considered equally important. GDAE overcomes these shortcomings. GDAE adopts graph convolution to incorporate the dynamic information of the samples and uses Euclidean distance normalized by softmax to assign different weights to the samples within the lag time. In addition, the encoder-decoder structure is used to extract the low-dimensional features of the data. GDAE combines the advantages of graph structure and stacked autoencoder.

GDAE are compared with several other dynamic methods, such as DPCA, DKPCA, and DSAE. According to the experiment results in Section 5, it is clear that GDAE performs much better than the other three methods. GDAE can be considered a promising alternative for nonlinear dynamic process monitoring.

CRediT authorship contribution statement

Lu Liu: Methodology, Investigation, Software, Writing – original draft, Writing – review & editing. **Haitao Zhao:** Methodology, Writing – review & editing. **Zhengwei Hu:** Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research is sponsored by National Natural Science Foundation of China (62173143 and 61973122).

References

Ammiche, Mustapha, Kouadri, Abdelmalek, Bakdi, Azzeddine, 2018. A combined monitoring scheme with fuzzy logic filter for plant-wide Tennessee Eastman process fault detection. Chem. Eng. Sci. 187, 269–279.
Bengio, Yoshua, Lamblin, Pascal, Popovici, Dan, Larochelle, Hugo, et al., 2007. Greedy layer-wise training of deep networks. Adv. Neural Informat. Process. Syst. 19, 153.
Bounoua, Wahiba, Bakdi, Azzeddine, 2021. Fault detection and diagnosis of nonlinear dynamical processes through correlation dimension and fractal analysis based dynamic kernel PCA. Chem. Eng. Sci. 229, 116099.
Choi, Sang Wook, Lee, In-Beum, 2004. Nonlinear dynamic process monitoring based on dynamic kernel PCA. Chem. Eng. Sci. 59 (24), 5897–5908.
Downs, James J, Vogel, Ernest F, 1993. A plant-wide industrial process control problem. Comput. Chem. Eng. 17 (3), 245–255.
Fan Jicong, Wang Wei, Zhang Haijun, 2017. Autoencoder based high-dimensional data fault detection system. In: 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), pp. 1001–1006.
Ge, Zhiqiang, 2017. Review on data-driven modeling and monitoring for plant-wide industrial processes. Chemomet. Intell. Lab. Syst. 171, 16–25.
Ge, Zhiqiang, Song, Zhihuan, Gao, Furong, 2013. Review of recent research on data-based process monitoring. Ind. Eng. Chem. Res. 52 (10), 3543–3562.
Härdle, Wolfgang Karl, Simar, Léopold, 2015. Applied multivariate statistical analysis. Springer, Berlin.
He Xiangnan, Deng Kuan, Wang Xiang, Li Yan, 2020. Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 639–648.
Hu, Zhengwei, Peng, Jingchao, Zhao, Haitao, 2021. Dynamic neural orthogonal mapping for fault detection. Int. J. Machine Learn. Cybernet. 12 (5), 1501–1516.
Jiang, Li, Ge, Zhiqiang, Song, Zhihuan, 2017. Semi-supervised fault classification based on dynamic Sparse Stacked auto-encoders model. Chemomet. Intell. Lab. Syst. 168, 72–83.
Jiang, Qingchao, Yan, Xuefeng, Huang, Biao, 2019. Review and perspectives of data-driven distributed monitoring for industrial plant-wide processes. Ind. Eng. Chem. Res. 58 (29), 12899–12912.
Jiang, Qingchao, Yan, Xuefeng, Huang, Biao, 2019. Deep discriminative representation learning for nonlinear process fault detection. IEEE Trans. Autom. Sci. Eng. 17 (3), 1410–1419.
Joe Qin, S., 2012. Survey on data-driven industrial process monitoring and diagnosis. Annual Revi. Control 36 (2), 220–234.
Kingma Diederik P., Ba Jimmy, 2015. Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations (ICLR).
Ku, Wenfu, Storer, Robert H., Georgakis, Christos, 1995. Disturbance detection and isolation by dynamic principal component analysis. Chemomet. Intell. Lab. Syst. 30 (1), 179–196.
LeCun, Yann, Bengio, Yoshua, Hinton, Geoffrey, 2015. Deep learning. Nature 521 (7553), 436–444.
Li, Gang, Joe Qin, S., Zhou, Donghua, 2014. A new method of dynamic latent-variable modeling for process monitoring. IEEE Trans. Ind. Electron. 61 (11), 6438–6445.
Lv, Feiya, Wen, Chenglin, Liu, Meiqin, Bao, Zhejing, 2017. Weighted time series fault diagnosis based on a stacked sparse autoencoder. J. Chemom. 31 (9), e2912.
Malinowski, Edmund R, Howery, Darryl G, 1980. Factor analysis in chemistry. Wiley.
Samuel, Raphael Tari, Cao, Yi, 2016. Nonlinear process fault detection and identification using kernel pca and kernel density estimation. Syst. Sci. Control Eng. 4 (1), 165–174.
Shang, Jun, Chen, Maoyin, Zhang, Hanwen, 2018. Fault detection based on augmented kernel Mahalanobis distance for nonlinear dynamic processes. Comput. Chem. Eng. 109, 311–321.
Tanjin Amin, Md., Imtiaz, Syed, Khan, Faisal, 2018. Process system fault detection and diagnosis using a hybrid technique. Chem. Eng. Sci. 189, 191–211.
Wu, Zonghan, Pan, Shirui, Chen, Fengwen, Long, Guodong, Zhang, Chengqi, Philip, S Yu, 2021. A comprehensive survey on graph neural networks. IEEE Trans. Neural Networks Learn. Syst. 32 (1), 4–24.
Yin, Jie, Yan, Xuefeng, 2019. Mutual information–dynamic stacked sparse autoencoders for fault detection. Ind. Eng. Chem. Res. 58 (47), 21614–21624.
Yin, Shen, Ding, Steven X., Xie, Xiaochen, Luo, Hao, 2014. A review on basic data-driven approaches for industrial process monitoring. IEEE Trans. Industr. Electron. 61 (11), 6418–6428.
Yin, Xueyan, Genze, Wu., Wei, Jinze, Shen, Yanming, Qi, Heng, Yin, Baocai, 2021. Multi-stage attention spatial-temporal graph networks for traffic prediction. Neurocomputing 428, 42–53.
Ying Rex, He Ruining, Chen Kaifeng, Eksombatchai Pong, Hamilton William L., Leskovec Jure, 2018. Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 974–983.
Zhang, Zehan, Jiang, Teng, Li, Shuanghong, Yang, Yupu, 2018. Automated feature learning for nonlinear process monitoring–An approach using stacked denoising autoencoder and k-nearest neighbor rule. J. Process Control 64, 49–61.
Zhang, Qi, Li, Peng, Lang, Xun, Miao, Aimin, 2020. Improved dynamic kernel principal component analysis for fault detection. Measurement 158, 107738.
Zhao, Haitao, 2018. Dynamic graph embedding for fault detection. Comput. Chem. Eng. 117, 359–371.
Zhao, Haitao, Yuen, Pong Chi, Kwok, James T, 2006. A novel incremental principal component analysis and its application for face recognition. IEEE Trans. Syst., Man, Cybernet., Part B (Cybernetics) 36 (4), 873–886.
Zheng, Shaodong, Zhao, Jinsong, 2020. A new unsupervised data mining method based on the stacked autoencoder for chemical process fault diagnosis. Comput. Chem. Eng. 135, 106755.
Zhu, Jiazhen, Shi, Hongbo, Song, Bing, Tan, Shuai, Tao, Yang, 2020. Deep neural network based recursive feature learning for nonlinear dynamic process monitoring. Can. J. Chem. Eng. 98 (4), 919–933.