Process Safety and Environmental Protection 150 (2021) 93–109

Contents lists available at ScienceDirect

Process Safety and Environmental Protection

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

Process topology convolutional network model for chemical process fault diagnosis

Deyang Wu[a], Jinsong Zhao[a,b,*]

[a] State Key Laboratory of Chemical Engineering, Department of Chemical Engineering, Tsinghua University, Beijing, China
[b] Beijing Key Laboratory of Industrial Big Data System and Application, Tsinghua University, Beijing, China

ARTICLE INFO

Article history:
Received 8 January 2021
Received in revised form 29 March 2021
Accepted 30 March 2021
Available online 8 April 2021

Keywords:
Fault diagnosis
Chemical process
Process topology convolutional network
Explainable deep learning
Process safety

ABSTRACT

There always exists potential safety risk in chemical processes. Abnormalities or faults of the processes can lead to severe accidents with unexpected loss of life and property. Early and accurate fault detection and diagnosis (FDD) is essential to prevent these accidents. Many data-driven FDD models have been developed to identify process faults. However, most of the models are black-box models with poor explainability. In this paper, a process topology convolutional network (PTCN) model is proposed for fault diagnosis of complex chemical processes. Experiments on the benchmark Tennessee Eastman process showed that PTCN improved the fault diagnosis accuracy with simpler network structure and less reliance on the amount of training data and computation resources. In the meantime, the model building process becomes much more rational and the model itself is much more understandable.

1. Introduction

Modern chemical processes are generally complex and integrated systems with many different unit operations. Under distributed control systems (DCS), chemical processes can be operated steadily in most of the time. But there always exists potential risk that abnormal events or process faults may happen, which is beyond the control capability of DCS and requires the intervention of operators. They can lead to severe accidents with unexpected loss of life and property if no actions are taken to bring the processes back to normal operation.

Abnormal situation management (ASM) of chemical processes is critical to handle these abnormal events or faults and prevent processes from further accidents. (Venkatasubramanian et al., 2003) defined ASM as a key component of supervisory control. ASM involves the timely detection of an abnormal event, diagnosing causal origins and then taking appropriate decisions and actions to bring the process back to a normal, safe, operating state. (Arunthavanathan et al., 2020a) proposed a framework from safety perspectives to analyze the interconnections between fault detection and diagnosis (FDD), risk assessment (RA) and ASM. In this framework, FDD is the initial step to identify the possible hazard. RA uses failure models, accident models and risk models to provide the basis of decision making for ASM. With the feedback information, ASM changes the decision to control the operation and brings the processes back to normal state. (Khan et al., 2015) mentioned that integration of dynamic fault detection and diagnosis with risk assessment had significantly improved safety in process facilities. According to (Dai et al., 2016), process monitoring including fault detection and diagnosis should be effectively integrated to achieve a reliable, robust, scalable ASM platform for smart chemical process operations. As can be seen, FDD is highly valued in ASM and the field of process safety.

Although DCS makes chemical processes highly automatic nowadays, the automation of ASM has not been realized yet (Shu et al., 2016). It heavily relies on human operators' monitoring. Additionally, real-time FDD is getting more and more difficult for the increasingly complex chemical processes (Venkatasubramanian et al., 2003). As the chemical industries evolve towards smart manufacturing, it's necessary to build intelligent FDD system for safer and more automatic process operations.

From the perspectives of process safety and risk assessment, many researchers have paid attention to the applications of FDD, especially in the field of dynamic risk assessment these years. (Zadakbar et al., 2013) extended principle component analysis (PCA) based FDD to a risk based FDD framework targeting the safety issues of a process system. The combination of PCA and a quantitative operational risk assessment model made this method robust to false alarms. (Madakyaru et al., 2017) proposed a statistical

* Corresponding author at: Department of Chemical Engineering, Tsinghua University, Beijing, China.
E-mail address: jinsongzhao@tsinghua.edu.cn (J. Zhao).

https://doi.org/10.1016/j.psep.2021.03.052
0957-5820/© 2021 Institution of Chemical Engineers. Published by Elsevier B.V. All rights reserved.

approach that exploited the advantages of multiscale partial least squares (MSPLS) models and generalized likelihood ratio (GLR) tests for fault detection in processes. (Amin et al., 2020) proposed a novel methodology for dynamic risk analysis, integrating the multivariate data-based process monitoring and logical dynamic failure prediction model. This framework combined the naïve Bayes classifier, Bayesian network, and event tree analysis. It provided robust performance on the application of a binary distillation column and the RT 580 experimental setup in four fault scenarios.

From the perspective of modeling, many FDD models have been proposed by researchers over the past few decades. (Venkatasubramanian et al., 2003) classifies FDD models into three categories: quantitative model-based, qualitative model-based and process history based. Process history-based models are further classified into qualitative and quantitative models. The latter is commonly termed as data-driven models. Easy access to big data and the rapid development of computing power have made data-driven models get excellent performance in terms of accuracy. Additionally, data-driven models can be easily constructed without first principles and expert knowledge.

Basic data-driven FDD methods are mainly statistical methods, such as principle component analysis (PCA), independent component analysis (ICA), fisher discriminant analysis (FDA), partial least squares (PLS), canonical variate analysis (CVA), subspace aided approach (SAP), *etc.* and their variants. There're some machine learning FDD models that treat fault diagnosis as a classification problem, such as support vector machine (SVM), artificial neural network (ANN), artificial immune system (AIS) (Dai and Zhao, 2011), *etc*. In order to compare different models' performance, the Tennessee Eastman (TE) process (Downs and Vogel, 1993) was gradually exploited by researchers as a benchmark. A comparison study on basic data-driven fault diagnosis methods has been conducted by (Yin et al., 2012).

With the recent development of artificial intelligence and big data technologies, deep learning-based FDD has attracted a lot of interests from researchers. Deep learning is a general term for a large class of algorithms utilizing multi-layer neural network models. Alternate linear transformations and nonlinear activation functions give deep learning models strong ability of nonlinear fitting. (Xie and Bai, 2015) proposed a hierarchical deep neural network (HDNN) based on deep belief network (DBN). HDNN reached an average fault classification accuracy of 80.5 % excluding faults 3, 9, 15 on the TE process. (Lv et al., 2016) used sparse auto encoder (SAE) and support vector machine (SVM) for fault detection and classification. (Zhang and Zhao, 2017) proposed a fault diagnosis model based on DBN. It utilized DBN sub-networks for features extraction and a global two-layer back-propagation network for fault classification. The model reached an average fault classification accuracy of 82.1 % for all the 20 fault classes of the TE process. (Wu and Zhao, 2018) treated two-dimension data matrices as images and proposed a deep convolutional neural network (DCNN) model. DCNN extracted features in spatial and temporal domains simultaneously. It reached an average fault classification accuracy of 88.2 % for all of the 20 fault classes. (Zhang et al., 2020a) utilized bidirectional recurrent neural network (BiRNN) to process time series from both positive and negative directions. The model reached an average fault classification accuracy of 92.7 % considering all the 20 fault classes. These're all supervised deep learning-based approaches. For unsupervised deep learning, (Cheng et al., 2019) proposed a process monitoring method based on variational recurrent autoencoder with a new monitoring metric named negative variational score, proving that unsupervised deep learning methods can be adapted for complex process monitoring. (Zheng and Zhao, 2020) proposed an unsupervised data mining method that can make use of unlabeled historical data to deal with the problem of lacking labeled data in real situations.

Despite of the advantages mentioned above, data-driven fault diagnosis models still have room for improvement. Firstly, these models are mainly based on pure data and utilize no process knowledge. Without extra information from the process, pure data-driven models tend to have higher degree of freedom thus more model parameters. Overly complex models generally suffer from the problem of overfitting (Lever et al., 2016). The model building process is not rational without guidance from process knowledge. Secondly, these pure data-driven models lack explainability and are still black boxes to humans, which limits their application in real industrial processes. Thirdly, the training process and online application of these models are prone to consume more resources on computation.

As techniques in machine learning develop quickly these years, there is a consensus among many researchers that innovation is needed to integrate data analytics tools with fundamental knowledge to create robust and scalable solutions for industrial processes (Qin and Chiang, 2019). The current status of machine learning is more like alchemy, a collection of *ad hoc* methods. But this limitation could be offset by the use of first-principles knowledge, which can impose some rigor and discipline on purely data-driven models (Venkatasubramanian, 2019). (Bikmukhametov and Jäschke, 2020) proposed 5 methods to enhance accuracy and explainability of data-driven models through combining machine learning and process engineering physics. However, these methods mainly made use of feature engineering or the mismatch between the developed models and first-principle models. Process knowledge was not utilized in the building phase of machine learning models.

There're some hybrid approaches for FDD combining both model-based approaches and data-driven approaches. Among them, the Bayesian network (BN) has been used widely for FDD and root cause analysis of process systems in the past few decades. (Yu et al., 2015) proposed a dynamic operational risk management framework integrating modified independent component analysis and a BN model. (Gharahbagheri et al., 2017) proposed a methodology for operators to diagnose the root cause of abnormal. It was based on kernel principal component analysis, and process knowledge was combined through Bayesian belief network. (Lou et al., 2020) utilized a condition Gaussian network (CGN), a special form of BN, with an adaptive threshold scheme to detect and diagnose the process faults. For cognitive modeling, (Chen et al., 2014) employed deterministic reservoir models to fit the multiple-input and multiple-output signals in the TE process, which firstly investigated the TE process in a cognitive way. (Arunthavanathan et al., 2020b) focused on the integration of unsupervised learning and cognitive modeling to detect and diagnose unknown fault conditions. However, hybrid approaches generally take more effort in the phrase of modeling and can't be trained end-to-end easily.

The graph neural network (GNN) is a type of deep learning model that can process data in the graph domain or non-Euclidean space, which has drawn much attention of researchers recently. In mathematics, graph is a kind of data structure consisting of nodes and edges. These nodes are connected with edges to describe their relationships. Differences of data in Euclidean space and graph domain are showed in Fig. 1. GNN has a wide range of applications because data in the graph domain are far more common than structured data in the real world. Among GNNs, graph convolutional network (GCN) (Kipf and Welling, 2017) has been studied widely in the field of deep learning. GCN defines graph convolutions by information propagation. It requires input of a general data matrix with stacked node features and an extra adjacent matrix that describes nodes' connectivity in the graph. Within a graph convolutional layer in GCN, a node in the graph will aggregate feature information from its neighborhood nodes (Wu et al., 2020). Stacked graph convolutional layers enable GCN to propagate information of nodes along the edges in the graph as predefined. This helps GCN learn about

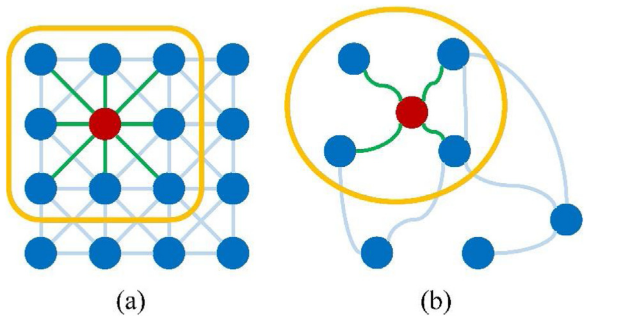

Fig. 1. Data and convolution operations in (a) Euclidean space and (b) graph domain.

the relationships among different nodes under clear guidance. The predefined graph is an entry where we can integrate extra knowledge or information from the processes. However, to the authors' knowledge, GCN's applications in chemical processes have been rarely reported.

The major contributions of this work are summarized as followed:

(1) A fault diagnosis model named process topology convolutional network (PTCN) is proposed based on GCN. Detailed steps are given to describe how to construct graphs from chemical processes that are required by PTCN.
(2) The framework of fault diagnosis method based on PTCN is designed for practical application. Case study on the TE process showed that PTCN improved fault diagnosis performance and reduced the reliance on model parameters, training data and computation resources.

The rest of the paper is organized as followed: Section 2 introduces the brief history and the basic theory of graph convolutional network. Section 3 introduces graph construction steps, PTCN model structure, data structure and the fault diagnosis method in detail. Section 4 is a case study on the TE process that introduces the dataset preparation and experiments result of the PTCN model. Finally, conclusions are drawn in Section 5.

2. Graph convolutional network

In recent years, convolutional neural networks (CNNs) have received considerable attention from researchers in the field of machine learning. CNNs have performed much better than many traditional algorithms in many tasks, such as image classification (Krizhevsky et al., 2017) and object detection (Redmon et al., 2016). However, CNNs are good at processing the data in the Euclidean space, such as images, but can hardly fulfil the tasks where the data are represented in the non-Euclidean space, such as graphs or networks (Wu et al., 2020). Fig. 1 shows the differences between the data and convolutional operations represented in the Euclidean space and graph domain.

In practical application, the semi-structured data represented as graph or network are more common. For instance, the relationships among the users in some social media platforms can be represented as a graph. In chemical researches, a molecular structural formula can also be regarded as a graph, where the nodes represent the atoms and the edges represent the chemical bonds. To process data in graph domain, graph neural networks (GNNs) were invented (Wu et al., 2020), which have attracted the interests of many researchers in the field of deep learning.

Table 1
Commonly used notations.

Notations	Descriptions
G	A graph
V	The set of nodes in a graph
v	A node $v \in V$
E	The set of edges in a graph
e_{ij}	An edge $e_{ij} \in E$
$\mathcal{N}(v)$	The neighborhoods of node v
n	The number of nodes in a graph
m	The number of edges in a graph
d	The dimension of a node feature vector
b	The dimension of a hidden state vector
c	The dimension of an edge feature vector
$\boldsymbol{A} \in \mathbb{R}^{n\times n}$	The graph adjacency matrix
$\boldsymbol{A}^T$	The transpose of the matrix $\boldsymbol{A}$
$\boldsymbol{A}_i$	The i^{th} row of the matrix $\boldsymbol{A}$
A_{ij}	The element in the i^{th} row and j^{th} column of the matrix $\boldsymbol{A}$
$\boldsymbol{D} \in \mathbb{R}^{n\times n}$	The degree matrix of $\boldsymbol{A}$. $D_{ii} = \sum_{j=1}^{n} A_{ij}$
$\boldsymbol{X} \in \mathbb{R}^{n\times d}$	The node feature matrix
$\boldsymbol{X}_t \in \mathbb{R}^{n\times d}$	The node feature matrix at time step t
$\boldsymbol{x}_v \in \mathbb{R}^{d}$	The node feature vector of node v
$\boldsymbol{x}_{(v,u)} \in \mathbb{R}^{c}$	The edge feature vector of edge (v, u)
$x_i \in \mathbb{R}$	The i^{th} element of vector $\boldsymbol{x}$
$\boldsymbol{H}^k \in \mathbb{R}^{n\times b}$	The hidden state matrix in the k^{th} graph convolutional layer
$\boldsymbol{h}_v \in \mathbb{R}^{b}$	The hidden state vector of node v

2.1. *The definition and notations of graph*

A graph is a set of vertices (or nodes) and edges, and is applied as a special data structure in computing sciences that can represent objects and the connections among them. In this paper, the commonly used notations are illustrated in Table 1 unless otherwise specified. And the formal definitions of graphs and directed graphs are in accordance with (Wu et al., 2020).

2.1.1. *Definition of* graph

A graph is represented as $G = (V,\ E)$ where V is the set of vertices or nodes (we will use nodes throughout the paper) and E is the set of edges. Let $v_i \in V$ to denote a node and $e_{ij} = \left(v_i,\ v_j\right) \in E$ to denote an edge pointing from v_j to v_i. The neighborhood of a node v is defined as $\mathcal{N}(v) = \left\{u \in V \mid (v,\ u) \in E\right\}$. The adjacency matrix $\boldsymbol{A}$ is a $n \times n$ matrix with $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ if $e_{ij} \notin E$. A graph may have node attributes $\boldsymbol{X}$, where $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is a node feature matrix with $\boldsymbol{x}_v \in \mathbb{R}^d$ representing the feature vector of a node v. Meanwhile, a graph may have edge attributes $\boldsymbol{X}^e$, where $\boldsymbol{X}^e \in \mathbb{R}^{m\times c}$ is an edge feature matrix with $\boldsymbol{x}_{(v,u)} \in \mathbb{R}^c$ representing the feature vector of an edge (v, u).

2.1.2. *Definition of directed graph*

A directed graph is a graph with all edges directed from one node to another. An undirected graph is considered as a special case of directed graphs where there is a pair of edges with inverse directions if two nodes are connected. A graph is undirected if and only if the adjacency matrix is symmetric.

2.2. *A brief history of graph neural network*

Early study on neural networks applied to graphs can date back to (Sperduti and Starita, 1997). After first proposed by (Gori et al., 2005), *graph neural network (GNN)*[1] was further developed by (Scarselli et al., 2009), which was extended from neural net-

[1] To avoid confusion, graph neural networks (GNNs) represent general neural network models that can process data in graph domains, while italic *graph neural network (GNN)* stands for the specific graph neural network model proposed by (Gori et al., 2005) and further developed by (Scarselli et al., 2009).

works for processing the data represented in graph domains. GNNs for supervised learning can be mainly categorized into recursive graph neural networks (RecGNNs) and convolutional graph neural networks (ConvGNNs).

2.2.1. *Recursive graph neural network*

Many graph neural network models proposed in early time were extended from recursive neural network (RNN). A typical case, *GNN* (Scarselli et al., 2009), is an extension of RNN with random walk model, and is based on an information diffusion mechanism. In *GNN*, the nodes of a graph will exchange information through the connections between them and update their hidden states until a stable equilibrium is reached, and a unique stable equilibrium can always be guaranteed by Banach's fixed point theorem (Khamsi and Kirk, 2001). The hidden state corresponding to a node v can be updated by

$$\boldsymbol{h}_v^t = \sum_{u \in N(v)} f_{\boldsymbol{w}}(\boldsymbol{x}_v, \boldsymbol{x}_{(v,u)}, \boldsymbol{h}_u^{t-1}, \boldsymbol{x}_u) \tag{1}$$

Where $\boldsymbol{h}_v^t$ is the hidden state of the node v in the t^{th} iteration. The hidden state of a node can be regarded as a high-level node representation that combines the information of neighbor nodes and the graph's topological structure. The information diffusion mechanism in RecGNNs has made profound effect on the development of ConvGNNs.

2.2.2. *Convolutional graph neural network*

The convolutional graph neural network (ConvGNN) is another main category of graph neural network, and draws far more attention from researchers than RecGNNs. Convolutional graph neural networks introduce convolutional operation from Euclidean space to data in graph domains, which can hardly be processed by CNNs directly. Based on the domain that convolutional operations are performed on, ConvGNNs can be categorized further into spectral-based ConvGNNs and spatial-based ConvGNNs.

2.2.3. *Spectral-based ConvGNN*

Spectral-based ConvGNNs utilize the knowledge from graph signal processing. For a graph, the normalized Laplacian matrix $\boldsymbol{L}$ is

$$\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{D}^{\left(-\frac{1}{2}\right)}\boldsymbol{A}\boldsymbol{D}^{\left(-\frac{1}{2}\right)} \tag{2}$$

$$\boldsymbol{D} = \sum_{\boldsymbol{j}=1}^{\boldsymbol{n}} A_{ij} \tag{3}$$

And $\boldsymbol{D}$ is called degree matrix of the adjacency matrix $\boldsymbol{A}$. $\boldsymbol{I}_n$ is a n-order identity matrix. When we perform eigenvalue decomposition to the normalized Laplacian matrix $\boldsymbol{L}$

$$\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T \tag{4}$$

We can get an orthonormal matrix $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n] \in \mathbb{R}^{n\times n}$ where $\boldsymbol{u}_i$ is the eigenvector arranged by the value of eigenvalues, and $\boldsymbol{\Lambda}$ stands for the diagonal matrix where $\Lambda_{ii} = \lambda_i$. If we have a graph signal $\boldsymbol{x} = [x_1, x_2, \ldots x_n]^T \in \mathbb{R}^n$ where x_i is the signal value of node i, the graph Fourier transformation and its inverse transformation can be defined as

$$\boldsymbol{x}' = \mathcal{F}(\boldsymbol{x}) = \boldsymbol{U}^T\boldsymbol{x} \tag{5}$$

$$\boldsymbol{x} = \mathcal{F}^{-1}(\boldsymbol{x}') = \boldsymbol{U}\boldsymbol{x}' \tag{6}$$

This can be derived from $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}$ since $\boldsymbol{U}$ is an orthonormal matrix. And a graph convolution of graph signal $\boldsymbol{x}$ with filter $\boldsymbol{g}$ can be defined as (Henaff et al., 2015)

$$x *_G \boldsymbol{g} = \boldsymbol{U}(\boldsymbol{U}^T\boldsymbol{x} \odot \boldsymbol{U}^T\boldsymbol{g}) \tag{7}$$

$*_G$ stands for the convolutional operator performed on graphs and $\odot$ means element-wise product.

Since $\boldsymbol{U}^T\boldsymbol{x}$, $\boldsymbol{U}^T\boldsymbol{g} \in \mathbb{R}^n$, the spectral-based graph convolutional operation can be simplified as

$$x *_G \boldsymbol{g_\theta} = \boldsymbol{U}(\boldsymbol{g_\theta}\boldsymbol{U}^T\boldsymbol{x}) \tag{8}$$

Where $\boldsymbol{g_\theta} = diag\left(\boldsymbol{U}^T\boldsymbol{g}\right)$ is a parametric filter. And nearly all the spectral-based ConvGNNs identify with this formulation (Wu et al., 2020).

The efficiency of spectral-based ConvGNNs is affected by the eigenvalue decomposition of the normalized Laplacian matrix $\boldsymbol{L}$. (Defferrard et al., 2016) proposed Chebyshev Spectral CNN (ChebNet) to solve the efficiency problem through approximating the filter $\boldsymbol{g_\theta}$ by Chebyshev polynomials of $\boldsymbol{\Lambda}$. And (Kipf and Welling, 2017) utilized a first-order approximation of ChebNet to propose Graph Convolutional Network (GCN). For a graph G of n nodes with d-dimensional features, the graph convolutional operation of GCN on a graph signal $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is

$$\boldsymbol{Z} = \tilde{\boldsymbol{D}}^{\left(-\frac{1}{2}\right)}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{\left(-\frac{1}{2}\right)}\boldsymbol{X}\boldsymbol{\Theta} \tag{9}$$

$$\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n \tag{10}$$

$$\tilde{D}_{ii} = \sum_{\boldsymbol{j}=1}^{\boldsymbol{n}} \tilde{A}_{ij} \tag{11}$$

$\tilde{\boldsymbol{A}}$ is the adjacent matrix of graph G added with self-loops, and $\tilde{\boldsymbol{D}}$ is the degree matrix of $\tilde{\boldsymbol{A}}$. $\boldsymbol{\Theta} \in \mathbb{R}^{d\times b}$ is a parametric matrix that embeds d-dimensional node features to a b-dimensional space.

2.2.4. *Spatial-based ConvGNN*

Different from spectral-based ConvGNNs, spatial-based ConvGNNs inherit more from the information diffusion mechanism of RecGNNs in early time. Spatial-based ConvGNNs propagate information through the connections, and aggregates the information of neighbor nodes and corresponding edges to a certain node to update the hidden state of it. However, spatial-based ConvGNNs do not require a unique stable equilibrium of the graph under strict constraint conditions. They define convolutional layers for information passing and aggregation operations, then use graph readout technologies to obtain node-level or graph-level representation of the information. From Fig. 2 we can see that the spatial-based convolutional operation on a directed graph updates a node's hidden state with its neighbor nodes' representations. This is similar to convolutional operation on images in CNNs if we consider the image as a special type of graph with nodes fixed in the grid. So spatial-based ConvGNNs can be regarded as a generalization of traditional CNNs.

(Gilmer et al., 2017) proposed message passing neural network (MPNN) as a framework for spatial-based ConvGNNs. It describes the process of message passing and aggregation on a graph. In MPNN, the spatial-based convolutional operation on graphs can be formulated as

$$\boldsymbol{m}_v^{t+1} = \sum_{u \in N(v)} M_t(\boldsymbol{h}_v^t, \boldsymbol{h}_u^t, \boldsymbol{x}_{(v,u)}) \tag{12}$$

$$\boldsymbol{h}_v^{t+1} = U_t(\boldsymbol{h}_v^t, \boldsymbol{m}_v^{t+1}) \tag{13}$$

The function M_t is called message functions for passing the hidden states of neighbor nodes and the features of connected edges to node v. U_t is a node update function for updating the hidden state of node v, and t means the t^{th} iteration or the t^{th} graph convolutional operation. After T convolutional operations, readout function

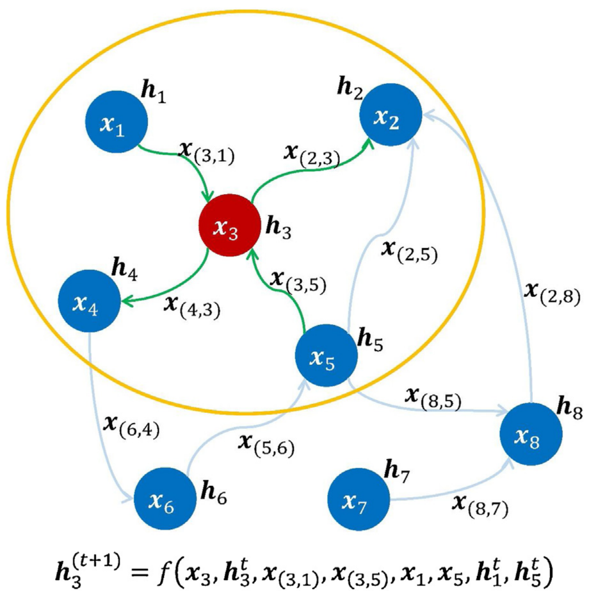

Fig. 2. Spatial-based convolutional operation on a directed graph.

$R(\{\boldsymbol{h}_v^T | v \in G\})$ can generate the representation of the graph from all the nodes, or generate node-level representations.

If we look into the graph convolutional operation of GCN we can find that it's in accordance with the MPNN framework. The equation Eq. (9) can be expressed as

$$\boldsymbol{Z}_i = \sum_j \frac{1}{\sqrt{\tilde{D}_{ii}} \cdot \sqrt{\tilde{D}_{jj}}} \tilde{A}_{ij} \boldsymbol{X}_j \boldsymbol{\Theta} = \sum_{j \in \{i\} \cup N(i)} \frac{1}{\sqrt{\tilde{D}_{ii}} \cdot \sqrt{\tilde{D}_{jj}}} \boldsymbol{X}_j \boldsymbol{\Theta} \tag{14}$$

$$\boldsymbol{h}_i = U(\boldsymbol{Z}_i) = U(M_{\boldsymbol{\Theta}}(\frac{1}{\sqrt{\tilde{D}_{ii}} \cdot \sqrt{\tilde{D}_{ii}}} \boldsymbol{X}_i, \sum_{j \in \mathcal{N}(i)} \frac{1}{\sqrt{\tilde{D}_{ii}} \cdot \sqrt{\tilde{D}_{jj}}} \boldsymbol{X}_j)) \tag{15}$$

The graph convolutional operation of GCN has two phases of message passing and aggregation like general spatial-based ConvGNNs. From this perspective, GCN can be regarded as a connection between spectral-based and spatial-based ConvGNNs, when the spectral filter is approximated to first-order (Zhang et al., 2020b). The spectral-based and spatial-based graph convolutions are equivalent under specific conditions, but spatial-based ones can benefit from the research progresses of traditional CNNs. This maybe the reason why there're more researches studying spatial-based ConvGNNs than spectral-based ConvGNNs.

3. PTCN based fault diagnosis method

3.1. Graph and chemical processes topology

In mathematics, topology describes the way how nodes and edges are arranged within a network. In this paper, chemical process topology is used to describe the way how different parts of the process are arranged. In chemical processes, unit operations and streams are physically connected with pipes. Control loops link up the process variables and manipulated variables with sensors, controllers, actuators, *etc*. Process topology contains a large amount of knowledge about the relationships between different variables, since they are physical connected with pipes or signal transmission wires. To utilize the information of process topology, the process should be transformed to a graph first.

To construct a graph from a chemical process, the P&ID is required to figure out different unit operations, streams and corresponding measurements. Control loops should also be studied to confirm process variables, manipulated variables, sensors, controllers, actuators, *etc*. Detailed steps are listed below to describe how to construct graphs from chemical processes. For better understanding, a subgraph of the graph constructed from Tennessee Eastman process is showed in Fig. 3 as an example. And the complete graph is showed as Fig. 7 in Section 4.2.

Step 1: Figure out all the unit operations and streams that we concern about. Create nodes for every unit operation and stream. These nodes are called unit operation nodes and stream nodes respectively.

Step 2: Figure out all the corresponding measurements of these unit operations and streams in Step 1. Create nodes for every measurement. These nodes are called measurement nodes.

Step 3: Create directed edges pointing from every unit operation node or stream node to corresponding measurement node.

Step 4: For every stream node, create a directed edge pointing from the upstream unit operation node to the stream node, and a directed edge pointing from the stream node to the downstream unit operation node.

Step 5: Figure out all the process variables and manipulated variables of concerned control loops. Process variables are included in the measurement nodes in Step 2. Create nodes for every manipulated variable. These nodes are called manipulated variable node. For every control loop, create an abstract node called controller node.

Step 6: In a control loop, create a directed edge pointing from the corresponding measurement node to the controller node, and a directed edge pointing from the controller node to the corresponding manipulated variable node.

Step 7: For simplicity of the graph, delete the stream nodes that have no corresponding measurements, and then connect the upstream and downstream unit operation nodes with a directed edge directly.

The graph describes the physical connectivity among the streams, unit operations and control loops. From the perspective of information transferring, different nodes generate or carry information about the process. The directed edges describe the directions of information transferring or massage passing. Compared to complicated first-principle mathematical models, the graph only concerns about the connections between different parts of the process, which can qualitatively indicate the relationships among different variables.

Following the steps above, the graph constructed from the process will be cyclic graph considering recycle streams and control loops in the process. This is reminiscent of other fault diagnosis methods based on graphs such as the Bayesian network. But different from PTCN, the Bayesian network is based on acyclic graphs. The graph in PTCN is not used for inferring the probability of the presence of a variable, so it can be a cyclic graph. In another word, PTCN is not a probabilistic graphical model.

The graph in PTCN is mainly used to describe the physical relationships between different variables. It doesn't intend to model the exact dependency between variables. Though the physical connections can't accurately model the dependency between variables, the dependency must be based on physical links. The exact relationships between variables are decided by learnable network parameters, which are learnt from data following the paradigm of deep learning. Thus, the graph in PTCN works as loose constraints for the relationships between variables.

3.2. Data structures and data preprocessing

Generally, in data-driven fault diagnosis methods, a data sample in the training dataset is consist of *n* observation variables

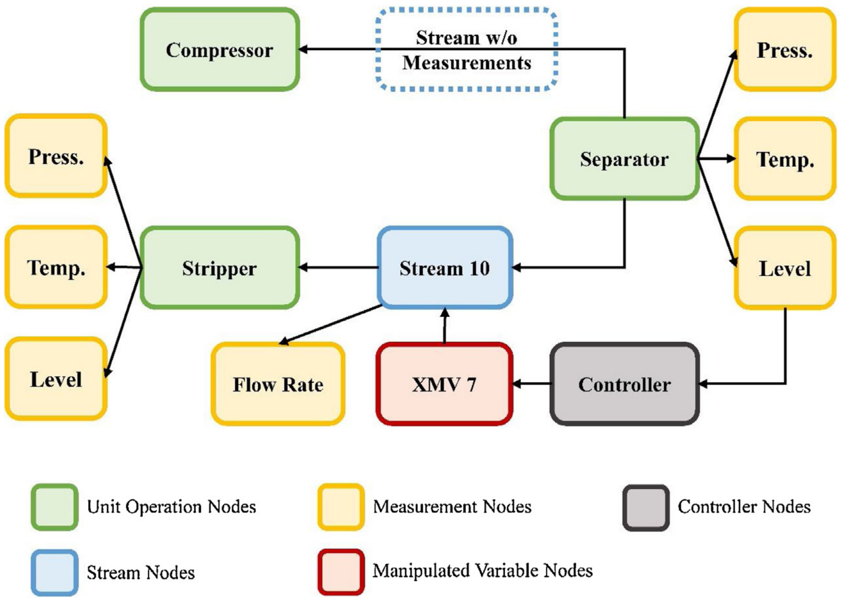

Fig. 3. Subgraph of the graph constructed from Tennessee Eastman process.

and w observations for each variable, which forms a data matrix $\boldsymbol{X}_t \in \mathbb{R}^{n\times w}$ at time step t like Eq. (16).

$$\boldsymbol{X}_t = \begin{bmatrix} x_{1,t} & x_{1,(t-1)} & \cdots & x_{1,(t-w+1)} \\ x_{2,t} & x_{2,(t-1)} & \cdots & x_{2,(t-w+1)} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,t} & x_{n,(t-1)} & \cdots & x_{n,(t-w+1)} \end{bmatrix} \tag{16}$$

Every column of the matrix is a n -dimensional observation vector at a certain time. Every row is a w -dimensional time serial of a certain observation variable. While columns in $\boldsymbol{X}_t$ should be arranged in chronological order to exploit serial correlation, the rows do not have to follow a certain sequence. In another word, all the observation variables are equally treated in $\boldsymbol{X}_t$ because no prior knowledge or constraint conditions are given. However, all the observation variables are collected from a certain chemical process, so they are highly correlated under the constraints of material balance, heat balance, energy balance and decentralized control system. And the process knowledge can hardly be involved in a single $\boldsymbol{X}_t$. Thus, the data-driven model may suffer from overfitting of the data without enough prior knowledge.

To integrate process knowledge about the correlation among the observation variables, the data matrix $\boldsymbol{X}_t$ is input with an adjacency matrix $\boldsymbol{A}'$ of the graph G constructed from the concerned chemical process topology. According to *Definition 1* in Section 2.1, the adjacency matrix $\boldsymbol{A}'$ can be calculated as Eq. (17).

$$A'_{ij} = \begin{cases} 1, \text{ if } e_{ij} \in E \\ 0, \text{ if } e_{ij} \notin E \end{cases} \tag{17}$$

E is the set of edges in the graph G.

It needs to be pointed out carefully that the best directions for message passing are not necessary the same as the physical transferring directions of flows and control signals. They can also be the reverse directions. Experiments in Section 4.6 indicates that the reverse directions are more appropriate for message passing. Thus, the actual adjacency matrix input with data matrix $\boldsymbol{X}_t$ is, the adjacency matrix of the reverse graph of G. And in mathematics, $\boldsymbol{A}$ can be easily deviated from $\boldsymbol{A}'$ as Eq. (18).

$$\boldsymbol{A} = \left(\boldsymbol{A}'\right)^T \tag{18}$$

After the graph is prepared, the historical data of the process should be collected for further training and testing a fault diagnosis model. Historical data of measurements and manipulated variables form time series and are sampled at a certain frequency. The number of measurements and manipulated variables is m. For every variable, all the original historical data $\boldsymbol{X}'_i$ should first be normalized with the sample mean $\bar{x}_i$ and sample standard deviation s_i calculated using data in normal state as Eq. (19), which is also known as standard score normalization.

$$\boldsymbol{X}_i = \frac{\boldsymbol{X}'_i - \bar{x}_i}{s_i} \ (i = 1, 2, \ldots m) \tag{19}$$

The normalized data $\boldsymbol{X}_i$ are then sliced with a time window. At a certain time t, the data slice for every variable will be a vector $\boldsymbol{X}_{i,t} \in \mathbb{R}^{1\times w}$ of w observations including current sample and $w-1$ samples before t.

$$\boldsymbol{X}_{i,t} = \begin{bmatrix} x_{i,t} & x_{i,(t-1)} & \cdots & x_{i,(t-w+1)} \end{bmatrix} \ (i = 1, 2, \ldots m) \tag{20}$$

It should be noted that the n nodes in the graph include not only measurements and manipulated variables that are observed in the process, but also some abstract nodes such as stream nodes, unit operation nodes and controller nodes. These abstract nodes can't be observed directly so have no corresponding historical data. For computational requirements, the time series or node features of these abstract nodes should be initialized properly. The appropriate initialization technologies will be further discussed in Section 4.5. Then all the time series $\boldsymbol{X}_{i,t}\ (i = 1, 2, \ldots n)$ are stacked into the data matrix $\boldsymbol{X}_t \in \mathbb{R}^{n\times w}$. The data matrix $\boldsymbol{X}_t$ can also be regarded as the node feature matrix, and every row is a node feature vector, which is also a time serial of w observations.

Labels are essential in supervised learning. Since the fault diagnosis is treated as a multi-classification problem, it has to be figured out when a fault happened in the historical data and what the fault type is. If there exists c types of faults, every data matrix $\boldsymbol{X}_t$ will be assigned with a corresponding label $\boldsymbol{y} \in \mathbb{Z}$ varying from 0 to c that indicates the fault type of the process at time t. And $\boldsymbol{y} = 0$ means the process is in normal state, while other integers stand for different types of faults. As a result, a data sample input to the model will be $\boldsymbol{X}_t, \boldsymbol{A}, \boldsymbol{y}$. After all the historical data are sliced and arranged into data samples, the whole dataset $\left\{\boldsymbol{X}_t, \boldsymbol{A}, \boldsymbol{y}\right\}_{i=1}^{N}$ will be divided into training dataset and testing dataset with the ratio of 4:1 for further training and testing process.

3.3. PTCN model for fault diagnosis

The proposed model for fault diagnosis is named as process topology convolutional network (PTCN) in this paper, and the structure is showed as Fig. 4. The model name PTCN comes from that the graph convolutional operations are applied on the graph constructed from the chemical process topology. PTCN is mainly composed of stacked graph convolutional layers and a multi-layer perceptron classifier. A graph convolutional layer will include a complete graph convolutional operation as Eq. (9).

Within a graph convolutional layer, every node in the graph will aggregate feature information from its neighborhood nodes. When graph convolutions are applied to the chemical process topology, information will propagate among different parts of the process along the physical connections. This is how the knowledge of the chemical process topology can be utilized in the building of the model. Compared to pure data-driven models, the feature extraction of PTCN is more understandable and reasonable to humans, because the graph convolutions are based on the physics of the chemical process. Thus, PTCN is more of explainability with the guidance of the chemical process topology.

Within a training epoch, dataset $\left\{\boldsymbol{X}_t, \boldsymbol{A}, \boldsymbol{y}\right\}^N$ is input to the network model. For every data sample, self-loops are first added to $\boldsymbol{A}$ to get $\tilde{\boldsymbol{A}}$ as Eq. (21) and corresponding degree matrix $\tilde{\boldsymbol{D}}$ is calculated as Eq. (22).

$$\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n \tag{21}$$

$$\tilde{D}_{ij} = \begin{cases} \sum_{k=1}^{n} \tilde{A}_{kj}, & if\ i = j \\ 0, & else \end{cases} \tag{22}$$

It's worth mentioning that $\tilde{\boldsymbol{A}}$ is not symmetric since G is a directed graph. The degree matrix $\tilde{\boldsymbol{D}}$ is calculated as an out-degree matrix, which counts how many nodes a node j is pointing to after self-loops are added.

The hidden states of nodes in the graph are initialized with the node feature matrix $\boldsymbol{X}_t$.

$$\boldsymbol{H}^0 = \boldsymbol{X}_t \tag{23}$$

Every graph convolutional layer applies the graph convolutional operation to the hidden state matrix, and the embedding dimension of the hidden states in the kth layer is decided by the parameter matrix $\boldsymbol{\Theta}^k \in \mathbb{R}^{d \times b}$. The embedding dimension should be determined according to the scale of the problem. What's more, a nonlinear activation function Rectified Linear Unit (ReLU) (Glorot et al., 2011) is applied after every graph convolutional layer to give the network ability of nonlinear fitting:

$$\boldsymbol{Z}^k = \tilde{\boldsymbol{D}}^{\left(-\frac{1}{2}\right)} \boldsymbol{A} \tilde{\boldsymbol{D}}^{\left(-\frac{1}{2}\right)} \boldsymbol{H}^{k-1} \boldsymbol{\Theta}^k \, (k = 1, 2, \ldots, K) \tag{24}$$

$$\boldsymbol{H}^k = ReLU\left(\boldsymbol{Z}^k\right) \tag{25}$$

After K graph convolutional layers, the final high-level representations of node features are extracted as $\boldsymbol{H}^K$. Then a multi-layer perceptron classifier is applied for the task of fault classification, which can be regarded as stacked fully connected (FC) neural layers. $\boldsymbol{H}^K$ is firstly flatten to a vector $\boldsymbol{a}^0$ and then input to the classifier:

$$\boldsymbol{a}^0 = flat\left(\boldsymbol{H}^K\right) = \left[\boldsymbol{H}_1^K, \boldsymbol{H}_2^K, \ldots, \boldsymbol{H}_n^K\right]^T \tag{26}$$

$$\boldsymbol{z}^l = \boldsymbol{\Theta}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \, (l = 1, 2, .., L) \tag{27}$$

$$\boldsymbol{a}^l = ReLU\left(\boldsymbol{z}^l\right) \, (l = 1, 2, .., L-1) \tag{28}$$

$$\boldsymbol{a}^l = Dropout\left(\boldsymbol{a}^l\right) \, (l = 1, 2, .., L-1) \tag{29}$$

To avoid overfitting in the multi-layer perceptron classifier, a technology named Dropout (Srivastava et al., 2014) is applied. Simply, the dropout technology randomly ignores a certain ratio p of neural units in a FC layer when training the network model, and the neural layers of $1-p$ left neurons form the valid network structure for forward and backward propagation.

The L^{th} FC layer in the classifier is consist of $c+1$ neurons that stand for 1 normal state and c fault states in the TE process respectively. The output vector $\boldsymbol{z}^L \in \mathbb{R}^{c+1}$ of the L^{th} neural layer is

$$\boldsymbol{z}^L = [z_0, z_1, \ldots, z_c]^T \tag{30}$$

To further judge the class or type of the faults, a Softmax layer follows the L^{th} fully connected neural layer, which transforms the real numbers of output vector $\boldsymbol{z}^L$ into probability distribution as

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \, (i = 0, 1, \ldots, c) \tag{31}$$

$$\sum_i \sigma(z_i) = \frac{\sum_i e^{z_i}}{\sum_j e^{z_j}} = 1 \tag{32}$$

$$\sigma\left(\boldsymbol{z}^L\right) = [\sigma(z_0), \sigma(z_1), \ldots, \sigma(z_c)]^T \tag{33}$$

The corresponding index of maximal value in $\sigma\left(\boldsymbol{z}^L\right)$ is regarded as the fault class for the input data sample. After the forward propagation, the loss is calculated using $\sigma\left(\boldsymbol{z}^L\right)$ and one-hot encoded vector $\boldsymbol{y} \in \mathbb{R}^{c+1}$ of the actual fault class. If the actual fault class corresponding to the input is $i(0, 1, 2, \ldots c)$, the i^{th} element of $\boldsymbol{y}$ is set to 1 and other elements are all set to 0. The loss function commonly used in multi-class classification problem is cross entropy loss function, which is also used in this paper. And the loss can be calculated for backward propagation then for parameters updating with Adam optimizer (Kingma and Ba, 2017).

$$\mathcal{L}\left(\boldsymbol{\Theta}^k, \boldsymbol{\Theta}^l, \boldsymbol{b}^l\right) = -\frac{1}{c+1}\sum_{i=0}^{c}\left[y_i \log\sigma(z_i) + (1-y_i)\log(1-\sigma(z_i))\right] \tag{34}$$

The complete algorithm for training PTCN is show as Table 2.

After trained for certain epochs, the PTCN model is ready for fault type inference. Given a test data sample $\left\{\boldsymbol{X}_t, \boldsymbol{A}\right\}$, the data will be forward propagated through the layers with all the calculation procedures as Eq. (21) ~ (32), and the model will output the vector $\sigma\left(\boldsymbol{z}^L\right)$. The index for the maximal element in $\sigma\left(\boldsymbol{z}^L\right)$ will be the fault type that the model infers in term of the test data sample.

3.4. Fault diagnosis method based on PTCN

The fault diagnosis method based on PTCN is divided into off-line stage and on-line stage showed as Fig. 5. The procedures are described in detail as below.

3.4.1. Off-line stage

Step 1: Historical process data in normal state and different faulty states are collected for PTCN model building. Pipes & instruments diagram is collected for graph modeling.

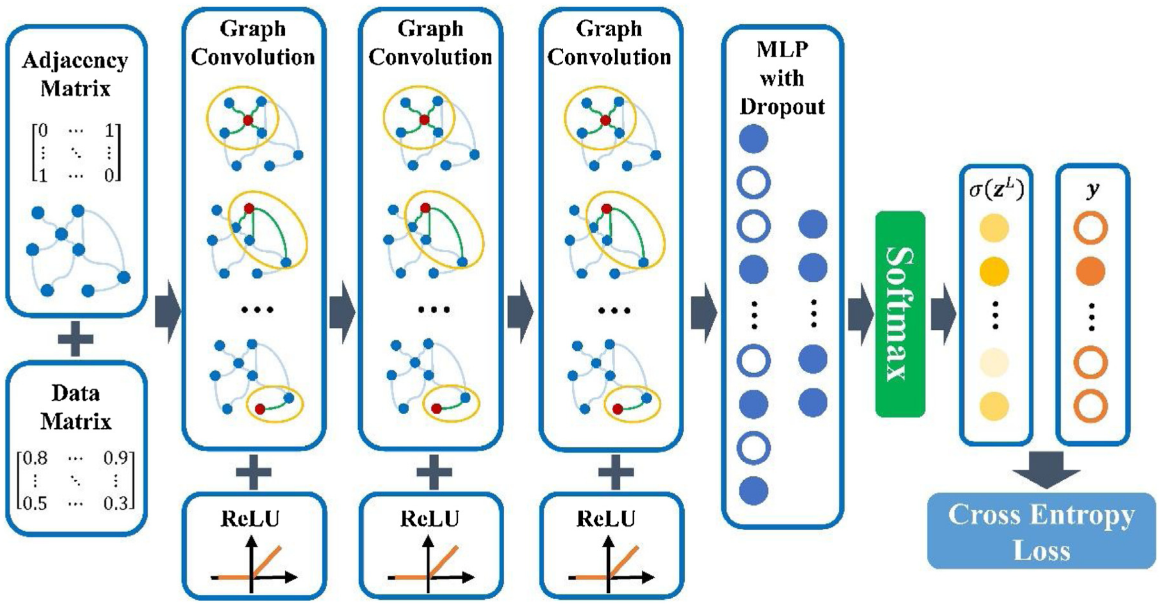

Fig. 4. PTCN model structure.

Table 2
PTCN training algorithm.

```
INPUT: Dataset $\{\boldsymbol{X}_t, \boldsymbol{A}, \boldsymbol{y}\}_{n=1}^{N}$
  $\boldsymbol{\Theta}^k (k = 1,2,\ldots,K), \boldsymbol{\Theta}^l (l = 1,2,\ldots,L), \boldsymbol{b}^l (l = 1,2,\ldots,L) \leftarrow$ Initialize parameters
  for $epoch = 1$ to $E$ do
    for $n = 1$ to $N$ do
      $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$
      $\widetilde{D}_{jj} = \sum_k \tilde{A}_{kj}$
      $\boldsymbol{H}^0 = \boldsymbol{X}_t$
      for $k = 1$ to $K$ do
        $\boldsymbol{Z}^k = (\widetilde{\boldsymbol{D}})^{-\frac{1}{2}} \boldsymbol{A} (\widetilde{\boldsymbol{D}})^{-\frac{1}{2}} \boldsymbol{H}^{k-1} \boldsymbol{\Theta}^k$
        $\boldsymbol{H}^k = ReLU(\boldsymbol{Z}^k)$
      end for
      $\boldsymbol{a}^0 = flat(\boldsymbol{H}^K) = [\boldsymbol{H}_1^K, \boldsymbol{H}_2^K, \ldots, \boldsymbol{H}_n^K]^T$
      for $l = 1$ to $L - 1$ do
        $\boldsymbol{z}^l = \boldsymbol{\Theta}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$
        $\boldsymbol{a}^l = ReLU(\boldsymbol{z}^l)$
        $\boldsymbol{a}^l = Dropout(\boldsymbol{a}^l)$
      end for
      $\boldsymbol{z}^L = \boldsymbol{\Theta}^L \boldsymbol{a}^{L-1} + \boldsymbol{b}^L = [z_0, z_1, \ldots, z_c]^T$
      for $i = 1$ to $c$ do
        $\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$
      end for
    end for
    $\mathcal{L}(\boldsymbol{\Theta}^k, \boldsymbol{\Theta}^l, \boldsymbol{b}^l) = \frac{1}{N} \sum_N \left(-\frac{1}{c+1} \sum_{i=0}^{c} [y_i \log \sigma(z_i) + (1 - y_i) \log(1 - \sigma(z_i))]\right)$
    $\boldsymbol{\Theta}^k (k = 1,2,\ldots,K), \boldsymbol{\Theta}^l (l = 1,2,\ldots,L), \boldsymbol{b}^l (l = 1,2,\ldots,L) \leftarrow$ Backpropagation and Update
parameters using gradients of $\mathcal{L}(\boldsymbol{\Theta}^k, \boldsymbol{\Theta}^l, \boldsymbol{b}^l)$ (e.g. Adam optimizer)
  end for
```

Step 2: Historical process data are normalized with mean and standard deviation of every variable. Different types of faults are recognized from the historical data. A graph is constructed from the chemical process topology according to P&ID and the design of decentralized control system.

Step 3: Preprocessed historical data are cut into data matrices $\boldsymbol{X}_t \in \mathbb{R}^{n \times w}$ and every data matrix is labeled with a one-hot encoded vector $\boldsymbol{y}$ according to its fault type. The adjacency matrix $\boldsymbol{A}$ is calcu-

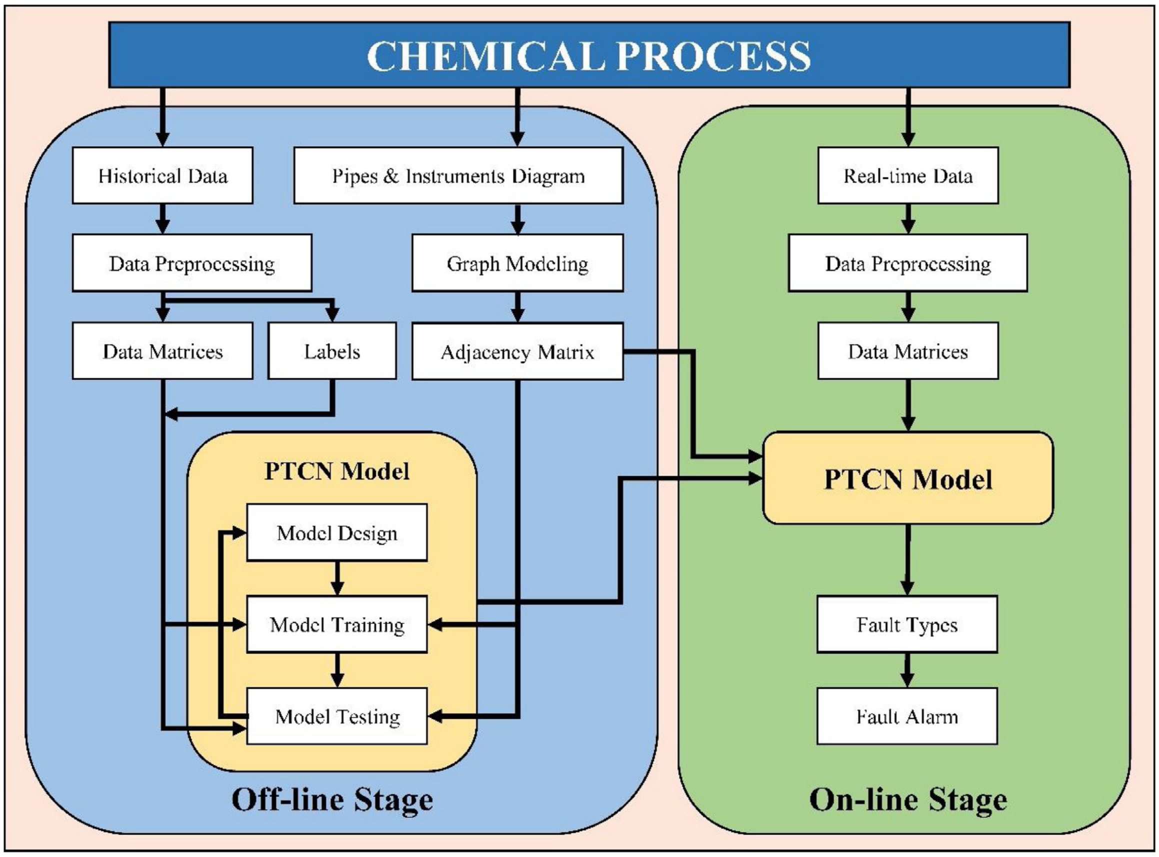

Fig. 5. The framework of fault diagnosis method based on PTCN.

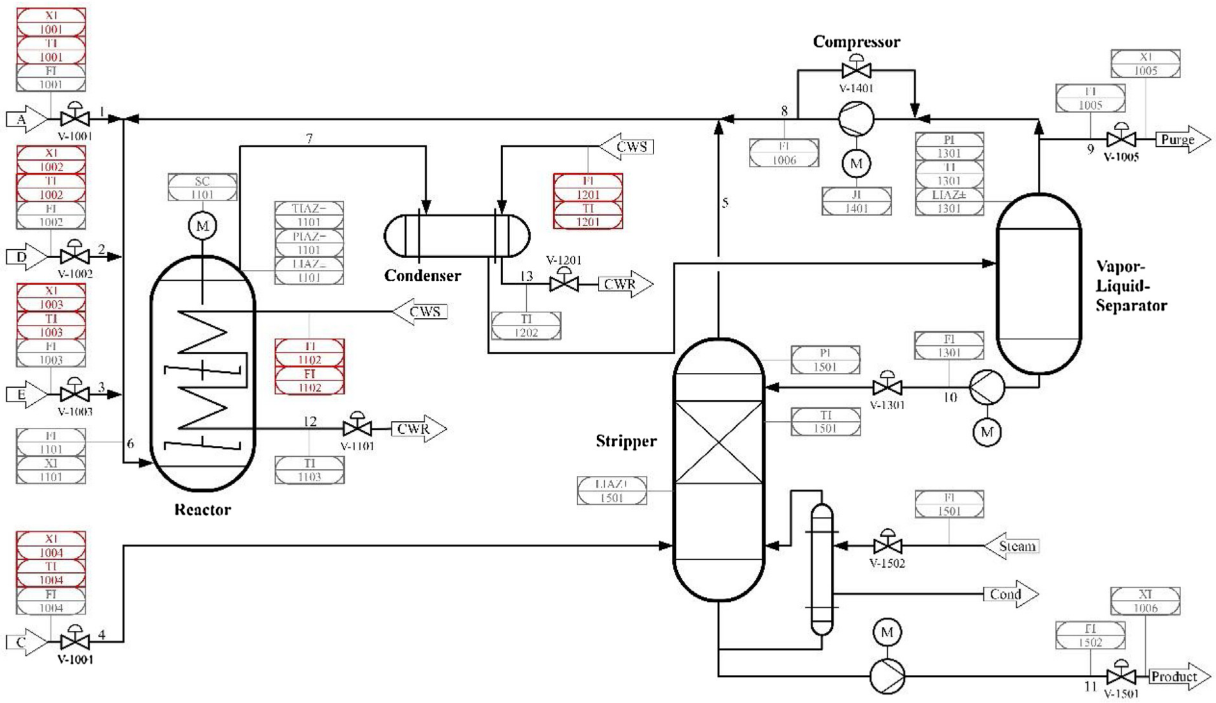

Fig. 6. P&ID of the revised process model; additional measurements in red (Bathelt et al., 2015) (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

lated from the graph. Dataset $\left\{\boldsymbol{X}_t, \boldsymbol{A}, \boldsymbol{y}\right\}_{i=1}^{N}$ is divided into training dataset and testing dataset then.

Step 4: The PTCN model is built as the structure showed in Fig. 4. The number of graph convolutional layers and the dimension of node embedding should be chosen according to the scale of the problem.

Step 5: The model is trained with training dataset.

Step 6: The model is tested with testing dataset.

Step 7: If the testing results are not satisfactory, the number of graph convolutional layers and the dimension of node embedding should be modified in Step 4 before retrained and retested. Otherwise, the model should be saved and prepared for further application at the on-line stage.

3.4.2. On-line stage

Step 1: The PTCN model saved at the off-line stage is loaded and prepared for real-time fault diagnosis.

Step 2: The real-time process data are collected continuously from the process for fault diagnosis.

Step 3: The real-time process data are normalized with the same mean and standard deviation at the off-line stage.

Step 4: The preprocessed real-time data are cut into data matrices and packed with the adjacency matrix calculated at the off-line stage.

Step 5: The data matrices and the adjacency matrix are input into the loaded PTCN model.

Step 6: The PTCN model outputs the diagnosis results. If a fault occurs, the model will recognize and output the fault type.

4. Case study: Tennessee Eastman process

4.1. Introduction

Tennessee Eastman (TE) process is utilized as a benchmark to show the advantages of the proposed PTCN model in this paper. (Downs and Vogel, 1993) first introduced TE process that was modified based on an actual industrial process of Eastman Chemical Company in Tennessee, USA for testing process control technologies. It developed as a platform to examine or compare different algorithms in chemical process study later. The process model describes the non-linear relationships in the unit operation and the material and energy balances. (Ricker, 1995) studied optimal steady-state operations and (Lawrence Ricker, 1996) developed a decentralized control system of TE process shortly after it was published. After two decades, (Bathelt et al., 2015) revised TE process model for inconsistent computational results with different solvers, and extended the process model with additional process measurements and runtime outputs, which is available at https://depts.washington.edu/control/LARRY/TE/download.html. The process is mainly consisted of five unit operations: a reactor, a condenser, a vapor-liquid separator, a recycle compressor and a product stripper. The P&ID of the revised process model is showed as Fig. 6.

The process produces two products from four reactants, including an inert and a byproduct. There're six modes of process operations at different products mass ratio and production rate. The process has 41 measurements and 12 manipulated variables in (Downs and Vogel, 1993), and (Bathelt et al., 2015) added 32 more measurements. For further testing and evaluation of different algorithms, 20 process disturbances showed in Table 3 can be added to the process, which will make the process operate at abnormal conditions of faults. Although 8 more kinds of disturbances were designed in (Bathelt et al., 2015), most papers used only the original 20 process disturbances to test their proposed models. And in this paper, only the original set of 41 measurements, 11 manipulated variables (agitator speed XMV (12) was excluded because it stays constant during simulations) and 20 process disturbances were utilized. The data for evaluating the proposed model were simulated from the MATLAB/Simulink model provided in (Bathelt et al., 2015) at mode 1 (base case).

Table 3
Process disturbances in TE process.

Variable Number	Process variable	Type
IDV (1)	A/C feed ratio, B composition constant (stream 4)	Step
IDV (2)	B composition. A/C ratio constant (stream 4)	Step
IDV (3)	D feed temperature (stream 2)	Step
IDV (4)	Reactor cooling water inlet temperature	Step
IDV (5)	Condenser cooling water inlet temperature	Step
IDV (6)	A feed loss (stream 1)	Step
IDV (7)	C header pressure loss-reduced availability (stream 4)	Step
IDV (8)	A, B, C feed composition (stream 4)	Random variation
IDV (9)	D feed temperature (stream 2)	Random variation
IDV (10)	C feed temperature (stream 4)	Random variation
IDV (11)	Reactor cooling water inlet temperature	Random variation
IDV (12)	Condenser cooling water inlet temperature	Random variation
IDV (13)	Reaction kinetics	Slow drift
IDV (14)	Reactor cooling water valve	Sticking
IDV (15)	Condenser cooling water valve	Sticking
IDV (16)	Unknown	Unknown
IDV (17)	Unknown	Unknown
IDV (18)	Unknown	Unknown
IDV (19)	Unknown	Unknown
IDV (20)	Unknown	Unknown

4.2. Graph constructed from TE process topology

As mentioned in Section 3.2, PTCN requires an adjacency matrix $\boldsymbol{A}$ as a part of the input data structures. To generate the adjacency matrix, the TE process topology should be formulated as a graph first. Referring to the P&ID diagram in Fig. 6 and the decentralized control system designed by (Lawrence Ricker, 1996), the steps mentioned in Section 3.1 were followed to construct a graph from the TE process topology. The graph was named as TE graph G^{TE} in this paper. The complete TE graph G^{TE} is showed as Fig. 7, which consists of 82 nodes and 113 edges. The TE graph G^{TE} was then transformed into corresponding adjacency matrix $\boldsymbol{A}^{TE} \in \mathbb{R}^{82\times 82}$ with Eq. (17) for further usage in the training process of PTCN.

As mentioned in Section 3.2, the set of' s nodes V includes not only the measurements and manipulated variables mentioned in TE process, but also some abstract nodes that are not observed. But in the PTCN model, every node in the graph has a related node feature to satisfy the requirement for message passing. For those abstract nodes that are not observed in the TE process, the node features should be initialized with certain initialization technologies, which will be discussed in Section 4.5.

4.3. TE process simulation and data preprocessing

As mentioned above, (Bathelt et al., 2015) provided a revised version of TE process simulation code on the platform of MATLAB/Simulink on the Tennessee Eastman Challenge Archive website https://depts.washington.edu/control/LARRY/TE/download.html#Updated_TE_Code. To generate enough training and testing process data for PTCN, the TE process model was simulated with MATLAB/Simulink referring to the methods in (Zhang and Zhao, 2017) and (Wu and Zhao, 2018). The simulation mode was set to mode 1 (base case). The simulation duration and data

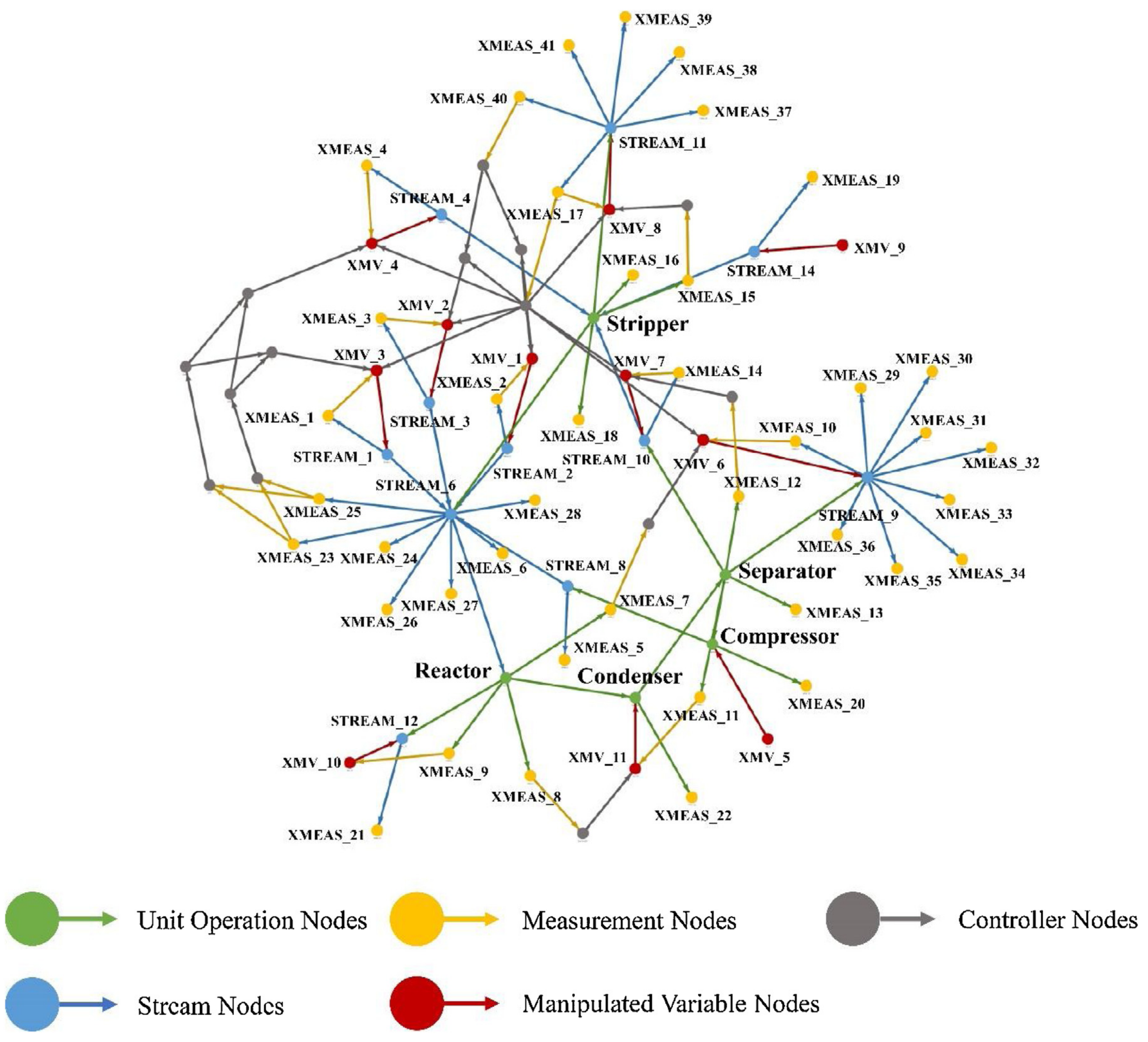

Fig. 7. The complete TE graph.

Table 4
Simulation dataset for training and testing.

Fault types	Simulation length (training) / h	Number of Data Matrices (training)	Simulation length (testing) / h	Number of Data Matrices (testing)
IDV 1~5 & 7~20	20 × 19 × 8	60800	20 × 19 × 2	15200
IDV 6	7 × 1 × 8	1120	7 × 1 × 2	280
Normal	3000	59981	750	14981
Total	6096	121901	1524	30461

samples are showed in Table 4. For 20 different faulty states, the process was simulated for 10 times. In every run, the simulation generated data of 28 h and the fault was inserted into the process after running in normal state for 8 h, which means there are 20 h of simulation data for every run of a fault type. The simulation data were sampled every 3 min. It should be noted that the simulation can only last for 7 h after the fault IDV 6 is inserted, because some variables will exceed allowable limits after 7 h' run, which will cause an early stop of the simulation program. For normal state, the model was simulated for 3750 h all at once without any fault inserted.

The simulation data was normalized with sample mean and sample standard deviation calculated with all the simulation data in normal state. As described in Section 3.2, the data matrix $\boldsymbol{X}_t$ in a data sample $\{\boldsymbol{X}_t, \boldsymbol{A}, \boldsymbol{y}\}$ was stacked time serials in a certain time window. In this paper, the time window was set to 60 min, which means $\boldsymbol{X}_t$ includes 20 data points for every variable.

For faulty data, 8 runs of simulation data were randomly chosen as training data and 2 runs were chosen as testing data. For normal data, 3000 h of simulation data were chosen as training data and 750 h were chosen as testing data. The ratio of faulty samples to normal samples was about 1:1 either in training dataset or testing dataset, which is considered to balance the numbers of faulty samples and normal samples for less false alarms.

4.4. Training parameters and evaluation criterion

To ensure the consistency of training conditions, all PTCN models were trained for 50 epochs. In every epoch, the dataset was divided into mini-batches with the size of 128 data samples. Cross

Table 5
Training parameters for the PTCN model.

Epochs	Mini-batch size	Loss function	Optimizer	Learning rate
50	128	Cross entropy loss	Adam	0.001

Table 6
Confusion matrix in multi-class classification.

		Predicted class	
’		Class c	Other Classes
Actual class	Class c	True Positive (TP)	False Negative (FN)
	Other Classes	False Positive (FP)	True Negative (TN)

entropy loss function was chosen for this multi-classification problem. For backpropagation, Adam optimizer (Kingma and Ba, 2017) was used to update the trainable parameters with the learning rate of 0.001 (Table 5).

There should be some criterions to evaluate the performance of fault diagnosis methods, so fault diagnosis rate (FDR) and accurate classification rate (ACR) (Zhang and Zhao, 2017) are introduced as followed. In multi-class classification, the confusion matrix for class c is defined as Table 6.

FDR and ACR are defined as

$$FDR = \frac{N_{TP}}{N_{TP} + N_{FN}} \tag{35}$$

$$ACR = \frac{\sum_c N_{TP}}{\sum_c N_{TP} + \sum_c N_{FN}} = \frac{\sum_c N_{TP}}{N_{total}} \tag{36}$$

FDR concerns how many samples of a certain class are recognized as the class itself, it's also called true positive rate (TPR) generally. The higher FDR is, the more accurate the fault diagnosis model is for recognize class c. ACR is a total classification accuracy of this fault diagnosis model considering all the testing samples. It's also a class-weighted average of FDR and will be used to evaluate the general performance of a model in this paper.

4.5. Hyper-parameters tuning

4.5.1. Direction of TE graph

In Section 4.2, the TE process topology is formulated as a directed graph G^{TE}, where all the edges' directions are determined by streams' flow directions or control signals' transmission directions. However, the best directions for message passing on G^{TE} are not necessarily the same as the physical conditions. Message may also be passed on a reverse graph of G^{TE} or an undirected graph of G^{TE} Fig. 8 shows the difference among a directed graph, its reverse graph and its undirected graph.

To choose the best directions for message passing, three types of adjacency matrix $\boldsymbol{A}^{\mathrm{original}}$, $\boldsymbol{A}^{\mathrm{reverse}}$, $\boldsymbol{A}^{\mathrm{undirected}}$ are added to the dataset for testing.

$$\boldsymbol{A}^{\mathrm{original}} = \boldsymbol{A}^{\mathrm{TE}} \tag{37}$$

$$\boldsymbol{A}^{\mathrm{reverse}} = \left(\boldsymbol{A}^{\mathrm{TE}}\right)^T \tag{38}$$

$$\boldsymbol{A}^{\mathrm{undirected}} = \boldsymbol{A}^{\mathrm{TE}} + \left(\boldsymbol{A}^{\mathrm{TE}}\right)^T \tag{39}$$

The base model structure is *Model 3* in Table 8, which consists of 3 graph convolutional layers with 20-dimension embedding and a 2-layer perceptron classifier with 300 neurons and a *Dropout* rate of 0.5 in the first layer. The last layer of the classifier has 21 neurons (1 for normal state and 20 for different types of fault). The variables in $\boldsymbol{X}_t$ that are not observed in TE process are all initialized with 0. The training parameters are introduced in Section 4.4. Table 7 shows the experiments results trained using the dataset in Section 4.3 with different type of graphs. The results indicate that the best message passing directions should be the same as the reverse of the original TE graph. Along the directions of streams' flow or control signals' transmission, the nodes in the downstream will carry the information propagated through the edges from the nodes in the upstream. With a reverse message passing directions in the PTCN model, one node's hidden state will fuse information of itself and the nodes under its influence that belongs to a serial of time since the downstream nodes carry its previous information. This helps the model to extract node representations from spatial and temporal domains simultaneously. Thus, the reverse TE graph was chosen for PTCN model in the experiments below and the final testing procedure in Section 4.6.

Table 7
Experiments results using different type of graphs.

Graph types	Original TE graph	Reverse TE graph	Undirected TE graph
ACR	0.9246	**0.9392**	0.9168

Table 8
PTCN model with different number of graph convolutional layers.

Model Name	Model structure
Model 1	*GConv(20)-FC(300, 0.5)-FC(21)*
Model 2	*GConv(20)-GConv(20)-FC(300, 0.5)-FC(21)*
Model 3	*GConv(20)-GConv(20)-GConv(20)-FC(300, 0.5)-FC(21)*
Model 4	*GConv(20)-GConv(20)-GConv(20)-GConv(20)-FC(300, 0.5)-FC(21)*
Model 5	*GConv(20)-GConv(20)-GConv(20)-GConv(20)-GConv(20)-FC(300, 0.5)-FC(21)*

Table 9
Experiments result of PTCN model with different number of graph convolutional layers.

Model Name	*Model 1*	*Model 2*	*Model 3*	*Model 4*	*Model 5*
ACR	0.9294	0.9283	**0.9392**	0.9311	0.9271

4.5.2. Number of graph convolutional layers

The number of graph convolutional layers has a strong impact on the performance of PTCN since it decides the width of message passing. With too few graph convolutional layers, a node can only perceive nearby neighbor nodes so the information is not fully spread through the graph. With too many graph convolutional layers, every node will contain information from nearly the whole graph, which eliminates the differences among nodes and makes the graph too smooth to keep the distinguish information of nodes. To determine the appropriate number, 5 PTCN models with 1∼5 different number of graph convolutional layers are trained and tested with the dataset. The model candidates are shown as Table 8. In the structure of models, *GConv(n)* means a graph convolutional layer with hidden state embedding dimension of n, $FC(n, p)$ is a fully connected layer with n neurons using a dropout ratio of p. The variables in $\boldsymbol{X}_t$ that are not observed in TE process are all initialized with 0. Table 9 shows that 3 graph convolutional layers is appropriate for PTCN, and less *GConv* layers or more *GConv* layers will have an impact on decreasing the overall performance of the model.

4.5.3. Embedding dimension of node features

The embedding dimension of node features is another important parameter for graph convolutional layers, which has to match with the scale of problem. With a high embedding dimension, the hidden states can be very sparse and thus increases the computational cost for redundancy dimensions. With a low embedding

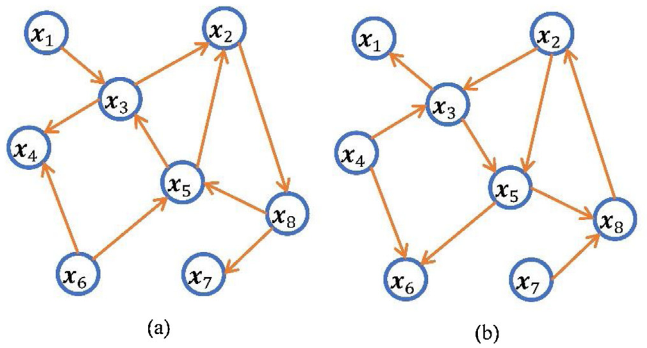

Fig. 8. Example graph with different edge directions. (a) original graph *G*; (b) the reverse graph of *G*; (c) undirected graph of *G*.

Table 10
PTCN model with different embedding dimension of graph convolutional layers.

Model Name	Model structure
Model 6	*GConv(16)-GConv(8)-GConv(4)-FC(300, 0.5)-FC(21)*
Model 7	*GConv(16)-GConv(10)-GConv(8)-FC(300, 0.5)-FC(21)*
Model 3	*GConv(20)-GConv(20)-GConv(20)-FC(300, 0.5)-FC(21)*
Model 8	*GConv(24)-GConv(28)-GConv(32)-FC(300, 0.5)-FC(21)*
Model 9	*GConv(32)-GConv(48)-GConv(64)-FC(300, 0.5)-FC(21)*

Table 11
Experiments result of PTCN model with different embedding dimension.

Model name	*Model 6*	*Model 7*	*Model 3*	*Model 8*	*Model 9*
ACR	0.9264	0.9269	**0.9392**	0.9305	0.9312

dimension, there will be information loss of node features that can degrade the performance of PTCN. The initialized node feature is 20-dimension for every variable, which is determined by the time window we choose to form a data matrix $\boldsymbol{X}_t$. To choose an appropriate dimension of embedding, different PTCN model candidates are set as Table 10. The variables in $\boldsymbol{X}_t$ that are not observed in TE process are all initialized with 0. The testing results are shown in Table 11 and it indicates that the dimension of hidden states should be kept to 20 all the time.

4.5.4. Initialization methods of unobserved variables

Section 3.2 mentioned that the data matrix $\boldsymbol{X_t} \in \mathbb{R}^{82\times20}$ includes not only the 41 measurements and the 11 manipulated variables (XMV (12) excluded), but also some nodes without corresponding observations in the TE process. These nodes should be initialized with some values for computational requirements, which is also known as missing value filling. Three initialization methods are tested to find an appropriate one, namely constant initialization, random initialization and standard normal initialization. Constant initialization will initialize the missing values with the constant 0, since all the historical data are normalized with standard score normalization. Random initialization will initialize the miss values with data randomly sampled from $[-1, 1]$, and standard normal initialization will use the data sampled from standard normal distribution $N(0,\ 1)$. *Model 3* was chosen as the base model for testing the initialization methods. The results are shown as Table 12. It can be seen that initialization with constant 0 is an appropriate method to fill the missing values relatively.

Table 12
Experiments result of PTCN model with different initialization methods for unobserved variables.

Initialization methods	Constant initialization	Standard normal initialization	Random initialization
ACR	**0.9392**	0.9250	0.9274

Table 13
ACR and average time cost for training and testing.

Model	Testing ACR	Time cost for training /second/epoch	Time cost for testing /second/epoch
Model 1	0.9294	23	7
Model 2	0.9283	24	7
Model 3	**0.9392**	25	7
Model 4	0.9311	29	8
Model 5	0.9271	31	8
Model 6	0.9264	26	8
Model 7	0.9269	26	8
Model 8	0.9305	27	8
Model 9	0.9312	27	8

4.6. Experiment results

Table 13 concludes all the ACR and the average time cost for training and testing within every epoch. All the experiments were conducted on the platform of CentOS Linux release 7.7 with Intel Xeon Gold 5118 processor and Titan RTX graphics card. To avoid the error caused by randomness, one type of model structure was trained for 10 times and the results showed in Table 13 are the average of all parallel experiments. It shows that the average time cost for training within every epoch is around 27 s, which shows no big difference between different models. And the PTCN with *Model 3* structure reaches the highest testing ACR of 0.9392.

The final chosen structure for PTCN is *Model 3* in Table 8. The adjacency matrix $\boldsymbol{A}^{\text{reverse}}$ required by PTCN is calculated with the reverse of original TE graph. And the missing values in data matrices are all initialized with the constant 0. With the training parameters introduced in Section 4.4, PTCN is trained and tested and Fig. 9 show the changes of average accuracy with the number of training epochs for both training and testing dataset. It can be seen that the curves are converged steadily. Even with 3 epochs, the testing accuracy almost reaches the maximum.

The confusion matrix of the testing results shown as Fig. 10 illustrates the diagnosis performance of PTCN for every fault class respectively. The labels in vertical axis are actual labels for fault types from 0 to 20, and fault 0 means normal state. The labels in horizontal axis are predicted labels. In previous studies of fault diagnosis models, FDRs for fault 3, 9, 15 are relatively low and the 3 types of faults can hardly be distinguished from normal state. Fig. 10 visually display that PTCN greatly improve the diagnosis for

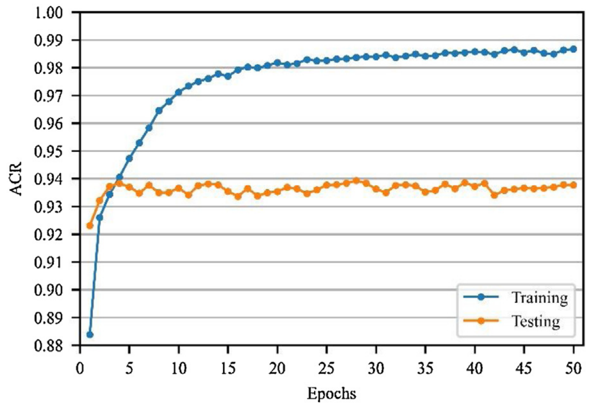

Fig. 9. Change of testing ACR with the number of training epochs.

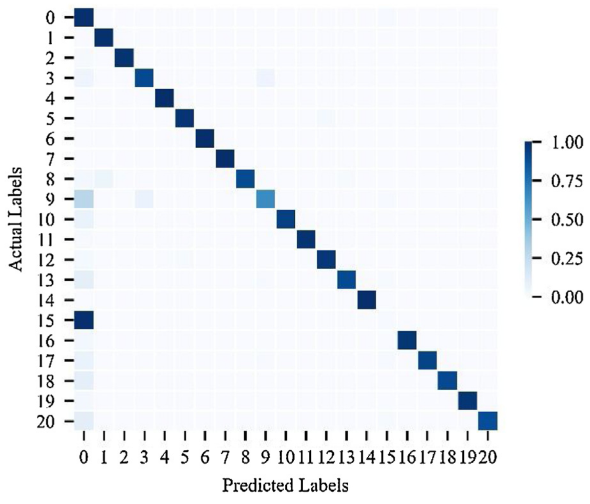

Fig. 10. Confusion matrix of the testing results.

fault 3, while fault 9 and 15 are still hard to recognized, especially for fault 15.

Table 14 gives the detailed experiments results for every fault type and the comparison with DBN-based (Zhang and Zhao, 2017), CNN-based (Wu and Zhao, 2018) and RNN-based (Zhang et al., 2020a) fault diagnosis models. These are all deep learning-based models. And the other approaches in Table 14 utilize graphs either, including Bayesian-based CGN (Lou et al., 2020) and SOM-based DFAE-SOM (Lu and Yan, 2020). The FDR for normal state of PTCN is much higher than other models, which will decrease false alarms in actual application. This is quite importance since a high frequency of false alarm will indeed increase the burden of the operators to confirm and thus lower their trust on the fault diagnosis models, which goes against the original intension. In all the 21 classes, there are 17 classes that corresponding FDR is or higher than 0.9. For fault 8 and 16 that FDRs are not satisfactory in some of other models, PTCN reaches the FDR of 0.9160 or higher, especially for fault 16 that the FDR reaches 0.9685. For the overall performance of the models, PTCN reaches the highest 0.9392 average accuracy among all the models, which is 5.72 % higher than DCNN and 1.22 % higher than BiGRU. Without the fault 9 and 15 that are constitutionally similar to normal state, the average accuracy of PTCN reaches 0.9729, and is 2.09 % higher than BiGRU, 3.89 % than CNN, 3.52 % higher than CGN and 2.32 % higher than DFAE-SOM. In the experiments of other graph-based approaches, CGN and DFAE-SOM ignored more than fault 9 and 15, which made the classification problem even easier.

With the knowledge of the chemical process topology, PTCN outperforms other pure data-driven fault diagnosis models in terms of the average performance. The adjacency matrix of the graph input to the model is indispensable, because the graph describes the connectivity of different parts of the chemical process. Essentially, the connectivity indicates the correlations of different process variables. This provides a guidance when the model is trained with data to fit the underlying relationships among different variables in the process, and it will reduce the risk of overfitting the training dataset. Compared to other approaches using graphs, PTCN also gets better results, which indicates its strong ability to learn about the relationships between variables from data.

4.7. Network parameters, computation cost and requirement for training data

To dig into the benefits of the utilization of the process topology, it's important to pay attention to time cost in training and testing, the number of network parameters and the requirement of training data. Time cost in training and testing is about the PTCN's efficiency. In practical applications, the faster PTCN can make inferences, the less time it'll take to detect existing faults. The number of network parameters decides the model's complexity. This is related to the consumption of computing resources and the required amount of training data. A simple model is always preferred because the fault data we can acquire for training are generally limited in real industrial applications.

Table 15 shows the numbers of network parameters and average time cost of training and testing of PTCN with different model structures. The table shows that most of the candidate models have less than 1 million trainable network parameters and the numbers are around 0.6 million. For the chosen structure *Model 3*, the number of network parameters is 0.597 million, while *Model 7* of DCNN in (Wu and Zhao, 2018) has 2.537 million parameters in total, which is more than 4 times than that of *Model 3* of PTCN. This indicates that the complexity of the network can be reduced with the guidance of process knowledge. Because the knowledge can reduce the degree of freedom and then cut down a large proportion of network parameters. Without any prior knowledge, the design of a network model is relatively blind so more trainable parameters will be added for uncertainty. In addition, a more lightweight network model with less parameters means less computing resources consumption including memory and time cost.

Table 15 shows that the average time cost for training within every epoch is around 27 s and it takes 25 s for *Model 3* to be trained for an epoch. Under the condition of parallel computing, the average time cost for testing a data matrix is no more than 0.263 milliseconds when the total number of testing samples is 30,461 and the mini-batch size is set to 128. But *Model 7* of DCNN in (Wu and Zhao, 2018) takes 30 s to train for every epoch. Average time cost for testing a data matrix is 1.5 milliseconds, which is about 6 times as long as PTCN takes. This indicates that the time cost for training and testing is closely related to the number of network parameters. With less parameters, the PTCN can be trained and tested faster, which is critical in the practical application.

To study the requirement of training data, *Model 3* of PTCN and *Model 7* of DCNN was trained with different number of runs of simulation data. And all the trained models were tested with the same testing dataset mentioned in Section 4.3. Table 16 shows the experiments results. The number of training data matrices and the testing ACRs are listed with different number of runs of simulation data. 8 runs of simulation data formed the original training dataset in Section 4.3. With less runs of simulation, the number of data matrices in the training dataset will decrease proportionately, but the ratio of data in normal state to faulty state is kept the same as original training dataset. And all the testing ACRs are the average testing result of 10 independent repeated training processes. Fig. 11 also shows the change of ACR of PTCN and DCNN with different number

Table 14
FDR and ACR comparison of fault diagnosis experiments results on testing dataset.

Fault type	DBN (Zhang and Zhao, 2017)	DCNN (Wu and Zhao, 2018)	BiGRU (S. Zhang et al., 2020a)	CGN (Lou et al., 2020)	DFAE-SOM (Lu and Yan, 2020)	PTCN (ours)
Normal	–	0.978	0.969	0.985	0.748	0.9924
Fault 1	1	0.986	0.986	0.975	0.995	0.9931
Fault 2	0.99	0.985	0.972	0.980	0.987	0.9819
Fault 3	0.95	0.917	0.935	–	–	0.8804
Fault 4	0.98	0.976	0.974	0.824	0.996	0.9956
Fault 5	0.86	0.915	0.998	0.980	1	0.9786
Fault 6	1	0.975	1	1	1	1
Fault 7	1	0.999	1	1	0.998	1
Fault 8	0.78	0.922	0.753	0.966	–	0.9160
Fault 9	0.57	0.584	0.807	–	–	0.6601
Fault 10	0.98	0.964	1	0.881	–	0.9276
Fault 11	0.87	0.984	0.965	0.778	–	0.9798
Fault 12	0.85	0.956	0.961	0.981	0.781	0.9704
Fault 13	0.88	0.957	0.953	0.758	–	0.8969
Fault 14	0.87	0.987	0.996	0.986	0.978	0.9964
Fault 15	0	0.28	0.541	–	–	0.0035
Fault 16	0	0.442	0.788	0.814	0.855	0.9685
Fault 17	1	0.945	0.97	0.848	0.928	0.9254
Fault 18	0.98	0.939	0.923	0.685	–	0.9049
Fault 19	0.93	0.986	0.926	0.964	0.860	0.9650
Fault 20	0.93	0.933	0.981	0.871	0.788	0.8825
ACR	0.821	0.882	0.927	–	–	0.9392
ACR w/o Fault 9 & 15	0.889	0.934	0.952	0.904	0.916	0.9729

Table 15
Numbers of network parameters and average time cost for training and testing.

Model	Numbers of network parameters/M	Time cost for training /(s/epoch)	Time cost for testing /(ms/matrix)
Model 1	0.531	23	0.230
Model 2	0.564	24	0.230
Model 3	**0.597**	**25**	**0.230**
Model 4	0.630	29	0.263
Model 5	0.663	31	0.263
Model 6	0.144	26	0.263
Model 7	0.249	26	0.263
Model 8	0.962	27	0.263
Model 9	2.011	27	0.263
Model 7 of DCNN(Wu and Zhao, 2018)	2.537	30	1.5

Table 16
ACR of PTCN models trained with different number of runs of simulation data.

Number of runs of simulation data	Number of data matrices	Testing ACR (PTCN, ours)	ACR drop compared to the original training dataset (%)	Testing ACR (DCNN, Wu and Zhao, 2018)	ACR drop compared to the original training dataset (%)
8	121,901	0.9392	–	0.9309	–
7	106,661	0.9389	0.03	0.9294	0.15
6	91,421	0.9373	0.19	0.9260	0.49
5	76,181	0.9347	0.45	0.9227	0.82
4	60,941	0.9281	1.11	0.9165	1.44
3	45,701	0.9227	1.65	0.9133	1.76
2	30,461	0.9142	2.50	0.9040	2.69
1	15,221	0.8943	4.49	0.8738	5.71

of runs of training data. The confidence interval is calculated under the significance level of 0.05.

For both PTCN and DCNN, the testing ACR will drop with the reduction of training data, which is as expected because data-driven models rely on patterns learnt from data. The testing ACR will drop more quickly when more training data are reduced. This means that the more data the training dataset has, the more redundant information the dataset has. In another word, the marginal benefit of adding data to training dataset will diminish in terms of the model's performance.

Trained with different number of runs of simulation data, the testing ACR of PTCN is always about 1% higher than DCNN. As the training data decrease, the testing ACR of DCNN drops more quickly than PTCN. This means that the knowledge of process topology integrated in PTCN plays a role in reducing the impact of the decrease of training data. To get the same testing ACR, training data required by PTCN is less than DCNN. For example, to get the testing ACR of 0.93, it can be estimated from Fig. 11 that PTCN requires only 54 % of training data needed by DCNN. Namely, PTCN relies less on training data with the guidance of information from the chemical process. This feature of PTCN is very critical for the situations where data are relatively difficult to obtain but the topology of the process can be utilized to construct the model.

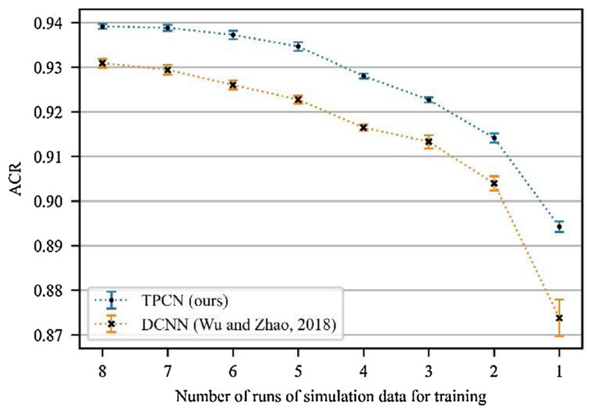

Fig. 11. ACR of PTCN and DCNN trained with different number of runs of simulation data.

5. Conclusion

In this paper, process topology convolutional network (PTCN) model is proposed for fault diagnosis of chemical processes. Different from pure data-driven models, PTCN utilizes the knowledge of the chemical process topology. The process topology is transformed into a graph, which reflects the relationships among different process variables. The graph plays a role of bridge to integrate process knowledge to the model.

To illustrate the performance of PTCN, its application to the benchmark TE process is used as a case study. The TE process topology is first transformed into a directed graph named TE graph. Corresponding adjacency matrix which describes nodes and their connectivity in the graph is established. The training and testing data are generated by running the simulation model of the TE process. The experiments show that the ACR of the PTCN reaches 0.9392, which is higher than existing fault diagnosis models such as CNN-based, RNN-based, Bayesian-based and SOM-based models.

PTCN has only 0.597 million of network parameters, which is less than a quarter of the number of DCNN (Wu and Zhao, 2018). This means less memory consumption and time cost in training and testing processes. Integrated with knowledge of process topology, the PTCN model relies less on training data. Trained with different amount of data, the testing ACR of PTCN is always about 1% higher than DCNN. To get the testing ACR of 0.93, PTCN requires only 54 % of training data needed by DCNN.

From the perspective of process safety, the PTCN model can be further applied in RA and ASM. As an initial step in ASM, the PTCN model can monitor the operating states of chemical processes in real time. Once a fault happens, the model will complete detection and diagnosis simultaneously and will give out the possibilities of different types of fault. These possibilities can be used for quantitative dynamic risk assessment. The combination of PTCN and quantitative risk assessment models can provide the basis of decision for ASM, which is the final step to bring the processes back to normal states.

The knowledge from chemical process topology helps improve the performance of the data-driven model in terms of diagnosis accuracy and the requirements for training data and computation resources. What's more, the feature extraction process of the PTCN model is more rational and understandable than other data-driven fault diagnosis models, since it follows the guidance of the process topology. The future research work will be focused on analyzing PTCN's inference process and enhancing explainability for the PTCN model.

Declaration of Competing Interest

The authors report no declarations of interest.

Acknowledgements

The authors gratefully acknowledge support from the National Natural Science Foundation of China (No. 21878171) and the National Science and Technology Innovation 2030 Major Project (2018AAA0101605) of the Ministry of Science and Technology of China.

References

Amin, M.T., Khan, F., Ahmed, S., Imtiaz, S., 2020. A novel data-driven methodology for fault detection and dynamic risk assessment. Can. J. Chem. Eng. 98, 2397–2416, http://dx.doi.org/10.1002/cjce.23760.

Arunthavanathan, R., Khan, F., Ahmed, S., Imtiaz, S., 2020a. An analysis of process fault diagnosis methods from safety perspectives. Comput. Chem. Eng., http://dx.doi.org/10.1016/j.compchemeng.2020.107197, 107197.

Arunthavanathan, R., Khan, F., Ahmed, S., Imtiaz, S., Rusli, R., 2020b. Fault detection and diagnosis in process system using artificial intelligence-based cognitive technique. Comput. Chem. Eng. 134, http://dx.doi.org/10.1016/j.compchemeng.2019.106697, 106697.

Bathelt, A., Ricker, N.L., Jelali, M., 2015. Revision of the Tennessee eastman process model. IFACPapersOnLine 48, 309–314, http://dx.doi.org/10.1016/j.ifacol.2015.08.199.

Bikmukhametov, T., Jäschke, J., 2020. Combining machine learning and process engineering physics towards enhanced accuracy and explainability of data-driven models. Comput. Chem. Eng. 138, 106834, http://dx.doi.org/10.1016/j.compchemeng.2020.106834.

Chen, H., Tiňo, P., Yao, X., 2014. Cognitive fault diagnosis in Tennessee Eastman process using learning in the model space. Comput. Chem. Eng. 67, 33–42, http://dx.doi.org/10.1016/j.compchemeng.2014.03.015.

Cheng, F., He, Q.P., Zhao, J., 2019. A novel process monitoring approach based on variational recurrent autoencoder. Comput. Chem. Eng. 129, 106515, http://dx.doi.org/10.1016/j.compchemeng.2019.106515.

Dai, Y., Wang, H., Khan, F., Zhao, J., 2016. Abnormal situation management for smart chemical process operation. Curr. Opinion Chem. Eng. Biotechnol. Bioprocess Eng. Proc. Syst. Eng. 14, 49–55, http://dx.doi.org/10.1016/j.coche.2016.07.009.

Dai, Y., Zhao, J., 2011. Fault diagnosis of batch chemical processes using a dynamic time warping (DTW)-Based artificial immune system. Ind. Eng. Chem. Res. 50, 4534–4544, http://dx.doi.org/10.1021/ie101465b.

Defferrard, M., Bresson, X., Vandergheynst, P., 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 29. Curran Associates, Inc., pp. 3844–3852.

Downs, J.J., Vogel, E.F., 1993. A plant-wide industrial process control problem. Comput. Chem. Eng. 17, 245–255.

Gharahbagheri, H., Imtiaz, S.A., Khan, F., 2017. Root cause diagnosis of process fault using KPCA and bayesian network. Ind. Eng. Chem. Res. 56, 2054–2070, http://dx.doi.org/10.1021/acs.iecr.6b01916.

Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E., 2017. Neural message passing for quantum chemistry. arXiv, 1704.01212 [cs].

Glorot, X., Bordes, A., Bengio, Y., 2011. Deep sparse rectifier neural networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. Presented at the Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference ProceedIngs, 315–323.

Gori, M., Monfardini, G., Scarselli, F., 2005. A new model for learning in graph domains. In: Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005. Presented at the International Joint Conference on Neural Networks 2005, IEEE, Montreal, Que., Canada, pp. 729–734, http://dx.doi.org/10.1109/IJCNN.2005.1555942.

Henaff, M., Bruna, J., LeCun, Y., 2015. Deep convolutional networks on graph-structured data. arXiv, 1506.05163 [cs].

Khamsi, M.A., Kirk, W.A., 2001. An Introduction to Metric Spaces and Fixed Point Theory. John Wiley & Sons, Inc., http://dx.doi.org/10.1002/9781118033074.

Khan, F., Rathnayaka, S., Ahmed, S., 2015. Methods and models in process safety and risk management: past, present and future. Process. Saf. Environ. Prot. 98, 116–147, http://dx.doi.org/10.1016/j.psep.2015.07.005.

Kingma, D.P., Ba, J., 2017. Adam: a method for stochastic optimization. arXiv, 1412.6980 [cs].

Kipf, T.N., Welling, M., 2017. Semi-supervised classification with graph convolutional networks. arXiv, 1609.02907 [cs, stat].

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 84–90, http://dx.doi.org/10.1145/3065386.

Lawrence Ricker, N., 1996. Decentralized control of the Tennessee eastman challenge process. J. Process Control 6, 205–221, http://dx.doi.org/10.1016/0959-1524(96)00031-5.

Lever, J., Krzywinski, M., Altman, N., 2016. Model selection and overfitting. Nat. Methods 13, 703–704, http://dx.doi.org/10.1038/nmeth.3968.

Lou, C., Li, X., Atoui, M.A., 2020. Bayesian network based on an adaptive threshold scheme for fault detection and classification. Ind. Eng. Chem. Res. 59, 15155–15164, http://dx.doi.org/10.1021/acs.iecr.0c02762.

Lu, W., Yan, X., 2020. Deep fisher autoencoder combined with self-organizing map for visual industrial process monitoring. J. Manuf. Syst. 56, 241–251, http://dx.doi.org/10.1016/j.jmsy.2020.05.005.

Lv, F., Wen, C., Bao, Z., Liu, M., 2016. Fault diagnosis based on deep learning. 2016 American Control Conference (ACC). Presented at the 2016 American Control Conference (ACC), 6851–6856, http://dx.doi.org/10.1109/ACC.2016.7526751.

Madakyaru, M., Harrou, F., Sun, Y., 2017. Improved data-based fault detection strategy and application to distillation columns. Process. Saf. Environ. Prot. 107, 22–34, http://dx.doi.org/10.1016/j.psep.2017.01.017.

Qin, S.J., Chiang, L.H., 2019. Advances and opportunities in machine learning for process data analytics. Comput. Chem. Eng. 126, 465–473, http://dx.doi.org/10.1016/j.compchemeng.2019.04.003.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Las Vegas, NV, USA, pp. 779–788, http://dx.doi.org/10.1109/CVPR.2016.91.

Ricker, N.L., 1995. Optimal steady-state operation of the Tennessee Eastman challenge process. Comput. Chem. Eng. 19, 949–959, http://dx.doi.org/10.1016/0098-1354(94)00043-N.

Scarselli, F., Gori, M., Chung Tsoi, Ah, Hagenbuchner, M., Monfardini, G., 2009. The graph neural network model. IEEE Trans. Neural Netw. 20, 61–80, http://dx.doi.org/10.1109/TNN.2008.2005605.

Shu, Y., Ming, L., Cheng, F., Zhang, Z., Zhao, J., 2016. Abnormal situation management: challenges and opportunities in the big data era. Comput. Chem. Eng. 91, 104–113, http://dx.doi.org/10.1016/j.compchemeng.2016.04.011.

Sperduti, A., Starita, A., 1997. Supervised neural networks for the classification of structures. IEEE Trans. Neural Netw. 8, 714–735, http://dx.doi.org/10.1109/72.572108.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 1929–1958.

Venkatasubramanian, V., 2019. The promise of artificial intelligence in chemical engineering: Is it here, finally? Aiche J. 65, 466–478, http://dx.doi.org/10.1002/aic.16489.

Venkatasubramanian, V., Rengaswamy, R., Yin, K., Ka, S.N., 2003. A review of process fault detection and diagnosis Part I: quantitative model-based methods. Comput. Chem. Eng., 19.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S., 2020. A comprehensive survey on graph neural networks. IEEE Trans. Neural Netw. Learn. Syst., 1–21, http://dx.doi.org/10.1109/TNNLS.2020.2978386.

Wu, H., Zhao, J., 2018. Deep convolutional neural network model based chemical process fault diagnosis. Comput. Chem. Eng. 115, 185–197, http://dx.doi.org/10.1016/j.compchemeng.2018.04.009.

Xie, D., Bai, L., 2015. A hierarchical deep neural network for fault diagnosis on Tennessee-eastman process. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). Presented at the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 745–748, http://dx.doi.org/10.1109/ICMLA.2015.208.

Yin, S., Ding, S.X., Haghani, A., Hao, H., Zhang, P., 2012. A comparison study of basic data-driven fault diagnosis and process monitoring methods on the benchmark Tennessee Eastman process. J. Process Control 22, 1567–1581, http://dx.doi.org/10.1016/j.jprocont.2012.06.009.

Yu, H., Khan, F., Garaniya, V., 2015. Modified independent component analysis and bayesian network-based two-stage fault diagnosis of process operations. Ind. Eng. Chem. Res. 54, 2724–2742, http://dx.doi.org/10.1021/ie503530v.

Zadakbar, O., Imtiaz, S., Khan, F., 2013. Dynamic risk assessment and fault detection using principal component analysis. Ind. Eng. Chem. Res. 52, 809–816, http://dx.doi.org/10.1021/ie202880w.

Zhang, S., Bi, K., Qiu, T., 2020a. Bidirectional recurrent neural network-based chemical process fault diagnosis. Ind. Eng. Chem. Res. 59, 824–834, http://dx.doi.org/10.1021/acs.iecr.9b05885.

Zhang, Z., Cui, P., Zhu, W., 2020b. Deep learning on graphs: a survey. IEEE Trans. Knowl. Data Eng., http://dx.doi.org/10.1109/TKDE.2020.2981333, 1–1.

Zhang, Z., Zhao, J., 2017. A deep belief network based fault diagnosis model for complex chemical processes. Comput. Chem. Eng. 107, 395–407, http://dx.doi.org/10.1016/j.compchemeng.2017.02.041.

Zheng, S., Zhao, J., 2020. A new unsupervised data mining method based on the stacked autoencoder for chemical process fault diagnosis. Comput. Chem. Eng. 135, 106755, http://dx.doi.org/10.1016/j.compchemeng.2020.106755.