creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학박사 학위논문

Probabilistic Machine Learning Approach to Process Systems Engineering through Parametric Distribution Approximation

매개분포근사를 통한 공정시스템 공학에서의 확률기계학습 접근법

2021년 8월

서울대학교 대학원
화학생물공학부
박 담 대

Probabilistic Machine Learning Approach to Process Systems Engineering through Parametric Distribution Approximation

지도교수 이 종 민

이 논문을 공학박사 학위논문으로 제출함

2021년 8월

서울대학교 대학원

화학생물공학부

박 담 대

박 담 대의 공학박사 학위논문을 인준함

2021년 8월

위 원 장 _____ 이 원 보 _____

부위원장 _____ 이 종 민 _____

위 원 _____ 남 재 욱 _____

위 원 _____ 나 종 걸 _____

위 원 _____ 정 동 휘 _____

Abstract

Probabilistic Machine Learning Approach to Process Systems Engineering through Parametric Distribution Approximation

Damdae Park

School of Chemical and Biological Engineering

The Graduate School

Seoul National University

With the rapid development of measurement technology, higher quality and vast amounts of process data become available. Nevertheless, process data are ‘scarce’ in many cases as they are sampled only at certain operating conditions while the dimensionality of the system is large. Furthermore, the process data are inherently stochastic due to the internal characteristics of the system or the measurement noises. For this reason, uncertainty is inevitable in process systems, and estimating it becomes a crucial part in engineering tasks as the prediction errors can lead to misguided decisions and cause severe casualties or economic losses. A popular approach to this is applying probabilistic inference techniques that can model the uncertainty in terms of a probability. However, most of the existing probabilistic inference techniques are based on recursive

sampling, which makes it difficult to use them for industrial applications that require processing a high-dimensional and massive amount of data. To address such an issue, this thesis proposes probabilistic machine learning approaches based on parametric distribution approximation, which can model the uncertainty of the system and circumvent the computational complexity as well. The proposed approach is applied for three major process engineering tasks: process monitoring, system modeling, and process design.

First, a process monitoring framework is proposed that utilizes a probabilistic classifier for fault classification. To enhance the accuracy of the classifier and reduce the computational cost for its training, a feature extraction method called probabilistic manifold learning is developed and applied to the process data ahead of the fault classification. We demonstrate that this manifold approximation process not only reduces the dimensionality of the data but also casts the data into a clustered structure, making the classifier have a low dependency on the type and dimension of the data. By exploiting this property, non-metric information (e.g., fault labels) of the data is effectively incorporated and the diagnosis performance is drastically improved.

Second, a probabilistic modeling approach based on Bayesian neural networks is proposed. The parameters of deep neural networks are transformed into Gaussian distributions and trained using variational inference. The redundancy of the parameter is autonomously inferred during the model training, and insignificant parameters are eliminated *a posteriori*. Through a verification study, we demonstrate that the proposed

approach can not only produce high-fidelity models that describe the stochastic behaviors of the system but also produce the optimal model structure.

Finally, a novel process design framework is proposed based on reinforcement learning. Unlike the conventional optimization methods that recursively evaluate the objective function to find an optimal value, the proposed method approximates the objective function surface by parametric probabilistic distributions. This allows learning the continuous action policy without introducing any cumbersome discretization process. Moreover, the probabilistic policy gives means for effective control of the exploration and exploitation rates according to the certainty information. We demonstrate that the proposed framework can learn process design heuristics during the solution process and use them to solve similar design problems.

Keywords: Probabilistic machine learning, parametric distribution approximation, uncertainty quantification, probabilistic manifold learning, Bayesian neural networks, Bayesian inference, reinforcement learning, distributional networks.

Student Number: 2015-21060

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Motivation

Data classification, model prediction, and finding optimal solutions constitute the three principal components of the process system engineering tasks. For instance, model predictive control, process optimization, and process design, in essence, seek to find optimal solutions with prediction models; they only differ in optimization targets. For another instance, we can notice that process diagnosis tasks are simply data classification tasks where dimensionality reduction models are employed to predict the systems' behaviors in their intrinsic dimensions. As such tasks are based on models constructed from the process data, it is a natural corollary that the amount of information a model can draw from the data has significance to its performance on executing the given tasks.

One of such information is *uncertainty*, which can be divided into two different kinds: *aleatoric uncertainty* from the stochastic process and *epistemic uncertainty* due to insufficient data [1]. Owing to the

development of measurement technology, higher quality and large amounts of process data are now available and the aleatoric uncertainty has become less concerning. However, epistemic uncertainty caused by data scarcity is unavoidable because the process systems normally operate around some specific operating conditions while they have high dimensionality – which is often referred to as *the curse of dimensionality*. Considering that erroneous predictions can lead to misguided decisions which can cause severe casualties or economic losses in process systems, it is of utmost important to derive high-fidelity models that are able to describe and quantify the uncertainty. To concretize the discussion, such concerns will now be specifically dealt with some important issues found in the process monitoring, process modeling, and process design tasks.

In the process monitoring, particularly when fault classification tasks are given, uncertainty is on the data class (e.g., normal or fault) of a queried data point. A possible solution to model such uncertainty is to train a parametric classifier (such as GMM [2]) with available training data and evaluate the posterior probability of the queried point to each data class. In this case, typically a few important features of the data are extracted ahead of the classification as the classifiers usually show high computational complexity to the data dimension. Statistical projection methods based on PCA [3]–[5] and FDA [6], [7] have been widely used for this purpose. However, these methods solely focus on preserving the variance of the data, and do not preserve the structure of its original *manifold*. As we will see later, this often leads to losing original data distribution in the input space and incurs misclassification (see Box 1 in

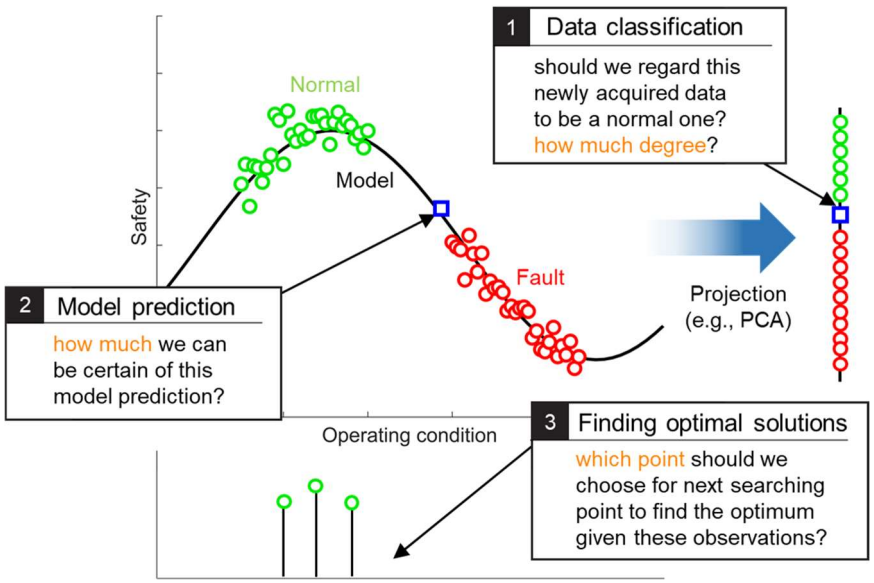

Figure 1.1 Uncertainties found in process systems engineering tasks.

Figure 1.1), especially when the data of two different classes are in proximity. Therefore, a feature extraction method that can capture the original data distribution becomes an indispensable element for achieving high process monitoring performance.

In terms of system modeling, the model predictions (see Box 2 in Figure 1.1), or equivalently the estimates on the model parameters, always entail the uncertainty. To obtain a probabilistic model, which can describe the stochastic nature of the system and quantify the uncertainty, one can employ probabilistic inference techniques such as Markov chain Monte Carlo (MCMC) [8]. However, such techniques usually involve recursive sampling, and this makes it difficult to use them for industrial applications that require processing a large number of data samples and model parameters [9]. Furthermore, advanced modeling techniques, such as surrogate modeling based on neural networks [10], [11] and system identification methods based on model library [12], [13], require the estimation of a myriad number of parameters. In order to address such an issue, a computationally efficient probabilistic modeling approach needs to be devised.

Finally, for process design problems, we are uncertain about the objective function values at unevaluated points, whereas it is obvious that estimates for those values can facilitate the searching process. Note that the conventional optimization approaches such as mathematical programming [14], [15] and metaheuristics [16], [17] query the next guess based on the latest evaluation point and do not exploit the full evaluation record. This seems somewhat inefficient considering that the objective

function surfaces often show simple patterns and we can suppose that the unevaluated points would have similar values to those of the evaluated points nearby. This implies that the objective function surfaces can be efficiently 'encoded' with some parametric distributions [18], [19]. A similar approach can be found in Bayesian optimization [20], except that it is nonparametric and not scalable to the data size.

Summarizing our discussions so far, we can elicit that the uncertainty found in process system engineering can be efficiently modeled by approximating the target objects with some parametric distributions, where the target can be varied from process data and model parameters to objective function values. This constitutes the main theme of this thesis, and we will demonstrate that it can effectively address some major issues found in the three process engineering tasks.

1.2. Outline of the thesis

The remainder of this thesis is organized as follows. Chapter 2 provides the backgrounds on the probabilistic inference and the preliminaries required to follow the next chapters.

In Chapter 3, a feature extraction method, called probabilistic manifold learning, is developed for probabilistic fault classification and process monitoring. The proposed method approximates a data manifold prior to the projection by setting the distance on the manifold with the pairwise likelihoods between the data points (see Box 1 in Figure 1.2). This allows simultaneous utilization of limited labeled data and abundant unlabeled data within a unified scheme. Moreover, through calibrating

the data distribution during the manifold approximation, it produces nearly data-independent projection results. It leads to superior monitoring and fault classification performance compared to the conventional dimensionality reduction methods.

In Chapter 4, a process modeling methodology based on Bayesian deep neural networks is proposed. The proposed method represents the parameters of the neural networks by Gaussian distributions and learns the distribution of the data. The optimal model structure and parameters are obtained through *a posteriori* elimination of the parameters with lower importance (see Box 2 in Figure 1.2). Through an experimental study conducted on a semiconductor manufacturing process, it is demonstrated that the proposed method can not only learn the complex and stochastic behavior of the processes but also is able to derive an optimal architecture of the neural network model.

In Chapter 5, a reinforcement learning-based process design framework is proposed with distributional actor-critic network. Unlike the conventional approach, which only employs recursive evaluation of objective function values to find the optimal solution, the proposed approach approximates the objective function surface with distributional deep neural networks (see Box 3 in Figure 1.2). This enables using behavioral cloning [21] with Monte Carlo tree search [22] and promotes stable learning of continuous policy. We demonstrate that the process design heuristics can be learned during the solution process, and we can use them to solve similar design problems.

Finally, Chapter 6 summarizes the contributions made by the thesis

and discusses their limitations and possible directions for further work.

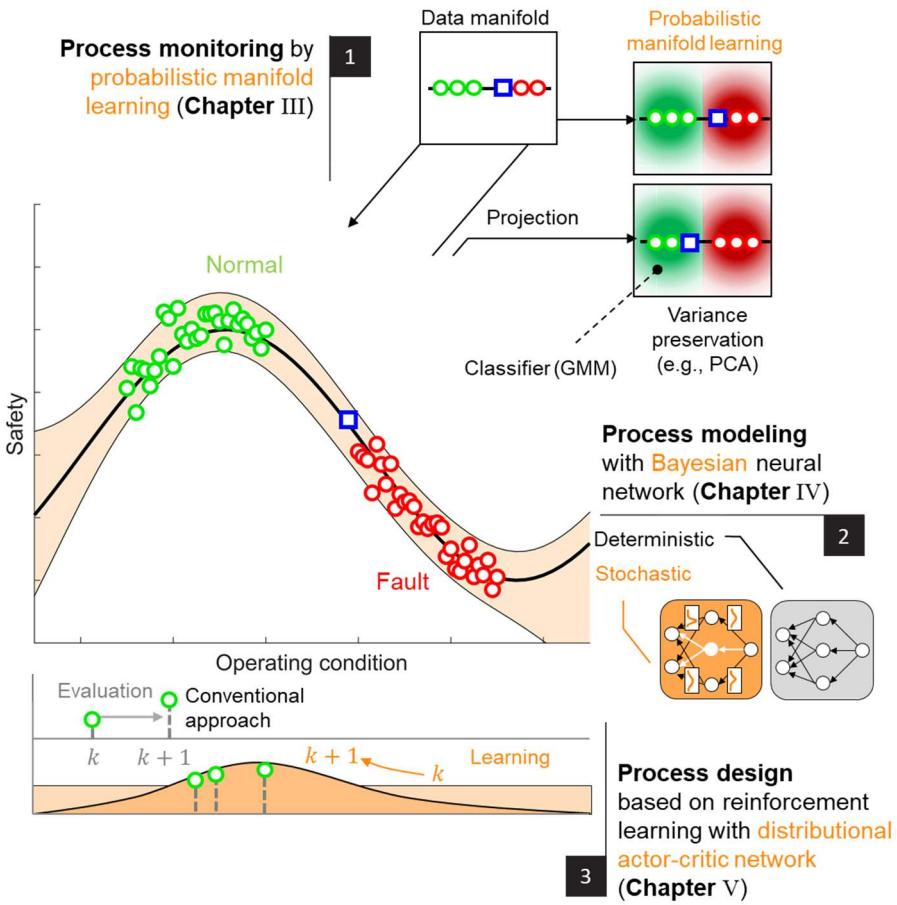

Figure 1.2 Outline of the proposed methods.

Chapter 2

Backgrounds and preliminaries

2.1. Bayesian inference

In the Bayesian inference framework, any task involving learning from data is given with some prior knowledge, based on which the model is updated to incorporate the information from the data. More precisely, we encode our initial belief as probabilities for different possible instances of some variables or parameters θ, and represent it as a *prior* $p(\theta)$. Given observed data $\mathcal{D}$, we can also describe how likely different values of θ are to have given rise to that data using a *likelihood* function $p(\mathcal{D}|\theta)$. These can then be combined using Bayes' rule to derive a *posterior* $p(\theta|\mathcal{D})$, which represents our updated belief on θ.

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)d\theta} = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \tag{2.1}$$

The denominator $p(\mathcal{D})$ here is a normalization constant known as the

marginal likelihood and ensures that $p(\theta|\mathcal{D})$ is a valid probability distribution. Therefore, we can think of Bayes' rule into a much simpler form, where the posterior being proportional to the prior times the likelihood.

A key feature of Bayes' rule is that it can be used in a recursive fashion where the posterior from one task becomes the prior, when the model is updated with more data, that is,

$$p(\theta|\mathcal{D}_1,\mathcal{D}_2) = \frac{p(\mathcal{D}_2|\theta,\mathcal{D}_1)p(\theta|\mathcal{D}_1)}{p(\mathcal{D}_2|\mathcal{D}_1)} = \frac{p(\mathcal{D}_2|\theta,\mathcal{D}_1)p(\mathcal{D}_1|\theta)p(\theta)}{p(\mathcal{D}_2|\mathcal{D}_1)p(\mathcal{D}_1)} \quad (2.2)$$

and this represents the core of the Bayesian framework: the model learns the system by updating its beliefs with the observations. If the newly acquired observations are against the prior experiences, it will not make drastic changes to the underlying belief. It will only change its view if multiple corroborating observations are obtained. Once a strong belief about the system has been developed from consecutive learning, we can take substantial confidence to change our mind, even if the learned belief seems highly illogical.

2.2. Monte Carlo

Monte Carlo is an approximation method that can numerically estimate probability distributions through random sampling. The most common use of Monte Carlo is to estimate the expectations of the distributions, which is often referred to as Monte Carlo integration.

Consider the problem of calculating the expectation of some function

$f(\theta)$ under the distribution $\theta \sim \pi(\theta)$ (that is usually $p(\theta|\mathcal{D})$ for the Bayesian inference case), which is given as

$$I \coloneqq \mathbb{E}_{\pi(\theta)}[f(\theta)] = \int f(\theta)\pi(\theta)d\theta \tag{2.3}$$

We can approximate I using the Monte Carlo estimator I_{MC} where

$$I \cong I_{MC} \coloneqq \frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n) \tag{2.4}$$

and $\hat{\theta}_n \sim \pi(\theta)$. The I_{MC} is an unbiased estimator for I, that is, we have

$$\mathbb{E}[I_{MC}] = \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N} f(\hat{\theta}_n)\right] = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}[(\hat{\theta}_n)] = I \tag{2.5}$$

This indicates that Monte Carlo does not introduce any systematic error (i.e., bias) into the approximation. We can get the true value of I if the estimation is repeated with a large number N [23].

2.3. Kullback-Leibler divergence

The *Kullback-Leibler* (KL) *divergence* [24], also known as *relative entropy*, is a measure of the similarity between two probability density functions $P(x)$ and $Q(x)$, given as below.

$$\mathrm{KL}(P\|Q) \coloneqq \int P(x)\log\left(\frac{P(x)}{Q(x)}\right)dx \tag{2.6}$$

For a discrete variable x, KL divergence is computed as

$$\mathrm{KL}(P\|Q) \coloneqq \mathbb{E}_{x\sim P(x)}\left[\log\left(\frac{P(x)}{Q(x)}\right)\right] = \sum_{x} P(x)\log\left(\frac{P(x)}{Q(x)}\right) \tag{2.7}$$

A few important properties of KL divergence are:

- it is not symmetric, that is, $\mathrm{KL}(P\|Q) \neq \mathrm{KL}(Q\|P)$, and thus, it is not a distance metric;
- it can take on values in $[0, \infty]$, and particularly, if P and Q are the exact same distribution, then $\mathrm{KL}(P\|Q) = \mathrm{KL}(Q\|P) = 0$. In other words, if $\mathrm{KL}(P\|Q) = 0$, then $P \equiv Q$;
- for the KL divergence to be finite, the support of P needs to be contained in the support of Q. If a point x exists with $Q(x) = 0$ but $P(x) > 0$, then $\mathrm{KL}(P\|Q) = \infty$.

The KL divergence can be rewritten as

$$\begin{aligned} \mathrm{KL}(P\|Q) &= \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right) \\ &= \sum_x P(x) \log P(x) - \sum_x P(x) \log Q(x) \\ &= -H(P) + H(P, Q) \end{aligned} \tag{2.8}$$

where $H(P)$ is the *entropy* of P and $H(P, Q)$ is the *cross-entropy* between P and Q. In Bayesian inference, $P(x)$ is regarded as some true distribution to be estimated and $Q(x)$ is an approximate distribution. In this case, the entropy term is given as a constant, and the cross-entropy minimization is given equivalent to the KL divergence minimization.

2.4. Variational inference

In *variation inference* (VI) [25], [26], the posterior distribution of the parameters θ for the data $\mathcal{D}$, $P(\theta|\mathcal{D})$, is approximated by a variational posterior distribution $Q(\theta|\Phi)$, which is a model described by *variational*

parameters Φ (see Figure 2.1). Variational inference seeks the optimal Φ by minimizing reverse KL divergence between the posterior distribution and the approximated posterior distribution.

$$\mathrm{KL}(Q||P) = \mathbb{E}_{\theta\sim Q}\big[\log\big(Q(\theta|\Phi)/P(\theta|\mathcal{D})\big)\big] \tag{2.9}$$

For a loss function $\mathcal{L}(\mathcal{D},\Phi) = \mathrm{KL}\big(Q(\theta|\Phi)||P(\theta|\mathcal{D})\big)$, the optimal estimate of Φ is given by

$$\begin{aligned}\Phi_{\mathrm{VI}} &= \underset{\Phi}{\operatorname{argmin}}\,\mathrm{KL}\big(Q(\theta|\Phi)||P(\theta|\mathcal{D})\big)\\ &= \underset{\Phi}{\operatorname{argmin}}\Big[\mathrm{KL}\big(Q(\theta|\Phi)||P(\theta)\big) - \mathbb{E}_{Q(\theta|\Phi)}[P(\mathcal{D}|\theta)] + \log P(\mathcal{D})\Big]\end{aligned} \tag{2.10}$$

and removing Φ-independent term yields an effective loss function

$$\mathcal{L}_E(\mathcal{D},\Phi) = \mathrm{KL}\big(Q(\theta|\Phi)||P(\theta)\big) - \mathbb{E}_{Q(\theta|\Phi)}[P(\mathcal{D}|\theta)] \tag{2.11}$$

which is often referred as the *variational free energy* [27] or *evidence lower bound* (ELBO) [28]. Note that the effective loss function is a lower bound on the marginal log-likelihood $\log P(x)$, and it is given by the sum of the prior dependent part that serves as a regularizer and the data-dependent part given by the likelihood cost.

2.5. Riemannian manifold

A *Riemannian manifold* is an n-dimensional differentiable smooth manifold $\mathcal{M}$ equipped with a positive-definite inner product g_p on the tangent space $T_p\mathcal{M}$ for each point $p \in \mathcal{M}$. Here, $\mathcal{M}$ is a topological space wherein the local space near each point can be approximated as

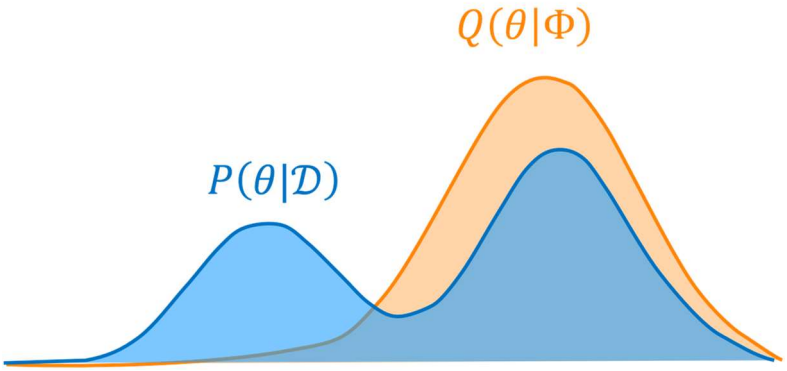

Figure 2.1. Graphical illustration of the variational inference.

Euclidean space. The process data $X = \{x_i\}_{i=1}^N$ in $\mathbb{R}^n$ can be regarded as being sampled from a Riemannian manifold $(\mathcal{M}, g)$ by some injective maps $\varphi_n: \mathcal{M} \hookrightarrow \mathbb{R}^n$.

By assuming that the metric g on $\mathcal{M}$ is locally constant, the shortest path, or *geodesic distance*, between the data points on the local $\mathcal{M}$ is derived by the following Lemma 1 [29].

Lemma 1. Let $(\mathcal{M}, g)$ be a Riemannian manifold embedded in an ambient $\mathbb{R}^n$, and let $x \in \mathcal{M}$ be a point. Suppose that g is locally constant in an open neighborhood U of x such that g is a constant diagonal matrix in $\mathbb{R}^n$. Let B be a ball in $\mathbb{R}^n$ centered at x, whose volume is $\frac{\pi^{n/2}}{\Gamma(n/2+1)}$ with respect to g. The geodesic distance from x to a point $y \in B$ is given by $d_{\mathcal{M}}(x, y) = \frac{1}{r} d_{\mathbb{R}^n}(x, y)$, where r is the radius of B and $d_{\mathbb{R}^n}(x, y)$ is the distance from x to y in $\mathbb{R}^n$.

See Appendix A.1 for a proof of Lemma 1. According to Lemma 1, if the given dataset is uniformly distributed on $\mathcal{M}$, any ball of fixed volume in $\mathcal{M}$ should contain the same number of data points. Conversely, let $B_k(x_i)$ be the ball in $\mathbb{R}^n$ around the data point x_i that contains its k nearest neighbors. Then for any data point x_i, the neighborhood on $\mathcal{M}$, $\varphi_n^{-1}\left(B_k(x_i)\right)$, should have the same volume. That is, the distances from x_i to its k nearest neighbors on $\mathcal{M}$ can be measured approximately by $\frac{1}{r_i} d_{\mathbb{R}^n}\left(x_i, x_{i_j}\right)$ for $1 \leq j \leq k$ if $r_i = d_{\mathbb{R}^n}\left(x_i, x_{i_k}\right)$ where x_{i_k} is the k^{th}

nearest neighbor of a given point x_i.

2.6. Finite extended-pseudo-metric space

A *finite extended-pseudo-metric space* (FEPMS) is defined as a metric space (X, d) where X is a set with finite elements, and the function $d: X \times X \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a metric on X such that for any $x, y, z \in X$, the following hold:

1. $d(x, y) \geq 0$, and $x = y$ implies $d(x, y) = 0$
2. $d(x, y) = d(y, x)$
3. $d(x, z) \leq d(x, y) + d(y, z)$ or $d(x, z) = \infty$

2.7. Reinforcement learning

Reinforcement learning (RL) is a subfield of machine learning that aims at training an *agent*, which is a learner and a decision-maker, to optimally behave in the system outside the agent, called *environment.* Once the agent applies an action to the environment, the status of the environment, referred to as a *state*, is changed and the environment feed-backs a numerical value, *reward*, to the agent (see Figure 2.2). The agent and environment repeatedly interact with each other for a sequence of discrete time steps, and the discounted sum of the rewards obtained during which is defined as a *return*. The goal of reinforcement learning can be described as to find an optimal *policy* function that maps the actions to given states [30].

More specifically, given a time step $t = 0,1, \dots, T$, the agent receives

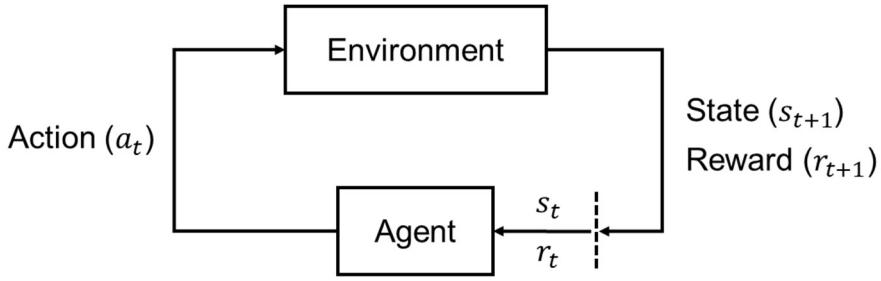

Figure 2.2 Schematic of reinforcement learning.

some representation of the environment's state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states, and selects an action $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of valid actions in state s_t. One time-step later, as a consequence of applying the action, the environment transits into a new state s_{t+1} and the agent receives a reward $r_{t+1} \in \mathcal{R}$: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Given a bounded action sequence with maximal step T and a trajectory of state/action pairs $\tau = \{(s_t, a_t)\}_{t=0}^{T}$, *return* R_t for $0 \leq t \leq T-1$ is defined as

$$R_t = r_{t+1} + \gamma\, r_{t+2} + \cdots = \sum_{k=0}^{t+k+1 \leq T} \gamma^k\, r_{t+k+1} \tag{2.12}$$

where $\gamma \in [0,1)$ is *discount factor*.

The policy function is given as either a deterministic policy $a = \pi(s)$ or a stochastic policy $\pi(a|s) = \mathbb{P}[a_t = a|s_t = s]$. As our interest is in developing a probabilistically behaving agent, we will use the word policy hereafter to denote the stochastic policy.

As the states or actions are uncountable in many cases, it is often hard to derive the exact solution of π^* with iterative refinement methods based on value iteration or policy iteration. In this case, we can approximate the optimal policy using parameterized functions such as neural networks, i.e., we can set $\pi_{\theta^*}(a|s) \cong \pi^*(a|s)$ and find θ^* by training.

With the same terminology, a *deterministic Markov decision process* (DMDP) is defined by a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \gamma >$. In DMDP, taking a given action a for a given state s always results in the same next state s'. In other words, choosing an action a is equivalent to choosing the reachable state s'.

2.8. Directed graph

A *directed graph* is a pair (V, E), where V is a finite non-empty set of *vertices* and E is a set of ordered pairs of distinct *edges*. For $e \in E$ where $e: v \to w$ and $v, w \in V$, we respectively call v and w *tail* and *head*. All the vertices and edges in a graph can be uniquely indexed using ordered numbers. That is, for an indexing function I, $I(v) \in \{1,2, \dots, |V|\}$ and $I(e) \in \{1,2, \dots, |E|\}$ for $v \in V$ and $e \in E$, where $|V|$ and $|E|$ represent the number of vertices and edges, respectively.

Chapter 3

Process monitoring and fault classification with probabilistic manifold learning[1]

3.1. Introduction

As modern industrial processes have become more complex and process safety and high-quality products are demanded as prerequisites, effective monitoring of chemical processes is receiving considerably increasing attention. Providing human-readable information is of great importance for pragmatic diagnostic systems because it enables plant operators and maintenance workforces to be better informed of the process status. Such information also provides rationales for the diagnosis results, which facilitates making better decisions in taking remedial actions to

[1] This chapter is an adapted version of D. Park, J. Na, and J. M. Lee, "Clustered Manifold Approximation and Projection for Semi-supervised Fault Diagnosis and Process Monitoring," *Ind. Eng. Chem. Res., vol. 60, no. 26, pp. 9521–9531, Jul. 2021.* and D. Park and J. M. Lee, "Robust Probabilistic Manifold Learning for Fault Diagnosis and Process Monitoring," *Comput. Chem. Eng.,* In preparation.

bring the plant back to its normal state. The process records usually include the information that indicates the operating conditions they have been sampled from, and these *a priori* information, or labels, can be incorporated to enhance the diagnosis performance [31]–[34]. However, labeling is costly and requires considerable expertise in many cases, and only a few labeled data are likely to be available.

Faced with this situation, many semi-supervised methods have recently been introduced that can utilize both labeled data and unlabeled data for training the models. Some of those are based on deep ladder networks [35], deep generative models [36], and multitask learning [37], which share a functional structure where a feature extractor and a classifier are serially connected. They utilize labeled data by imposing discriminant objectives to the classifier, thereby achieve superior performance such as low classification error or faster training speed compared to some competing methods. However, for the process monitoring tasks, they may be restrictively used or require subsidiary dimensional reduction processes because they only yield high-dimensional features. Furthermore, as their feature extractors are only trained subordinate to the classifier's learning process, only discriminative patterns (e.g., clusters) are likely to be produced without reflection of the transient behavior of the process.

In contrast, traditional statistical monitoring techniques, such as principal component analysis (PCA) and Fisher discriminant analysis (FDA), can distill visually informative features of the process data with a few principal dimensions. As their feature spaces are linearly projected space

of the input space, the process statuses, such as the degree of deviation from normal operating conditions, can be visually tracked based on the extracted features [38]. Therefore, they can provide interpretable grounds for the diagnosis results that are conducted based on the features. For this reason, several studies have been conducted to extend the existing PCA and FDA algorithms to be compatible with semi-supervised fault classification tasks [39], [40]. One common approach they take is to formulate an objective function weighted with separate goals for unlabeled data and labeled data. In this case, the visual characteristics of the extracted features can be controlled by the weight values.

Despite the appealing benefits and successful applications, the statistical process monitoring methods suffer from several drawbacks. First, it is difficult to comprehend the direct effects of providing additional label information and they can only be inferred by the projection results. Thus, choosing the weights usually requires a repetitive tuning procedure. Second, as they are linear methods, whose coordinates are given to maximally explain the variances of a given data set, a few data samples having wider distributions are likely to dominate the projection. This often incurs the problem of overlapping clusters (e.g. the normal samples and nearby incipient fault samples overlap in the feature space) [41], especially when multiple faults need to be taken into account. Such a problem not only hampers the discrimination by the human eye in monitoring the process but also deteriorates the discrimination performance of the classifiers that conduct fault classification based on the extracted features. The problem is alleviated in FDA because it promotes cluster separation

by maximizing the between-class variance. However, the local patterns (e.g., time trajectories) of the original data are liable to be lost after feature extraction because FDA minimizes the variances within classes at the same time. Moreover, they are known to be ineffective when the data manifold has nonlinear curvatures, which can often be found in industrial data. Although some kernel variants [42]–[44] are introduced that employ nonlinear kernel mappings in order to "unfold" such nonlinear data prior to the projection, the improvement is not always guaranteed, as is shown in the following verification study. Furthermore, designing kernel function generally requires careful inspection of data and an additional costly optimization procedure [45].

In order to address the issues mentioned above, we propose a manifold approximation-based feature extraction method, named probabilistic manifold learning (PML) and clustered manifold approximation and projection (CMAP). These two methods employ a probabilistic manifold approximation process before the projection. During this process, they draw the metric and non-metric attributes of the data and translates them into one common measure, which can be viewed as *likelihood,* thereby utilizes the labeled and unlabeled data simultaneously. The projection target of those methods is the clustered data manifold, not the raw data, and this prevents the aforementioned overcrowding problem. The nonlinear structures of the dataset are extracted by topology-preserving projection without costly nonlinear kernel mapping. The effectiveness of the proposed methods is demonstrated by the application to complex chemical process data, and the results are compared with those obtained from

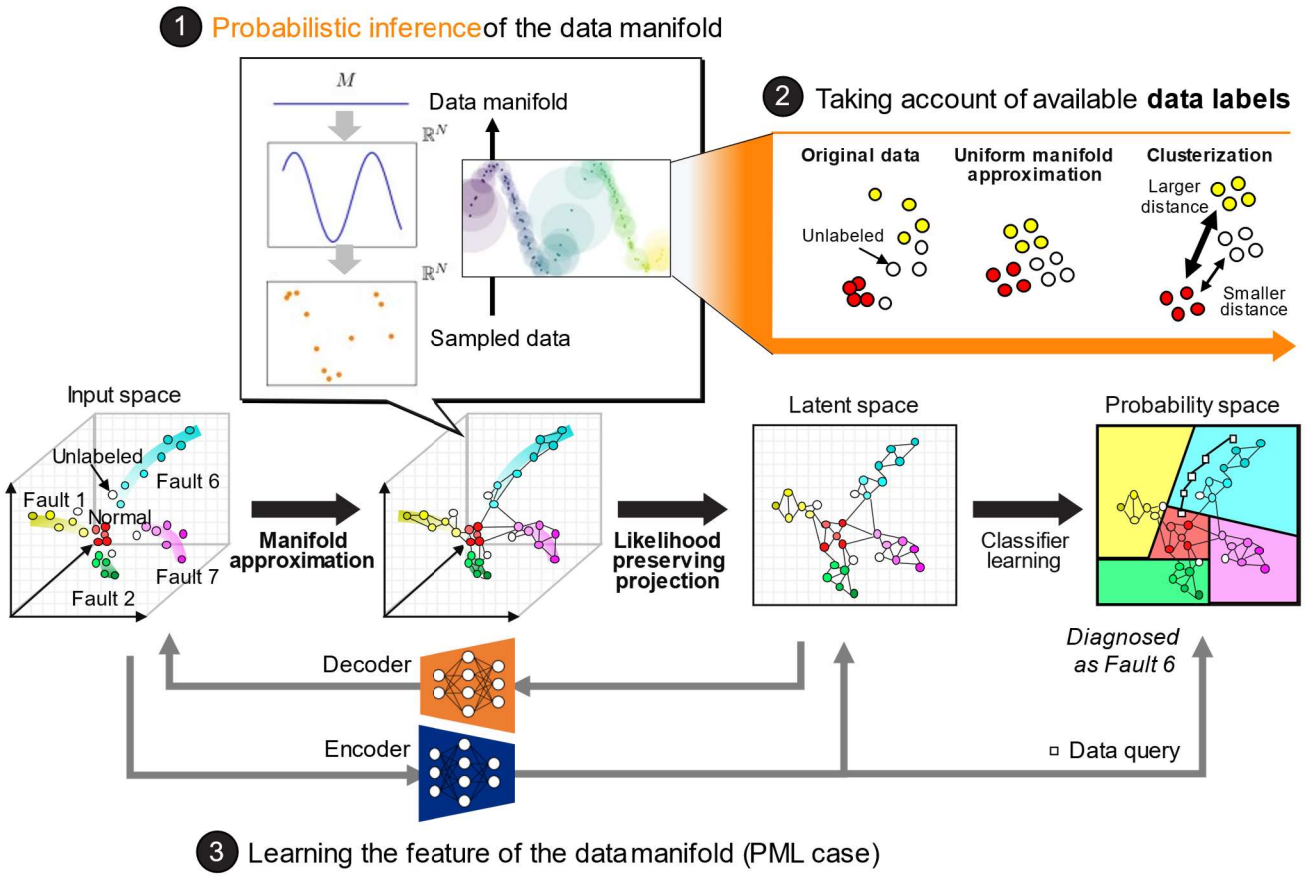

Figure 3.1 The schematic of the proposed fault diagnosis framework.

five competing methods.

The remainder of this chapter is organized as follows. Section 3.2 details the proposed methods and fault diagnosis framework. A verification study is performed in Section 3.3.

3.2. Methods

PML and CMAP is developed based on the framework of the Uniform Manifold Approximation and Projection (UMAP) in [29]. The core of UMAP is to approximate a manifold where the data points are uniformly distributed and to project its simplicial structure into low-dimensional space with minimal loss in structural properties. In this study, we consider the simplest case (1-simplices only), where the problem is reduced to the projection of pairwise distances. In this case, we can exploit the available labels to re-evaluate the distances between points. Motivated by this, in PML and CMAP, it is further assumed that the data manifold forms clusters by their labels. Specifically, a uniform data manifold is first approximated and repulsive displacement between the data points is applied according to their label interactions. Note that we adopt the algorithms of UMAP without modification, to examine the gain from incorporating partially labeled data in the process monitoring and fault diagnosis tasks. Instead, we provide concise descriptions for the uniform manifold approximation and projection algorithms of their simplest case in 3.2.1. and 3.2.3. For more details, one can refer to the original paper.

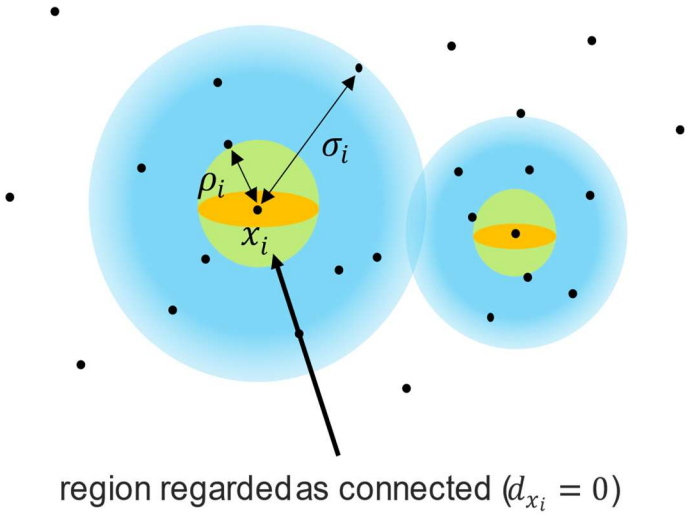

Figure 3.2 Visual interpretation of local FEPMS.

3.2.1. Uniform manifold approximation

For a given dataset $X = \{x_i\}_{i=1}^{N}$ with $x_i \in \mathbb{R}^n$, a local FEPMS $\left(X_i, d_{x_i}\right)$, which has properties described in Section 2.6, is defined at each data point x_i. Here, X_i is a set containing k nearest neighbors of x_i and the metric d_{x_i} is given by

$$d_{x_i}(x_i, x_j) = \max\left(0,\ \left(d_{\mathbb{R}^n}(x_i, x_j) - \rho_i\right)/\sigma_i\right) \tag{3.1}$$

for $x_j \in X_i$ where σ_i and ρ_i are specified to satisfy the following equations.

$$\rho_i = \min_j\left\{d_{\mathbb{R}^n}(x_i, x_j) \middle| d_{\mathbb{R}^n}(x_i, x_j) > 0\right\} \tag{3.2}$$

$$\log_2(k) = \sum_j \exp\left(-d_{x_i}(x_i, x_j)\right) \tag{3.3}$$

where $d_{\mathbb{R}^n}(x_i, x_j)$ denotes the Euclidean distance between the points x_i and x_j, ρ_i is the distance to the nearest neighbor, and σ_i is a scaler. Figure 3.2 illustrates the local FEPMS in the input space.

The parameter ρ_i ensures that x_i is locally connected to at least one of its nearest neighbors (i.e., the distance between them is 0) so that no point in X is isolated. Assuming the local FEPMS forms a uniform manifold, the scaler σ_i corresponds to the r in Section 2.5, and the distance d_{x_i} is the modified geodesic distance on the manifold. A simple experiment illustrated in Figure 3.3 demonstrates that defining local FEPMS makes the distances between the data points uniformly distributed regardless of the type and dimensionality of the datasets. The penicillin and yeast

fermentation process datasets can be found in [46] and [47].

For $x_i, x_j \in X$, the distance measured from x_i to x_j will be, in general, different from that from x_j to x_i as each data point defines its own FEPMS with different σ_i. To merge these incompatible local views, each of the local FEPMS is first translated into a fuzzy set $(\hat{X}_i, \mu_{j|i})$ where $\hat{X}_i = \{(x_i, x_j)\}$ and $\mu_{j|i}: \hat{X}_i \to [0,1]$ is a membership function that measures the likelihood between two points represented by the metric attributes of the data.

$$\mu_{j|i} = \begin{cases} \exp\left(-d_{x_i}(x_i, x_j)\right), & \text{if } x_j \in X_i \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

Then, a fuzzy union operation (probabilistic sum) is applied across the local fuzzy sets. The membership function values defined on $\hat{X} = \bigcup_i \hat{X}_i$ are given as

$$\mu_{ij}^{\mathcal{U}} = \mu_{j|i} + \mu_{i|j} - \mu_{j|i} \cdot \mu_{i|j} \tag{3.5}$$

Note that the resulting membership function values are symmetric. The fuzzy set $(\hat{X}, \mu_{ij}^{\mathcal{U}})$ descreibes the overall structure of the uniformly approximated manifold.

3.2.2. Clusterization

Let the labels of the point x_i by $y_i \in \{-1, 0, 1, 2\ldots\}$, where $y_i = -1$, $y_i = 0$, and $y_i = 1, 2, \ldots$ denote normal, unlabeled, and fault samples, respectively. We define a membership function for each element $(x_i, x_j) \in$

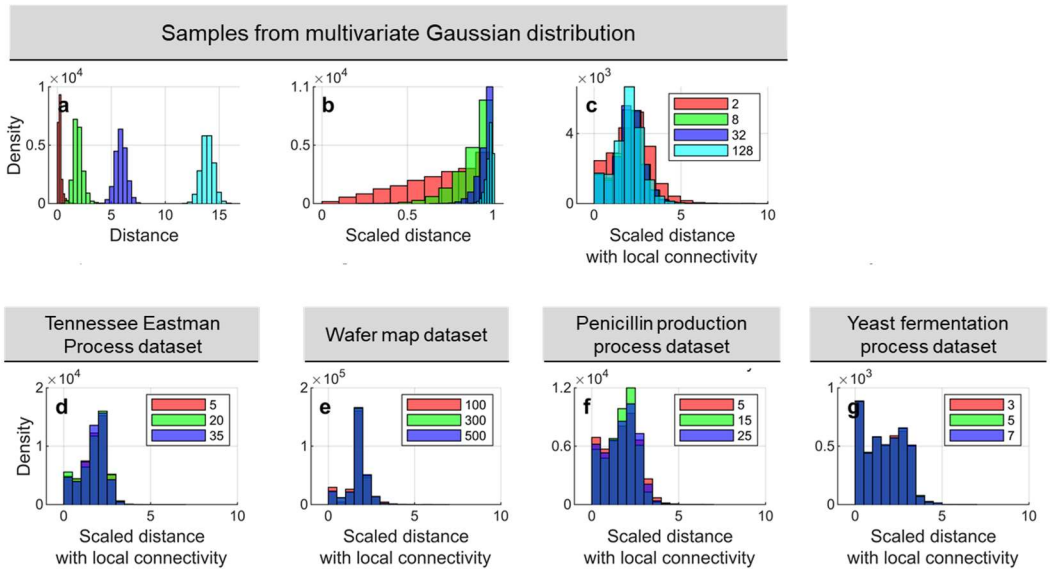

Figure 3.3 A thousand samples were drawn from a multivariate Gaussian distribution with a zero mean vector and a unit covariance matrix, varying the dimensions from 2 to 128. For each point, the distances to its 20 nearest neighbors are measured. (a) As the dimension of the space increases, the average distance increases exponentially. (b) Simple scaling renders all distances in the high dimensional spaces to be approximately the same. It implies that all data points are located far away from each other at almost the same degree. (c) If the local connectivity assumption is further applied, the distances are well distributed irrespective of the dimension of the space. Similar results can be obtained by (d) chemical process data used in this study, (e) WM-811k wafer image data, (f) penicillin production process dataset, and (g) yeast fermentation process dataset.

$\hat{X}$ that measures the likelihood induced by their labels by

$$\mu_{ij}^{\mathcal{L}} = \begin{cases} \exp(-d_u), & \text{if } y_i \neq y_j \text{ and } y_i + y_j < 0 \\ \exp(-d_v), & \text{if } y_i \neq y_j \text{ and } y_i \cdot y_j \neq 0 \\ 1, & \text{otherwise} \end{cases} \tag{3.6}$$

where $d_u, d_v > 0$ are hyperparameters that can be interpreted as virtual distance between the samples. Note that we let the unlabeled samples have less affinity with the normal samples than the fault samples. This is to treat those "indistinguishable" samples more like fault samples from a conservative point of view.

The clusterization operation is equivalent to intersecting two fuzzy sets, having membership function values of $\mu_{ij}^{\mathcal{U}}$ and $\mu_{ij}^{\mathcal{L}}$. In this study, probabilistic product is employed – that is dual to the probabilistic sum.

$$\mu_{ij} = \mu_{ij}^{\mathcal{U}} \cdot \mu_{ij}^{\mathcal{L}} \tag{3.7}$$

The resulting membership function values describe the overall affinity between two points. The distance between the two points x_i and x_j on the clustered data manifold $\mathcal{M}$, which embeds all the metric and non-metric (label) information, can be approximated by

$$d_{\mathcal{M}}(x_i, x_j) = -\log \mu_{ij} \tag{3.8}$$

The effect of the clusterization can be viewed as the repulsion between the labeled points (see Figure 3.1b). As the projection in Section 3.2.3 is performed to preserve all the pairwise distances, clusterization not only disgregates the data points manipulated by the repulsion operation ($\mu_{ij}^{\mathcal{L}} < 1$) but also migrates unmanipulated neighboring points ($\mu_{ij}^{\mathcal{L}} = 1$).

3.2.3. Projection

By letting $Z = \{z_i\}_{i=1}^{N}$ with $z_i \in \mathbb{R}^m$ be the projection of X, the corresponding fuzzy set $\hat{Z} = \{(z_i, z_j) | \, z_i, z_j \in Z\}$ and membership function $v_{ij}: \hat{Z} \to [0,1]$ can also be defined. In this case, the manifold for Z is $\mathbb{R}^m$ itself and its metric d is directly defined by the Euclidean distance on $\mathbb{R}^m$. To facilitate the upcoming optimization procedure, v_{ij} is given by a smooth differentiable function.

$$v_{ij} = \left(1 + a\delta_{ij}^{2b}\right)^{-1} \tag{3.9}$$

where $\delta_{ij} = \left\| z_i - z_j \right\|_2$ and a and b are hyperparameters that determine the dispersion of the layout.

The goal of the projection is to find the layout of Z that minimizes the difference between μ_{ij} and v_{ij}. To this end, a loss function is defined by binary cross-entropy between $\hat{X}$ and $\hat{Z}$.

$$H(\hat{X}, \hat{Z}) = \sum_{i \neq j} \left[\mu_{ij} \log\left(\frac{\mu_{ij}}{v_{ij}}\right) + (1 - \mu_{ij}) \log\left(\frac{1 - \mu_{ij}}{1 - v_{ij}}\right) \right] \tag{3.10}$$

The minimization of the loss function forces nearby (distant) points on the manifold to be placed nearby (distant) in the reduced space. Z is first initialized through spectral embedding [48] and optimized by iterative stochastic gradient descent procedure. Dropping constant terms and taking partial derivatives with respect to z_i, the gradient of the loss function is given by

$$\frac{\delta H}{\delta z_i} = \sum_{i \neq j} \left[\mu_{ij} \cdot \frac{2ab\delta_{ij}^{2(b-1)}}{1 + a\delta_{ij}^{2b}} + (1 - \mu_{ij}) \cdot \frac{-2b}{(\epsilon + \delta_{ij}^2)(1 + a\delta_{ij}^{2b})} \right] (z_i - z_j) \tag{3.11}$$

where $\varepsilon = 10^{-3}$ is added to prevent division by zero.

The optimization process has been simplified and accelerated in two ways. In CMAP, we utilize edge sampling [49] and negative sampling [50] techniques on Equation (3.11) as in UMAP. For PML, we construct a deep Autoencoder [51] with a reconstruction loss. Given the encoder and decoder networks $\mathrm{Enc}(x)$ and $\mathrm{Dec}(z)$ respectively, the reconstruction loss is formulated as

$$\mathcal{L}_{\mathrm{recon}} = \sum_i \left(x_i - \mathrm{Dec}\big(\mathrm{Enc}(x_i)\big) \right)^2 \tag{3.12}$$

The loss functions for PML and CMAP are given as

$$\mathcal{L}_{\mathrm{PML}} = H + \mathcal{L}_{\mathrm{recon}} \tag{3.13}$$

$$\mathcal{L}_{\mathrm{CMAP}} = H \tag{3.14}$$

respectively.

3.2.4. Mapping of unknown data query

The low-dimensional layout $Z' = \{z_i'\}_{i=1}^M$ for unlabeled data query $X' = \{x_i'\}_{i=1}^M$ is determined by the regression based on k-nearest neighbors. For each query point $x_i' \in X'$, its k nearest neighbors, denoted by X_i', are drawn from X. The referential points for $z_i' \in Z'$ are given by

$Z_i' = \pi(X_i')$, where π: $X \to Z$ is the map yielded from the projection in 3.2.3. Then for $x_j' \in X_i'$ and $z_j' \in Z_i'$, following local membership functions are defined.

$$\mu_{ij}' = \exp\left(-d_{x_i}\left(x_i', x_j'\right)\right) \tag{3.15}$$

$$\nu_{ij}' = \left(1 + a\left(\delta_{ij}'\right)^{2b}\right)^{-1} \tag{3.16}$$

where $\delta_{ij}' = \left\|z_i' - z_j'\right\|_2$. The optimal layout of Z' is determined by minimizing cross-entropy between μ_{ij}' and ν_{ij}' following the optimization procedure in 3.2.3.

3.2.5. Inference

To demonstrate that the faults can be visually discriminated in the DML's feature space, we employ the Gaussian mixture model (GMM), an unsupervised classifier whose classification performance is highly dependent on the distribution of the data. Note that the input data for the classifier are the extracted features, as illustrated in Figure 3.1.

The GMM is a superposition of Gaussian components whose probability density function is given by the weighted sum of Gaussian density functions.

$$p\left(z|\hat{\mu}, \hat{\Sigma}\right) = \sum_{c=1}^{C} \pi_c \mathcal{N}\left(z\middle|\hat{\mu}_c, \hat{\Sigma}_c\right) \tag{3.17}$$

where z is a data point in the feature space $\mathbb{R}^m$, C is the number of Gaussian components, and π_c is the prior probability of selecting the cth Gaussian component.

Each Gaussian density $\mathcal{N}\left(z\middle|\hat{\mu}_c,\hat{\Sigma}_c\right)$ measures the probability of sample z conditioned on c, with its own mean $\hat{\mu}_c \in \mathbb{R}^m$ and covariance $\hat{\Sigma}_c \in \mathbb{R}^{m\times m}$. Once trained, the GMM allows inference on unknown data queries. The posterior probability of the cth component is given as

$$p(c|z) = \frac{p(c)p(z|c)}{\sum_i p(i)p(z|i)} = \frac{\pi_c \mathcal{N}\left(z\middle|\hat{\mu}_c,\hat{\Sigma}_c\right)}{\sum_i \pi_c \mathcal{N}\left(z\middle|\hat{\mu}_i,\hat{\Sigma}_i\right)} \tag{3.18}$$

Another frequently used classification measure is the Mahalanobis distance [27], which is a unitless and scale-invariant measure of the distance between a point and a distribution. The squared Mahalanobis distance d_M from the point z to the cth Gaussian component is defined as

$$d_M^2(z|z \in c) = (z-\hat{\mu}_c)^T \hat{\Sigma}_c^{-1}(z-\hat{\mu}_c) \sim \chi_m^2 \tag{3.19}$$

where χ_m^2 is the Chi-square distribution with m degrees of freedom. The confidence bound can then be defined in terms of the Mahalanobis distance to each Gaussian component that follows the χ_m^2 distribution. The confidence bound of $100\alpha\%$ $(0 < \alpha < 1)$ defines a region that encompasses $100\alpha\%$ of the given data as the sample size tends to infinity.

$$d_{M,\alpha} = F^{-1}(\alpha|m) = \{z: F(z|m) = \alpha\} \tag{3.20}$$

where $F(z|m) = \int_0^z \frac{t^{(m-2)/2}e^{-t/2}}{2^{m/2}\,\Gamma(m/2)}\,dt$ is the cumulative distribution function of the Chi-square distribution.

Note that the parameters of the Gaussian components are only tuned to describe the entire distribution of the data. Thus, we can expect high classification performance only if each Gaussian component can exclu-

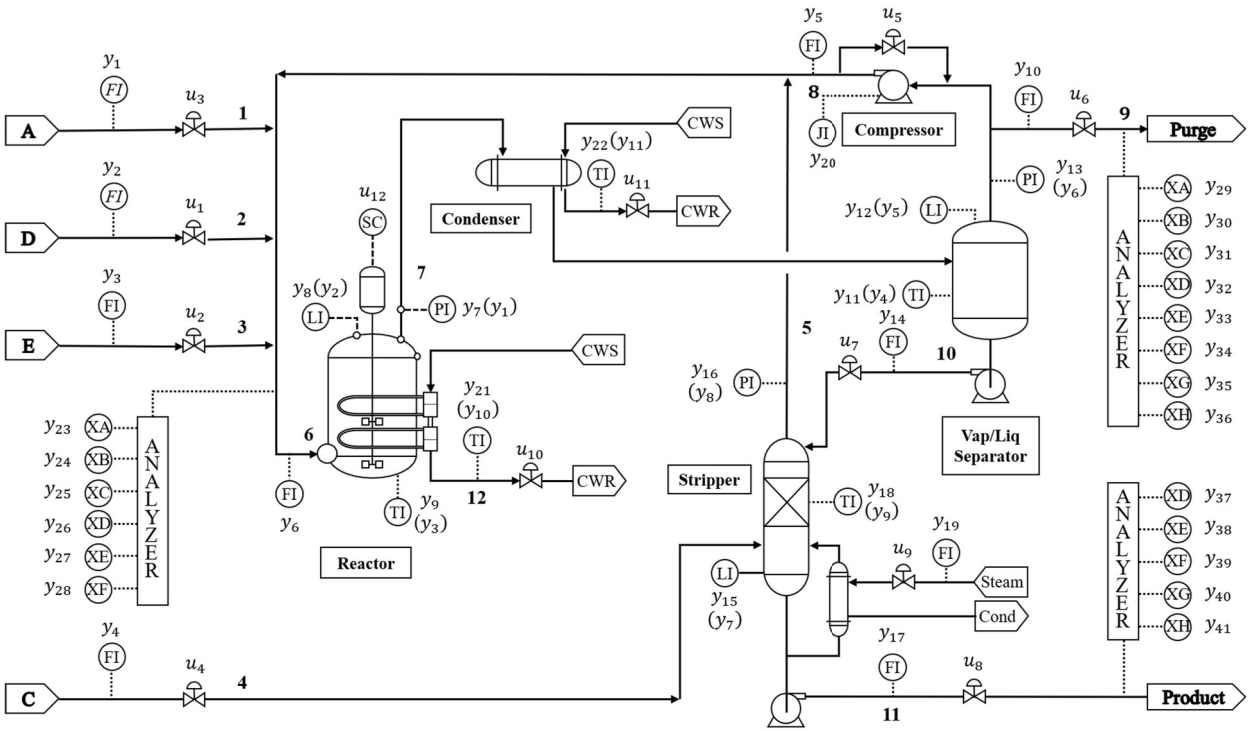

Figure 3.4 Process flow diagram of Tennessee Eastman Process.

Table 3.1 Fault scenarios used in dataset generation.

Name	Process Disturbances	Type
Fault 1	A/C feed ratio, B composition constant (stream 4)	Step
Fault 2	B composition, A/C ratio constant (stream 4)	Step
Fault 6	A feed loss (stream 1)	Step
Fault 7	C header pressure loss (stream 4)	Step

Table 3.2 Simulation configuration used in training and test dataset generation.

Dataset type	Operation scenario	Sampling period	# of sampled points per simulation	Random seeds used
Training	Normal	19 – 24 h	500	[1, …, 10]
	Fault 1	24 – 29 h		[11, …, 20]
	Fault 2			[21, …, 30]
	Fault 6			[31, …, 40]
	Fault 7			[41, …, 50]
Test	Normal	19 – 29 h	1000	[51, …, 60]
	Fault 1			[61, …, 70]
	Fault 2			[71, …, 80]
	Fault 6			[81, …, 90]
	Fault 7			[91, …, 100]

sively capture the distribution of each class. Such a case holds if the input data are clustered by their labels and the number of components is given by the number of data classes. In this study, we satisfy the second condition so that the classification performance solely depends on the clusterization degree of the features.

3.3. Verification study

3.3.1. Dataset description

Tennessee Eastman process (TEP) produces two products and one by-product from four reactants, and consists of five major operation units: reactor, partial condenser, vapor/liquid separator, product stripper, and recycle compressor. The schematic of the process is illustrated in Figure 3.4. We ran the process by operation mode 1 with a closed-loop control system [52], [53].

A total of 53 process variables, composed of 12 manipulated variables and 41 measured variables, were sampled every 0.01 h. Four process fault scenarios listed in Table 3.1 were used to generate the datasets [54], [55]. We ran the process a complete 72 h, where the process was initially run for the first 24 h under normal operating conditions and then for 48 h with faults. The training and test datasets were constructed as stated in Table 3.2. The distribution of the data is represented in Figure 3.5. To take account of the stochastic behavior of actual plants, we applied small random variations to the reaction kinetics coefficients during the simulation. We used different random seeds for each simulation and repeated

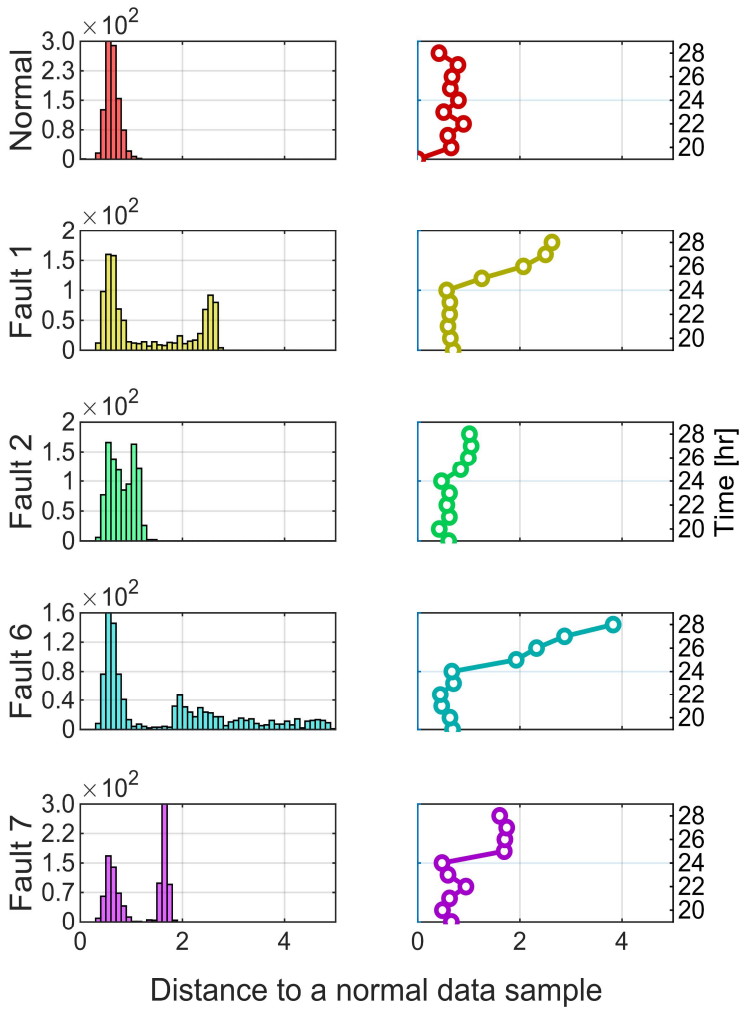

Figure 3.5 Distribution of the data in the input space (left), and their trajectories over time (right).

simulations 10 times for each operation scenario to assess the reliability of the results. Each feature of the datasets was scaled to [0, 1] with min-max normalization. All the points are labeled while the first 10% of the fault samples are unlabeled. This is because the time when the fault occurred is unknown, and it is often hard to distinguish such incipient fault samples from those of normal or fault in practice.

3.3.2. Experimental setup

To highlight the effectiveness of PML and CMAP, we compared those to UMAP and seven methods described in Table 3.3. For unsupervised methods, the dataset is provided without labels, whereas, for supervised methods, the dataset is provided by labeling the unlabeled points.

The reduction dimension has been set to $n = 2$. For PML, CMAP, and UMAP, $k = 50$, $d_u = 2$, $d_v = 5$, $a = 1.58$, and $b = 0.9$ were used. The number of epochs for the iterative optimization in the projection was set to 500, which was enough number for the layout to converge in our experiments. The autoencoder architecture used in PML is illustrated in Figure 3.6. For KPCA and KFDA, the Gaussian kernel, which has been reported to be suitable for nonlinear process monitoring [56], was used for a kernel function. The kernel coefficient is given *a priori* by the optimal value [45] derived for TEP.

To encourage each Gaussian component to capture the distribution of data corresponding to each operating condition, we initialize each $\hat{\mu}_c$ as the average coordinate value of each operating condition data in the

Table 3.3 Dimension reduction methods used in the benchmark.

Abbrevi-ation	Full name	Type	Mapping	Training method
PML	Probabilistic manifold learning	Manifold Approxi-mation	Nonlinear	Semi-supervised
CMAP	Clustered manifold approximation and projection			
UMAP	Uniform manifold approximation and projection [29]			Unsupervised
PTSNE	Parametric t-distributed stochastic neighbor embedding [57]	Graph embedding		
LLE	Locally linear embedding [58]			
ISOMAP	Isometric feature mapping [59]			
CVAE	Conditional variational autoencoder [60]	Feature learning	Nonlinear	Semi-supervised
VAE	Variational autoencoder [61]			Unsupervised
AE	Autoencoder [51]			
KFDA	Kernel Fisher discriminant analysis [62]	Statistical projection	Nonlinear	Supervised
FDA	Fisher discriminant analysis [6]		Linear	
KPCA	Kernel principal component analysis [63]		Nonlinear	Unsupervised
PCA	Principal component analysis [3], [4]		Linear	

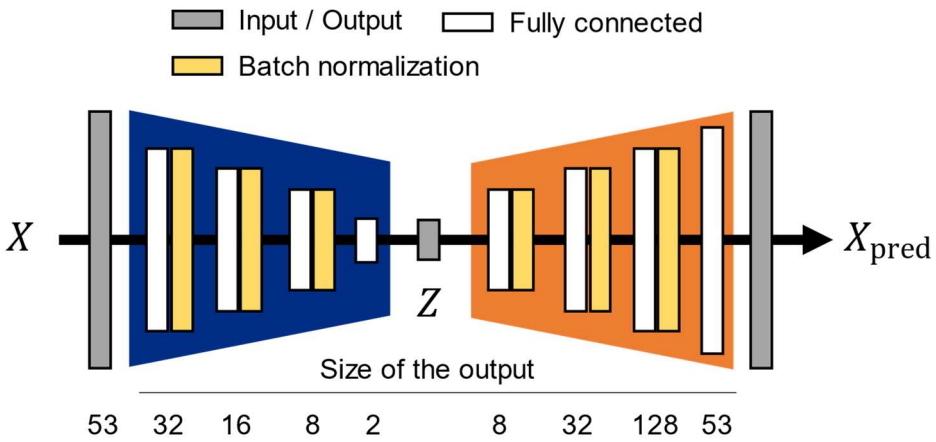

Figure 3.6 Autoencoder architecture used for parametric methods.

feature space. The covariance matrix $\hat{\Sigma}_c$ is initialized with the identity matrix of size m, and the number of iterations is set to 1000, which was a sufficient number for the GMM to converge in all the experiments conducted in this study. To ensure that the estimated covariance matrices are positive definite, we regularized the covariance matrices to have at least a small value $\epsilon = 2 \times 10^{-2} \cdot \mathrm{tr}(\Sigma)/m$, where Σ is the covariance matrix of the data in the feature space. The parameters of the GMM are found by the expectation-maximization (EM) algorithm [2].

3.3.3. Process monitoring

Figure 3.7 shows the projection results of the training dataset for the methods compared. Note that Fault 6 has the widest distribution in the input space (see Figure 3.5) and it dominates the projection direction of PCA and FDA-based methods. This is because these methods find a low-dimensional representation that maximizes the variance of data (PCA) or the distances between the classes with their respective mean values (FDA). Hence, the normal and Fault 2 data samples, which lie in close proximity to each other in the input space, are overlapped in the feature space, making it hard to distinguish them. Note also that using nonlinear kernel mapping does not improve the results. In contrast, the extracted features clearly separated by clusters in PML and CMAP, providing high resolution for monitoring the processes. As similar results can also be found in UMAP, we can deduce that data such characteristic is largely attributed to the uniform manifold approximation and topology-preserv-

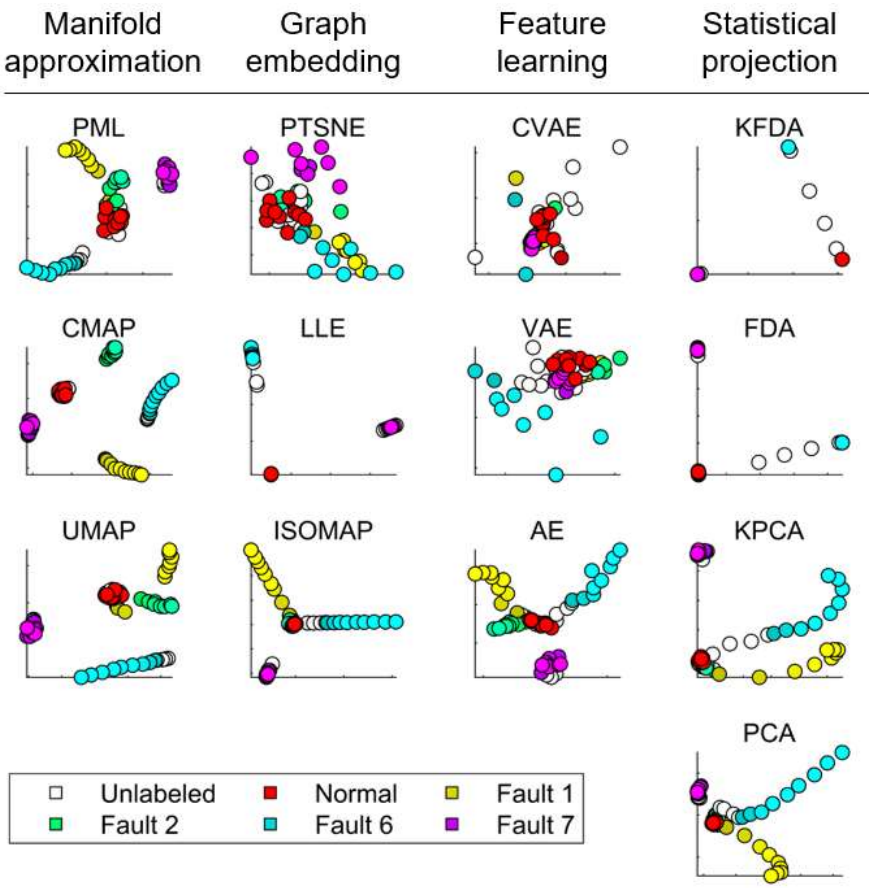

Figure 3.7 Time trajectories (from dark to light) of the training dataset in the feature space.

ing projection operations.

In terms of effectiveness in utilizing label information, the proposed methods are found to excel the conventional FDA or KFDA-based methods. The projection results of FDA and KFDA are confined to preserving the points that are located far away or very close. This is because, aside from the fact that they are linear methods, their objectives – maximize the ratio of between-class variance to within-class variance – can only either shrink or set apart the points. This results in losing the mid-scaled distances that involve the data trajectories. In the proposed methods, the label information is exploited in advance of the projection process and only by imposing repulsions on "calibrated" points – note that all the local distances were rendered to the range [0, 5] in the uniform manifold approximation process in advance of clusterization (see Figure 3.3c and Figure 3.3d). This enables controlling two competing objectives in label utilization, which are to preserve the features of the metric data such as data trajectory and to induce discriminative features using non-metric information (labels). For example, giving large values on d_u and d_v can foster clustering, resulting in a higher separation degree between the clusters. The values can be appropriately chosen by a user within the range [0, 5].

By comparing the feature space of CMAP with UMAP (see Figure 3.8), we can notice that clusterization operation is effective for preserving the trajectories and partitioning the feature space with respect to the clusters. Such property is a crucial aspect for visually discerning faults in the feature space and tracing their progress. A direct measure to

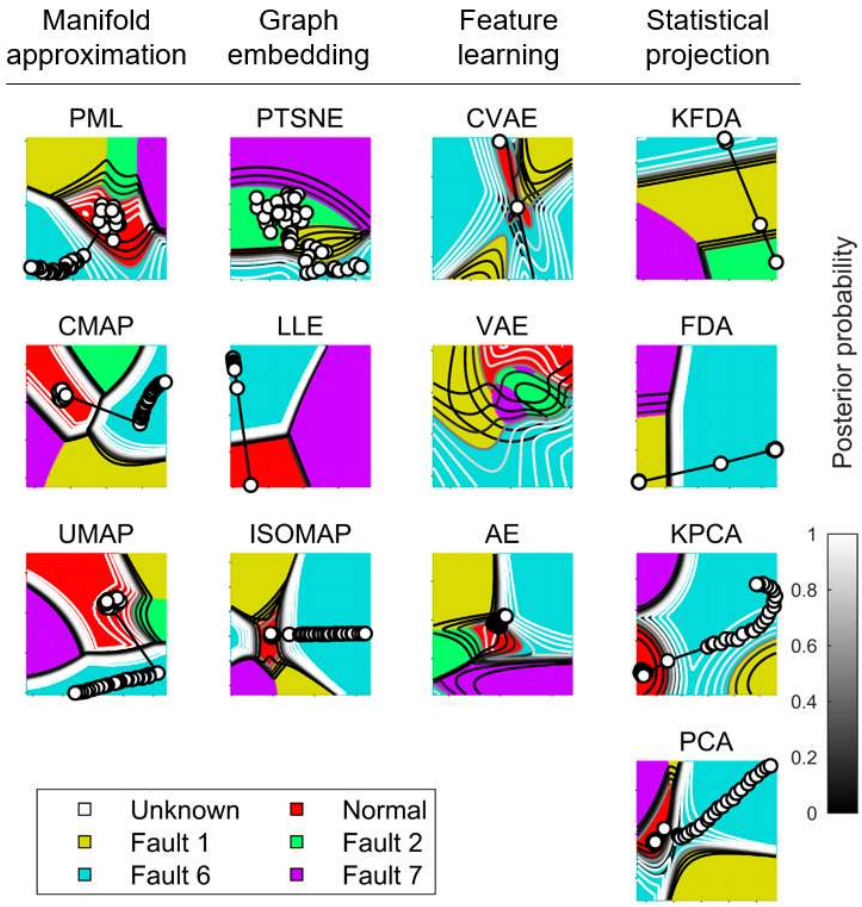

Figure 3.8 Trajectories of Fault 2 and Fault 6 occurrence scenarios in the feature spaces. The colored region represents the class that gives the maximum posterior probability.

analyze such difference is to compare their projection targets (i.e., the data distributions on their manifolds). However, as the manifolds are only described by the pairwise local distances, we can infer such, as a workaround, by inspecting the changes in pairwise distances. Figure 3.9 shows the distribution of the pairwise distances in the input space and their adjustment made in each method. We can notice that especially larger separation is made for closely located data points (having distances of 0.5 – 1.5) in CMAP. Therefore, we can infer that the clusterization operation is an effective measure for disgregating this "overcrowded" region.

3.3.4. Projection characteristics

The projection characteristics of the proposed methods can be mined further with the pairwise distance plot. Here, the Pearson correlation coefficient (PCC) values represent the preservation degree of the (metric) data structure. We can notice that PCA projects nearby (faraway) points in the input space to nearby (faraway) points into the feature space and this leads to a larger PCC value. The "ascending" trend in PCA can also be found in PML and CMAP, which indicates that they preserve the global structure of the data. The diverging trajectories of Fault 1 and Fault 6 are shown in PML and CMAP in proper orientations (see Figure 3.5 and Figure 3.7). This implies that the global behaviors of the data, such as the deviation degree from the normal operating condition, are trackable in the feature spaces of the proposed methods. Note that this

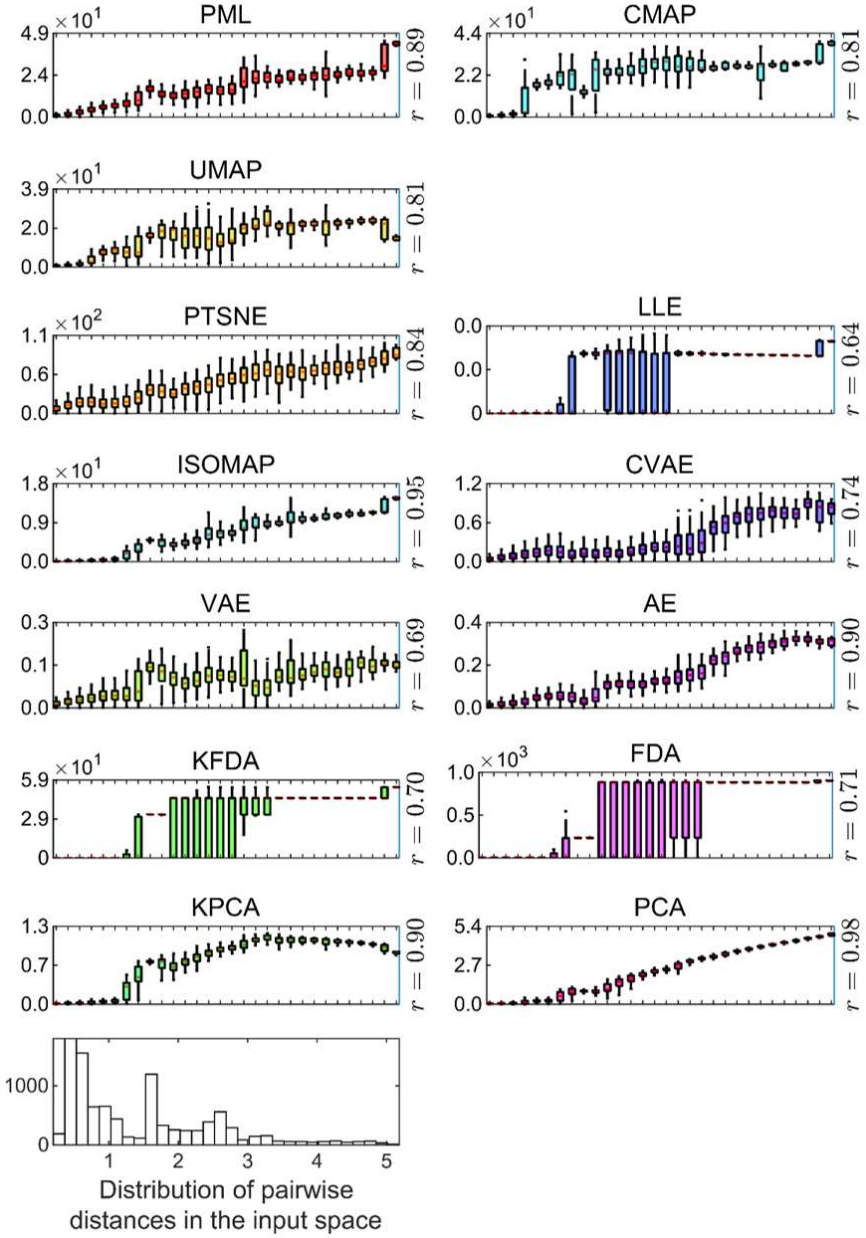

Figure 3.9 Preservation of pairwise distances in the embeddings. The last row represents the distribution of the pairwise distances in the input space. The Pearson correlation coefficient value for the pairwise distances between the input space and feature space is denoted by r.

Table 3.4 Evaluation metric values. The bold entries denote the best and second most scored.

	DI	**DBI**	**SE**	**LCMC**
Range	$[0, \infty)$	$[0, 1]$	$[0, \infty]$	$[0, 1]$
Best	∞	0	0	1
PML	0.17	**1.07**	37.0	**0.29**
CMAP	**0.39**	**1.12**	91.0	**0.29**
UMAP	0.29	1.15	37.5	0.28
PTSNE	4.6×10^{-3}	20	381	0.2
LLE	8.1×10^{-16}	1.9×10^{12}	0.99	0.13
ISOMAP	1.8×10^{-11}	8.3×10^{8}	2.3	0.21
CVAE	2.5×10^{-3}	220	0.80	0.08
VAE	3.8×10^{-3}	259	0.92	0.14
AE	**1.25**	1.9	0.92	0.11
KFDA	5.7×10^{-16}	1.3×10^{12}	143	0.06
FDA	5.3×10^{-13}	3.5×10^{9}	3.5×10^{4}	0.04
KPCA	2.0×10^{-6}	8.2×10^{3}	**0.55**	0.20
PCA	1.9×10^{-6}	9.0×10^{3}	**0.23**	0.20

trend is not found in UMAP and all the distances are adjusted to have almost the same length (see Figure 3.9).

To quantify the visualization performance in the process monitoring tasks, we introduced four evaluation metrics. We used Dunn index (DI) and Davies-Bouldin index (DBI) [64] to evaluate how well the data clusters were separated, and Sammon's error (SE) and the local continuity meta-criterion (LCMC) [65] to measure how well the structures of the data have been preserved during the feature extraction. whereas LCMC measures the preservation degree of the local structures only. See Performance Indices in the Supporting Information to refer to their equations. The evaluation results are summarized in Table 3.4. It is shown that CMAP performs the best in all the visualization properties of interest, except structure preservation (SE), and PML ranked second in overall performance. From the large whiskers (large deformation) in the pairwise distance plot (see Figure 3.9), we can infer that its small SE value has been incurred from the recast made in the manifold approximation. Compared to UMAP, CMAP has a larger SE value, implying that clusterization inhibits data structure preservation. But at the same time, it is found that this improves all the other visualization properties, including local structure preservation.

3.3.5. Fault diagnosis

We conduct two fault diagnosis tasks in this study: fault detection and fault classification. In fault detection tasks, the process is diagnosed as

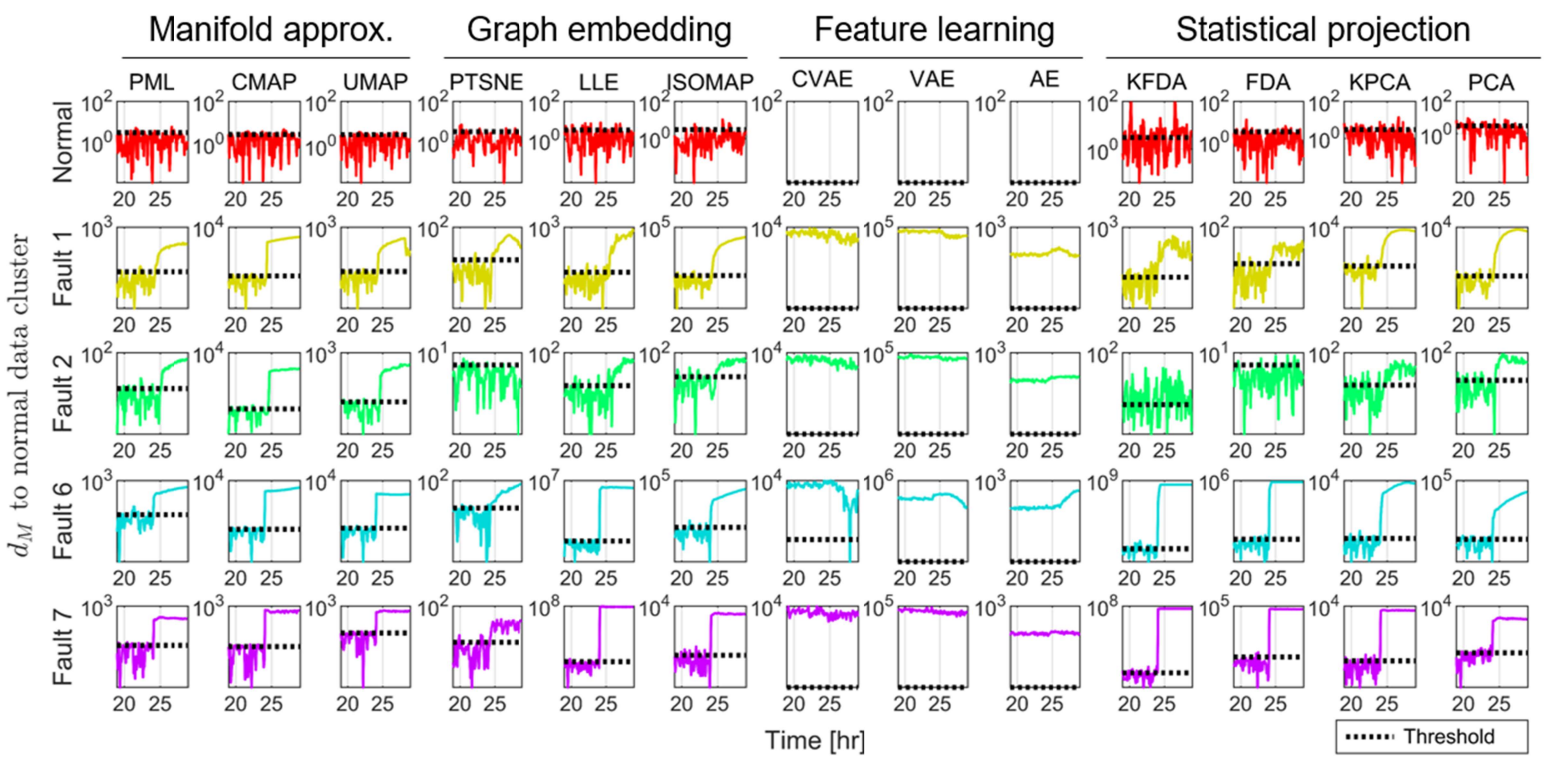

Figure 3.10 Process monitoring results on fault detection tasks. The y-axis is represented in the logarithmic scale.

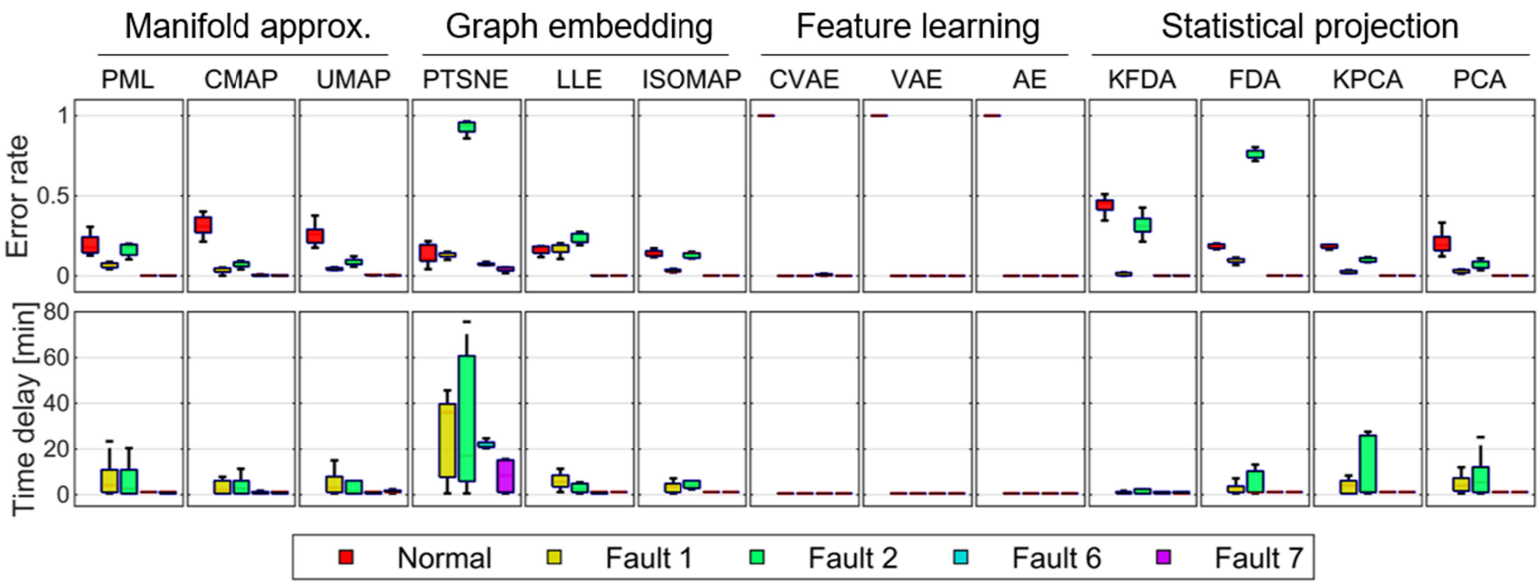

Figure 3.11 Detection error rates and the time taken to detect a fault after its occurrence.

deviating from normal operating conditions when the Mahalanobis distance from a queried data point to the normal operation cluster is larger than the confidence bound in (3.20). The threshold for the fault alarm is set by the 99% confidence bound of the normal operation data. Two types of errors are taken into account: false alarm, where a process under normal operation is diagnosed as a fault, and missing alarm, where a process under faulty operation is diagnosed as normal. The error rate is defined as the number of misdiagnosed points divided by the number of total data points.

Figure 3.10 and Figure 3.11 summarize the fault detection results on the test dataset. Considering both the error rate and detection time, we can notice that CMAP shows the most robust online monitoring performance and superior detection performance. However, CMAP shows a relatively larger false alarm rate compared to UMAP. This is because its mapping amplifies the deviation from the normal operating condition; a small deviation from the normal operating condition is mapped far away from the normal region in the feature space. Since the detection tasks have been performed based on the distance to the normal cluster, CMAP is bound to have more chance to occur false alarms. But conversely, it has a lower chance to miss the faults.

In fault classification tasks, the process status is identified with the class for which the GMM gives the maximum posterior probability Figure 3.12 and Figure 3.13 summarize the fault classification results on the test dataset. CMAP shows the lowest misclassification rates and time delay in classifying the faults, especially for the *hard-to-classify* faults:

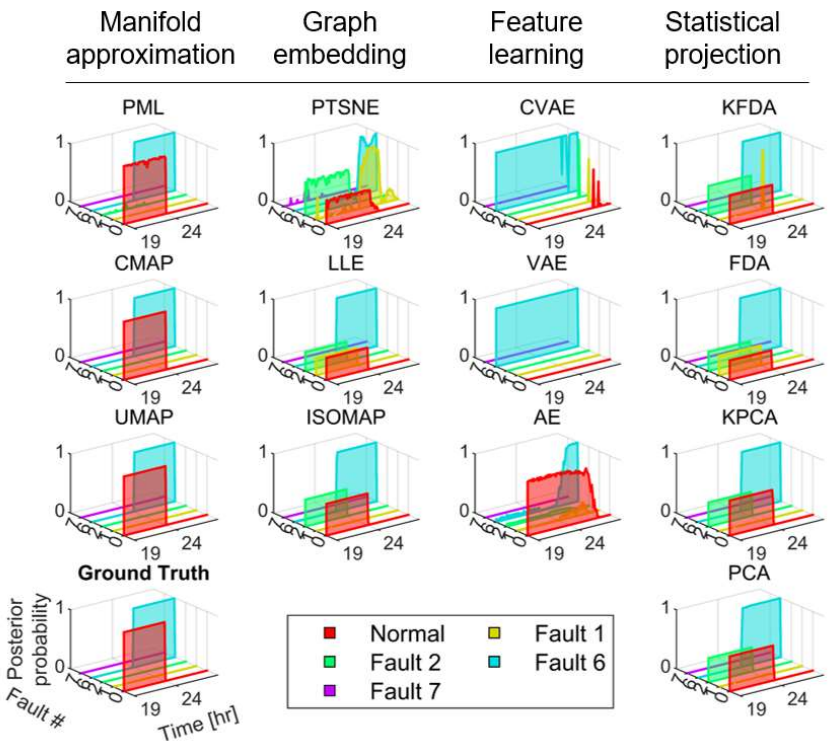

Figure 3.12 Process monitoring and fault classification results for Fault 6 occurrence scenario.

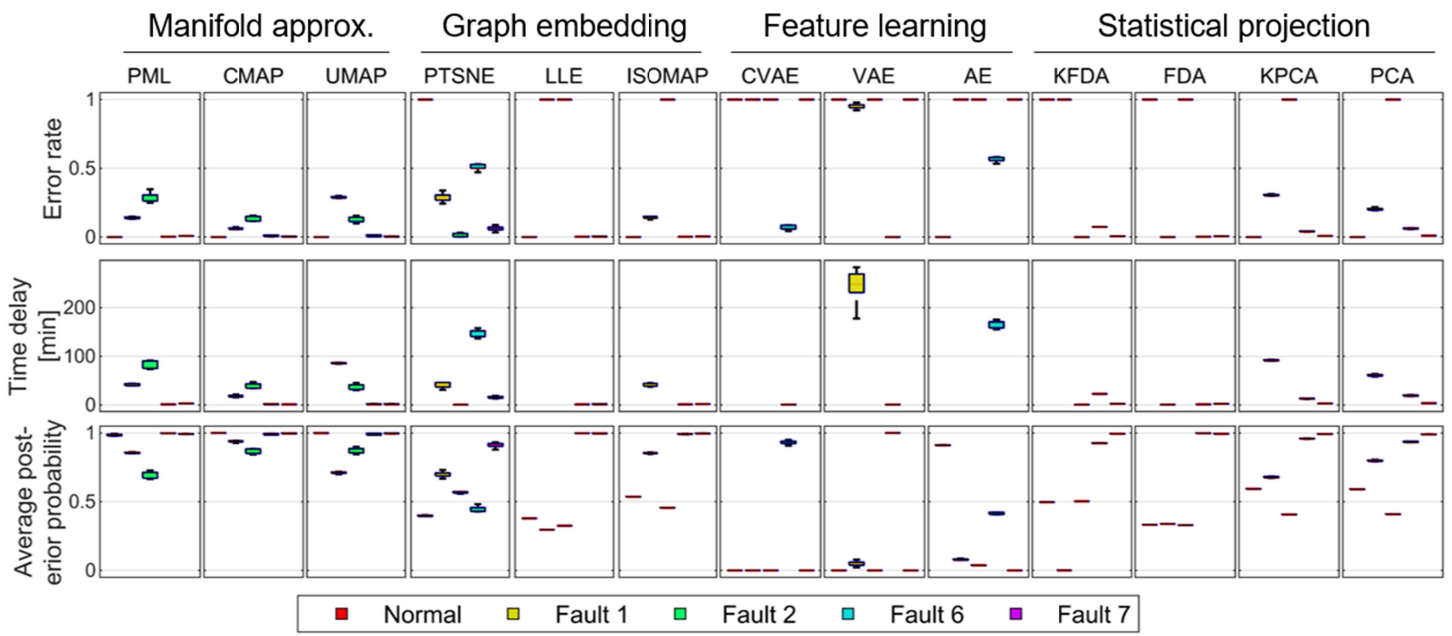

Figure 3.13 Misclassification rates, the time delay in classifying the faults, and the average posterior probabilities to the ground truth faults.

Fault 1 and Fault 2. See Figure 3.8 that the clusterization characteristics of CMAP in feature extraction facilitates GMM's training process.

CMAP has less data dependency as it recasts data to a particular shape (clustered manifold) first (see Figure 3.3) and sets it as its projection target. From a functional point of view, this manifold approximation process can be viewed as a nonlinear mapping in conventional monitoring techniques. In contrast, applying kernel function – one popular approach to handle nonlinear data – is shown to have only a random effect on the diagnosis performance of the classifier. See Figure 3.7 that the dataset has not been linearized in KFDA and KPCA after the projection.

Note that we use posterior probability in fault classification tasks to diagnose the process and, unlike the fault detection case, CMAP yieldsrobust performance in classifying normal operating conditions (no false alarms). It suggests that better diagnostic results can be attained by using tailored classifiers and classification approaches.

3.3.6. Computational Aspects

To assess the computational aspects of the proposed methods for online implementation, we evaluated their CPU runtimes with different data sizes and compared with the other methods. The benchmark was performed in Python 3.8 with a 4.3GHz Intel i9-7900X CPU, NVIDIA GeForce GTX 1080Ti GPU, and 64 GB RAM. Only a single CPU core was utilized, and the time limit was set by 50 min. The results are summarized in Figure 3.14.

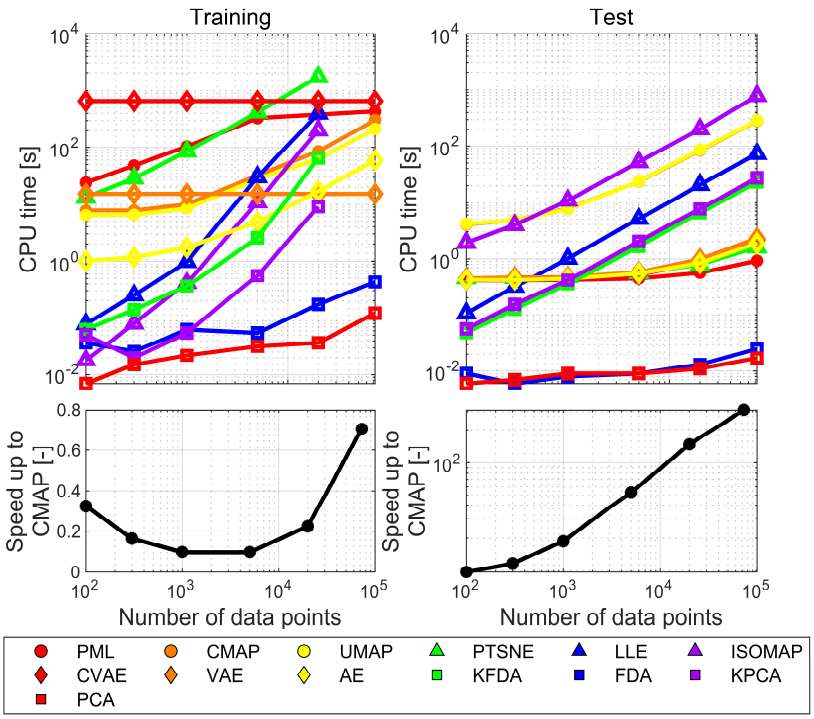

Figure 3.14 CPU time cost for (a) training and (b) test datasets and the speed-up ratio in PML against CMAP.

From an implementation point of view, a model desired is one that can process a large quantity of data during the training phase. In this case, the time cost required to train the model can be ignored. In terms of scalability to data size, the proposed methods are shown to work when 72,900 data points are fetched (see Figure 3.14a). Note that kernel-based conventional nonlinear methods, namely KFDA and KPCA, suffer an out-of-memory error in this case. This is because they construct a kernel matrix, which costs space complexity $O(N^2)$, whereas the manifold approximation-based methods handle the nonlinearity of data via sparse distance matrices, which cost less space complexity. It is observed that CMAP and UMAP show linear time complexity of $O(N^{1.02})$, and PML costs a bit more computation time. This is promising because conventional statistical projection methods have time complexities of $O(N^3)$ and have been reported to be impractical to use with large amounts of data [29].

Contrary to the training phase, in the test phase, the time cost to execute projection becomes a major concern. The number of data to be projected in each time step can be assumed to be relatively small. Figure 3.14b illustrates that the strategy of learning the mapping between the input and feature spaces in PML can effectively reduce the time cost in the order of magnitudes compared to CMAP.

Chapter 4

Process system modeling with Bayesian neural networks[2]

4.1. Introduction

The trends in microelectronics manufacturing have been toward extreme miniaturization of device geometries and higher transistor density. Plasma etching is one of the key processes in semiconductor manufacturing using for fabricating finer patterns. In the plasma etching, the process first applies electric power to the electrodes in a vacuum reactor, disassociating the etchant gas into electrodes, ions, photons, and radicals. Electromagnetic fields are then applied, transferring kinetic energy to the charged particles toward the wafer, and the wafer surface is etched via physicochemical reactions [66], [67].

[2] This chapter is an adapted version of D. Park, S. Ryu, G.-H. Kim, and J. M. Lee, "Sparse Bayesian Long Short-Term Memory Networks for Computationally Efficient Modeling of Stochastic Plasma Etch Processes," *Comput. Chem. Eng.*, In preparation.

The main goal of the process is to achieve a desired etch profile that is described by the performance metrics including etch rate, uniformity, selectivity, and anisotropy. As the etch profile is resulted from the time accumulation of the plasma condition [68]–[70], many studies have recently been exerted on monitoring [71], [72] and controlling the etching process [73]–[76] in real-time.

In control system design and verification, the most crucial component is the accuracy of the system model because the predictions obtained from the model are used in deriving optimal control actions [77], [78]. Followings are the most crucial characteristics considered for modeling the plasma etch processes:

- *Nonlinearity* occurred from the physicochemical reactions of the plasma,
- *Hybrid dynamics* occurred from discrete control logics of the etching equipment,
- *Process drifts* incurred from deposition/desorption of chemical species to the reactor wall [79]–[81],
- *Time delay* from the control loop, and
- *Stochastic behaviors* (e.g., noise) of the system.

There have been a number of studies to simulate these system characteristics by formulating physical equations and conducting computationally intensive numerical simulations [82]. Some of these works have succeeded in estimating the distribution of chemical species in the reactor and deriving the strategies to enhance etch profile [83], as well as have

investigated the effect of the wall condition on the etch profile [84]. However, most of them are based on a large number of simplifying assumptions due to the lack of understanding of the inherent physicochemical reactions and the numerical complexity. This can incur a large discrepancy between model predictions and actual measurement, limiting their use in control applications [85].

An applicable solution to the aforementioned problem is using the surrogate models, which are mathematically simpler and empirically map the input-output relationships of the system or computationally intensive model [86]. For the plasma etch process, state-space representation [87], response surface model [88], and the first-order-plus-time-delay (FOPTD) model [73], [85], [89] have been employed. Among those, the FOPTD model has been the most popular choice for practical use owing to its mathematical simplicity and interpretability. However, the FOPTD model is inherently limited because it can only describe the local response of the system with linear approximation and requires auxiliary measures such as gain scheduling to use it as a global model [90]. Furthermore, the multivariable interactions cannot be taken into account in the FOPTD model, that is, if it is under the conditions where such interactions severely affect the system, it requires a separate procedure of designing interaction decouplers [91].

As an alternative measure, the techniques that utilize artificial neural networks have been employed to model the process. Some studies have demonstrated its effectiveness in modeling the deterministic characteristics of the etch process, as well as the interactions between multiple input

and output variables of the process without prior knowledge of the system dynamics [92]–[95]. Nevertheless, these studies have not taken into account the stochastic behaviors of the process, and the strategy to derive the optimal structure of the model has not yet been suggested.

Modeling the stochastic process behavior and quantifying the uncertainty of the model prediction is crucial for the control system design and verification because such model enables deriving robust control parameters by considering possible stochastic system behaviors. Yielding a lightweight model is of importance for practical use because neural network architectures are typically comprised of a myriad number of weights (i.e., high memory complexity), which may constrain its use on industrial sites.

We develop Bayesian long short-term memory (LSTM) based on LSTM, which is known to be effective in modeling the nonlinear dynamical processes [96]. A preliminary work has been presented in [97]. In Bayesian LSTM, each weight of the LSTM is expressed as a Gaussian distribution, and the distributional parameters are trained to maximize the posterior probability to the dataset. After then, structural optimization was executed by eliminating *redundant* weights (i.e., weights having larger standard deviations). In a case study, the resulting sparse Bayesian LSTM is found to preserve prediction accuracy even after eliminating insignificant 90% weights of the Bayesian LSTM.

The remainder of this chapter is organized as follows. Section 4.2 describes the proposed method. A verification study is performed in Section 4.3.

4.2. Methods

4.2.1. Long Short-Term Memory (LSTM)

The LSTM [98] is a type of recurrent neural network [99] that is designed to model temporal sequences and their long-range dependencies more accurately than conventional recurrent neural networks. Such property is attained by allowing the network to learn when to forget and update the hidden states given new information. We used an LSTM architecture described in [100], whose schematic is illustrated in Figure 4.1. By given the system input $u_t \in \mathbb{R}^n$ and the system output $y_t \in \mathbb{R}^m$, the updates for LSTM unit t are formulated as

$$i_t = \text{sigmoid}(W_{u\to i}u_t + W_{y\to i}h_{t-1} + b_i) \tag{4.1}$$

$$f_t = \text{sigmoid}(W_{u\to f}u_t + W_{y\to f}h_{t-1} + b_f) \tag{4.2}$$

$$o_t = \text{sigmoid}(W_{u\to o}u_t + W_{y\to o}h_{t-1} + b_o) \tag{4.3}$$

$$g_t = \tanh(W_{u\to c}u_t + W_{y\to c}h_{t-1} + b_c) \tag{4.4}$$

$$x_t = f_t \odot x_{t-1} + i_t \odot g_t \tag{4.5}$$

$$h_t = o_t \odot \tanh(x_t) \tag{4.6}$$

where $\text{sigmoid}(x) = (1 + e^{-x})^{-1} \in [0, 1]$, $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x}) \in [-1, 1]$, and $a \odot b$ denotes the element-wise products of vectors a and b; i_t, f_t, o_t, g_t are respectively the return values at input gate, forget gate, and output gate; $W_{i\to j}$ and b_j denote the weights from the gate i to the gate j; x_t and h_t are long-term and short-term memory states.

The system output is given by a fully connected layer:

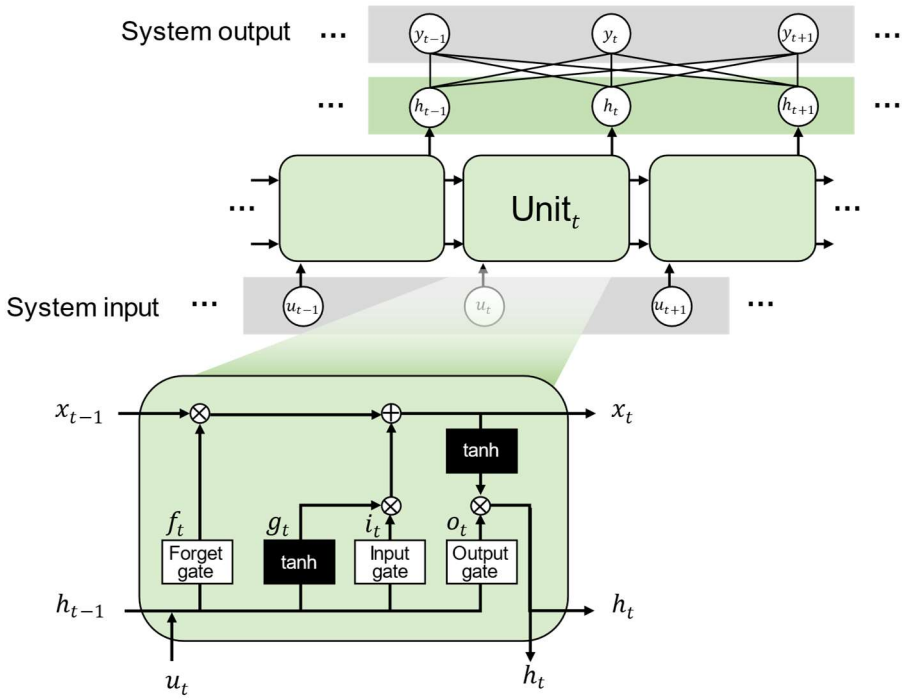

Figure 4.1. Schematic of the LSTM network used in this study.

$$y = W_o h + b_o \tag{4.7}$$

where $y = \{y_t\}_{t=1}^{T}$, $h = \{h_t\}_{t=1}^{T}$, T is the number of LSTM units or *windowsize*, and W_{fc} and b_{fc} are the weights of fully connected layer.

An LSTM can be viewed as a parametric model that assigns conditional probability $P(\mathcal{D}|\theta)$ of some dataset $\mathcal{D} = (u_i, y_i)_{i=1}^{N}$ conditioned on the network weights $\theta = \{\theta_i\}_{i=1}^{M}$ where N is the number of sample points and M is the number of weights. If we assume that the sample points are drawn independently from a joint distribution $P(u, y)$, the optimal weights can be found by maximum likelihood estimation (MLE):

$$\theta_{\text{MLE}} = \underset{\theta}{\text{argmax}} \log P(\mathcal{D}|\theta) = \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log P(y_i|u_i, \theta) \tag{4.8}$$

If we assume Gaussian observation noise on the prediction, i.e., $y = f_\theta(u) + \epsilon$ where f_W denotes LSTM and $\epsilon \sim \mathcal{N}(0, \sigma^2)$, the maximum likelihood estimate can be found from the prediction error minimization with mean squared error.

$$\begin{aligned}
\theta_{\text{MLE}} &= \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log P(y_i|u_i, \theta) \\
&= \underset{\theta}{\text{argmax}} \sum_{i=1}^{N} \log \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(y_i - f_\theta(u_i)\right)^2}{2\sigma^2} \right) \right] \\
&= \underset{\theta}{\text{argmax}} \left[-\frac{N}{2} \log 2\pi\sigma^2 - \sum_{i=1}^{N} \frac{\left(y_i - f_\theta(u_i)\right)^2}{2\sigma^2} \right] \\
&= \underset{\theta}{\text{argmin}} \sum_{i=1}^{N} \left(y_i - f_\theta(u_i)\right)^2
\end{aligned} \tag{4.9}$$

4.2.2. Bayesian LSTM (BLSTM)

Unlike conventional LSTM that finds a point estimate of θ, Bayesian LSTM seeks the posterior distribution of the weights to the training data, $p(\theta|\mathcal{D})$, using the variational inference described in Section 2.2. Given a loss function $\mathcal{L}(\mathcal{D}, \Phi) = \mathrm{KL}\big(Q(\theta|\Phi)||P(\theta|\mathcal{D})\big)$, the optimal estimate of Φ is given as

$$\Phi_{\mathrm{VI}} = \underset{\Phi}{\operatorname{argmin}} \Big[\mathrm{KL}\big(Q(\theta|\Phi)||P(\theta)\big) - \mathbb{E}_{Q(\theta|\Phi)}[P(\mathcal{D}|\theta)] + \log P(\mathcal{D})\Big] \quad (4.10)$$

The optimal Φ can be found using stochastic gradient descent on the loss function in Equation (4.10), where the gradient on variational parameter is given as

$$\frac{\partial}{\partial \Phi}\mathcal{L}(\mathcal{D}, \Phi) = \frac{\partial}{\partial \Phi}\mathbb{E}_{\theta\sim Q(\theta|\Phi)}[\log Q(\theta|\Phi) - \log P(\theta) - \log P(\mathcal{D}|\theta)] \quad (4.11)$$

However, a gradient estimator for $\nabla_{\Phi}\mathbb{E}_{Q(\theta|\Phi)}[\log P(\mathcal{D}|\theta)]$ is known to exhibits high variance and hinder the training process [61]. To circumvent the posed problem, reparameterization trick [101] is used, where θ is factorized using differentiable posterior distribution. First, $Q(\theta|\Phi)$ is chosen to be a Gaussian with diagonal covariance. For each weight component θ_i, the mean θ_i^{μ} and standard deviation θ_i^{σ} are defined, where the weight is sampled by $\theta_i = \theta_i^{\mu} + \theta_i^{\sigma} \cdot \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, 1)$. Here, to ensure non-negative σ_i, we further factorized $\theta_i^{\sigma} = \log\big(1 + \exp\big(\theta_i^{\rho}\big)\big)$. We define variational parameters as $\Phi = \big\{\theta_i^{\mu}, \theta_i^{\rho}\big\}_{i=1}^{M}$. Since $\varepsilon = \{\varepsilon_i\}_{i=1}^{M}$ is independent to Φ, Equation (4.11) can be rewritten as

$$\begin{aligned}\frac{\partial}{\partial\Phi}\mathcal{L}(\mathcal{D},\Phi) &= \frac{\partial}{\partial\Phi}\mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}[\log Q(\theta|\Phi) - \log P(\theta) - \log P(\mathcal{D}|\theta)] \\ &= \mathbb{E}_{\varepsilon\sim\mathcal{N}(0,I)}\left[\frac{\partial}{\partial\Phi}\{\log Q(\theta|\Phi) - \log P(\theta) - \log P(\mathcal{D}|\theta)\}\right] \\ &\approx \frac{\partial}{\partial\Phi}\sum_{k=1}^{K}\left[\log Q\left(\theta^{(k)}\middle|\Phi\right) - \log P\left(\theta^{(k)}\right) - \log P\left(\mathcal{D}\middle|\theta^{(k)}\right)\right]\end{aligned} \tag{4.12}$$

where K is the number of the random samples and $\theta^{(k)}$ is the kth Monte Carlo sample drawn from the variational posterior distribution $Q(\theta|\Phi)$.

Let $\tilde{\mathcal{L}}(\theta,\Phi) = \sum_{k=1}^{K}\left[\log Q\left(\theta^{(k)}\middle|\Phi\right) - \log P\left(\theta^{(k)}\right) - \log P\left(\mathcal{D}\middle|\theta^{(k)}\right)\right]$ for given data $\mathcal{D}$, then the gradient for each distributional parameter $\theta = \{\theta_i\}_{i=1}^{M}$ in Φ can be found as

$$\frac{\partial}{\partial\Phi}\mathcal{L}(\mathcal{D},\Phi) = \frac{\partial\tilde{\mathcal{L}}(\theta,\Phi)}{\partial\theta}\frac{\partial\theta}{\partial\Phi} + \frac{\partial\tilde{\mathcal{L}}(\theta,\Phi)}{\partial\Phi} \tag{4.13}$$

and the training can be performed by using the usual backpropagation process.

For minibatch training, the gradient is given as

$$\begin{aligned}\frac{\partial}{\partial\Phi}\mathcal{L}(\mathcal{D},\Phi) &= \sum_{j=1}^{B}\left[\frac{\partial}{\partial\Phi}\Big|_j\,\mathcal{L}(\mathcal{D}_j,\Phi)\right] \\ \frac{\partial}{\partial\Phi}\Big|_j\,\mathcal{L}(\mathcal{D}_j,\Phi) &= \frac{\partial}{\partial\Phi}\sum_{k=1}^{K}\left[\left(\log Q\left(\theta^{(k)}\middle|\Phi\right) - \log P\left(\theta^{(k)}\right)\right)/B\right. \\ &\qquad \left. - \log P\left(\mathcal{D}\middle|\theta^{(k)}\right)\right]\end{aligned} \tag{4.14}$$

where B is the number of mini-batches per epoch. Algorithm 1 summarizes the optimization process during the minibatch training.

After training the network with a training dataset, we can expect that

the weights are separated into Gaussian distributions having different parameters. In such case, the redundancy of a weight θ can be measured by the following signal-to-noise ratio (SNR)

$$\theta^{SNR} = \left|\frac{\theta^{\mu}}{\theta^{\sigma}}\right| \tag{4.15}$$

and removing the weights having small SNR yields a sparse Bayesian LSTM.

4.3. Verification study

4.3.1. System description

Figure 4.2 describes the schematic of a plasma etching reactor employed in this paper. A capacitively coupled plasma reactor equipped with a 300 mm Si wafer was used. The reactor was powered by a 60 MHz radio frequency (RF) generator. The generator was applied to the bottom electrode and the bottom electrode was grounded. The gap between top and bottom electrodes was 25mm and the area ratio between the top showerhead and the bottom electrode was 1.33. The pressure of the chamber was controlled by manipulating the throttle valve position of the vacuum pump. Two optical emission spectrometers (AvaSpec-ULS2048L, Avantes) were used in estimating the plasma variables, where low wavelength range (255 – 523 nm) and high wavelength range (492 – 1030 nm) were measured with 0.2 nm spectral resolution.

Algorithm 1 Optimization procedure for a minibatch training in Bayesian LSTM

Input	θ: Weights of the LSTM network
Parameter	B: number of minibatches K: number of random samples α: learning rate

1: **for** $k = 1,2,\dots,K$

2: **Sample** $\varepsilon^{(k)} \sim \mathcal{N}(0, I)$

3: $\theta^{(k)} = \theta^{\mu} + \log(1 + \exp(\theta^{\rho})) \circ \varepsilon^{(k)}$

4: $$\tilde{\mathcal{L}}(\theta, \Phi) = \sum_{k=1}^{K} \left[\left(\log Q\left(\theta^{(k)} \middle| \Phi\right) - \log P\left(\theta^{(k)}\right) \right) / B - \log P\left(\mathcal{D} \middle| \theta^{(k)}\right) \right]$$

5: $$\Delta_{\mu} = \frac{\partial \tilde{\mathcal{L}}(\theta, \mu)}{\partial \theta} + \frac{\tilde{\mathcal{L}}(\theta, \mu)}{\partial \mu}$$

6: $$\Delta_{\rho} = \frac{\partial \tilde{\mathcal{L}}(W, \rho)}{\partial \theta} + \frac{\tilde{\mathcal{L}}(\theta, \rho)}{\partial \rho}$$

7: $\mu \leftarrow \mu - \alpha \Delta_{\mu}$

8: $\rho \leftarrow \rho - \alpha \Delta_{\rho}$

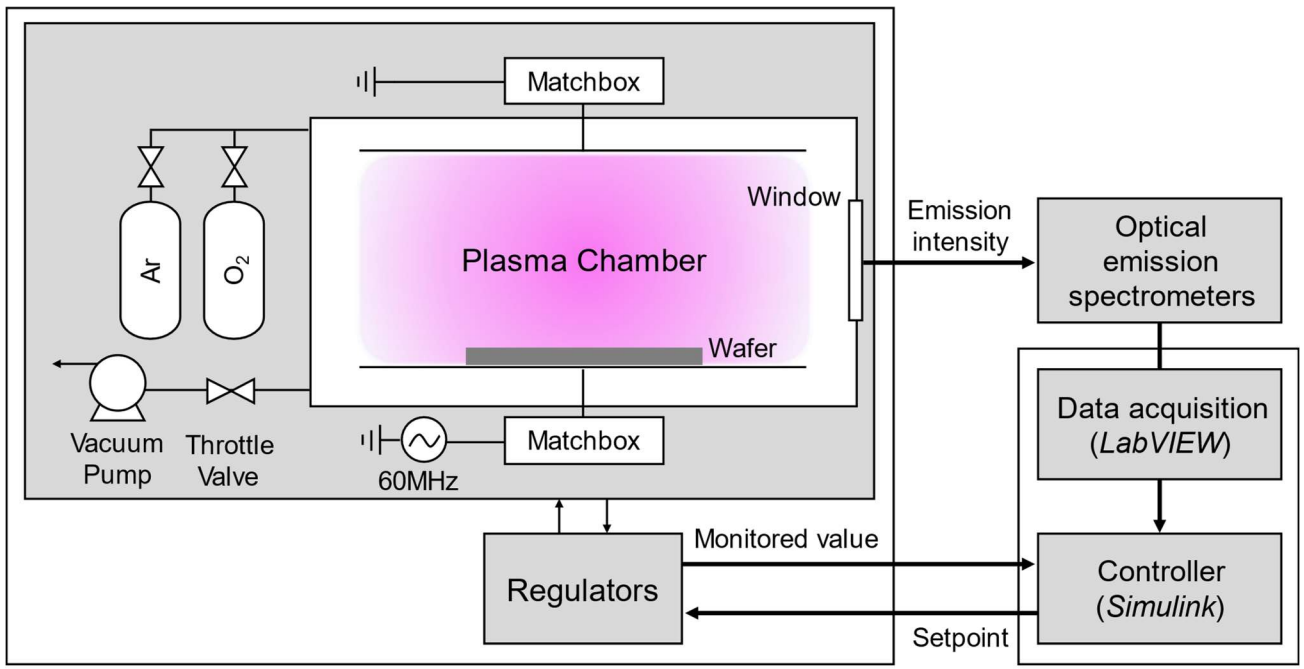

Figure 4.2 Schematic of etching equipment and data acquisition systems employed.

4.3.2. Estimation of the plasma variables

The operating conditions specified above form weakly-ionized plasma inside the chamber [102], where the electron excitation rate is equal to the spontaneous photo decay process [103]. At this condition, the light emission intensity from an i^{th} state to j^{th} state is given by

$$\phi_{ij} = n_0 n_e \int_{E_{ij}^{\mathrm{th}}}^{\infty} \sigma_{ij}(\varepsilon)(2\varepsilon/m)^{1/2} f(\varepsilon) d\varepsilon \tag{4.16}$$

where n_0 is the number density of ground state atoms, n_e is the electron density, E_{ij}^{th} is the excitation threshold energy from level i to level k, σ_{ij} is the excitation cross section, ε is the electron energy, m is the electron mass, and $f(\varepsilon)$ is the electron energy distribution function (EEDF) [104].

Here, $f(\varepsilon)$ is given by one-parameter Maxwellian distribution characterized by an electron temperature, T_e,

$$f(\varepsilon, T_e) = \frac{2\sqrt{\varepsilon}}{\sqrt{\pi}(kT_e)^{3/2}} \exp\left(-\frac{\varepsilon}{kT_e}\right) \tag{4.17}$$

where k is the Boltzmann constant.

The electron temperature can be measured using line-ratio method [105]. The ratio of two emission lines from different excited states is given as

$$\Phi(i \to j, k \to l) \equiv \frac{\phi_{ij}}{\phi_{kl}} = \frac{\int_{E_{ij}^{\mathrm{th}}}^{\infty} \sigma_{ij}(\varepsilon) \cdot \exp\left(-\frac{\varepsilon}{kT_e}\right) \varepsilon \, d\varepsilon}{\int_{E_{kl}^{\mathrm{th}}}^{\infty} \sigma_{kl}(\varepsilon) \cdot \exp\left(-\frac{\varepsilon}{kT_e}\right) \varepsilon \, d\varepsilon} \tag{4.18}$$

where 750.4 nm (Ar, $2p_1 \to 1s_2$) and 425.9 nm (Ar, $3p_1 \to 1s_2$) lines were used.

The electron density was estimated via a linear correlation between

the emission intensity of the wavelength at 750.4 nm and the measurement from a Langmuir probe [103].

4.3.3. Dataset description

Table 4.1 describes the input and output variables of the system. The manipulated variables were the setpoints for internal regulators in Figure 4.2. We generated data by applying bounded random fluctuations on the setpoints for RF power, Ar flow rate, and O_2 flow rate with different changing intervals (see Figure 4.3 and Table 4.2). A total of 5995 data points were obtained from running the process for 300s with 50ms sampling rate. The first 70% samples were used training dataset and the remaining 30 % samples were used for test dataset. We employed *window-marching* sampling for each dataset to recast the dataset into an LSTM-compatible form.

4.3.4. Experimental setup

Based on an assumption that the delayed output is available when making a prediction, we augmented the input data with 1-step delayed output data. The input and output data were normalized into Gaussian distribution with zero mean and identity covariance matrices prior to the model training.

The initial values for memory states, i.e., x_0 and h_0, were given by zeros. For both LSTM and BLSTM, we used parameter settings described in Table 4.3. Both the initial distributions of the weights and the

Table 4.1 Input and output variables of the system.

Input						Output (OV)		
Manipulated (MV)		Unit	Unmanipulated (UV)		Unit			Unit
1	Pressure setpoint	mTorr	5	RF power [W]	W	1	Electron density	cm^{-3}
2	RF power setpoint	W	6	Forward RF power	W	2	Electron temperature	eV
3	Ar flow rate setpoint	sccm	7	Reflected RF power	W			
4	O_2 flow rate setpoint	sccm	8	Pressure	mTorr			
			9	Pumping valve pressure	mTorr			
			10	Pumping valve position	%			
			11	Ar flow rate	sccm			
			12	O_2 flow rate	sccm			

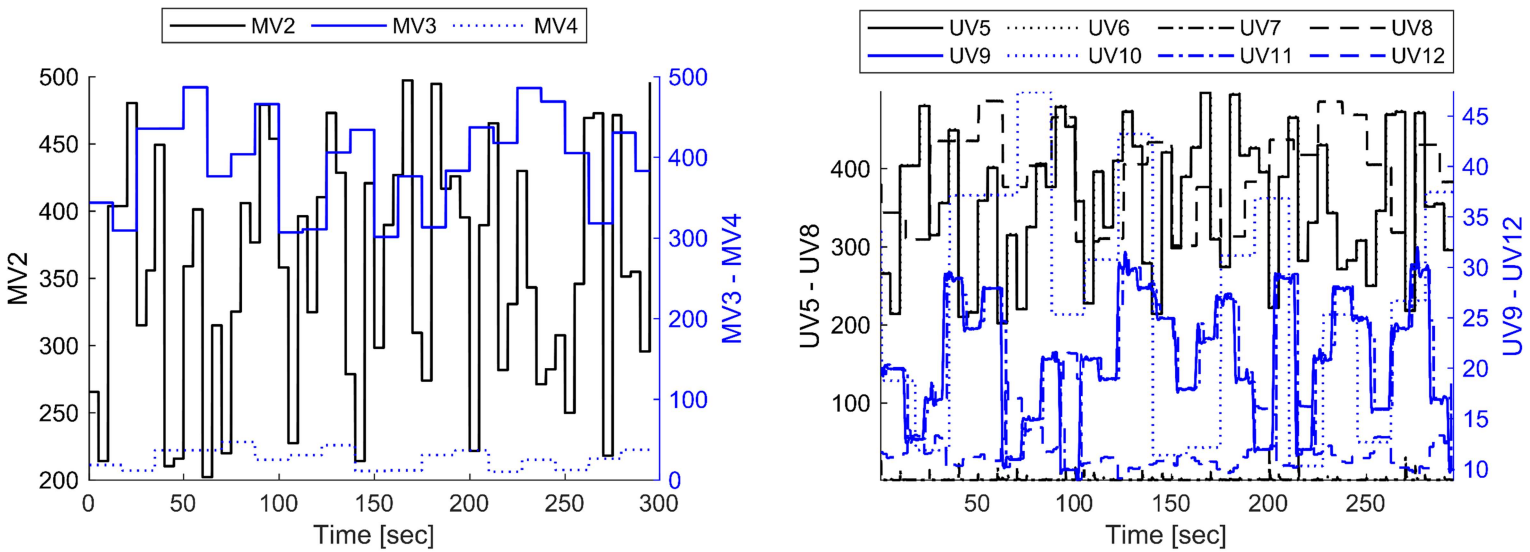

Figure 4.3 Time trajectories of the system input variables

Table 4.2 Operation ranges and change intervals applied.

	Unit	Min	Nominal	Max	Change interval [s]	Note
Pressure setpoint	mTorr	–	30	–	–	Fixed during the operation
RF power setpoint	W	200	350	500	5	
Ar flow rate setpoint	sccm	300	400	500	12.5	
O_2 flow rate setpoint	sccm	10	30	50	17.5	

Table 4.3 Parameter values used in LSTM and BLSTM.

Parameter		Value
LSTM Architecture	Dimension of the long-term memory states (x) Dimension of the short-term memory states (h)	50
	Number of LSTM units	20
Training	Number of random samples (K)	10
	Number of samples per mini-batch	1000
	Number of mini-batches per epoch (B)	5
	Maximum epoch number	500
	Learning rate	0.01

likelihood distributions are given by the Gaussian distributions with zero mean and unit covariance matrices multiplied by 0.1. The termination criterion for the training was given as

$$|\mathcal{L}_{k+1} - \mathcal{L}_k| \leq 10^{-6} \cdot |1 + \mathcal{L}_k| \tag{4.19}$$

where k is the epoch number. In BLSTM, Gaussian distributions with zero mean and identity covariance matrices were used for the prior and likelihood function, and the variational inference was applied only to the LSTM units (not to the fully connected layer).

To benchmark the prediction accuracy of the proposed method, the FOPTD model [106] and fully connected neural networks (FC) were used. The FOPTD model is formulated as

$$\tau \cdot \frac{dy(t)}{dt} = -y(t) + K \cdot u(t - \tau_d) \tag{4.20}$$

where K is the process gain, τ is the time constant, and τ_d is the time delay, respectively. The FC model is given as (4.7), except for the hyperbolic tangent activation function is applied to the outputs.

Since FC, LSTM, and BLSTM work in a *time-marching window* manner, multiple predictions are possible at a time point. Considering the practical situations of how prediction is made, the value obtained from the terminal unit was used and the other prediction values were disregarded.

To benchmark the *a posteriori* regularization proposed in 4.2.2, L1 and dropout-based regularization were respectively employed on LSTM. The loss function of the L1-regularized LSTM was given as

$$\theta_{\mathrm{MLE}} = \underset{\theta}{\operatorname{argmin}} \left[\sum_{i=1}^{N} \left(y_i - f_\theta(u_i) \right)^2 + \sum_{j=1}^{M} |\theta_j| \right] \tag{4.21}$$

The compression rate was choosen as 10 (removing 90% of the weights). For instance, in dropout-based LSTM, the dropout rate was choosen as 0.9.

4.3.5. Weight regularization during training

Figure 4.4 illustrates the loss function values of the LSTM and BLSTM and their weight distributions during the training. Posterior loss, prior loss, and likelihood loss denote $\sum_{k=1}^{K}\left[\log Q\left(\theta^{(k)}|\Phi\right)/B\right]$, $\sum_{k=1}^{K}\left[-\log P\left(\theta^{(k)}\right)/B\right]$, and $\sum_{k=1}^{K}\left[-\log P\left(\mathcal{D}|\theta^{(k)}\right)\right]$ in Equation (4.14), respectively. Both networks show robust convergence and are shown to primarily minimize the likelihood loss. As the training progresses, the parameter distribution gradually deviates from the prior distribution, and the prior loss gradually increases. The posterior loss in BLSTM fluctuates over epochs, where the inverse pattern can be found in mean values of W^{ρ}, denoted in $\bar{W}^{\rho}$. As $\bar{W}^{\rho}$ accounts for the *stiffness* of the overall parameter distributions in BLSTM, the trajectory of the posterior loss can be viewed as a measure of the overall *flatness* of the parameter distributions in the BLSTM.

We can observe that the SNRs of the weights, θ^{SNR}, spread out as optimization proceeds in BLSTM, which signifies that the BLSTM can regularize the weights during the training. The training of the weights is found to be faster in BLSTM than LSTM, where the maximum absolute

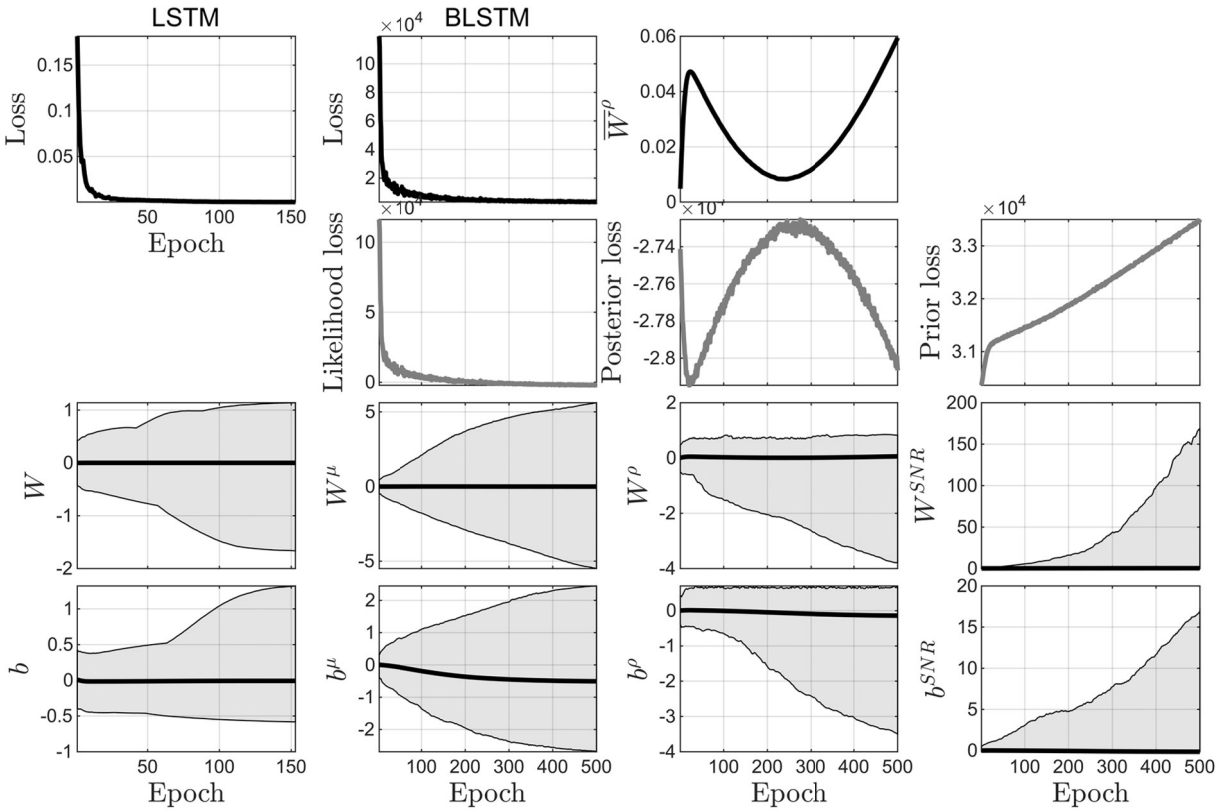

Figure 4.4 LSTM and BLSTM losses and weight distribution changes during the training. The shaded areas and bold lines in the third and fourth rows represent the parameter distribution and the mean value, respectively.

values of the weights at 100 epochs were $\max(|W|, |b|) = (1.0, 0.8)$ for LSTM and $\max(|W^{\mu}|, |b^{\mu}|) = (2.8, 1.4)$ for BLSTM.

4.3.6. Modeling complex behaviors of the system

Figure 4.5 shows fitting results of the FOPTD, FC, LSTM, and BLSTM to the training dataset, as well as their predictions on the test dataset. FC, LSTM, BLSTM models show high prediction accuracy, whereas the FOPTD model exhibits large model-plant-mismatch error. Note that the FOPTD model primarily describes the relationship between the RF power setpoint (MV1) and electron density (OV1) (see Table 4.4), and the combinatorial effects of the input variables to the system are not taken into account. This implies that the conditions where the large error occurred in the FOPTD predictions are the conditions where the interactions other than MV1-OV1 have not been correctly modeled or those where the combinatorial effects significantly affect the system. In other words, the results are shreds of evidence that the proposed method can effectively model such complex system characteristics.

In terms of the structure of the neural networks, Figure 4.6 demonstrates that the LSTM architecture is more suitable in modeling the nonlinearity and dynamical behaviors of the system, compared to the fully connected neural networks. From Figure 4.7, we can further ascertain that the proposed Bayesian LSTM architecture better models the stochastic characteristics of the process such as noise. No significant difference has been found in the results of Bayesian LSTM and sparse

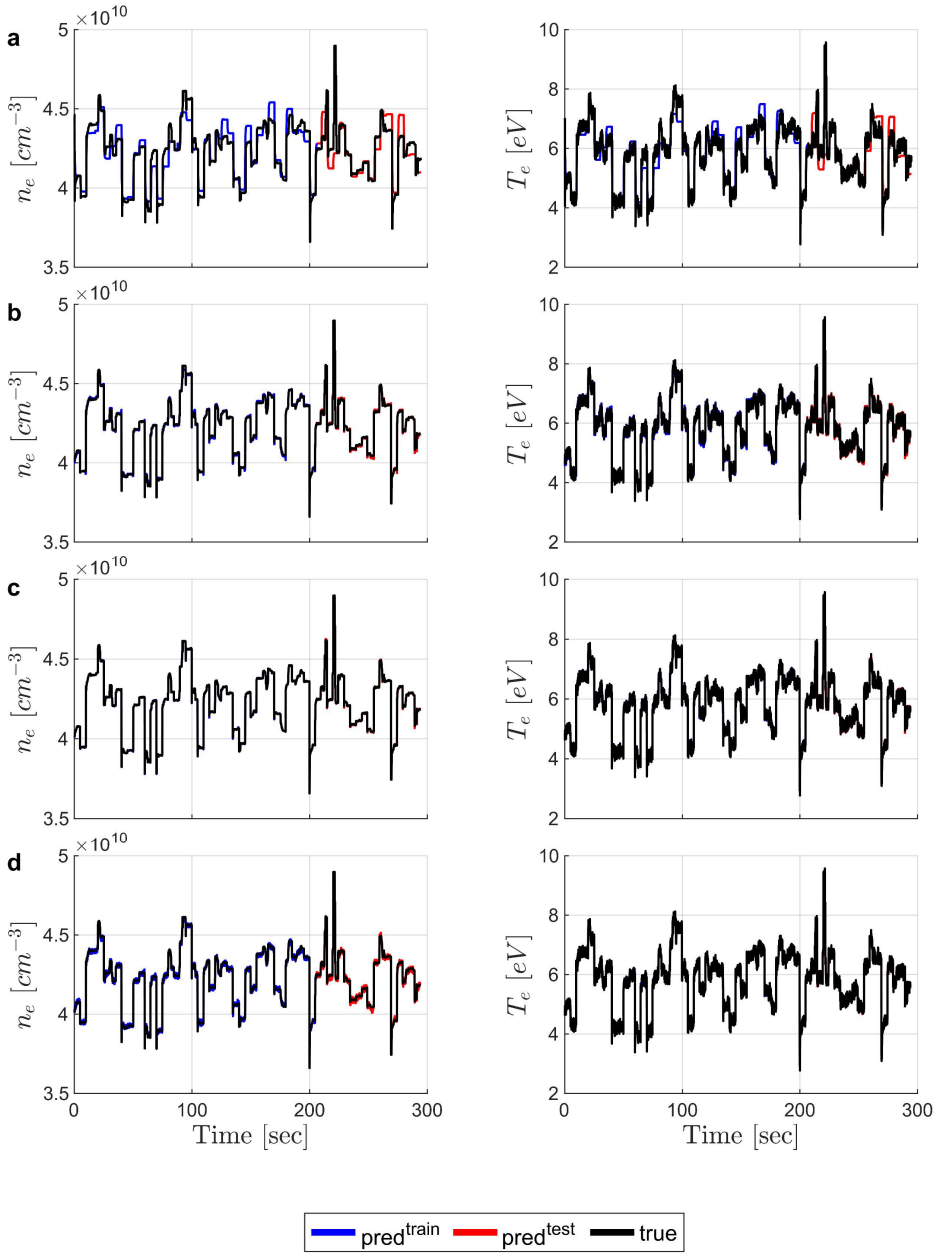

Figure 4.5 Fitting and prediction results of the (a) FOPTD, (b) FC, (c) LSTM, and (d) BLSTM models on training and test datasets.

Table 4.4 Estimated FOPTD model parameter values.

		RF power setpoint	**Ar flow rate setpoint**	**O2 flow rate setpoint**
Electron density	K	0.93	-8.3×10^{-3}	-0.13
	τ	0.22	1.7×10^{-4}	6.2×10^{-3}
	τ_d	0.26	2.2	0.35
Electron temperature	K	0.92	-0.18×10^{-2}	-0.13
	τ	0.21	9.4×10^{-6}	2.2×10^{-3}
	τ_d	0.27	0.74	0.35

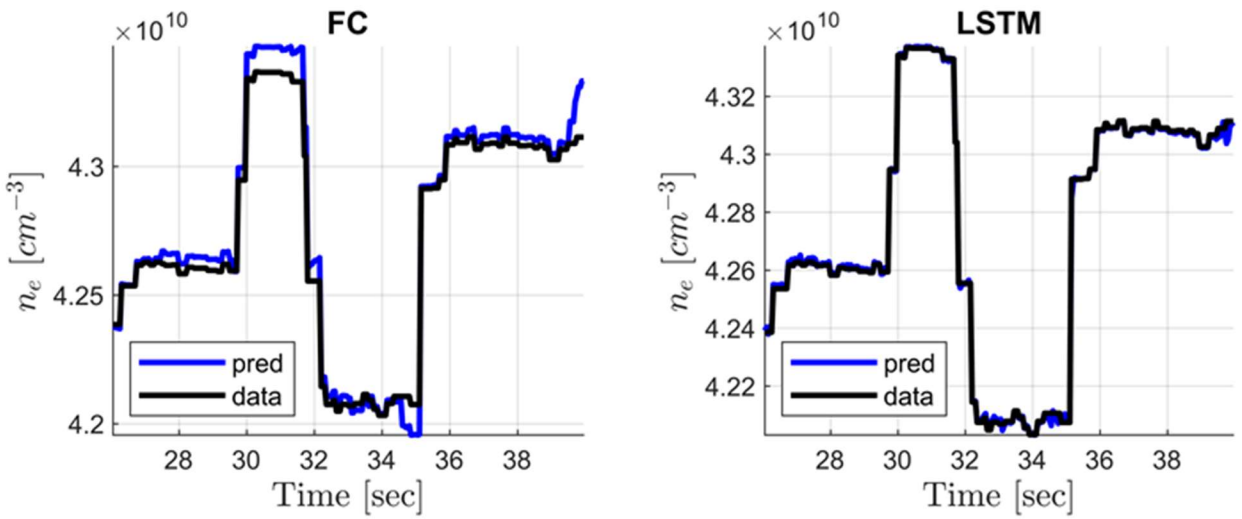

Figure 4.6 Comparison of the fitting results of FC and LSTM models for electron density within the time interval [26, 40].

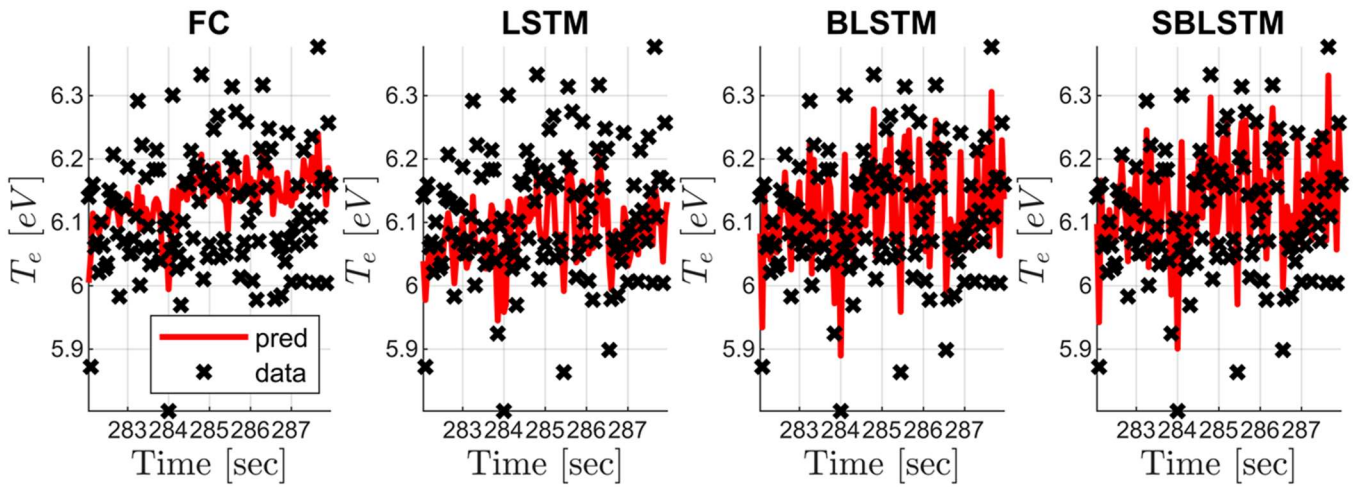

Figure 4.7 Comparison of the prediction results of FC, LSTM, BLSTM, and SBLSTM models for electron temperature within the time interval [282, 288].

LSTM models. Note that the noisy predictions in FC and LSTM have been originated from the noises in the system input data and the random initialization of the weights.

4.3.7. Uncertainty quantification and model compression

As we trained the BLSTM with the distributional data acquired from each operating condition, the model can learn the uncertainty inherent in the process data from the training variational parameters. As a result, even if the same input has been applied, it produces different outputs with respect to the random seeds employed, and these outputs reflect the stochastic behaviors of the process. Figure 4.8 shows the posterior predictive distribution of BLSTM to the datasets. Note that larger uncertainty is observed for the test dataset than the training dataset, especially when the system input changes drastically. As such conditions are where only a small number of samples are accessible, the results manifest that the model can capture the uncertainty from the process data.

In BLSTM, variational parameters are regularized by priors having a standard normal distribution during the training, and hereby relatively redundant weights are identified as described in Section 3.1. This enables eliminating relatively redundant weights by computing the SNR of each weight according to Equation (4.15). From the histograms of weight SNRs in Figure 4.9, we can notice that most of the weights are insignificant and only a small fraction of the weights is valid in prediction, where the prediction accuracy has not decreased at all even

after removing 90%

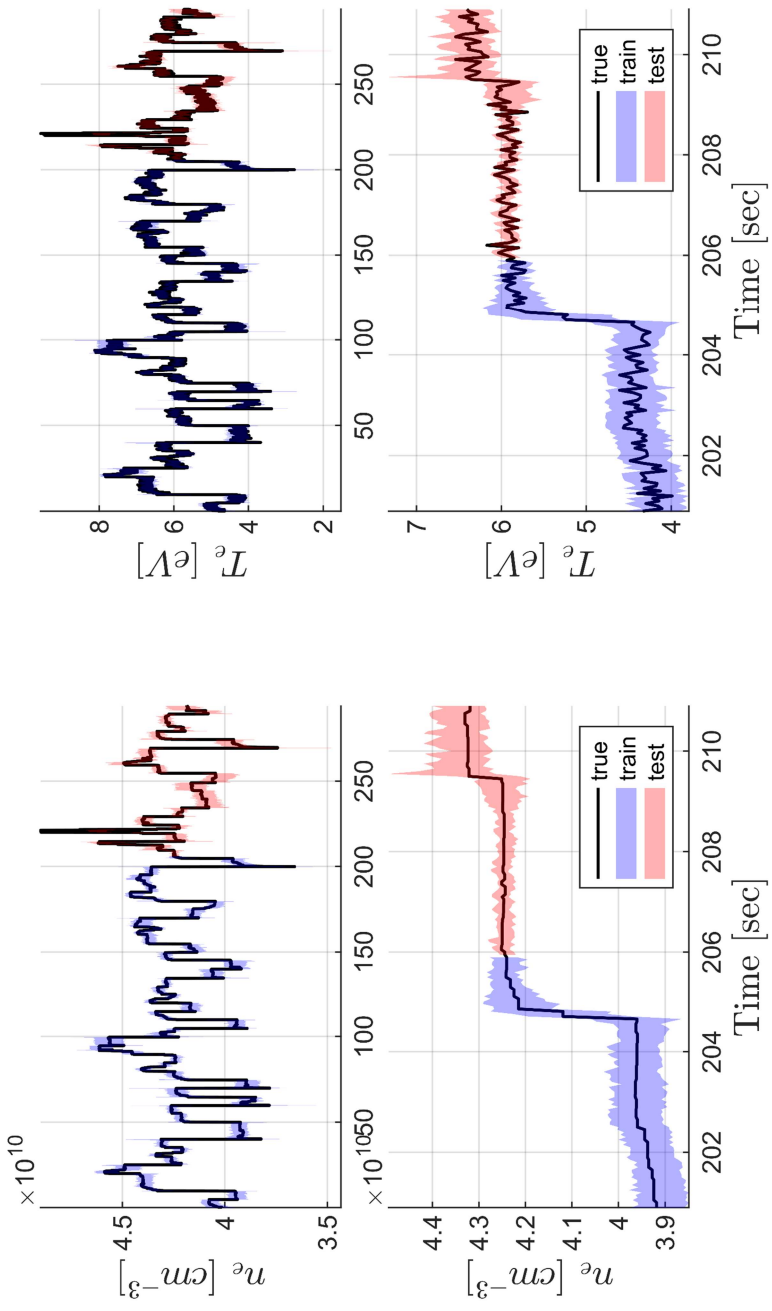

Figure 4.8 Posterior predictive distribution of BLSTM to the training and test datasets. For visualization, the distributions were magnified three times by expanding minimum and maximum values from the mean values.

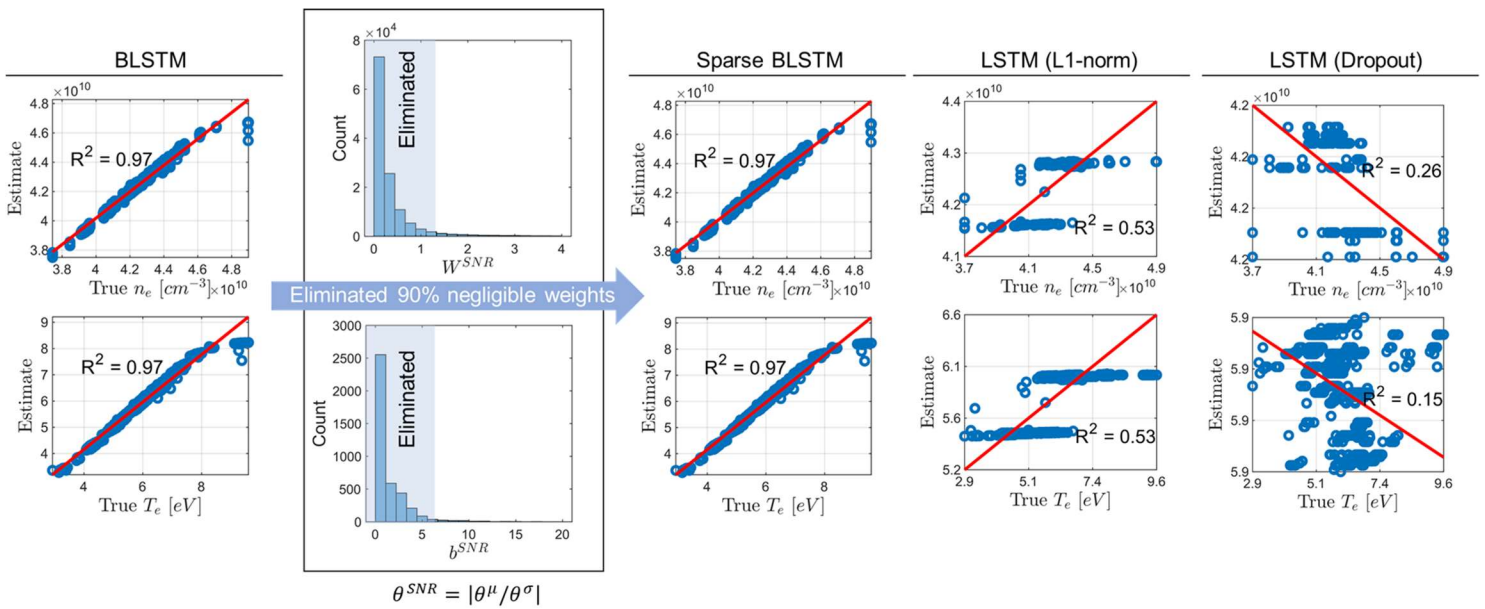

Figure 4.9 Procedure of instantiating a sparse BLSTM and the results of employing regularization techniques.

weights. Note that such selective pruning of the weights also enables avoiding a time-consuming process of determining the optimal number of weights in designing neural network models because the procedure is executed *a posteriori*. This *a posteriori* regularization is shown to be highly effective in obtaining an optimal model structure compared to conventional L1-norm or dropout-based regularization.

Chapter 5

Process design based on reinforcement learning with distributional actor-critic networks[3]

5.1. Introduction

Recently, discussions have been made on the machine inference-based autonomous chemical compound synthesis and process design [107], [108] in the aim of meeting the rapid changes in market demands, such as diversified products and customized drug manufacturing. The process design problems can be described as finding an optimal layout called flowsheet and the specifications of the units for given raw materials, target products, and the requirements specified. A design objective is usually given with the maximization of the profit obtained from process operation.

[3] This chapter is an adapted version of D. Park and J. M. Lee, "Autonomous Process Design based on Behavioral Cloning between Distributional Actor-Critic Networks and Monte Carlo Tree Search," *Comput. Chem. Eng.,* In Preparation.

Two process design approaches, namely decomposition method [109] and optimization [15], [110], have been widely used for this purpose. In the decomposition method, a chemical process is designed through a sequential and hierarchical decisions based on the design heuristics. The resulting flowsheet, often referred to as base case design, is then modified by performing case studies in a way to yield a better process design. The decomposition method can quickly locate a feasible solution from exploiting design expertise, but it cannot guarantee its optimality as the interactions between the design steps of the different hierarchies are not taken into account.

In contrast, the optimization-based method first postulates a process layout, so-called *superstructure*, that includes a set of alternatives of process designs encoded by integer variables. The given process design problem is then solved using optimization techniques such as mathematical programming or metaheuristic methods. As the sequential decisions are simultaneously considered in the optimization framework, the reciprocal interactions can be taken into account. Because of this, optimization-based methods have been established as state-of-the-art methods for process design tasks. That said, as the superstructure construction heavily relies on design expertise, there is possibility to miss the counter-intuitive but still relevant process designs. Particularly for using mathematical programming with commercial solvers, there exists a fatal shortcoming that the interim solutions cannot be obtained until the solver converges. That is, unless the solver converged to a solution, the user cannot even obtain suboptimal solutions.

A possible solution to avoid potential bias from search space restrictions and obtain interim solutions is employing *ab initio* approach [111], which does not construct a superstructure but instead sequentially manipulates unit operations (e.g., addition, removal, …) from a blank flowsheet.

Nevertheless, all the aforementioned methods are *solution searching* techniques that can only find problem-specific solutions. Contrary to this, RL-based process design approach [30] can provide means to perform *solution learning*. That is, the agent can learn the design heuristics during performing designing tasks and utilize it to solve similar design tasks. Nevertheless, relevant studies so far have only demonstrated the applicability of RL to the process design tasks, confined to some simplest design problems [112], [113].

This study first proposes a learnable process design framework based on RL. To this end, the process design problem is first formulated in the RL syntax. By following the terminology and definitions described in Section 2.7, we define the *state* as a process flowsheet, the *reward* as a profit obtained from the process operation, and the *action* as a selection of unit type and its specification. We define implementing action as *linking* a unit to one of the product stream existing in the flowsheet. The schematic of the proposed framework is illustrated in Figure 5.1. The design process is given as a DMDP described in Section 2.7

The key feature of the framework is behavioral cloning [21], where a solution finder and a learner work in collaboration to find and learn the optimal policy (see Figure 5.1). To be more specific, the learner guides

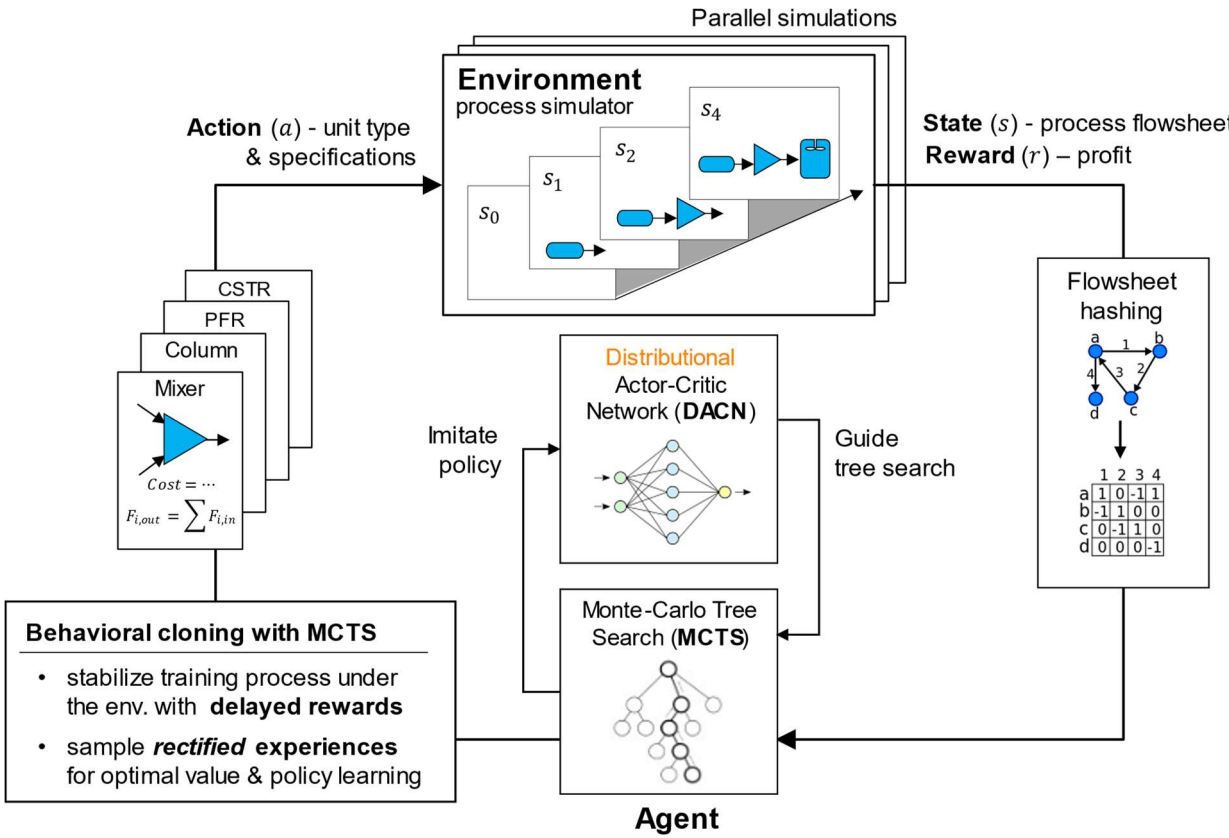

Figure 5.1 Schematic of the proposed framework.

the solution finder for an efficient solution process, and the data obtained from the solution processes are used to train the learner. As the training data are made up of suboptimal solutions rather than random experiences, efficient learning is facilitated.

For the learner, distributional actor-critic networks (DACN) were developed to consider the hybrid action spaces. For solution finder, Neural Monte Carlo tree search (N-MCTS) algorithm, which is known to stabilize the training process with delayed rewards [114], was utilized. In order to employ the neural networks, a flowsheet hashing algorithm that can convert graphical process flowsheets into numeric tensors was developed.

The remainder of this chapter is organized as follows. Section 5.2 describes the proposed framework in detail. A verification study is performed in Section 5.3.

5.2. Methods

5.2.1. Flowsheet hashing

The essence of the flowsheet hashing algorithm is to convert a given process flowsheet $G \in \mathcal{G}$ to a DACN-compatible tensor $T \in \mathcal{T}$ where $\mathcal{G}$ and $\mathcal{T}$ represent the possible set of flowsheet and tensors, respectively., the images of the two different process flowsheets should differ. In this case, the mapping from the process flowsheets to the numeric tensors should be given be injective. As the process flowsheets can be represented by directed graphs, where the nodes and edges respectively

Table 5.1 Example of defining the invariant numbers for unit type.

Type	Invariant number	Type	Invariant number
FEED	1	SPLT	6
PRDT	2	CSTR	7
NULL	3	PFR	8
LINK	4	DSTL1	9
MIX	5	DSTL2	10

Algorithm 2	Rank evaluation
Input	G: graphical representation of the given process v_i: i^{th} node in the graph G
Parameter	$\overline{n}$: maximum number for the iteration
Function	T: return the unit type number D: return the in-degree in the transitive closure of the graph P: return the port number of the input streams with respect to their parent node C: return the centrality O: return ordinal orders for the given number array, e.g., [4,2,1,3,4] = O([5,2,1,3,5]) S: return the 2-digit encoding for the given number array, e.g., $010302 = S([1,3,2])$ and $011203 = S([1,12,3])$ L: count the number of elements in the given array N: count the number of the nodes in the graph U: return the unique elements of the given array K: return the k^{th} prime number for given number k p: return the parental nodes s: return the successor nodes Π: return the product of the elements
Output	$R = [r_i]_{i=1}^{N(G)}$: array of the ranks

$t_i = T(v_i)$
$d_i = D(v_i)$
for $i = 1, 2, \dots, N(G)$
  if $t_i == 1$ # unit type is FEED
    $a_i = n_i$
  else
    $a_i = \sum P(v_i)$
  end if
  $c_i = C(v_i)$
end for
$[\bar{c}_i]_{i=1}^{N} = O([c_i]_{i=1}^{N})$
$I_i = S([t_i, d_i, a_i, \bar{c}_i])$ # invariant number for v_i
$\bar{r}_i = O(I_i)$
$n = 0$
while $L(\bar{r}_i) \neq L\big(U(\bar{r}_i)\big)$ # tie breaking
  $\hat{r}_i = K(\bar{r}_i)$
  for $i = 1, 2, \dots, N(G)$

if $L\big(p(v_i)\big)$
$r_i^p = 1$
else
$r_i^p = \hat{r}_{p(v_i)}$
end if
if $L\big(s(v_i)\big)$
$r_i^s = 1$
else
$r_i^s = \hat{r}_{s(v_i)}$
end if
$\tilde{r}_i = \hat{r}_i \cdot \Pi\big(r_i^p\big) \cdot \Pi(r_i^s)$
end for
$r_i = O(\tilde{r}_i)$ # rank for v_i
if $r_i == \bar{r}_i$ for $\forall i = 1,2, \ldots N(G)$ **or** $n > \overline{n}$
break
else
$\bar{r}_i = r_i$
$n = n + 1$
end if
end while

Algorithm 3 Flowsheet hashing	
Input	G: graphical representation of the given process V: array for the nodes in the graph G
Parameter	$\|T\|$: number of possible node types $W(L)$: width(length) of the 3D tensor H_C: number of embedding layers for configurations H_S: number of embedding layers for specifications $\overline{F}$: maximum flow rates $\underline{F}$: minimum flow rates
Function	*zeros*: return the tensor filled with zeros by given dimensions $rank$: return the array of the ranks using Algorithm 2 $sort(\cdot, idx)$: return the array sorted with the index idx $sortidx$: return the indices for sorting N: count the number of the nodes in the graph M: count the number of the edges in the graph T: return the unit type number E: return True if a node specification is given; otherwise, return False $\overline{s}$: return the maximum specification value $\underline{s}$: return the minimum specification value P: return the port number P_I: return the maximum inlet port number P_O: return the maximum outlet port number h: return the index of the head node t: return the index of the tail node F: return the flow rates
Output	S: tensor representation of the process

$S = zeros(W,\ L,\ H_C + H_S)$
$R = rank(G)$
$idx = sortidx(R)$
$V = sort(V, idx)$
  for $i = 1, 2, \dots, N(G)$
    $v_i = V[i]$
    $t_i = T(v_i)/|T|$
    if $E(v_i)$
      $\hat{s}_i = \left(s_i - \underline{s}(t_i)\right) / \left(\overline{s}(t_i) - \underline{s}(t_i)\right)$
    else
      $\hat{s}_i = 0$

12: **end if**
13: **end for**
14: $S[1{:}N(G),\ 1{:}N(G),\ 1{:}H_C] = t_i$
15: $S[1{:}N(G),\ 1{:}N(G),\ H_C+1{:}H_C+H_S] = \hat{s}_i$
16: **for** $j = 1, 2, \ldots, M(G)$
17: $S[t(j),\ h(j),\ 1] = P\big(t(j)\big)/P_O\big(t(j)\big)$
18: $S[t(j),\ h(j),\ 2] = P\big(h(j)\big)/P_I\big(h(j)\big)$
19: $S[t(j),\ h(j),\ 3{:}H_C+H_S] = \big(F(j)-\underline{F}\big)/\big(\overline{F}-\underline{F}\big)$
20: **end for**

represent process units and streams, we will first follow the terminology presented in Section 2.8 and describe the algorithm in terms of the graph theory.

An equivalent way to ensure the injective mapping is to uniquely determine the orders or *ranks* of the nodes in the graph [115]. To this end, we first define the invariant numbers for possible unit types as in Table 5.1 so that the number indicating unit type t_i can be assigned to each node v_i. A name identifier n_i is attached in the increasing order, i.e., FEED-01, FEED-02, to distinguish the units with the same type. We also assign the invariant numbers to indicate the inlet/outlet ports of the units. For a distillation tower with no side streams, for example, the port invariant numbers can be assigned as input(1), top(1), bottom(2). Finally, the centrality c_i is defined for each node following equation.

$$c_i = \left(\frac{A_i}{N-1}\right)^2 \frac{1}{B_i} \tag{5.1}$$

where A_i is the number of reachable nodes from v_i, B_i is the sum of the distances from v_i to all reachable nodes, and N is the number of nodes in the graph. The rank $r_i \in \{1,2,\dots,|V|\}$ is then evaluated for each node v_i following Algorithm 2. The evaluated ranks provide canonical numbering of the nodes and allow the injective hashing of the flowsheets. The complete flowsheet hashing algorithm is described in Algorithm 3.

5.2.2. Behavioral cloning

The training of the agent is proceeded by a modified version of the

behavioral cloning algorithm used in AlphaZero [116]. Specifically, for a given state s, the N-MCTS outputs the estimates for the return, $v(s)$, and optimal policy function value, $\pi(a|s)$. The neural networks are trained by reducing the following loss function

$$\mathcal{L} = \big(v(s) - v_\theta(s)\big)^2 - \pi(a|s) \cdot \log\big(\pi_\theta(a|s)\big) + c\|\theta\| \qquad (5.2)$$

where $v_\theta(s)$ and $\pi_\theta(a|s)$ are approximate functions given by the neural networks. Note that the loss function forces the neural networks to *clone* the behavior of the N-MCTS. As the agent has hybrid action space – action is given as a combinatorial selection of discrete unit type and continuous unit specifications – the policy samples produced by the tree search are given as Figure 5.2.

5.2.3. Neural Monte Carlo tree search (N-MCTS)

Starting from an empty tree, N-MCTS repeatedly performs the selection, expansion, rollout, and backpropagation phases to find a better action. Specifically, a modified version of the PUCT algorithm used in AlphaZero was utilized. In N-MCTS, every action node in the tree stores statistics $\{N(s,a), W(s,a), Q(s,a)\}$ where $N(s,a)$ is the visitation count, $W(s,a)$ is the cumulative return over all rollouts through (s,a), and

$$Q(s,a) = \frac{W(s,a)}{N(s,a)} = \frac{1}{N(s,a)} \sum\nolimits_{s'|s,a\to s'} v_\theta(s') \qquad (5.3)$$

is the estimate for action-value where $v_\theta(s')$ denotes the estimated

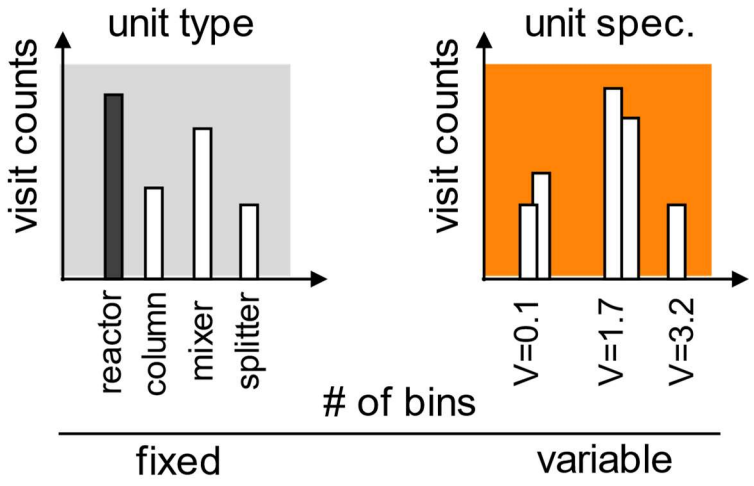

Figure 5.2 Schematic of MCTS policy.

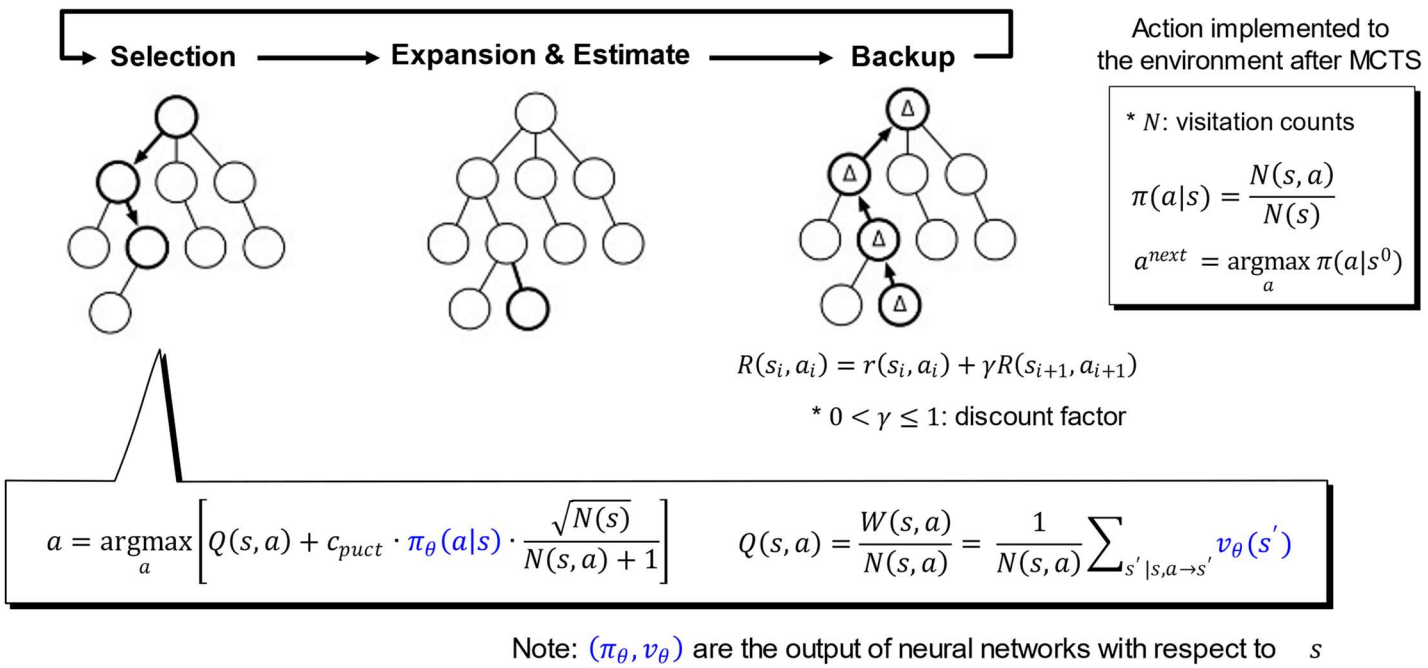

Figure 5.3 Schematic of MCTS algorithm used in this study.

return at s' given by the neural networks. The graphical description of the algorithm is given in Figure 5.3. The N-MCTS algorithm iterates the following four phases:

1. **Selection** In selection phase, the agent descents the tree from the root node according to:

$$a = \underset{a}{\operatorname{argmax}} \left[Q(s,a) + c_{puct} \cdot \pi_\theta(a|s) \cdot \frac{\sqrt{N(s)}}{N(s,a)+1} \right] \tag{5.4}$$

where $N(s) = \sum_a N(s,a)$ is the total number of visits to state s, $c_{puct} \in \mathbb{R}^+$ is a constant for scaling the amount the exploration/exploitation. The selection is performed until either a terminal state is reached or an action that have not been tried before is selected.

2. **Expansion and Estimate** The agent then expands the tree by appending a new leaf state s_L and the value $v(s_L)$ is approximated by the neural network.

3. **Backup** The results in the tree nodes are recursively backed-up. Let the forward searching trace as $\{s_0, a_0, \dots, s_{L-1}, a_{L-1}, s_L\}$. For each state-action edge (s_i, a_i) where $L > i \geq 0$, the agent recursively estimates the state-action value as

$$R(s_i, a_i) = r(s_i, a_i) + \gamma R(s_{i+1}, a_{i+1}) \tag{5.5}$$

where $R(s_L, a_L) \coloneqq R(s_L)$. The cumulative return $W(s_i, a_i)$ is incremented with the new estimate $R(s_i, a_i)$ and the visitation count $N(s_i, a_i)$ is increased by 1. Lastly, state-action value is updated by $Q(s_i, a_i) = W(s_i, a_i)/N(s_i, a_i)$. This backup process is applied

backward along the trace until we reach the root node s_0.

The searching procedure is repeated until the iteration reaches pre-defined tree search budget N_{tree}. After finishing the tree search, it returns a root action set $A_0 = \{a_{0,0}, a_{0,1}, \dots, a_{0,n}\}$ with associated counts $N_0 = \{N(s_0, a_{0,0}), N(s_0, a_{0,1}), \dots, N(s_0, a_{0,m})\}$ where m denotes the number of child actions implemented at the root node s_0. The agent selects the action for the implementation to the environment, a^*, following the probability distribution based on the visitation counts at the root node s_0, that is,

$$a^* \sim \pi(a|s_0) \tag{5.6}$$

where $\pi(a|s_0) = \frac{N(s_0,a)}{N(s_0)}$ and $N(s_0) = \sum_{a_i \in A_0} N(s_0, a_i)$. We can store the subtree that belongs to the picked action a^* for the MCTS at the next time step. Therefore, $N(s_0)$ can be larger than N_{tree}.

As we cannot enumerate all the actions in the continuous domain, we utilize progressive widening [22] that confines the possible number of child actions at state s, $m(s)$, by a function of the total number of visit to that state $n(s)$. This facilitates exploiting actions with larger visitation counts (which indicate producing better returns) to get more child actions for consideration. Specifically, we employed the progressive widening method described in [117], which is given as

$$m(s) = c_{pw} \cdot N(s)^{\kappa} \tag{5.7}$$

where constants $c_{pw} \in \mathbb{R}^+$ and $\kappa \in (0,1)$. To prevent excessive

exploitation and facilitate exploration, the expansion counts for each node is restricted by $\overline{E}_{node}$.

5.2.4. Distributional actor-critic networks (DACN)

DACN are composed of two functional layers called *actor* and *critic* that respectively approximates the policy and value functions. We employ Siamese architecture, where the two functional layers share the feature extraction layers. The schematic diagram of DACN is described in Figure 5.4.

For the value function estimator, a fully connected layer can simply be used. For the policy function estimator, however, a special measure is required to compute the loss function (5.2) with the data obtained from tree search (see Figure 5.2). Note that $\pi(a|s) \cdot \log\big(\pi_\theta(a|s)\big)$ can be calculated only when the value $\pi_\theta(a|s)$ is accessible, and the neural network layers can only output a fixed tensor. For unit type, the number of slots for a can be configured in advance so a soft-max layer can be used to compute $\pi_\theta(a|s)$. However, for unit specifications having continuous domain, a is uncountable and $\pi_\theta(a|s)$ is generally inaccessible. A naïve solution to this is to discretize the action space [18], but it is known to be highly dependent on the quality of the grid [19].

To address this problem, we can approximate the continuous policy with parameterized distribution functions (see *Distributional layer* in Figure 5.5a). In practice, the action space is constrained by physical

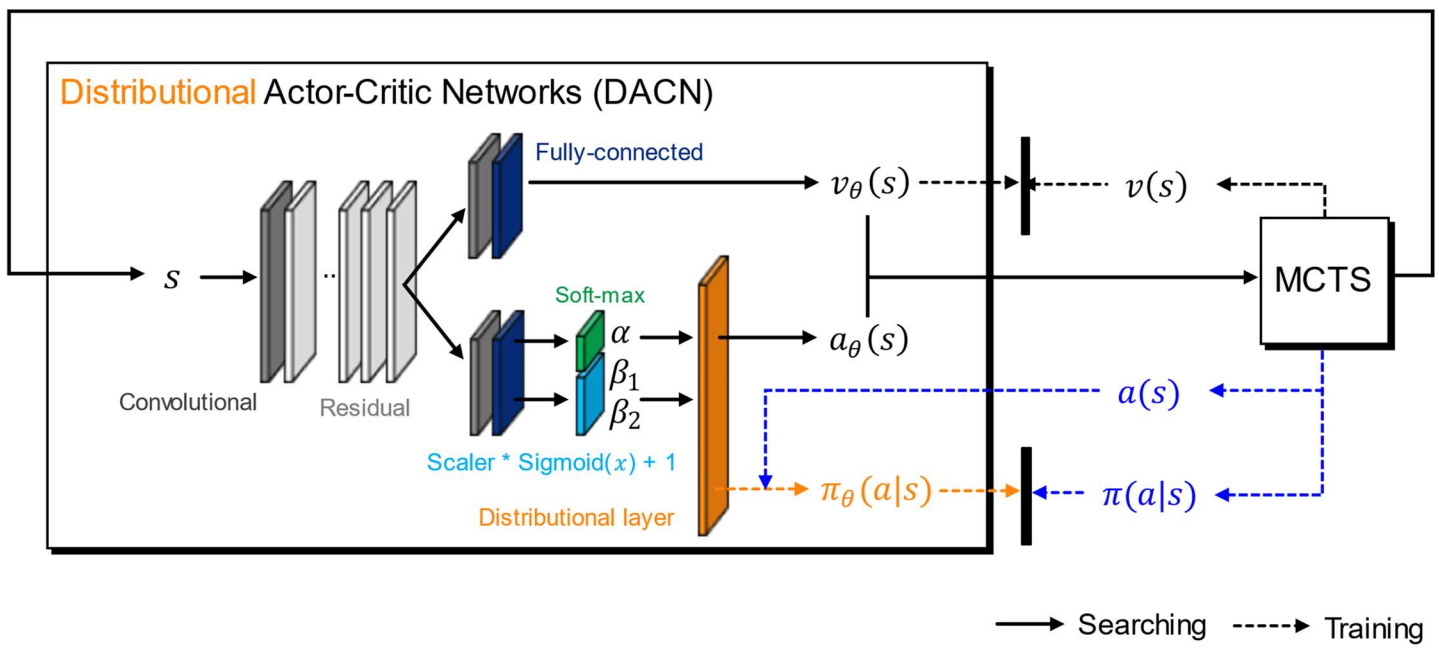

Figure 5.4. Behavioral cloning of DACN and MCTS.

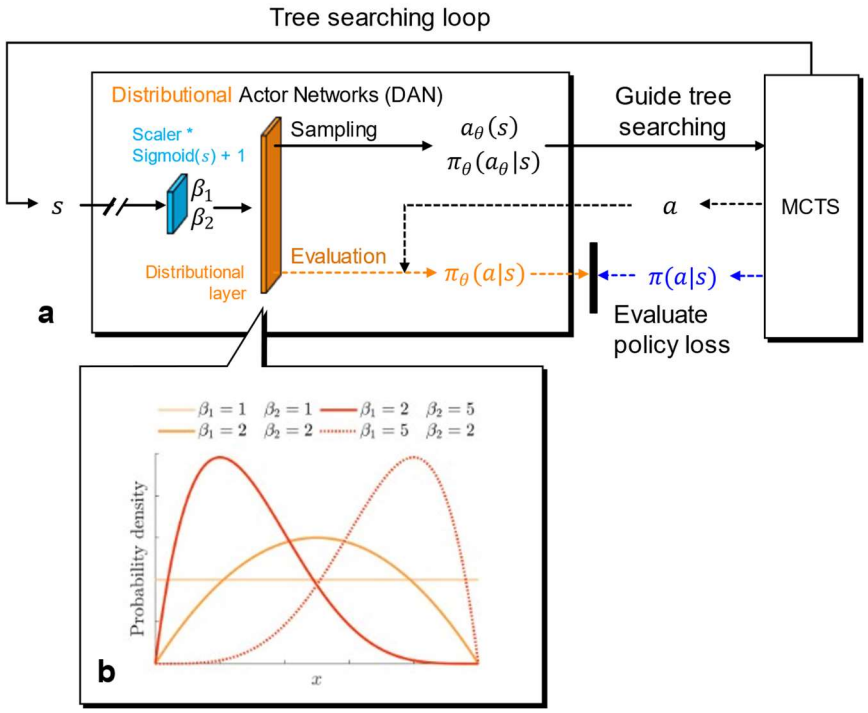

Figure 5.5 (a) Schematic of DAN and its interactions with MCTS. (b) Illustration of probability density of beta distribution.

restrictions (e.g., agitator speed of a reactor), and thereby we can choose probability distributions with bounded supports. One of such is Beta distribution given as

$$p(x|\,\beta_1,\,\beta_2) = \frac{\Gamma(\beta_1+\beta_2)}{\Gamma(\beta_1)\Gamma(\beta_2)} x^{\beta_1-1}(1-x)^{\beta_2-1} \tag{5.8}$$

where $x \in [0,1]$ and β_1, β_2 are hyperparameters that determine the shape of the distribution. Figure 5.5b illustrates the effect of the hyperparameter values.

The underlying rationale for using Beta distribution approximation is that the optimal policy at given state s would be given as a Dirac function, where its shape can be approximated by the Beta distribution with high β_1 and β_2 values. Moreover, the Beta distribution has finite support so it does not suffer from the boundary effect that is known to slow down the training progress [19]. The distribution approximation is also numerically differentiable [118], so we can use simply perform backpropagation to train the network, which enables rapid training.

Unlike the conventional approach that uses deterministic layers and additive stochastic processes (e.g., Ornstein-Uhlenbeck process [119]) to perturb the action in promoting exploration [120], [121], the resulting policy function, named distributional actor networks (DAN), is inherently stochastic and perform random exploration probabilistically. Furthermore, the exploitation/exploration rate can be autonomously controlled over the training. That is, we can initialize $\beta_1 = 1$ and $\beta_2 = 1$ to promote exploration in the earlier stage of the learning process. As the training progresses, the beta values are increased and the greedy actions

Algorithm 4 Action masking	
Input	G: graphical representation of the given process a: action candidate
Function	N: count the number of the nodes in the graph T: return the unit type number P: count the number of PRDT nodes in the graph F: count the number of FEED nodes in the graph L: count the number of NULL nodes in the graph
Output	V: flag indicating validity of the action

```
if N(G) == 0 and T(a) ≠ 1  # only FEED action allowed
    V = 0
elseif P(G) == 0
    V = 0
elseif F(G) ≠ 0 and T(a) == 1  # FEED already exists
    V = 0
elseif T(a) == 4  and L(G) == 0  # LINK selected
    V = 0                          but NULL does
end if                             not exist
```

are encouraged (see Figure 5.5b).

5.2.5. Action masking

To avoid repeatedly sampling invalid actions and promote rapid simulation, action masking [122] technique is employed as described in Algorithm 4.

5.3. Verification study

5.3.1. System description

The algorithm is demonstrated with a reactor-separator-recycle system synthesis problem, Case 4 of benzene chlorination process in [123]. The chemical reactions of this liquid-phase process are given as:

$$\begin{aligned} &C_6H_6\ (A) + Cl_2 \xrightarrow{k_1} C_6H_5Cl\ (B) + HCl \\ &C_6H_5Cl + Cl_2 \xrightarrow{k_2} C_6H_4Cl_2\ (C) + HCl \end{aligned} \tag{5.9}$$

where $k_1 = 0.412/h$ and $k_2 = 0.05/h$ are the kinetic constants. The hydrochloric acid produced is eliminated at the reaction level output by a stripping operation whose cost is not taken into account. In the separation level, unreacted A is separated and recycled toward the reactor network, valuable product B, of which the demand is assumed to be 50 kmol/h, and a side product C. The volatility ranking of these components is given as $\alpha_A > \alpha_B > \alpha_C$, and the possible separation tasks are defined as A/BC, AB/C, B/C, and A/B.

5.3.2. Experimental setup

To reduce the complexity of the task, we only considered two types of separation, namely DSTL1 (A/BC) and DSTL2 (AB/C) in the following case studies. Refer to A.3 for the model equations of the process units. The minimum and maximum possible operation conditions are specified as Table 5.2. Stopping the designing process is considered by defining additional action. As a result, a total of 11 environmental objects is defined in this study.

When applying MIX unit, we do not merge two PRDT nodes but instead create a NULL node and attached it to the input of MIX unit. The NULL node here is virtual node and prepared to consider possible recycle operation. The action RECY links a PRDT to a NULL.

The CAPEX and OPEX of the process units are given as in Table 5.3, and the reward is defined as the profit of the process given as

$$\text{Profit} = 720(S - P) - \text{Cost}$$

where

$$\begin{aligned} \text{Sales } (S) &= 92.67 \sum_{i=\text{PRDT}} F_{i,B} \\ \text{Purchase } (P) &= 47.86 \sum_{i=\text{FEE}} F_{i,A} \\ \text{Cost} &= \sum_{i=\text{UNIT}} \frac{\text{CAPEX}_i}{2.5} + 0.52 \cdot \text{OPEX}_i \end{aligned} \tag{5.10}$$

The hyperparameters for training, reward, MCTS, and DACN are given as in Table 5.4, and Wegstein method [124] is used to simulate the processes with process loops.

A total of four case studies have been conducted as below:

Table 5.2 Specifications for process units and streams.

Type	Specification	Unit	Min	Default	Max
FEED	Flowrate of A	kmor/hr	50	75	100
PRDT	-	-	-	-	-
NULL	-	-	-	-	-
LINK	Target NULL	-	0	0	1
MIX	-	-	-	-	-
SPLT	Split ratio	-	0.05	0.5	0.5
CSTR	Reactor volume	m^3	0.01	25	50
PFR	Reactor volume	m^3	0.01	25	50
DSTL1	-	-	-	-	-
DSTL2	-	-	-	-	-
F_A					
F_B	Flow rates	kmor/hr	0	0	10
F_C					

Table 5.3 Environmental objects defined and CAPEX and OPEX of the units.

	IsAction	IsUNIT	CAPEX [$]	OPEX [$]
NULL	0	1	-	-
PRDT	0	1	-	-
FEED	1	1	-	-
RECY	1	1	-	-
STOP	1	0	-	-
MIX	1	1	1,000	-
SPLT	1	1	1,000	-
CSTR	1	1	$25{,}794 + 8{,}178 \cdot V$	-
PFR	1	1	$3{,}895 + 49{,}332 \cdot V$	-
DSTL1	1	1	$\begin{cases} 132,718 + F_{In}(369 \cdot x_{In,A} - 1114 \cdot x_{In,B}) \text{ if } x_{In,C} \neq 0 \\ 86{,}944 + F_{In}(1136^2 \cdot x_{In,A}) \text{ if } x_{In,C} = 0 \end{cases}$	$\begin{cases} F_{In}(3 + 36 \cdot x_{In,A} + 7.7 \cdot x_{In,B}) \text{ if } x_{In,C} \neq 0 \\ F_{In}(11 + 28 \cdot x_{In,A}) \text{ if } x_{In,C} = 0 \end{cases}$
DSTL2	1	1	$\begin{cases} 211,547 - F_{In}(1010 \cdot x_{In,A} + 479 \cdot x_{In,B}) \text{ if } x_{In,A} \neq 0 \\ 25{,}000 + F_{In}(6984 \cdot x_{In,B} - 3870 \cdot x_{In,B}{}^2) \text{ if } x_{In,A} = 0 \end{cases}$	$\begin{cases} F_{In}(16 + 17 \cdot x_{In,A} + 42 \cdot x_{In,B}) \text{ if } x_{In,A} \neq 0 \\ F_{In}(26 + 29 \cdot x_{In,B}) \text{ if } x_{In,A} = 0 \end{cases}$

Table 5.4 Hyperparameter settings.

Type	Parameter	Value
Search space	Number of episodes	40,000
	Number of actions for each episode	9
State	$\overline{n}$	100
	L	20
	H_C	2
	H_S	3
Reward	γ	0.99
	r for penalty	-10^4
MCTS	N_{tree}	100
	$\overline{E}_{node}$	10
	c_{puct}	10^8
	c_{pw}	3
	κ	1.2
DACN	Hidden nodes per residual layer	10
	Number of the residual layers	19
	scaler_β	1000

- Case 1:

  solve the base case problem.

- Case 2 (transferability test 1):

  CAPEX of CSTR = 25,794 + 7,000V

- Case 3 (transferability test 2):

  CAPEX of CSTR = 8,000 + 10,000V

- Case 4 (hyperparameter sensitivity test):

  $c_{puct} = 10^6$

5.3.3. Result and discussions

Figure 5.6 describes the results obtained from Case 1. Note that the proposed method yields an almost identical process design to the ground truth stated in the reference. As intended, we can observe that the agent initially selects all the actions with almost the same probability and becomes to choose the optimal action (e.g., MIX at s_1) exclusively as training progresses. Table 5.5 details the policy of DACN at episode 33230 where the selectivity represents the 'sharpness' of the policy distribution. We can find a few notable observations as below:

- The DACN selects FEED action with a high probability at the initial state, presumably due to action masking.
- The second most optimal action for s_1 is found to be CSTR. This is plausible as we can speculate that the suboptimal solutions could be non-loop processes.
- At s_2, s_3, and s_4, the DACN selects DIST1 with the second-

highest probability and the selectivity to CSTR is getting lowered. This indicates that the suboptimal solutions may include early termination of the design process without the implementation of CSTRs.

- At s_6, the DACN has low selectivity on DSTL2, which reflects the redundancy of selecting DSTL2 and LINK at s_6.
- At s_7, the DACN selects STOP with a probability comparable to choosing LINK. This indicates that the suboptimal solutions may include non-loop processes.

Figure 5.7 shows the profit trajectories for the case studies. The result of Case 2 demonstrates that the proposed methodology can locate a high return solution faster at the early stages of solving a similar process design problem to that encountered in the training process. Such transferability seems also valid at the early stages in Case 3 but becomes to lose its property as the learning progresses. The results reveal that transferability is not always guaranteed, especially if the given problem is significantly different from the problem used in the training.

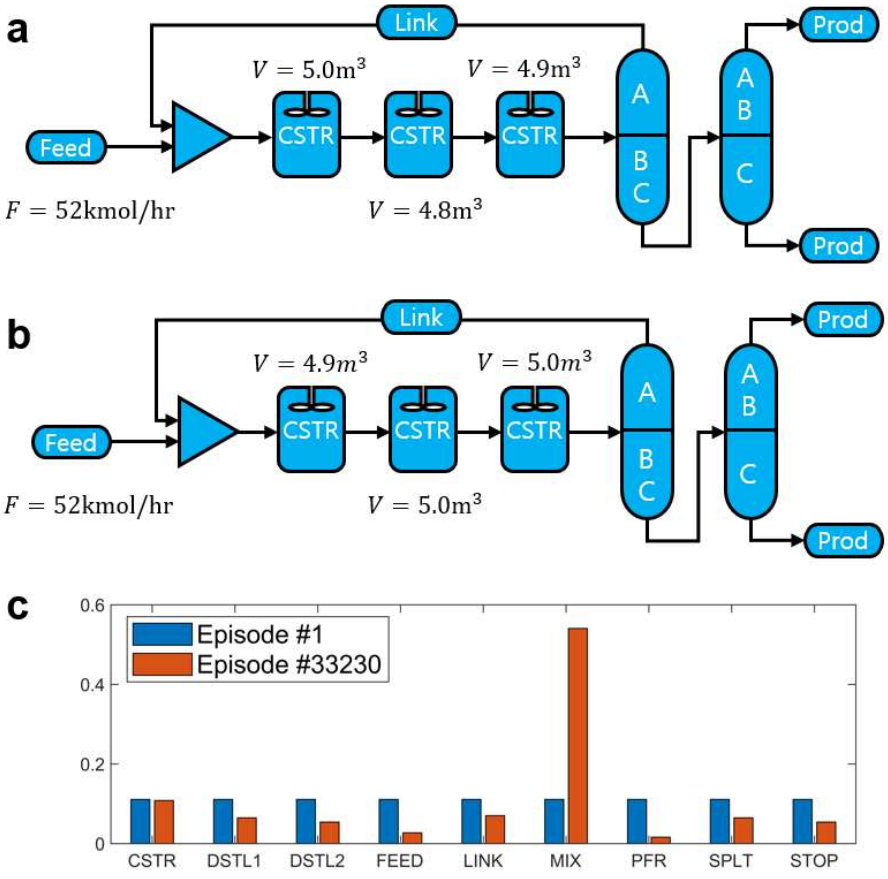

Figure 5.6 (a) Optimal solution obtained and (b) its ground truth. (c) The action policies for s_1 at episode 1 and 33230. The policies were calculated by normalizing 1,000 samples obtained from DACN.

Table 5.5 Policy of DACN at episode 33230. The policy was obtained by taking 1,000 samples from the DACN and normalizing them. a^* and a^{**} respectively denote the foremost and second most optimal actions.

			i								
		Note	0	1	2	3	4	5	6	7	8
Optimal action	$a^*_{i,\text{type}}$		FEED	MIX	CSTR	CSTR	CSTR	DSTL1	DSTL2	LINK	STOP
	$p(a^*_{i,\text{type}}\|s)$	A	0.85	0.54	0.77	0.67	0.63	0.57	0.41	0.4	0.31
	$a^*_{i,\text{spec}}$		52.0	-	4.88	4.97	5.04	-	-	0.13	-
	$p(a^*_{i,\text{spec}}\|s)$	E	0.9	-	0.78	0.83	0.88	-	-	0.99	-
	Unit		kmol /hr	-	m^3	m^3	m^3	-	-	-	-
Second most optimal action	$a^{**}_{i,\text{type}}$		STOP	CSTR	DSTL1	DSTL1	DSTL1	DSTL2	LINK	STOP	SPLT
	$p(a^{**}_{i,\text{type}}\|s)$	B	0.02	0.10	0.11	0.32	0.36	0.23	0.37	0.31	0.22
	A+B	C	0.87	0.64	0.88	0.99	0.99	0.8	0.78	0.71	0.53
Selectivity	A/C	D	0.97	0.84	0.88	0.67	0.63	0.71	0.53	0.56	0.58

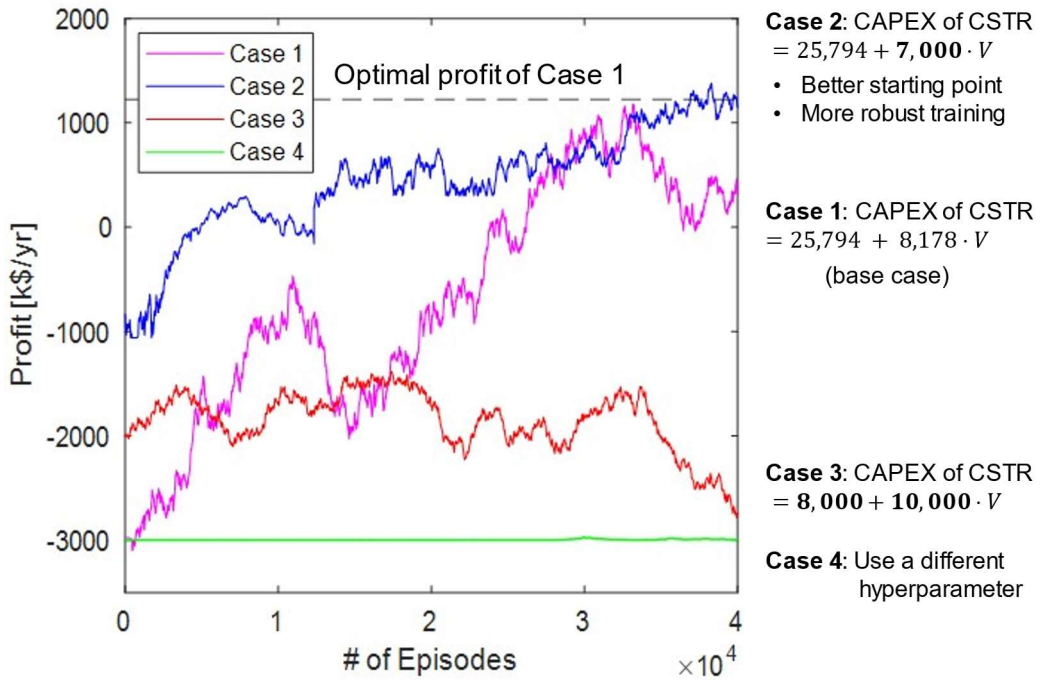

Figure 5.7 Profit trajectories for the case studies.

Chapter 6

Concluding remarks

6.1. Summary of the contributions

In this thesis, probabilistic machine learning methods have been developed to model and quantify the uncertainty that arises from the scarcity of the process data. The underlying theme was to approximate the target distribution via parameterized distributions and circumvent the computational complexity that occurred in probabilistic inference.

In Chapter 3, we proposed a process monitoring methodology using PML. The process data was assumed to be realized from a Riemannian manifold where data are clustered with respect to their classes. This manifold was approximated from measuring the pairwise likelihoods of the data points and projected into low-dimensional space by preserving the likelihoods. To be specific, a uniform manifold was first approximated with the metric attributes of the data and the local distances between the data points were calibrated. Then the non-metric attributes were utilized by imposing lower likelihood among those points. This postulated

clustered manifold and induced the data to be clustered in the feature space. In this way, those *hard-to-distinguish* fault points were able to be efficiently discriminated after projection, which drastically improved the monitoring performance. We have also shown that it has much more flexibility in tracing and controlling the resultant effect from incorporating prior information of the data (i.e., data labels), compared to the conventional statistical projection methods that have been popular choices in practice.

In Chapter 4, a probabilistic modeling approach using Bayesian deep neural networks was proposed, and its performance has been demonstrated by the application to the plasma etch process. Assuming that the parameters of the neural networks have a Gaussian distribution, a stochastic model was derived by learning the distribution of the data. After training the model, the optimal model structure and parameters were obtained by *a posteriori* elimination of the parameters having lower importance. The resulting sparse Bayesian networks were found to preserve the prediction accuracy with respect to the *full* Bayesian networks even though we eliminate 90% of the weights. We have shown that it is not only possible to simultaneously learn the complex behavior and stochastic characteristics of the processes, but also to obtain an optimal model structure. Note that the resulting architecture is differentiable, that is, the Jacobian and Hessian matrices of the system model can also be easily obtained by automatic differentiation. These suggest that the proposed methodology can be utilized for designing model-based control systems.

In Chapter 5, a reinforcement learning-based process design

framework was proposed with distributional actor-critic network. Unlike the conventional approach, which only employs recursive objective function evaluation to find the optimal solution, the proposed approach approximates the objective function spaces with distributional deep neural networks. We demonstrated that the process design heuristics can be learned and transferred to solve similar design problems, rather than simply finding a solution as conventional approaches have done.

6.2. Future works

The directions for the future work can be made as follows:

- The PML proposed in Chapter 3 was devised to obtain the same low-dimensional representation as CMAP, while gaining computational efficiency by mapping the input and latent spaces using neural networks. However, as the results reveal, PML had difficulty in discriminating *hard-to-distinguish* two different class data. Note that the only difference between the algorithms between PML and CMAP was that CMAP utilizes pointwise optimization based on nearest neighbors while PML performs transformative mapping using neural networks. Therefore, we can infer that the performance degradation found in PML is stemmed from using neural architecture, which is well known to suffer from the high dependency on initial values and convergence to local optimum. A possible solution could be introducing automated machine learning techniques [125] to find out the optimal neural architecture and a hyperparameter set that are tailored to the given dataset.

In PML, we utilized the autoencoder networks and derived an inverse mapping from the latent space to the input space, which can generate synthetic data through simple sampling. Therefore, if there is an imbalance in the amount of data among the classes, we can utilize it to oversample the minor data classes and improve the diagnostic performance. Note that the proposed methods are based on that the local distances between the data points are calibrated to be within the range of [0, 5] by the uniform manifold approximation. As we have only demonstrated such property empirically through various datasets, theoretical prove should be made.

- Even though the probabilistic modeling technique presented in Chapter 4 provides uncertainty measures for the predictions, we can still raise fundamental questions on its robustness to the extrapolation. A possible solution for this may be using a first-principles model and applying the same modeling techniques – reparameterization and variational inference – to the model parameters. Some studies have already shown that the variational inference on physical parameters also works for the dynamical systems described by ordinary differential equations [126]. In this case, we can further anticipate that the proposed model compression technique can be used in discovering the governing equations of the system. To be specific, we can first construct a model library [127], which is composed of possible candidates of physicochemical terms, and then perform *a posteriori* model compression to eliminate redundant terms. Contrary to the existing approach that

employs LASSO (L1 regularization) [128] for the model compression, the suggested approach could be able to produce stochastic models and attain better accuracy (see Figure 4.9).

- The reinforcement learning-based process design approach presented in Chapter 5 are having limitations as much as the described potential benefits. First, it was observed that it suffers from the instability and hyperparameter sensitivity during the training, which are well known problems of reinforcement learning. Even though many solution approaches such as *recovering* [129] and *averaging* [130] are proposed, the issue has not solved yet and being the biggest drawback against the conventional mathematical programming approach. Second, the generalization of the problem-dependent process constraints and that of the state representation for varying number of process units (number of actions) have not been developed. Promising solution strategies for these may be employing reward shaping [131] and hierarchical graph convolutions [132], respectively. Finally, note that the proposed method based on policy approximation using distributed neural networks. That is, if a large number of actions should be taken into account, *distribution flattening* occurs, and it can slow down the learning process. Introducing hierarchically structured policy networks [133] could be a fundamental solution to this problem, but a more promising approach would be directly incorporating the domain knowledge into the agent [109].

Appendix

A.1. Proof of Lemma 1

Let x^1, …, x^n be the coordinate system for the ambient space. A ball B in $\mathcal{M}$ under Riemannian metric g has volume given by

$$\int_B \sqrt{\det(g)}\, dx^1 \ldots dx^n = \sqrt{\det(g)}\frac{\pi^{n/2} r^n}{\Gamma(n/2+1)} \tag{A.1}$$

If we fix the volume of the ball to be $\frac{\pi^{n/2} r^n}{\Gamma(n/2+1)}$, we arrive at the requirement that

$$\det(g) = \frac{1}{r^{2n}} \tag{A.2}$$

and since g is assumed to be diagonal matrix, we can solve for g as

$$g_{ij} = \begin{cases} \frac{1}{r^2} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{A.3}$$

The geodesic distance on $\mathcal{M}$ under g from p to q where $p, q \in B$ is defined as

$$\inf_{c \in C} \int_a^b \sqrt{g\big(\dot{c}(t), \dot{c}(t)\big)} dt \tag{A.4}$$

where C is the class of smooth curves c on $\mathcal{M}$ such that $c(a) = p$ and $c(b) = q$, and $\dot{c}$ denotes the first derivative of c on $\mathcal{M}$. Given that g is as defined in (A.3), we can derive that this can be simplified to

$$\frac{1}{r}\inf_{c\in C}\int_a^b \sqrt{\langle \dot{c}(t),\dot{c}(t)\rangle}dt = \frac{1}{r}\inf_{c\in C}\int_a^b \langle \|\dot{c}(t),\dot{c}(t)\|\rangle dt = \frac{1}{r}\, d_{\mathbb{R}^n}(p,q) \tag{A.5}$$

■

A.2. Performance indices for dimension reduction

A total of four performance indices [65], [134], [135] were used to evaluate the clustering degree and data structure preservation. To measure the clustering degree, we introduced two clustering performance measures; the Dunn index (DI) and Davies-Bouldin index (DBI). These were originally designed to evaluate the performance of clustering algorithms, where the labeling is assessed based on the distances between given data points. Conversely, we used them to evaluate how well the data points were clustered in a situation where data labels are known *a priori*. We used Sammon's error (SE) and the local continuity meta-criterion (LCMC) to measure how well the global and local structures of the dataset in the input space were preserved in the feature space.

Dunn Index (DI)

For given clusters $\{C_k\}_{1\le k\le K}$ DI is defined as follows:

$$\mathrm{DI} = \frac{\min\limits_{i\neq j} \Delta(C_i, C_j)}{\max\limits_{1\le k\le K} \delta(C_k)} \tag{A.6}$$

where $\Delta(C_i, C_j)$ is the distance between clusters C_i and C_j, and $\delta(C_k)$ represents the size or diameter of a cluster C_k that can be defined in many different ways. We defined $\Delta(C_k)$ as the maximum distance between points in the cluster k:

$$\delta(C_k) = \max_{x_i, x_j \in C_k} d(x_i, x_j) \tag{A.7}$$

As shown in (A.6), DI measures whether the clusters are compact and well separated, and a larger DI value indicates better clustering.[136]

Davies-Bouldin index (DBI)

DBI measures the average separability by considering every pair of clusters, and a smaller DBI value indicates better clustering.

$$\text{DBI} = \frac{1}{K}\sum_{i=1}^{K} \underset{j \neq i}{max}\left(\frac{\delta(C_i) + \delta(C_j)}{\Delta(C_i, C_j)}\right) \tag{A.8}$$

where C_i, $\Delta(C_i)$, and $\delta(C_i, C_j)$ are defined the same as in DI.

Sammon's error (SE)

SE, which is also referred to as Sammon's stress, measures the degree of preservation of the global structure using the distances between data points in the observation and latent spaces.[137]

$$\text{SE} = \frac{1}{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} d_{ij}} \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{\left(d_{ij} - d_{ij}^{*}\right)^2}{d_{ij}} \tag{A.9}$$

where $d_{ij} = d_{\mathbb{R}^n}\left(x_i, x_j\right)$, $d_{ij}^{*} = d_{\mathbb{R}^m}\left(z_i, z_j\right)$, and an SE value of 0 indicates a perfect reduction.

Local continuity meta-criterion (LCMC)

LCMC measures the degree of preservation of the local structure based on a co-ranking matrix.[138], [139] First, the rank r_{ij} of x_i with respect to x_j is defined as

$$r_{ij} = \left|\{k: d_{ik} < d_{ij} \; or \; \left(d_{ik} = d_{ij} \; and \; k < j\right)\right| \tag{A.10}$$

where $|A|$ denotes the number of elements in the set A. Here, r_{ij} is an

integer indicating that x_i is the $r_{ij}{}^{\text{th}}$ closest neighbor of x_j. Analogously, the rank in the feature space is defined as

$$r_{ij}^* = \left|\left\{k: d_{ik}^* < d_{ij}^* \; or \left(d_{ik}^* = d_{ij}^* \; and \; k < j\right)\right\}\right| \tag{A.11}$$

The co-ranking matrix, whose elements are given by

$$q_{ij} = |\{(k,l): r_{kl}^* = i \; and \; r_{kl} = j\}| \tag{A.12}$$

represents the 2-dimensional histogram of the changes in ranks, i.e. q_{ij} is an integer that counts how many points of distance rank j are ranked i. Thus, if an embedding is carried out by a perfect reduction, the co-ranking matrix will only have non-zero entries on the diagonal. If most of the non-zero entries are in the lower triangle, we can interpret this as an embedding collapsed distant points onto each other, and vice versa.[140] The number of points belonging to the k^{th} nearest neighbors in both the observation space and feature space is computed as

$$\text{LC}(k) = \frac{1}{kN} \sum_{i=1}^{k} \sum_{j=1}^{k} q_{ij} \tag{A.13}$$

where N is the number of data points. Intuitively, this represents the degree of overlap between the neighboring sets of a data point and their corresponding embedding. LCMC adjusts this value by subtracting the expected overlap between two subsets of k elements from $N - 1$.

$$\text{LCMC}(k) = \text{LC}(k) - \frac{k}{N-1} \tag{A.14}$$

The higher the LCMC value, the better the local structure preservation, and an LCMC value of 1 indicates perfect preservation.

A.3. Model equations for process units

LINK

$$\left(F_{\text{out,A}},\ \ F_{\text{out,B}},\ \ F_{\text{out,C}}\right) = \left(F_{\text{in,A}},\ \ F_{\text{in,B}},\ \ F_{\text{in,C}}\right) \tag{A.15}$$

MIX

$$\begin{aligned}\left(F_{\text{out,A}},\ \ F_{\text{out,B}},\ \ F_{\text{out,C}}\right) = {} & \left(F_{\text{in1,A}},\ \ F_{\text{in1,B}},\ \ F_{\text{in1,C}}\right) \\ & + \left(F_{\text{in2,A}},\ \ F_{\text{in2,B}},\ \ F_{\text{in2,C}}\right)\end{aligned} \tag{A.16}$$

SPLT

For a split ratio r,

$$\begin{aligned}&\left(F_{\text{out1,A}},\ \ F_{\text{out1,B}},\ \ F_{\text{out1,C}}\right)\cdot r + \\ &\left(F_{\text{out2,A}},\ \ F_{\text{out2,B}},\ \ F_{\text{out2,C}}\right)\cdot (1-r) = \left(F_{\text{in,A}},\ \ F_{\text{in,B}},\ \ F_{\text{in,C}}\right)\end{aligned} \tag{A.17}$$

CSTR

For a reactor volume V,

$$F_{in} = \sum F_{\text{in},i} \quad \text{where } \ i \in \{\text{A, B, C}\} \tag{A.18}$$

If $F_{in} \neq 0$:

$$X_{\text{in},i} = \frac{F_{\text{in},i}}{\sum F_{\text{in},i}} \quad \text{for } \ i \in \{\text{A, B, C}\} \tag{A.19}$$

$$\bar{X} = 11.22 \cdot X_{\text{out,A}} + 9.86 \cdot X_{\text{out,B}} + 8.85 \cdot X_{\text{out,C}} \tag{A.20}$$

$$X_{\text{out,A}} = X_{\text{in,A}} \cdot \frac{\sum F_{\text{in},i}}{\sum F_{\text{in},i} + k_1 V \bar{X}} \tag{A.21}$$

$$X_{\text{out,B}} = \frac{X_{\text{in,B}} \sum F_{\text{in},i} + k_1 X_{\text{out,A}} V \bar{X}}{\sum F_{\text{in},i} + k_2 V \bar{X}} \tag{A.22}$$

$$X_{\text{out,C}} = \frac{X_{\text{in,C}} \sum F_{\text{in},i} + k_2 X_{\text{out,B}} V \bar{X}}{\sum F_{\text{in},i}} \tag{A.23}$$

Else:

$$F_{\text{out},i} = 0 \quad \text{for } i \in \{\text{A}, \text{B}, \text{C}\} \tag{A.24}$$

PFR

For a reactor volume V,

$$F_{in} = \sum F_{\text{in},i} \quad \text{where } \; i \in \{\text{A}, \text{B}, \text{C}\} \tag{A.25}$$

If $F_{in} \neq 0$:

$$\bar{X} = 11.22 \cdot X_{\text{out,A}} + 9.86 \cdot X_{\text{out,B}} + 8.85 \cdot X_{\text{out,C}} \tag{A.26}$$

$$\frac{dF_{\text{out},i}}{dV} = \begin{cases} -k_1 \bar{X} X_{\text{out,A}} & \text{for } i = A \\ k_1 \bar{X} X_{\text{out,A}} + k_2 \bar{X} X_{\text{out,B}} & \text{for } i = B \\ k_2 \bar{X} X_{\text{out,B}} & \text{for } i = C \end{cases} \tag{A.27}$$

Else:

$$F_{\text{out},i} = 0 \quad \text{for } i \in \{\text{A}, \text{B}, \text{C}\} \tag{A.28}$$

DSTL1

$$\begin{aligned} \left(F_{\text{out1,A}}, \;\; F_{\text{out1,B}}, \;\; F_{\text{out1,C}}\right) &= \left(F_{\text{in,A}}, \;\; 0, \;\; 0\right) \\ \left(F_{\text{out2,A}}, \;\; F_{\text{out2,B}}, \;\; F_{\text{out2,C}}\right) &= \left(0, \;\; F_{\text{in,B}}, \;\; F_{\text{in,C}}\right) \end{aligned} \tag{A.29}$$

DSTL2

$$\begin{aligned} \left(F_{\text{out1,A}}, \;\; F_{\text{out1,B}}, \;\; F_{\text{out1,C}}\right) &= \left(F_{\text{in,A}}, \;\; F_{\text{in,B}}, \;\; 0\right) \\ \left(F_{\text{out2,A}}, \;\; F_{\text{out2,B}}, \;\; F_{\text{out2,C}}\right) &= \left(0, \;\; 0, \;\; F_{\text{in,C}}\right) \end{aligned} \tag{A.30}$$

Bibliography

[1] E. Hüllermeier and W. Waegeman, “Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods,” *Mach. Learn.*, vol. 110, no. 3, pp. 457–506, Mar. 2021.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2006.

[3] H. Hotelling, “Analysis of a complex of statistical variables into principal components.,” *J. Educ. Psychol.*, vol. 24, no. 6, pp. 417–441, 1933.

[4] K. Pearson, “On lines and planes of closest fit to systems of points in space,” *London, Edinburgh, Dublin Philos. Mag. J. Sci.*, vol. 2, no. 11, pp. 559–572, Nov. 1901.

[5] S. W. Choi, J. H. Park, and I.-B. Lee, “Process monitoring using a Gaussian mixture model via principal component analysis and discriminant analysis,” *Comput. Chem. Eng.*, vol. 28, no. 8, pp. 1377–1387, Jul. 2004.

[6] R. A. FISHER, “THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS,” *Ann. Eugen.*, vol. 7, no. 2, pp. 179–188, Sep. 1936.

[7] Z. B. Zhu and Z. H. Song, “A novel fault diagnosis system using pattern classification on kernel FDA subspace,” *Expert Syst. Appl.*, vol. 38, no. 6, pp. 6895–6905, 2011.

[8] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, “An introduction to MCMC for machine learning,” *Mach. Learn.*, vol. 50, no. 1, pp. 5–43, 2003.

[9] L. V. Jospin, W. Buntine, F. Boussaid, H. Laga, and M. Bennamoun,

"Hands-on Bayesian Neural Networks -- a Tutorial for Deep Learning Users," vol. 1, no. 1, pp. 1–35, Jul. 2020, [Online]. Available: http://arxiv.org/abs/2007.06823.

[10] K. McBride and K. Sundmacher, "Overview of Surrogate Modeling in Chemical Process Engineering," *Chemie Ing. Tech.*, vol. 91, no. 3, pp. 228–239, Mar. 2019.

[11] D. M. Himmelblau, "Applications of artificial neural networks in chemical engineering," *Korean J. Chem. Eng.*, vol. 17, no. 4, pp. 373–392, Jul. 2000.

[12] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proc. Natl. Acad. Sci.*, vol. 113, no. 15, pp. 3932–3937, 2016.

[13] M. Raissi, "Deep hidden physics models: Deep learning of nonlinear partial differential equations," *J. Mach. Learn. Res.*, vol. 19, pp. 1–24, 2018.

[14] L. T. Biegler and I. E. Grossmann, "Retrospective on optimization," *Comput. Chem. Eng.*, vol. 28, no. 8, pp. 1169–1192, 2004.

[15] Q. Chen and I. E. Grossmann, "Recent Developments and Challenges in Optimization-Based Process Synthesis," *Annu. Rev. Chem. Biomol. Eng.*, vol. 8, no. 1, pp. 249–283, Jun. 2017.

[16] M. C. Aguitoni, L. V. Pavão, P. H. Siqueira, L. Jiménez, and M. A. da S. S. Ravagnani, "Heat exchanger network synthesis using genetic algorithm and differential evolution," *Comput. Chem. Eng.*, vol. 117, pp. 82–96, Sep. 2018.

[17] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: a review of algorithms and comparison of software implementations," *J.*

Glob. Optim., vol. 56, no. 3, pp. 1247–1293, Jul. 2013.

[18] Y. Tang and S. Agrawal, "Discretizing Continuous Action Space for On-Policy Optimization," 2019, [Online]. Available: http://arxiv.org/abs/1901.10500.

[19] P. W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," *34th Int. Conf. Mach. Learn. ICML 2017*, vol. 2, pp. 1386–1396, 2017.

[20] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," *NIPS Proc.*, pp. 2960–2968, Jun. 2012, [Online]. Available: http://arxiv.org/abs/1206.2944.

[21] F. Torabi, G. Warnell, and P. Stone, "Behavioral Cloning from Observation," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Jul. 2018, vol. 2018-July, no. July, pp. 4950–4957.

[22] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *International conference on computers and games*, 2006, pp. 72–83.

[23] T. Rainforth, "Automating Inference, Learning and Design using Probabilistic Programming," University of Oxford, 2017.

[24] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, Mar. 1951.

[25] A. Graves, "Practical Variational Inference for Neural Networks.," *Nips*, pp. 1–9, 2011, [Online]. Available: https://papers.nips.cc/paper/4329-practical-variational-inference-for-neural-networks.pdf.

[26] G. E. Hinton and D. van Camp, “Keeping the neural networks simple by minimizing the description length of the weights,” in *Proceedings of the sixth annual conference on Computational learning theory - COLT ’93*, 1993, pp. 5–13.

[27] R. M. Neal and G. E. Hinton, “A view of the EM algorithm that justifies incremental, sparse, and other variants,” in *Learning in graphical models*, 1998, pp. 355–368.

[28] L. K. Saul, T. Jaakkola, and M. I. Jordan, “Mean Field Theory for Sigmoid Belief Networks,” *J. Artif. Intell. Res.*, vol. 4, pp. 61–76, Mar. 1996.

[29] L. McInnes, J. Healy, and J. Melville, “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction,” *arXiv Prepr. arXiv1802.03426*, Feb. 2018, [Online]. Available: http://arxiv.org/abs/1802.03426.

[30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Bradford Books, 2018.

[31] J. Yu, “Local and Nonlocal Preserving Projection for Bearing Defect Classification and Performance Assessment,” *IEEE Trans. Ind. Electron.*, vol. 59, no. 5, pp. 2363–2376, May 2012.

[32] Y. Lei, F. Jia, J. Lin, S. Xing, and S. X. Ding, “An Intelligent Fault Diagnosis Method Using Unsupervised Feature Learning Towards Mechanical Big Data,” *IEEE Trans. Ind. Electron.*, vol. 63, no. 5, pp. 3137–3147, 2016.

[33] B. Song and H. Shi, “Fault Detection and Classification Using Quality-Supervised Double-Layer Method,” *IEEE Trans. Ind. Electron.*, vol. 65, no. 10, pp. 8163–8172, Oct. 2018.

[34] J. Zhu, Z. Ge, and Z. Song, “HMM-driven Robust Probabilistic

Principal Component Analyzer for Dynamic Process Fault Classification," *IEEE Trans. Ind. Electron.*, vol. 62, no. 6, pp. 3814–3821, 2015.

[35] R. Razavi-Far *et al.*, "Information Fusion and Semi-Supervised Deep Learning Scheme for Diagnosing Gear Faults in Induction Machine Systems," *IEEE Trans. Ind. Electron.*, vol. 66, no. 8, pp. 6331–6342, Aug. 2019.

[36] T. Ko and H. Kim, "Fault Classification in High-Dimensional Complex Processes Using Semi-Supervised Deep Convolutional Generative Models," *IEEE Trans. Ind. Informatics*, vol. 16, no. 4, pp. 2868–2877, Apr. 2020.

[37] L. Ma, J. Dong, and K. Peng, "A Novel Robust Semisupervised Classification Framework for Quality-Related Coupling Faults in Manufacturing Industries," *IEEE Trans. Ind. Informatics*, vol. 16, no. 5, pp. 2946–2955, May 2020.

[38] Q. P. He, J. Wang, and D. Shah, "Feature space monitoring for smart manufacturing via statistics pattern analysis," *Comput. Chem. Eng.*, vol. 126, pp. 321–331, Jul. 2019.

[39] K. Zhong, J. Li, J. Wang, and M. Han, "Fault Detection for Marine Diesel Engine Using Semi-supervised Principal Component Analysis," in *2019 9th International Conference on Information Science and Technology (ICIST)*, Aug. 2019, pp. 146–151.

[40] S. Zhong, Q. Wen, and Z. Ge, "Semi-supervised Fisher discriminant analysis model for fault classification in industrial processes," *Chemom. Intell. Lab. Syst.*, vol. 138, pp. 203–211, Nov. 2014.

[41] J. Yu, "Localized Fisher discriminant analysis based complex chemical process monitoring," *AIChE J.*, vol. 57, no. 7, pp. 1817–1828, Jul. 2011.

[42] Z. Ge, S. Zhong, and Y. Zhang, “Semisupervised Kernel Learning for FDA Model and its Application for Fault Classification in Industrial Processes,” *IEEE Trans. Ind. Informatics*, vol. 12, no. 4, pp. 1403–1411, Aug. 2016.

[43] X. Zeng, S.-B. Yin, Y. Guo, J.-R. Lin, and J.-G. Zhu, “A Novel Semi-Supervised Feature Extraction Method and Its Application in Automotive Assembly Fault Diagnosis Based on Vision Sensor Data,” *Sensors*, vol. 18, no. 8, p. 2545, Aug. 2018.

[44] C. M. Alaíz, M. Fanuel, and J. A. K. Suykens, “Convex Formulation for Kernel PCA and Its Use in Semisupervised Learning,” *IEEE Trans. Neural Networks Learn. Syst.*, vol. 29, no. 8, pp. 3863–3869, Aug. 2018.

[45] J.-D. Shao, G. Rong, and J. M. Lee, “Learning a data-dependent kernel function for KPCA-based nonlinear process monitoring,” *Chem. Eng. Res. Des.*, vol. 87, no. 11, pp. 1471–1480, Nov. 2009.

[46] S. Goldrick, C. A. Duran-Villalobos, K. Jankauskas, D. Lovett, S. S. Farid, and B. Lennox, “Modern day monitoring and control challenges outlined on an industrial-scale benchmark fermentation process,” *Comput. Chem. Eng.*, vol. 130, p. 106471, Nov. 2019.

[47] J. Hedengren, “Yeast Fermentation Bioreactor for Ethanol Production.” MATLAB Central File Exchange, 2021.

[48] M. Belkin and P. Niyogi, “Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering,” in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, 2001, pp. 585–591.

[49] J. Tang, J. Liu, M. Zhang, and Q. Mei, “Visualizing Large-scale and High-dimensional Data,” in *Proceedings of the 25th International Conference on World Wide Web*, Apr. 2016, pp. 287–297.

[50] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in *Advances in Neural Information Processing Systems*, 2013, vol. 26.

[51] G. E. Hinton, "Reducing the Dimensionality of Data with Neural Networks," *Science (80-.).*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[52] J. J. Downs and E. F. Vogel, "A plant-wide industrial process control problem," *Comput. Chem. Eng.*, vol. 17, no. 3, pp. 245–255, Mar. 1993.

[53] T. Larsson, K. Hestetun, E. Hovland, and S. Skogestad, "Self-Optimizing Control of a Large-Scale Plant: The Tennessee Eastman Process," *Ind. Eng. Chem. Res.*, vol. 40, no. 22, pp. 4889–4901, Oct. 2001.

[54] A. Bathelt, N. L. Ricker, and M. Jelali, "Revision of the Tennessee Eastman Process Model," *IFAC-PapersOnLine*, vol. 48, no. 8, pp. 309–314, 2015.

[55] F. Capaci, E. Vanhatalo, M. Kulahci, and B. Bergquist, "The revised Tennessee Eastman process simulator as testbed for SPC and DoE methods," *Qual. Eng.*, vol. 31, no. 2, pp. 212–229, Apr. 2019.

[56] J.-M. Lee, C. Yoo, S. W. Choi, P. A. Vanrolleghem, and I.-B. Lee, "Nonlinear process monitoring using kernel principal component analysis," *Chem. Eng. Sci.*, vol. 59, no. 1, pp. 223–234, Jan. 2004.

[57] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2625, Nov. 2008.

[58] S. T. Roweis, "Nonlinear Dimensionality Reduction by Locally Linear Embedding," *Science (80-.).*, vol. 290, no. 5500, pp. 2323–2326, Dec.

2000.

[59] J. B. Tenenbaum, V. de Silva, and J. C. Langford, “A Global Geometric Framework for Nonlinear Dimensionality Reduction,” *Science (80-.).*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000.

[60] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, “Semi-Supervised Learning with Deep Generative Models,” Jun. 2014, [Online]. Available: http://arxiv.org/abs/1406.5298.

[61] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014, no. Ml, pp. 1–14.

[62] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K. R. Mullers, “Fisher discriminant analysis with kernels,” in *Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop (Cat. No.98TH8468)*, 1999, pp. 41–48.

[63] B. Schölkopf, A. Smola, and K.-R. Müller, “Nonlinear Component Analysis as a Kernel Eigenvalue Problem,” *Neural Comput.*, vol. 10, no. 5, pp. 1299–1319, Jul. 1998.

[64] U. Maulik and S. Bandyopadhyay, “Performance evaluation of some clustering algorithms and validity indices,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 12, pp. 1650–1654, Dec. 2002.

[65] A. Gracia, S. González, V. Robles, and E. Menasalvas, “A methodology to compare Dimensionality Reduction algorithms in terms of loss of quality,” *Inf. Sci. (Ny).*, vol. 270, pp. 1–27, Jun. 2014.

[66] D. M. Manos and D. L. Flamm, *Plasma Etching: An Introduction*. Academic Press, 1989.

[67] V. M. Donnelly and A. Kornblit, “Plasma etching: Yesterday, today, and tomorrow,” *J. Vac. Sci. Technol. A Vacuum, Surfaces, Film.*, vol.

31, no. 5, p. 050825, Sep. 2013.

[68] K. J. McLaughlin, S. W. Butler, T. F. Edgar, and I. Trachtenberg, "Development of Techniques for Real-Time Monitoring and Control in Plasma Etching: I . Response Surface Modeling of and Etching of Silicon and Silicon Dioxide," *J. Electrochem. Soc.*, vol. 138, no. 3, pp. 789–799, Mar. 1991.

[69] K. A. Alshaltami and S. Daniels, "Investigation of etching optimization in capacitively coupled SF 6 –O 2 plasma," *AIP Adv.*, vol. 9, no. 3, p. 035047, Mar. 2019.

[70] P. Schoenborn, R. Patrick, and H. Baltes, "Numerical simulation of a CF 4/O 2 plasma and correlation with spectroscopic and etch rate data," *J. Electrochem.*, 1989, Accessed: Aug. 20, 2017. [Online]. Available: http://jes.ecsdl.org/content/136/1/199.short.

[71] D. Ha, D. Park, J. Koo, K. H. Baek, and C. Han, "Improvement of principal component analysis modeling for plasma etch processes through discrete wavelet transform and automatic variable selection," *Comput. Chem. Eng.*, vol. 94, pp. 362–369, Nov. 2016.

[72] S. A. Lynn, N. MacGearailt, and J. V. Ringwood, "Real-time virtual metrology and control of etch rate in an industrial plasma chamber," in *2012 IEEE International Conference on Control Applications*, Oct. 2012, no. Li, pp. 1658–1663.

[73] B. Keville, C. Gaman, Y. Zhang, A. M. Holohan, M. M. Turner, and S. Daniels, "Attenuation of wall disturbances in an electron cyclotron resonance oxygen–argon plasma using real time control," *J. Vac. Sci. Technol. A Vacuum, Surfaces, Film.*, vol. 32, no. 4, p. 041301, Jul. 2014.

[74] J. Koo, "Design of Adaptive Model Predictive Controller for Plasma Etching Reactor," Seoul National University, 2019.

[75] J. Koo *et al.*, “Design of optical emission spectroscopy based plasma parameter controller for real-time advanced equipment control,” *Comput. Chem. Eng.*, vol. 100, pp. 38–47, May 2017.

[76] B. Keville, Y. Zhang, C. Gaman, A. M. Holohan, S. Daniels, and M. M. Turner, “Real-time control of electron density in a capacitively coupled plasma,” *J. Vac. Sci. Technol. A Vacuum, Surfaces, Film.*, vol. 31, no. 3, p. 031302, May 2013.

[77] V. Botelho, J. O. Trierweiler, M. Farenzena, and R. Duraiski, “Methodology for Detecting Model–Plant Mismatches Affecting Model Predictive Control Performance,” *Ind. Eng. Chem. Res.*, vol. 54, no. 48, pp. 12072–12085, Dec. 2015.

[78] A. S. Badwe, R. S. Patwardhan, S. L. Shah, S. C. Patwardhan, and R. D. Gudi, “Quantifying the impact of model-plant mismatch on controller performance,” *J. Process Control*, vol. 20, no. 4, pp. 408–425, 2010.

[79] R. Khare, A. Srivastava, and V. M. Donnelly, “Interactions of chlorine plasmas with silicon chloride-coated reactor walls during and after silicon etching,” *J. Vac. Sci. Technol. A Vacuum, Surfaces, Film.*, vol. 30, no. 5, p. 051306, Sep. 2012.

[80] A. Agarwal and M. J. Kushner, “Seasoning of plasma etching reactors: Ion energy distributions to walls and real-time and run-to-run control strategies,” *J. Vac. Sci. Technol. A Vacuum, Surfaces, Film.*, vol. 26, no. 3, pp. 498–512, May 2008.

[81] G. Cunge, B. Pelissier, O. Joubert, R. Ramos, and C. Maurice, “New chamber walls conditioning and cleaning strategies to improve the stability of plasma processes,” *Plasma Sources Sci. Technol.*, vol. 14, no. 3, pp. 599–609, Aug. 2005.

[82] D. J. Economou, “Modeling and simulation of plasma etching reactors

for microelectronics," *Thin Solid Films*, vol. 365, no. 2, pp. 348–367, Apr. 2000.

[83] F. Hamaoka, T. Yagisawa, and T. Makabe, "Modeling of Si etching under effects of plasma molding in two-frequency capacitively coupled plasma in SF6/O2 for MEMS fabrication," *IEEE Trans. Plasma Sci.*, vol. 35, no. 5 I, pp. 1350–1358, 2007.

[84] T. W. Kim and E. S. Aydil, "Effects of Chamber Wall Conditions on Cl Concentration and Si Etch Rate Uniformity in Plasma Etching Reactors," *J. Electrochem. Soc.*, vol. 150, no. 7, p. G418, 2003.

[85] B. A. Rashap *et al.*, "Control of semiconductor manufacturing equipment: real-time feedback control of a reactive ion etcher," *IEEE Trans. Semicond. Manuf.*, vol. 8, no. 3, pp. 286–297, Aug. 1995.

[86] Á. Bárkányi, T. Chován, S. Németh, and J. Abonyi, "Modelling for Digital Twins—Potential Role of Surrogate Models," *Processes*, vol. 9, no. 3, p. 476, Mar. 2021.

[87] M. Sarfaty, C. Baum, M. Harper, N. Hershkowitz, and J. L. Shohet, "Real-Time Monitoring and Control of Plasma Etching," *Jpn. J. Appl. Phys.*, vol. 37, no. 4S, p. 2381, 1998, [Online]. Available: http://stacks.iop.org/1347-4065/37/i=4S/a=2381.

[88] G. S. May, J. Huang, and C. J. Spanos, "Statistical experimental design in plasma etch modeling," *IEEE Trans. Semicond. Manuf.*, vol. 4, no. 2, pp. 83–98, May 1991.

[89] B. Rashap, J. Freudenberg, and M. Elta, "Real-time feedback for sidewall profile control in reactive ion etching," *J. Vac. Sci. Technol. A Vacuum, Surfaces, Film.*, vol. 13, no. 3, pp. 1792–1796, May 1995.

[90] M. Hankinson, T. Vincent, K. B. Irani, and P. P. Khargonekar, "Integrated real-time and run-to-run control of etch depth in reactive

ion etching," *IEEE Trans. Semicond. Manuf.*, vol. 10, no. 1, pp. 121–130, 1997.

[91] S. W. Butler, K. J. McLaughlin, T. F. Edgar, and I. Trachtenberg, "Development of Techniques for Real-Time Monitoring and Control in Plasma Etching: II . Multivariable Control System Analysis of Manipulated, Measured, and Performance Variables," *J. Electrochem. Soc.*, vol. 138, no. 9, pp. 2727–2735, Sep. 1991.

[92] C. D. Himmel and G. S. May, "Advantages of plasma etch modeling using neural networks over statistical techniques," *IEEE Trans. Semicond. Manuf.*, vol. 6, no. 2, pp. 103–111, May 1993.

[93] D. Stokes and G. S. May, "Real-time control of reactive ion etching using neural networks," *IEEE Trans. Semicond. Manuf.*, vol. 13, no. 4, pp. 469–480, 2000.

[94] Y. L. Huang, T. F. Edgar, D. M. Himmelblau, and I. Trachtenberg, "Constructing a reliable neural network model for a plasma etching process using limited experimental data," *IEEE Trans. Semicond. Manuf.*, vol. 7, no. 3, pp. 333–344, 1994.

[95] B. Kim and S. Park, "An optimal neural network plasma model: a case study," *Chemom. Intell. Lab. Syst.*, vol. 56, no. 1, pp. 39–50, Apr. 2001.

[96] J. Gonzalez and W. Yu, "Non-linear system modeling using LSTM neural networks," *IFAC-PapersOnLine*, vol. 51, no. 13, pp. 485–489, 2018.

[97] D. Park and J. M. Lee, "Surrogate Modeling of Plasma System With Bayesian Deep Neural Network," 2018.

[98] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[99] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Internal Representations by Error Propagation (Vol. 1),” in *Parallel distributed processing: Explorations in the microstructure of cognition*, 1986, pp. 318–362.

[100] J. Donahue *et al.*, “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 677–691, 2017.

[101] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight Uncertainty in Neural Networks,” vol. 37, 2015, [Online]. Available: http://arxiv.org/abs/1505.05424.

[102] J. B. Boffard, C. C. Lin, and C. a DeJosephJr, “Application of excitation cross sections to optical plasma diagnostics,” *J. Phys. D. Appl. Phys.*, vol. 37, no. 12, pp. R143–R161, Jun. 2004.

[103] X.-M. Zhu and Y.-K. Pu, “Optical emission spectroscopy in low-temperature plasmas containing argon and nitrogen: determination of the electron temperature and density by the line-ratio method,” *J. Phys. D. Appl. Phys.*, vol. 43, no. 40, p. 403001, 2010.

[104] S. Park, J. M. Choe, H. J. Roh, and G. H. Kim, “Characteristics of a non-Maxwellian electron energy distribution in a low-pressure argon plasma,” *J. Korean Phys. Soc.*, vol. 64, no. 12, pp. 1819–1827, 2014.

[105] J. M. Stillahn, K. J. Trevino, and E. R. Fisher, “Plasma Diagnostics for Unraveling Process Chemistry,” *Annu. Rev. Anal. Chem.*, vol. 1, no. 1, pp. 261–291, Jul. 2008.

[106] D. E. Seborg, T. F. Edgar, D. A. Mellichamp, and F. J. Doyle, *Process dynamics and control*, 4th ed. John Wiley & Sons, 2011.

[107] C. W. Coley *et al.*, “A robotic platform for flow synthesis of organic compounds informed by AI planning,” *Science (80-.).*, vol. 365, no.

6453, 2019.

[108] N. Collins *et al.*, “Fully Automated Chemical Synthesis: Toward the Universal Synthesizer,” *Org. Process Res. Dev.*, vol. 24, no. 10, pp. 2064–2077, Oct. 2020.

[109] J. M. Douglas, “A hierarchical decision procedure for process synthesis,” *AIChE J.*, vol. 31, no. 3, pp. 353–362, Mar. 1985.

[110] S. Amaran, N. V. Sahinidis, B. Sharda, and S. J. Bury, “Simulation optimization: a review of algorithms and applications,” *Ann. Oper. Res.*, vol. 240, no. 1, pp. 351–380, May 2016.

[111] T. Neveux, “Ab-initio process synthesis using evolutionary programming,” *Chem. Eng. Sci.*, vol. 185, pp. 209–221, 2018.

[112] A. Khan and A. Lapkin, “Searching for optimal process routes: A reinforcement learning approach,” *Comput. Chem. Eng.*, vol. 141, p. 107027, 2020.

[113] Q. Göttl, D. G. Grimm, and J. Burger, “Automated Synthesis of Steady-State Continuous Processes using Reinforcement Learning,” no. 1, pp. 1–18, Jan. 2021, [Online]. Available: http://arxiv.org/abs/2101.04422.

[114] C. B. Browne *et al.*, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[115] N. Oury, M. Pedersen, and R. Petersen, “Canonical Labelling of Site Graphs,” *Electron. Proc. Theor. Comput. Sci.*, vol. 116, no. CompMod, pp. 13–28, Jun. 2013.

[116] D. Silver *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,” *Science (80-.).*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018.

[117] A. Couëtoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard, “Continuous upper confidence trees,” in *International Conference on Learning and Intelligent Optimization*, 2011, pp. 433–445.

[118] M. Jankowiak and F. Obermeyer, “Pathwise Derivatives Beyond the Reparameterization Trick,” 2018, [Online]. Available: http://arxiv.org/abs/1806.01851.

[119] G. E. Uhlenbeck and L. S. Ornstein, “On the Theory of the Brownian Motion,” *Phys. Rev.*, vol. 36, no. 5, pp. 823–841, Sep. 1930.

[120] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” 2015.

[121] G. Matheron, N. Perrin, and O. Sigaud, “The problem with DDPG: understanding failures in deterministic environments with sparse rewards,” pp. 1–19, Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.11679.

[122] S. Huang and S. Ontañón, “A Closer Look at Invalid Action Masking in Policy Gradient Algorithms,” *arxiv:2006.14171*, pp. 1–14, 2020, [Online]. Available: http://arxiv.org/abs/2006.14171.

[123] A. C. Kokossis and C. A. Floudas, “Synthesis of isothermal reactor—separator—recycle systems,” *Chem. Eng. Sci.*, vol. 46, no. 5–6, pp. 1361–1383, 1991.

[124] J. H. Wegstein, “Accelerating convergence of iterative processes,” *Commun. ACM*, vol. 1, no. 6, pp. 9–13, 1958.

[125] X. He, K. Zhao, and X. Chu, “AutoML: A survey of the state-of-the-art,” *Knowledge-Based Syst.*, vol. 212, p. 106622, Jan. 2021.

[126] D. Park, S. Park, J. H. Kim, and J. M. Lee, “Bayesian parameter estimation of nonlinear differential equations using automatic

differentiation," 2018.

[127] K. P. Champion, S. L. Brunton, and J. Nathan Kutz, "Discovery of nonlinear multiscale systems: Sampling strategies and embeddings," *SIAM J. Appl. Dyn. Syst.*, vol. 18, no. 1, pp. 312–333, 2019.

[128] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Inferring Biological Networks by Sparse Identification of Nonlinear Dynamics," *IEEE Trans. Mol. Biol. Multi-Scale Commun.*, vol. 2, no. 1, pp. 52–63, 2016.

[129] V. Dasagi, J. Bruce, T. Peynot, and J. Leitner, "Ctrl-Z: Recovering from Instability in Reinforcement Learning," *arXiv:1910.03732*, 2019, [Online]. Available: http://arxiv.org/abs/1910.03732.

[130] O. Anschel, N. Baram, and N. Shimkin, "Averaged-DQN: Variance Reduction and Stabilization for Deep Reinforcement Learning," *arXiv:1611.01929*, vol. 1, Nov. 2016, [Online]. Available: http://arxiv.org/abs/1611.01929.

[131] Z. Zhu, K. Lin, and J. Zhou, "Transfer Learning in Deep Reinforcement Learning: A Survey," *arxiv:2009.07888*, pp. 1–23, 2020, [Online]. Available: http://arxiv.org/abs/2009.07888.

[132] Y. Ma, C. C. Aggarwal, S. Wang, and J. Tang, "Graph convolutional networks with eigenpooling," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 723–731.

[133] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta Learning Shared Hierarchies," *arxiv:1710.09767*, Oct. 2017, [Online]. Available: http://arxiv.org/abs/1710.09767.

[134] B. Desgraupes, "Clustering Indices," *CRAN Packag.*, vol. 1, no. April, pp. 1–10, 2013, [Online]. Available: cran.r-

project.org/web/packages/clusterCrit.

[135] J.-O. Palacio-Niño and F. Berzal, “Evaluation Metrics for Unsupervised Learning Algorithms,” May 2019, [Online]. Available: http://arxiv.org/abs/1905.05667.

[136] N. Bolshakova and F. Azuaje, “Cluster validation techniques for genome expression data,” *Signal Processing*, vol. 83, no. 4, pp. 825–833, Apr. 2003.

[137] J. W. Sammon, “A Nonlinear Mapping for Data Structure Analysis,” *IEEE Trans. Comput.*, vol. C–18, no. 5, pp. 401–409, May 1969.

[138] L. Chen and A. Buja, “Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis,” *J. Am. Stat. Assoc.*, vol. 104, no. 485, pp. 209–219, 2009.

[139] J. A. Lee and M. Verleysen, “Quality assessment of dimensionality reduction: Rank-based criteria,” *Neurocomputing*, vol. 72, no. 7–9, pp. 1431–1443, Mar. 2009.

[140] G. Kraemer, M. Reichstein, and M. D. Mahecha, “dimRed and coRanking-unifying dimensionality reduction in R,” *R J.*, vol. 10, no. 1, pp. 342–358, 2018.

초 록

계측기술의 발달로 양질의, 그리고 방대한 양의 공정 데이터의 취득이 가능해졌다. 그러나 많은 경우 시스템 차원의 크기에 비해서 일부 운전조건의 공정 데이터만이 취득되기 때문에, 공정 데이터는 '희소'하게 된다. 뿐만 아니라, 공정 데이터는 시스템 거동 자체와 더불어 계측에서 발생하는 노이즈로 인한 본질적인 확률적 거동을 보인다. 따라서 시스템의 예측모델은 예측값에 대한 불확실성을 정량적으로 기술하는 것이 요구되며, 이를 통해 오진을 예방하고 잠재적 인명 피해와 경제적 손실을 방지할 수 있다. 이에 대한 보편적인 접근법은 확률추정기법을 사용하여 이러한 불확실성을 정량화 하는 것이나, 현존하는 추정기법들은 재귀적 샘플링에 의존하는 특성상 고차원이면서도 다량인 공정데이터에 적용하기 어렵다는 근본적인 한계를 가진다. 본 학위논문에서는 매개분포근사에 기반한 확률기계학습을 적용하여 시스템에 내재된 불확실성을 모델링하면서도 동시에 계산 효율적인 접근 방법을 제안하였다.

먼저, 공정의 모니터링에 있어 가우시안 혼합 모델 (Gaussian mixture model)을 분류자로 사용하는 확률적 결함 분류 프레임워크가 제안되었다. 이때 분류자의 학습에서의 계산 복잡도를 줄이기 위하여 데이터를 저차원으로 투영시키는데, 이를 위한 확률적 다양체 학습 (probabilistic manifold learning) 방법이 제안되었다. 제안하는 방법은 데이터의 다양체 (manifold)를 근사하여 데이터 포인트 사이의 쌍별 우도 (pairwise likelihood)를 보존하는 투영법이 사용된다. 이를 통하여 데이터의 종류와 차원에 의존도가 낮은 진단 결과를 얻음과 동시에 데이터 레이블과 같은 비거리적 (non-metric) 정보를 효율적으로 사용하여 결함 진단 능력을 향상시킬 수 있음을 보였다.

둘째로, 베이지안 심층 신경망(Bayesian deep neural networks)을 사용한 공정의 확률적 모델링 방법론이 제시되었다. 신경망의 각 매개변수는 가우스 분포로 치환되며, 변분추론 (variational inference)을 통하여 계산 효율적인 훈련이 진행된다. 훈련이 끝난 후 파라미터의 유효성을 측정하여 불필요한 매개변수를 소거하는 사후 모델 압축 방법이 사용되었다. 반도체 공정에 대한 사례 연구는 제안하는 방법이 공정의 복잡한 거동을 효과적으로 모델링 할 뿐만 아니라 모델의 최적 구조를 도출할 수 있음을 보여준다.

마지막으로, 분포형 심층 신경망을 사용한 강화학습을 기반으로 한 확률적 공정 설계 프레임워크가 제안되었다. 최적치를 찾기 위해 재귀적으로 목적 함수 값을 평가하는 기존의 최적화 방법론과 달리, 목적 함수 곡면 (objective function surface)을 매개화 된 확률분포로 근사하는 접근법이 제시되었다. 이를 기반으로 이산화 (discretization)를 사용하지 않고 연속적 행동 정책을 학습하며, 확실성 (certainty)에 기반한 탐색 (exploration) 및 활용 (exploitation) 비율의 제어가 효율적으로 이루어진다. 사례 연구 결과는 공정의 설계에 대한 경험지식 (heuristic)을 학습하고 유사한 설계 문제의 해를 구하는 데 이용할 수 있음을 보여준다.

주요어: 확률적 기계 학습, 매개 분포 근사, 불확실성 정량화, 확률적 다양체 학습, 베이지안 신경망, 베이지안 추정, 강화 학습, 분포형 신경망.

학번: 2015-21060