|  |  |
| --- | --- |
|  |  |
|  | Master-Thesis |
|  |  |
|  |  |
|  | Development of a deep learning method for anomaly detection in the Tennessee Eastman Process  Entwicklung einer Deep Learning Methode zur Detektion von Anomalien anhand des Tennessee Eastman Prozess |
|  |  |





Bearbeiter: cand. -Ing. Liang Zhao   
Matr. Nr.: 2657846

Betreuer: M. Sc. Nicolas Jourdan, M. Sc. Tobias Biegel  
Abgabe: Darmstadt, den 01.06.2021

**- Offizielle Aufgabenstellung einfügen -**

Die Version für die Bibliothek muss die unterschriebene Originalversion der Aufgabenstellung enthalten. Alle anderen Versionen enthalten Kopien der unterschriebenen Originalversion.

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Muster Mustermann, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.



(Datum) (Unterschrift)

**Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt**

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.



(Date) (Signature)

Abstract

For industrial processes, anomalies can affect production efficiency and even threaten the safety of workers. Monitoring and early anomalies detection of modern industrial processes require analysis and processing of big data. Deep learning has achieved great success in many challenging tasks and demonstrated the ability to process massive data. As a result, scholars and industry have begun to apply deep learning in anomaly detection. This thesis seeks to present a review of the current anomaly detection algorithms and develop an evaluation method for those algorithms. Then focusing on the Tennessee Eastman process, we implement series of deep learning anomaly detection algorithms and compare them with traditional machine learning algorithms. The developed anomaly detection algorithms can be categorized into two types: feature extraction and feature representation. Feature extraction algorithms first extract the input data samples' features and then identify the anomalies by calculating the anomaly scores with the features. One advantage is that several off-the-shelf algorithms are available, and we can directly combine those algorithms to detect anomalies. Feature representation detects anomalies by deploying autoencoders to represent the input data samples, then identify the anomalies by thresholding the reconstruction error. An important assumption for feature representation algorithms is that if the autoencoders are only trained with normal data samples, they could not reconstruct the fault data samples well. In experiments, we also compare the implemented deep learning algorithms with traditional machine learning algorithms. The experiment results show that the deep learning algorithms have exhibited advantages in fault detection, especially in accuracy and efficiency. We further explore the temporal relationship of data samples, and neural network classifiers with Long Short-Term Memory achieves the best anomaly detection performance. Finally, we point out several research directions for anomaly detection.

Keywords: industrial processes anomaly detection, algorithms evaluation, big data

Abstrakt

Bei industriellen Prozessen können Anomalien die Produktionseffizienz beeinträchtigen und sogar die Sicherheit der Arbeitnehmer gefährden. Analyse und Verarbeitung von Big Data ist bei der Überwachung und frühzeitige Erkennung von Anomalien moderner industrieller Prozesse erforderlich. Deep Learning hat bei vielen herausfordernden Aufgaben große Erfolge erzielt und die Fähigkeit demonstriert, massive Daten zu verarbeiten. Infolgedessen haben Wissenschaftler begonnen, tiefes Lernen bei der Erkennung von Anomalien anzuwenden. Diese Arbeit versucht, Deep-Learning-Anomalieerkennungsmethoden zu implementieren und Anomalieerkennungsmethoden mit den Bewertungsmetriken zu vergleichen. Die entwickelten Algorithmen können in zwei Typen eingeteilt werden: Autoencoder und Klassifikatoren. Autoencoder erkennen Anomalien durch Rekonstruktion der Eingabedatenproben. Klassifikatoren identifizieren Anomalien, indem sie eine Karte zwischen Eingabedatenproben und Etiketten erstellen. In Experimenten analysieren wir die Deep-Learning-Algorithmen mit herkömmlichen Algorithmen für maschinelles Lernen mit Bewertungsmetriken. Die Deep-Learning-Algorithmen haben Vorteile bei der Fehlererkennung gezeigt, insbesondere in Bezug auf Genauigkeit und Effizienz. Schließlich weisen wir auf verschiedene Forschungsrichtungen zur Erkennung von Anomalien hin.

Schlagwörter: industrial processes anomaly detection, algorithms evaluation, big data

Contents

[Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB der TU Darmstadt i](#_Toc72678860)

[Abstract i](#_Toc72678861)

[Abstrakt ii](#_Toc72678862)

[Contents iii](#_Toc72678863)

[List of Figures v](#_Toc72678864)

[List of Tables vii](#_Toc72678865)

[List of Abbreviations viii](#_Toc72678866)

[1 Introduction 1](#_Toc72678867)

[1.1 Motivation 1](#_Toc72678868)

[1.2 Problem Statement 2](#_Toc72678869)

[1.3 Research Questions 3](#_Toc72678870)

[1.4 Organization 3](#_Toc72678871)

[2 Background 4](#_Toc72678872)

[2.1 Machine Learning Concept 4](#_Toc72678873)

[2.2 Traditional Algorithms 5](#_Toc72678874)

[2.2.1 Principal Component Analysis 5](#_Toc72678875)

[2.2.2 Support Vector Machine 6](#_Toc72678876)

[2.2.3 Decision Tree 6](#_Toc72678877)

[2.2.4 Random Forests 7](#_Toc72678878)

[2.3 Deep Learning 7](#_Toc72678879)

[2.3.1 Artificial Neural Network 7](#_Toc72678880)

[2.3.2 Neural Network Training 8](#_Toc72678881)

[2.3.3 Recurrent Neural Network 11](#_Toc72678882)

[2.3.4 Generative Adversarial Networks 12](#_Toc72678883)

[2.3.5 Conditional GANs and Bidirectional GANs 13](#_Toc72678884)

[2.3.6 Generative vs. Discriminative Modeling 15](#_Toc72678885)

[2.4 Evaluation Metrics 16](#_Toc72678886)

[2.4.1 Confusion Matrix 16](#_Toc72678887)

[2.4.2 Receiver Operating Characteristics Curve 17](#_Toc72678888)

[2.4.3 Reliability Diagrams 19](#_Toc72678889)

[2.4.4 Log-loss Score 20](#_Toc72678890)

[2.5 Data Preparation 21](#_Toc72678891)

[3 Methodology 23](#_Toc72678892)

[3.1 Nature of Input Data 23](#_Toc72678893)

[3.2 What Are Anomalies? 23](#_Toc72678894)

[3.2.1 Point Anomalies 23](#_Toc72678895)

[3.2.2 Contextual Anomalies 24](#_Toc72678896)

[3.2.3 Collective Anomalies 24](#_Toc72678897)

[3.3 The Output of Anomaly Detection Models 25](#_Toc72678898)

[3.3.1 Anomaly score 25](#_Toc72678899)

[3.3.2 Labels 25](#_Toc72678900)

[3.4 Challenges in Anomaly Detection 25](#_Toc72678901)

[3.5 Anomaly Detection Algorithms 26](#_Toc72678902)

[3.5.1 Autoencoder 26](#_Toc72678903)

[3.5.2 Stacked Autoencoder 28](#_Toc72678904)

[3.5.3 Variational Autoencoder 29](#_Toc72678905)

[3.5.4 GANs in Anomaly Detection 30](#_Toc72678906)

[3.5.5 Long Short-Term Memory Networks 32](#_Toc72678907)

[3.6 Summary 32](#_Toc72678908)

[3.6.1 Feature Extraction Anomaly Detection 32](#_Toc72678909)

[3.6.2 Feature Representation Anomaly Detection 33](#_Toc72678910)

[4 Experiment and Analysis 34](#_Toc72678911)

[4.1 Tennessee Eastman Process 34](#_Toc72678912)

[4.1.1 TEP Data 35](#_Toc72678913)

[4.1.2 TEP Data Preparation 36](#_Toc72678914)

[4.1.3 TEP Data Exploration and Visualization 36](#_Toc72678915)

[4.2 Experiment Environment 40](#_Toc72678916)

[4.3 Baseline and Evaluation Metrics 40](#_Toc72678917)

[4.4 Experiment Result and Analysis 44](#_Toc72678918)

[4.4.1 Autoencoders and Regularization 44](#_Toc72678919)

[4.4.2 Deep Autoencoders 46](#_Toc72678920)

[4.4.3 Traditional Machine Learning Classifiers 48](#_Toc72678921)

[4.4.4 Neural Network Classifiers 49](#_Toc72678922)

[4.4.5 Autoencoders with Classifiers 52](#_Toc72678923)

[4.4.6 Generative Adversarial Networks 54](#_Toc72678924)

[5 Conclusion and Future Work 55](#_Toc72678925)

[5.1 Conclusion 55](#_Toc72678926)

[5.2 Future Work 55](#_Toc72678927)

[6 Appendix 57](#_Toc72678928)

[Bibliography 58](#_Toc72678929)

List of Figures

[Figure 1: Structure of an autoencoder (source: van der Maaten et al. (2007)). 2](#_Toc72678930)

[Figure 2: Performance comparison of deep learning algorithms and traditional algorithms (source: Bahnsen (2017)). 4](#_Toc72678931)

[Figure 3: Data for principal component analysis in 2D (source: Shlens (2014)). 5](#_Toc72678932)

[Figure 4: Maximum-margin for an SVM with two classes (Source: Carrasco (2019)). 6](#_Toc72678933)

[Figure 5: Decision tree for predicting customer‘s response (Source: Maimon and Rokach (2014, p. 14)). 7](#_Toc72678934)

[Figure 6: Schematic of an artificial neuron (Source: Sen et al. (2019, p. 6)). 8](#_Toc72678935)

[Figure 7: An artificial neural network with three layers (Adapt from Nilsson (1996, p. 62)). 9](#_Toc72678936)

[Figure 8: Relationship between training error and generalization error (Source: Goodfellow et al. (2017, p. 115)). 11](#_Toc72678937)

[Figure 9: Unfold an RNN (Source: Feng et al. (2017)). 11](#_Toc72678938)

[Figure 10: Generative adversarial network architecture (Source: Brownlee (2019, p. 10)). 12](#_Toc72678939)

[Figure 11: Training of the generator and discriminator (Source: Goodfellow et al. (2014)). 13](#_Toc72678940)

[Figure 12: Bidirectional generative adversarial networks (Source: Donahue et al. (2017)). 14](#_Toc72678941)

[Figure 13: Confusion matrix (Source: Budnarain (2020, p. 46)). 16](#_Toc72678942)

[Figure 14: ROC curve of a classifier. 18](#_Toc72678943)

[Figure 15: Relationship between ROC AUC score and Log-loss (Adapt from Dembla (2020b)). 19](#_Toc72678944)

[Figure 16: Reliability diagram of a classifier (Source: Own presentation). 20](#_Toc72678945)

[Figure 17: Log-loss of a no-skill classifier (Adapt from Dembla (2020a)). 21](#_Toc72678946)

[Figure 18: Anomalies and normal dataset (Source: Chandola et al. (2009)). 24](#_Toc72678947)

[Figure 19: Contextual anomalies example (Source: Chandola et al. (2009)). 24](#_Toc72678948)

[Figure 20: Collective anomalies example (Source: Chandola et al. (2009)). 25](#_Toc72678949)

[Figure 21: Autoencoder architecture (Adapt from Jordan (2018a)). 27](#_Toc72678950)

[Figure 22: Structure of stacked autoencoder (Adapt from Liu et al. (2018)). 29](#_Toc72678951)

[Figure 23: Structure of variational autoencoder (Source: Jordan (2018b)). 30](#_Toc72678952)

[Figure 24: GANomally structure (Source: Akcay et al. (2018)). 31](#_Toc72678953)

[Figure 25: Tennessee Eastman process (Source: Downs and Vogel (1993)). 34](#_Toc72678954)

[Figure 26: Standard deviation of TEP variables (Own presentation). 37](#_Toc72678955)

[Figure 27: xmeas\_1 distribution (Own presenatation). 37](#_Toc72678956)

[Figure 28: Scatter plot of xmeas\_7 and xmeas\_13 (Own presentation). 38](#_Toc72678957)

[Figure 29: PCA visualization of fault 1, 2, 3 (Own presentation). 39](#_Toc72678958)

[Figure 30: t-SNE visualization of fault 1, 2, 3 (Own presentation). 40](#_Toc72678959)

[Figure 31: Baseline autoencoder structure (Own presentation). 41](#_Toc72678960)

[Figure 32: xmeas\_11 and reconstructed xmeas\_11 in normal dataset. 42](#_Toc72678961)

[Figure 33: xmeas\_11 and reconstructed xmeas\_11 in fault dataset (fault 0). 42](#_Toc72678962)

[Figure 34: AUC ROC score of baseline autoencoder (Own presentation). 44](#_Toc72678963)

[Figure 35: Baseline autoencoder FDR and AUC ROC score (different hidden layer nodes). 45](#_Toc72678964)

[Figure 36: Autoencoder with progressively decreasing layer nodes (Own presentation). 47](#_Toc72678965)

[Figure 37: Autoencoder with overcomplete layer (Own presentation). 47](#_Toc72678966)

[Figure 38: LSTM classifier FDR of fault 3, 9 (different time step) (Own presentation). 51](#_Toc72678967)

[Figure 39: LSTM classifier FDR and ROC AUC score (different time step) (Own presentation). 52](#_Toc72678968)

[Figure 40: update figure 53](#_Toc72678969)

[Figure 41: update figure 54](#_Toc72678970)

List of Tables

[Table 1: Input data type and corresponding DL model architecture (Adapt from Chalapathy and Chawla (2019)). 23](#_Toc72678971)

[Table 2: TEP process faults (Source: Downs and Vogel (1993)). 35](#_Toc72678972)

[Table 3: TEP datasets structure (Source: Rieth et al. (2017)). 35](#_Toc72678973)

[Table 4: Number of samples in each TEP datasets (Source: Rieth et al. (2017)). 36](#_Toc72678974)

[Table 5: Pearson covariance coefficient of the normal dataset (Own presentation). 38](#_Toc72678975)

[Table 6: Experiment environment (Own presentation). 40](#_Toc72678976)

[Table 7: Baseline autoencoder fault detection performance (Own presentation). 43](#_Toc72678977)

[Table 8: Performance of baseline autoencoder (different hidden layer nodes) (Own presentation). 45](#_Toc72678978)

[Table 9: Performance of autoencoder (different regularization techniques) (Own presentation). 46](#_Toc72678979)

[Table 10: Performance of autoencoder (different size of the normal training dataset) (Own presentation). 46](#_Toc72678980)

[Table 11: Performance of autoencoder with three hidden layers (Own presentation). 48](#_Toc72678981)

[Table 12: Performance of traditional machine learning classifiers (Own presentation). 49](#_Toc72678982)

[Table 13: Performance of neural network classifiers (Own presentation). 50](#_Toc72678983)

[Table 14: Performance of LSTM classifiers (different time step size) (Own presentation). 51](#_Toc72678984)

[Table 15: Performance of stacked autoencoders (Own presentation). 53](#_Toc72678985)

List of Abbreviations

|  |  |
| --- | --- |
| ANN | Artificial neural network |
| AnoGAN | Anomaly detection method based on GAN |
| AUC | Area under the curve |
| BiGANs | Bidirectional generative adversarial networks |
| cGANs | Conditional generative adversarial networks |
| DL | Deep learning |
| EGBAD | Efﬁcient GAN-based anomaly detection |
| FAR | False alarm rate |
| FDR | Fault detection rate |
| FN | False negatives |
| FP | False positives |
| FPR | False positive rate |
| GAN | Generative adversarial networks |
| LSTM | Long short-term memory |
| ML | Machine learning |
| MLE | Maximum likelihood estimation |
| MSE | Mean squared error |
| PCA | Principal component analysis |
| RNN | Recurrent neural network |
| ROC | Receiver operating characteristics |
| SAE | Stacked autoencoder |
| SNR | Signal-to-Noise ratio |
| SVM | Support vector machine |
| TEP | Tennessee Eastman process |
| TN | True negatives |
| TP | True positive |
| TPR | True positive rate |
| VAE | Variational autoencoder |

# Introduction

## Motivation

The machine learning (ML) area has undergone a radical revolution in the last several years. Academia and industry have invested billions of dollars in developing massively complex machine learning algorithms. As an essential subset of machine learning, deep learning (DL) works by deploying multiple layers in the artificial neural network (ANN) and has achieved great success in challenging tasks such as image classification (Krizhevsky et al., 2017), natural language processing (Liu et al., 2020) and autonomous driving (Huang and Chen, 2020). At the early stage of big data, traditional machine learning algorithms dominated most machine learning tasks because of the high efficiency and decent performance. However, the traditional machine learning algorithms can hardly improve performance after trained with millions of samples. One fundamental difference between deep learning and traditional machine learning algorithms is the scalability regarding training data volume. In other words, deep learning algorithms can continuously improve performance and outperforms traditional machine learning algorithms with more training data (Chalapathy and Chawla, 2019). More importantly, device capabilities such as computing power, memory capacity have ushered in a significant improvement. Based on such background, deep learning is widely applied in various fields. For example, in terms of anomaly detection, deep learning has been deployed for tasks such as surveillance video analysis (Xie et al., 2017), cyber intrusion detection (Javaid et al., 2016), and medical diagnosis (Schlegl et al., 2017). This thesis focuses on evaluating the performance of different machine learning algorithms, especially deep learning, in industrial processes anomaly detection.

Industrial systems are becoming increasingly complex to fulfill the requirement of productiveness, reliability, efficiency, and safety. To ensure the normal operating, more and more variables are introduced to monitor and control the system, posing challenges for the system operator. Therefore, early detection of anomalies and emergency response plans are critical for the system's safe and effective operation. The anomaly detection methods are generally classified into data-driven and model-based (Ding et al., 2011). The latter usually requires expert-level knowledge and experience to build the sophisticated model, which is challenging to realize and economically unfeasible. Compared to the model-based method, the data-driven method such as machine learning algorithms relies on analyzing vast amounts of data, which provides a practical solution for various industrial processes. Due to the feasibility and practicability, machine learning has become an essential analysis method in the industrial process.

This thesis investigates the application of machine learning algorithms in anomaly detection regarding the Tennessee Eastman process (TEP). The TEP is first proposed by Downs and Vogel (1993). It is a typical industrial chemical process model and can be used as a benchmark to evaluate the control and monitoring methods. In general, the TEP is a nonlinear unstable system with multi-input/outputs variables and characterized by fast and slow dynamic behaviors (Martin-Villalba et al., 2018). Due to the above reasons, the TEP data has the following characteristics:

* high dimensionality,
* nonlinearity,
* non-gaussian distribution.

High-dimensional data pose a severe challenge for machine learning algorithms and may even make the algorithms ineffective. Such phenomena, which may not arise in low-dimensional data, are also termed the *curse of dimensionality (Bellman, 1966, pp. 34–37)*. Two descriptive explanations are given by Domingos (2012) for the *curse of dimensionality*. The first explanation is that the negative influence of noise accumulates as dimensionality increases, i.e., the influence of noise from irrelevant features may suppress the influence of relevant features. The second explanation is that the similarity between data points increases as dimension expands. For example, when using distance as the criteria to decide if data points are neighbors, the number of neighbor points increases as the dimensionality increases.

To mitigate the impact of high dimensionality, methods such as feature selection and feature extraction are proposed. The concept of feature selection is to select the most relevant features based on correlation metrics and expect to improve the machine learning algorithms' performance (Chebel-Morello et al., 2016). In contrast, the feature extraction method extracts the features by transforming the high-dimensional data into a low-dimensional space. The feature extraction method can be further categorized into linear and nonlinear approaches. A representative linear feature extraction approach is the principal component analysis (PCA) which extracts the low-dimensional features by representing the high-dimensional data into a linear subspace of lower dimensions (Shlens, 2014). However, the principal component analysis may not adequately deal with nonlinear data samples such as the TEP data. Hawkins et al. (2002) proposed autoencoders, which are also known as replicator neural networks, to handle the nonlinear high-dimensional data. As Figure 1 shows, an autoencoder consists of an encoder and a decoder part. The encoder learns to transform the input data to data , which is a low-dimensional representation of *.* Then*,* the decoder transforms the data back to the high-dimensional space and get . For autoencoders, it is restricted to reconstruct the input data only approximately. The encoder part of the autoencoder can be used to extract the data features. One research point of this thesis is to explore the application of autoencoders for anomaly detection in the TEP. Meanwhile, this thesis also seeks to provide a comprehensive comparison and analysis of anomaly detection methods in the TEP.

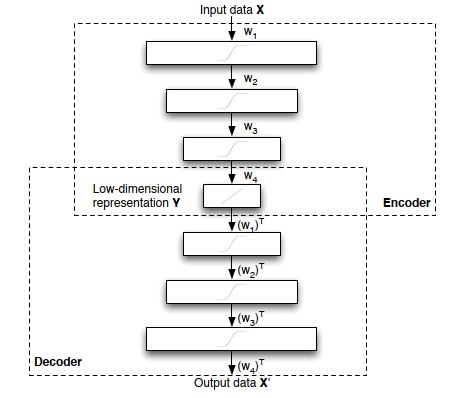


Figure 1: Structure of an autoencoder (source: van der Maaten et al. (2007)).

## Problem Statement

The operation and monitoring of modern industrial processes are based on the analysis of massive data. Machine learning algorithms and deep learning has been applied in anomaly detection. This thesis aims to review and analyze the performance of different machine learning algorithms, enhance the understanding of anomaly detection and point out the directions for further work.

## Research Questions

This thesis's main contribution is to comprehensively compare the traditional machine learning method and the deep learning method for TEP’s anomaly detection. Then analyze the reasons for the difference and implement an improved deep learning model for anomaly detection. More specifically, the goal of this research is to answer the following questions:

* What is the current status of research in TEP's anomaly detection?
* How to evaluate the performance of different algorithms regarding the TEP simulation data?
* What are the reasons causing the performance difference?
* How to implement deep learning algorithms for anomaly detection?
* What are the research directions for future works?

## Organization

The document is structured as follows. First, Chapter 1 introduces the topic and provides an overview of the research problem. Then, Chapter 2 explains the background knowledge about machine learning algorithms and evaluation metrics. Next, Chapter 3 illustrates the property of anomalies and introduces the anomaly detection algorithms in the present studies. Chapter 4 includes the data pre-processing, experiment setting, and performance analysis of different anomaly detection algorithms. Finally, Chapter 5 summarizes the works and points out the future research directions.

# Background

This chapter illustrates machine learning concepts and provides necessary background knowledge about machine learning. First, Section 2.2 introduces a series of traditional algorithms. Then, Section 2.3 provides the theoretical knowledge of deep learning, which is the basis for understanding the anomaly detection algorithms in Chapter 3. Next, Section 2.4 introduces the evaluation metrics to assess the performance of anomaly detection algorithms. Finally, Section 2.5 formalizes the data preparation process for machine learning algorithms.

## Machine Learning Concept

A machine learns if it can achieve self-adjustment and is expected to improve the performance given more external information (Nilsson, 1996, p. 1). Machine learning is a fast-evolving area in both academia and industry in the last decades. As an essential branch of machine learning, deep learning has achieved success in crucial fields such as pattern recognition and translation (Sejnowski, 2020). This is due to several reasons: 1) deep learning methods can realize sustained performance improvement when given more data exceeding the threshold of traditional learning algorithms, as Figure 2 shows; 2) deep learning can reconstruct the unknown structure and find appropriate representation based on input distribution (Bengio, 2012).

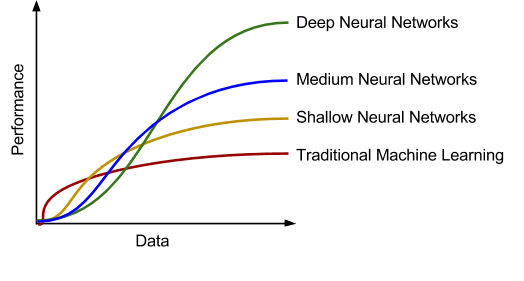


Figure 2: Performance comparison of deep learning algorithms and traditional algorithms (source: Bahnsen (2017)).

Machine learning algorithms can be grouped into different categories. For example, grouped by learning style (Behera and Das, 2017). it could be sub-divided into:

* Supervised learning: Each input data has a corresponded label, e.g., Positive/Negative. The machine learning algorithm can find the mapping between input data and labels by training. Further, the trained classifier or function can predict the label of data.
* Unsupervised learning: The input data has no label, unlike in supervised learning. Machine learning algorithms' task is to find the typical input data pattern or cluster it into groups.
* Semi-supervised learning: This can be seen as a hybrid version of supervised learning and unsupervised learning because the input data is partially labeled. The task is to predict the label of the data and execute the clustering.
* Reinforcement learning: The algorithms are trained to choose behaviors in a specific environment to obtain the most reward.

## Traditional Algorithms

In this thesis, traditional machine learning algorithms refer to the algorithms in which no artificial neural network is involved. Traditional machine learning algorithms are mature and widely deployed in many fields for clustering, classification, and data analysis. In this section, we introduce several traditional algorithms that could be deployed in anomaly detection.

### Principal Component Analysis

As a dimension reduction method, the principal component analysis represents the data so that only the most *meaningful* features remain. More specifically, the PCA transforms the high-dimensional data points into low-dimensional linearly uncorrelated data points while maintaining as much information as the original data points.

For example, we want to transform a set of two-dimensional data points into one-dimensional data points, as Figure 3 shows. However, the two-dimensional data points contain noise, and we want to mitigate the effect of noise and extract only the meaningful signal values in one dimension. To transform the two-dimensional data points, we define a new orthogonal coordinate with axis by rotating the original axis. is the variance of data points along the axis and is the variance of data points along the axis. Along the axis, the variance of data points has the maximum value. The one-dimensional data (signal) is generated by projecting the data points on the axis, and the data along axis is ignored. In this way, we reduce the influence of noise. PCA finds the axis by a series of linear algebra operations (covariance matrix, eigenvector, and singular value decomposition) and assumes that the data points are a linear combination of the basis vectors. Signal-to-Noise ratio (SNR) is used as a measure to evaluate the meaningfulness of data points in axis. A high SNR value indicates that the signal has high quality since the influence of noise is minor.

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.1 ) |
|  |  |  |

Although PCA is a non-parametric, easy to implement dimension reduction approach, it has several limitations: 1) The low-dimensional representation of data points can cause information loss and is less interpretable; 2) PCA may not achieve good performance for nonlinear data without further modification; 3) The orthogonality of basis may not hold in some cases (Shlens, 2014).

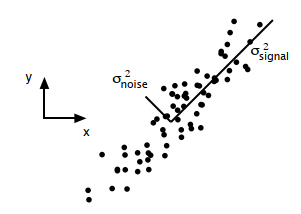


Figure 3: Data for principal component analysis in 2D (source: Shlens (2014)).

### Support Vector Machine

The support vector machine (SVM) is a classification model that can separate the data by transforming it into high dimensional space with the kernel technique. As Figure 4 shows, specific data points are chosen as the support vector, and the area between the support vectors is called margin. Thus, SVM solves the classification problem by finding a hyperplane (decision boundary) to separate the data and realize the maximal margin.

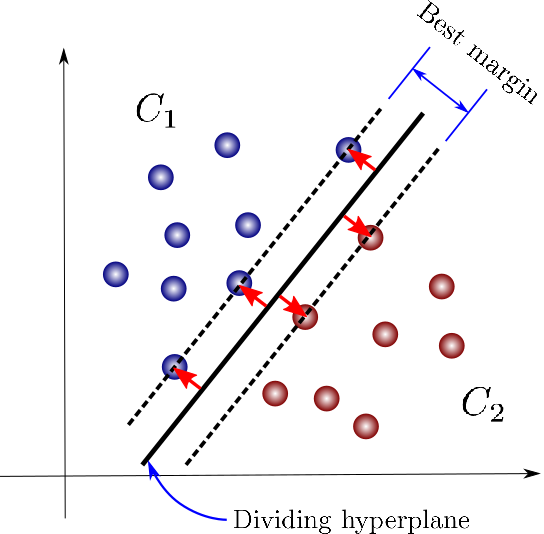


Figure 4: Maximum-margin for an SVM with two classes (Source: Carrasco (2019)).

One fact about the classification task is that mapping the data into another space may significantly improve the performance. SVM works effectively in high-dimensional space (Cristianini and Shawe-Taylor, 2006, p. 26) and solves the nonlinear separable issues using the kernel, transforming the data point into high- or infinite-dimensional space. However, SVM does not provide a probability estimate of the classification; instead gives the classification result directly. Besides, the selection of kernels has a crucial impact on classification performance. In the experiments, the computation cost for SVM is much higher than other classifiers, such as the random forest classifiers.

### Decision Tree

As a predictive model, the decision tree can be used for both classification and regression tasks. The decision tree works by partitioning the instance space recursively and constructs a decision boundary. As Figure 5 shows, a decision tree consists of nodes and edges. The nodes without outgoing edges are called leaves or terminal nodes, and the nodes without incoming edges are called the root. Classification begins from the root node, after which each node splits the instance space into sub-spaces using a specific evaluation function until it reaches the leaves. The leaves represent the classification result, and edges correspond with certain sub-spaces.

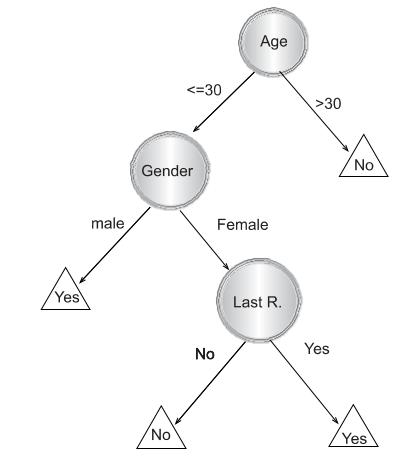


Figure 5: Decision tree for predicting customer‘s response (Source: Maimon and Rokach (2014, p. 14)).

The decision tree is an easy to explain machine learning algorithms because of the similarity to the human decision process. However, it is not very robust and has high variance because a minor change in the training data samples can lead to a different decision tree construction (Maimon and Rokach, 2014, p. 82). To improve the decision tree's performance, random forests are proposed (Breiman, 2001).

### Random Forests

Random forests are predictive models based on the decision tree for classification or regression. Since the decision tree has poor generalization performance because of high variance and overfitting problems, Ho (1998) and Breiman (2001) proposed random sub-space selection and bagging techniques, which leads to the invention of the random forests. The main idea of bagging is to construct many decision trees and make predictions using the *majority vote* (Gareth et al., 2013, p. 318). If the decision trees are not correlated after the training, the majority vote of trees can decrease the variance. Random sub-space selection solves the tradeoff between overfitting and high accuracy by selecting a random subset of features when splitting each decision tree's sub-space.

## Deep Learning

This section introduces the background knowledge of deep learning. Deep learning uses multiple layers of artificial neural networks to extract features from the input data samples. A deeper neural network has demonstrated better representational power regarding decision boundary (Montúfar et al., 2014) and classification (Bianchini and Scarselli, 2014). Further, fields such as object recognition (Krizhevsky et al., 2017) and strategy game (Silver et al., 2016) deep learning algorithms have produced comparable results or results beyond human experts.

### Artificial Neural Network

An artificial neural network is inspired by studies on biological neurons and built with connected neuron units. A neuron is as Figure 6 shows, the output is calculated by the following equation:

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.2 ) |

For , is the input signal, is the weights section which adjusts the signal strength and is the bias. A commonly used activation function is Sigmoid function ranging from -1 to 1 and is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.3 ) |

Stacking and connecting multiple neuron units in different directions regarding the signal flows can build various artificial neural networks. Universal approximation theorems (Csáji, 2001) imply that a sufficiently powerful artificial neural network could approximate a wide variety of functions. Based on the artificial neural network, deep learning deploys multiple layers to solve complicated problems. Unlike most traditional ML algorithms, which need the human's formal specification of knowledge to extract features, deep learning introduces end-to-end learning. End-to-End learning refers to training a complex learning system as a single model (Glasmachers, 2017). For example, manual feature selection or creation in computer vision becomes intractable as the training samples increase. End-to-End learning can bypass such procedures and be trained with the image samples and the image annotations (Mahony et al., 2020). Besides, the representation of input data has a strong influence on machine learning performance. Deep learning solved the representation problem by dividing it into simple subproblems, i.e., the complexity of representations at different layers is not the same. Such hierarchy working principle helps deep learning learn more high-level, abstract representations and achieve better performance (Goodfellow et al., 2017, p. 5).

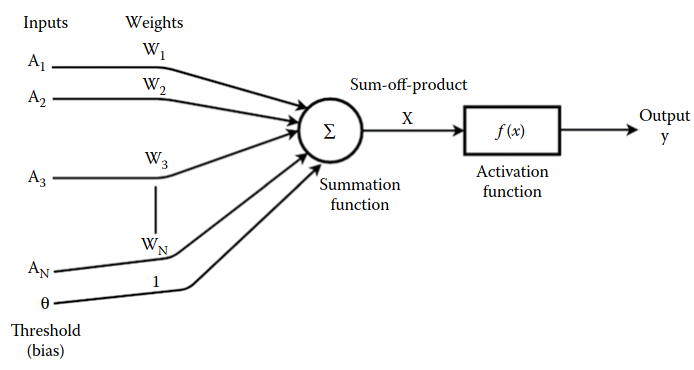


Figure 6: Schematic of an artificial neuron (Source: Sen et al. (2019, p. 6)).

### Neural Network Training

**Backpropagation**

An artificial neural network training refers to updating the model weight to build a mapping of inputs and outputs. Deep learning solves neural network training by transforming it into an optimization problem. First, we illustrate the training procedure with the network as Figure 7 shows, the activation of each layer is defined as:

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.4 ) |

Where is the activation of the neuron in layer , is the weight connecting the neuron in layer and the neuron in layer , is the bias of the neuron in layer and is the activation function.

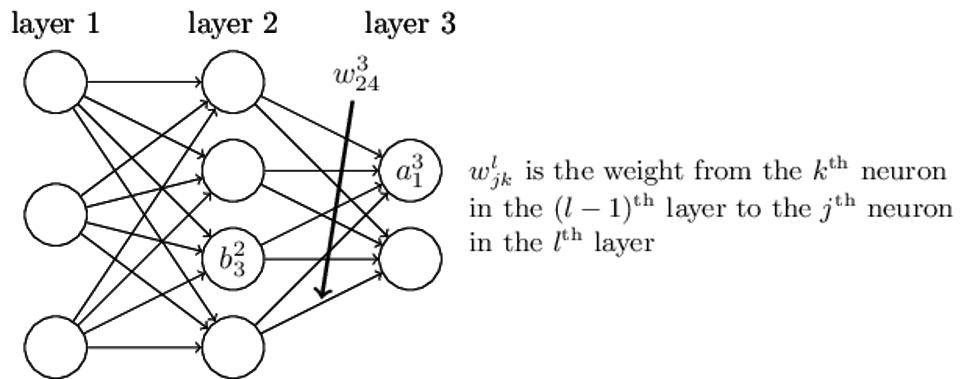


Figure 7: An artificial neural network with three layers (Adapt from Nilsson (1996, p. 62)).

We then define the cost function of the neural network as:

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.5 ) |

Where is the input data sample, is the number of training data samples, is the expected output and is the actual output in the output layer . Mean squared error (MSE) is used here to measure the deviation between expected output and actual output . We want to find the optimal and solving the following optimization problem:

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.6 ) |

We use partial derivative and the chain rule to find the relationship between cost function and and . Then use the gradient to optimize the parameters. After calculus transformation, we get the gradient of the cost function with respect to weight and bias :

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.7 ) |
|  |  | ( 2.8 ) |

Where (resp. ) is the error (resp. weighted input) of the neuron in the layer and defined as:

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.9 ) |
|  |  | ( 2.10 ) |

Finally, we update the weight and bias using the gradient from layer until layer 2:

|  |  |  |
| --- | --- | --- |
|  |  | ( 2.11 ) |
|  |  | ( 2.12 ) |

Where is the learning rate. The method that uses the partial derivative of the cost function with respect to all parameters and optimizes the parameters from the output layer to the forward layer is called backpropagation. In the deep neural network history, hard to train is the main obstacle for the development. When the deep neural network is trained with gradient-based learning methods by backpropagation, the gradient may become unstable and lead to gradient exploding or vanish problems (Nilsson, 1996, pp. 52–59). Lecun et al. (1998), Bengio et al. (2007), and Hinton et al. (2006) proposed several new approaches such as greedy layer-wise training and global training to solve that problem. Applying activation such as ReLU (Glorot et al., 2011) or residual networks (He et al., 2016) can also ease the training problem. The event that promotes deep learning development is the work from Krizhevsky et al. (2017), which achieves a new height for the image classification task. In addition to image classification, deep learning also makes significant progress in speech recognition (Amodei et al., 2015) and natural language processing (Sennrich et al., 2017).

#### Analysis of Model Performance

A machine learning model is first trained with the training dataset and later tested on the unseen test dataset. Generalization refers to the ability to achieve good performance when the model is tested with unobserved data (Goodfellow et al., 2017, p. 110). Underfitting refers to when the model fails to learn (high training error) from the training dataset when it is not trained sufficiently. In contrast, overfitting refers to when the model learns too well on the training dataset but performs poorly (high generalization error) on the test dataset. Figure 8 shows a typical relationship between training error and generalization error. When the model does not have enough capacity, it is prone to underfitting. However, when the model has too much capacity, it is likely to overfit the training dataset and performs poorly on the test dataset. Underfitting can be easily addressed by increasing the capacity of the model. In practice, a more common scenario is the model overfitting. Regularization could be applied to reduce the generalization error for deep learning and mitigate the overfitting problems. Regularization techniques include weight regularization (Krogh and Hertz, 1992), activity regularization (Glorot et al., 2011), weight constraints (Hinton et al., 2012), noise regularization (Bishop, 1995b), and dropout (Srivastava et al., 2014):

* Weight regularization: Encourage model with small weights by adding a regularizer term to the loss function. The regularizer term is usually the absolute or squared values of the weights.
* Activity regularization: Similar to weight regularization. Activity regularization improves the generalization by adding activation penalties to the loss function.
* Weight constraints: As an alternative to weight regularization, weight constraints restrict the weights vector of hidden units to be bound by a constant value or rescale it.
* Noise regularization: Add noise to the training samples during the training.
* Dropout: Randomly drop some outputs of nodes during training, which has a similar effect of training different network architectures in parallel.

In addition, the learning rate and training epochs play a vital role in affecting the model's generalization property. In experiments, we use the training epochs to control how many times the model is trained with the training dataset. The experiment results in Section 4.4.2 show that too many training epochs can lead to overfitting when the model has enough capacity.

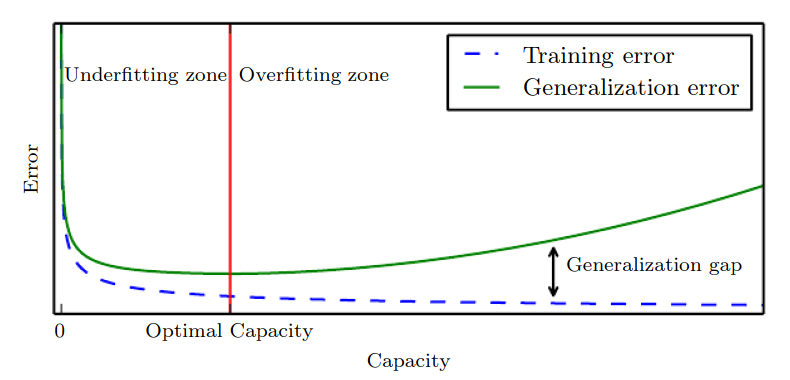


Figure 8: Relationship between training error and generalization error (Source: Goodfellow et al. (2017, p. 115)).

### Recurrent Neural Network

A recurrent neural network (RNN) is a class of artificial neural networks that can learn the mapping between input sequences and output sequences with the internal state . As Figure 9 shows, RNN uses loops to preserve previous computation's state information and process the input of the next time sequence. The formula for the operation in RNN is defined as follows:

|  |  |
| --- | --- |
|  | ( 2.13 ) |
|  | ( 2.14 ) |
|  | ( 2.15 ) |

Where , , are the weight matrices of different layers, , are the bias, and is the activation function. One observation of the RNN structure is that it can connect previous information with the current task. However, RNN can exhibit forgetting behavior, limiting its representation capability when dealing with long-range sequences (Frasconi et al., 1992). Besides, Bengio et al. (1994) and Robinson and Fallside (1987) have shown that RNN suffers from training problems when long-term memory is involved in the task. To mitigate the drawbacks of RNN, Hochreiter and Schmidhuber (1997) introduced Long Short-Term Memory.

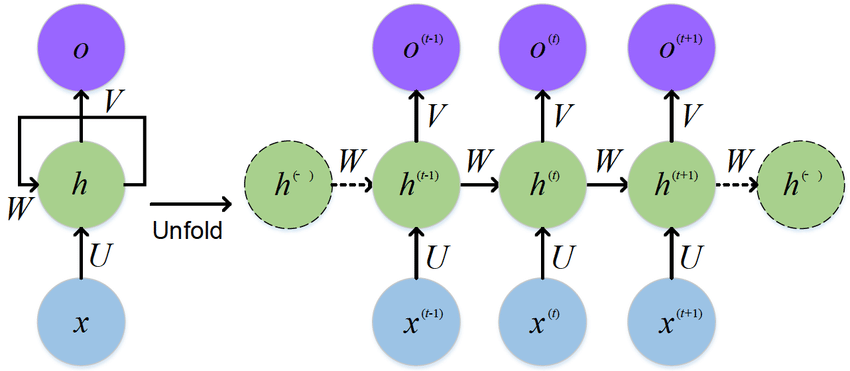


Figure 9: Unfold an RNN (Source: Feng et al. (2017)).

### Generative Adversarial Networks

Generative adversarial networks (GANs) proposed by Goodfellow et al. (2014) are neural network architectures composed of two neural networks: a generator and a discriminator. The generator generates synthetic data, while the discriminator attempts to classify data as real (from data space) or fake (generated by the generator). GANs work in a game-theoretic scenario, i.e., the generator and discriminator learn by competing with each other. As Figure 10 shows, the generator generates plausible data examples from a random input domain. Both the generated examples and real examples are provided to the discriminator to be identified as real or fake examples. The generator learns how to generate more *realistic* examples based on the classification result of the discriminator, i.e., the generator is penalized and updates the parameters when the discriminator successfully identifies the real or generated examples. If the discriminator failed, the generator would not update the model parameter, but the discriminator is penalized and updated its parameters.

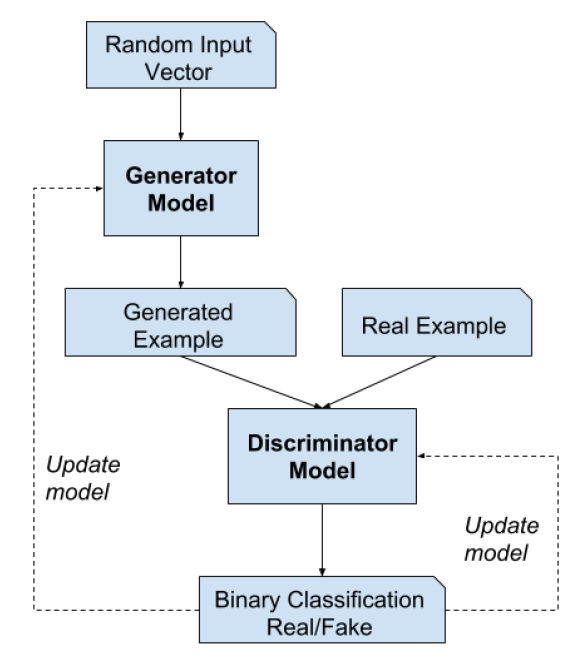


Figure 10: Generative adversarial network architecture (Source: Brownlee (2019, p. 10)).

Assume both the generator and discriminator are multilayer neural networks. For generator , the random input space is defined as and the mapping to data space is defined as , where are the parameters of generator . For discriminator , outputs a value representing the probability that data came from the real data distribution, where are the parameters. During the training of GANs, discriminator is trained to maximize the ability to correctly assign labels for the data from the real data distribution or generated by generator . Meanwhile, generator is trained to minimize , i.e., minimize the probability that discriminator correctly classify the data. In other words, generator and discriminator compete with each other in the minimax game of function :

|  |  |
| --- | --- |
|  | ( 2.16 ) |

Where is the expectation, is the real dataset distribution and is the latent space.

#### The Training Process

In practical implementation, the model parameters of and are updated iteratively. For example, discriminator is trained for steps followed by a one-step training of generator . Figure 11 from (a) to (d) shows the updating of discriminator (blue, dashed line) and generator (green, solid line) in a more intuitive way. The generator and discriminator are trained simultaneously to win in the min-max game. After an adequate number of training epochs, the generator and discriminator will reach a state where the discriminator cannot distinguish if the data distribution is from the generator or real data distribution, as (d) shows.

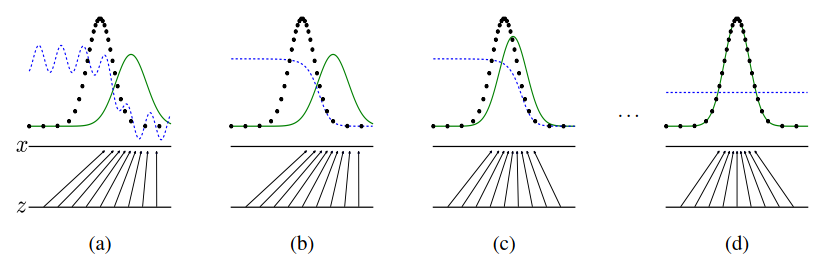


Figure 11: Training of the generator and discriminator (Source: Goodfellow et al. (2014)).

However, some practical issues remain unsolved in the GANs training process. At the early phase of training, if the learning progress of discriminator (learns to distinguish real/fake examples) is faster than the generator (learns to generate more realistic fake examples), generator G will fail to update the parameters. The reason that causes generator to fail is the loss function cannot provide sufficient gradient information for generator . Therefore, Goodfellow et al. (2014) suggests training generator to maximize rather than minimize to obtain stronger gradient information.

As a deep learning generative model, GANs have attracted attention from both practitioners and scholars. This is because GANs avoid the intractable problem of probabilistic computation, such as standard marginalization and conditioning operations (Hinton et al., 2006). Besides, GANs can be applied to various scenarios such as missing data imputation (Yoon et al., 2018), time-series prediction (Yoon et al., 2019), and anomaly detection (Geiger et al., 2020).

### Conditional GANs and Bidirectional GANs

Conditional GANs and bidirectional GANs are two innovative extensions of the GANs framework, which provide the cornerstone for applying GANs in anomaly detection. Conditional GAN helps build a generative model to conditionally generate an output by feeding both random input from latent space and some additional information as input to the generator and discriminator . The additional information can be the class labels or image styles. Bidirectional GANs extend the GANs by adding the *inverse mapping,* which maps from data space back to latent space with an encoder (Di Mattia et al., 2019).

#### Conditional GANs

The GANs framework can help train the generative models to generate images (Radford et al., 2016). However, there are no parameters to control the style or type of the generated images. Mirza and Osindero (2014) proposed Conditional GANs (cGANs) to generate images by feeding the generator with the class labels. Comparing to the original GANs (Goodfellow et al., 2014), the objective function is defined as follows:

|  |  |
| --- | --- |
|  | ( 2.17 ) |

Where is the auxiliary information such as class labels. During the training, the auxiliary information are first encoded as one-hot vectors and concatenated with the input , then fed to the discriminator . The input for the generator is similar to discriminator which is the concatenation of random input (sampled from latent space) and the auxiliary information . To generate a new data sample, feed the generator with input conditioned with the label information.

Gauthier (2015), Odena et al. (2016), and Isola et al. (2017) have demonstrated the great potential of cGANs in image synthesis. Odena et al. (2016) argued that the generated image could achieve better discriminability than the models that can only generate lower resolution images.

#### Bidirectional GANs

As an extension of GANs, Donahue et al. (2017) first proposed the Bidirectional Generative Adversarial Networks (BiGANs), which can learn the inverse mapping (from data space to latent space: ). The BiGANs adopt an encoder sub-network to map from latent space to data space. With the additional encoder, BiGANs can be further extended to unsupervised feature learning tasks such as anomaly detection (Akcay et al., 2018; Carrara et al., 2020). The objective function of BiGANs for training is defined as follows:

|  |  |
| --- | --- |
|  | ( 2.18 ) |
|  | ( 2.19 ) |

Comparing to the original GANs (Goodfellow et al., 2014), BiGAN use sample pairs such as or as the input for the discriminator . As Figure 12 shows, sampled from latent feature space is provided to the generator to generate fake sample pair and train the discriminator to identify fake data samples. On the other hand, sample pair force discriminator to identify real sample data samples. The output of is the probability result of classification. is supposed to be close to 1 if the input is real sample pair as .

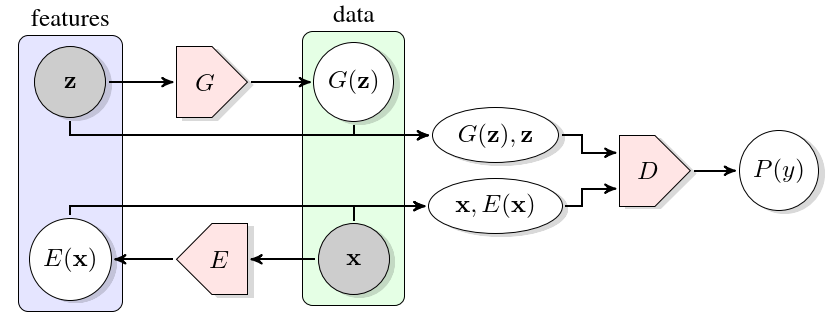


Figure 12: Bidirectional generative adversarial networks (Source: Donahue et al. (2017)).

### Generative vs. Discriminative Modeling

Generative and discriminative models are two essential types of machine learning models. For example, in statistical classification, given a series of independent training samples , , and . is a d-dimensional input data point and is the corresponding output data point such as the class label. We want to predict the class label of a new data point based on the observations. According to the Bayes rule, a new data point is categorized to class when the posterior probability achieves the highest value given . The posterior is defined as follows:

|  |  |
| --- | --- |
|  | ( 2.20 ) |

Where is a parametric model that models the class-conditional probability , and are the parameters of . are the multinomial distribution parameters of with , and is the full parametrization of the joint density. The difference between the generative and discriminative models is the estimation of parameters .

Generative classifiers estimate the parameters using the maximum likelihood estimation (MLE) method by solving the following optimization problems:

|  |  |
| --- | --- |
|  | ( 2.21 ) |
|  | ( 2.22 ) |

Where is the optimal parameters after maximum likelihood estimation.

Discriminative classifiers estimate the parameter by maximizing the conditional log-likelihood:

|  |  |
| --- | --- |
|  | ( 2.23 ) |
|  | ( 2.24 ) |

Where is the optimal parameters after maximum likelihood estimation.

By mathematical transformation, the relationship between and can also be expressed as

|  |  |
| --- | --- |
|  | ( 2.25 ) |

Where the first term is the generative objective function (Bouchard and Triggs, 2004).

In general, generative models attempt to estimate the joint distribution of both input and output , i.e., generating synthetic data points similar to the observed data points. Discriminative models learn to compute the mapping between input and output directly, i.e., calculate the conditional probability of given . In other words, generative models focus on understanding the data points’ composition and basic characteristics of the classes. In contrast, the discriminative models are designed to find the decision boundary between samples of different classes (Amamra et al., 2016).

In the practical application of machine learning algorithms, generative models have the following advantages:

* When the data point is missing or partially labeled, generative models can compensate for missing data points or augment original labels.
* A new class of data points can be learned independently by generative models without affecting the previously learned classes.
* Generative models can learn compositional features without taking samples of all combinations. For example, we want to classify animals based on body size and color, then a combination of all those features is necessary to train the discriminative models.

Compared to generative models, discriminative models make predictions by building maps between input and label directly. The success of applying discriminative models in image classification (Krizhevsky et al., 2017) seems to imply that discriminative models have better performance than generative models in the classification task. Burlina et al. (2019) compared the discriminative and generative models’ performance and demonstrated that when less information is available, the proposed generative model can still achieve comparable performance as the discriminative model. Ulusoy and Bishop (2006) showed generative models could achieve higher accuracy for object detection tasks while discriminative models are faster at prediction once trained, and the combination of both models is expected to achieve better performance. Further, Higa et al. (2019) overcame the shortcoming of discriminative models, which suffer from imbalanced data, by combining both generative and discriminative models.

## Evaluation Metrics

The evaluation metrics are crucial because they assess the performance of machine learning algorithms and guide the directions for better modeling. In this section, we introduce the evaluation metrics to quantify the performance of anomaly detection algorithms.

### Confusion Matrix

The confusion matrix is a commonly used tabular evaluation method in binary classification tasks that presents the classifier's predictions in a contingency table with four cells. To better understand it, we first define true positives, false positives, true negatives, and false negatives.

**True positive (TP).** The number of data points that are correctly predicted as the positive class.

**False positives (FP).** The number of data points that are incorrectly predicted as the positive class.

**True negatives (TN).** The number of data points that are correctly predicted as the negative class.

**False negatives (FN).** The number of data points that are incorrectly predicted as the negative class.

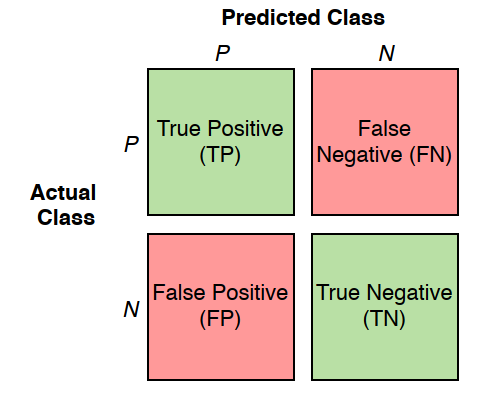


Figure 13: Confusion matrix (Source: Budnarain (2020, p. 46)).

One assumption made here is that the result of predicting and assessing is either positive or negative. As Figure 13 shows, the green cells represent the correct predictions, whereas the red cells are the incorrect predictions. The confusion matrix provides a visualization of the prediction performance of classifiers and forms the basis for other evaluation metrics.

**Precision**: Precision measures the proportion of predicted positive data points, which are correctly positive data points. Precision is defined as:

|  |  |
| --- | --- |
|  | ( 2.26 ) |

As can be seen from the formula, precision only considers positive instances and completely ignores the negative instances.

**Recall**: Recall measures how well the classifier predicts the positive data points. Recall is defined as:

|  |  |
| --- | --- |
|  | ( 2.27 ) |

**Accuracy**: Accuracy measures the fraction of correctly predicted data points in all predictions. Accuracy is defined as:

|  |  |
| --- | --- |
|  | ( 2.28 ) |

**F1 Score**: F1 score is a measure to balance both precision and recall. F1 score is defined as:

|  |  |
| --- | --- |
|  | ( 2.29 ) |
|  |  |

As can be seen from the formula, the F1 score reaches the highest value of 1.0 when either precision or recall is 0.

### Receiver Operating Characteristics Curve

The receiver operating characteristics (ROC) curve is a graph tool for evaluating binary classifiers’ performance as a function of the cut-off threshold. The axis of the graph is (true positive rate) in, and the axis is (false positive rate). and are defined as:

|  |  |
| --- | --- |
|  | ( 2.30 ) |
|  |  |
|  | ( 2.31 ) |

For each threshold value, there is a point consists of (, ) on the graph. Connecting all such points forms the ROC curve. At the lower-left point of the graph, the classifier makes no predictions as positives (no false alarms are detected). At the upper-right point , the classifier predicts all data samples as positive unconditionally (raise false alarms on all negative samples). The diagonal line (red dotted line) connecting both points and represents the classifiers that make predictions by randomly guessing. For example, if a random guessing classifier predicts the probability of a data instance being positive is 30%, it is expected to achieve a 30% and a 30% . Therefore, when the classifier performs better than a random guessing classifier, the classifier’s ROC curve (blue line) should be higher than the diagonal line (red dotted line) (Spackman, 1989).

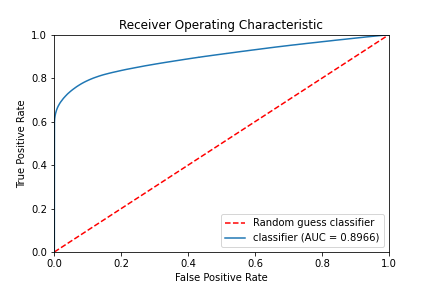


Figure 14: ROC curve of a classifier.

**ROC AUC Score**: Althougha ROC curve can provide a visual description of the classifiers’ performance under different thresholds, It is more convenient to use a single scalar value for comparison. Bradley (1997) proposed the ROC AUC (area under the curve) score, which calculates the proportion of the area underneath the ROC curve in a unit square. A random guessing classifier has an AUC score of 0.5, and no realistic classifier should have an AUC score less than 0.5. A perfect classifier has an AUC score close to 1. Generally speaking, the ROC AUC score implies a degree of separability between the predictions of different classes. As Figure 15 shows, there is a clear boundary (threshold) that can separate the predictions of classifier A (with ROC AUC score=1). In contrast, for the predictions of classifier C (ROC AUC score=0.952), no threshold can separate the predictions of the two classes. Thus the higher the AUC score of a classifier, the better separation performance it has.

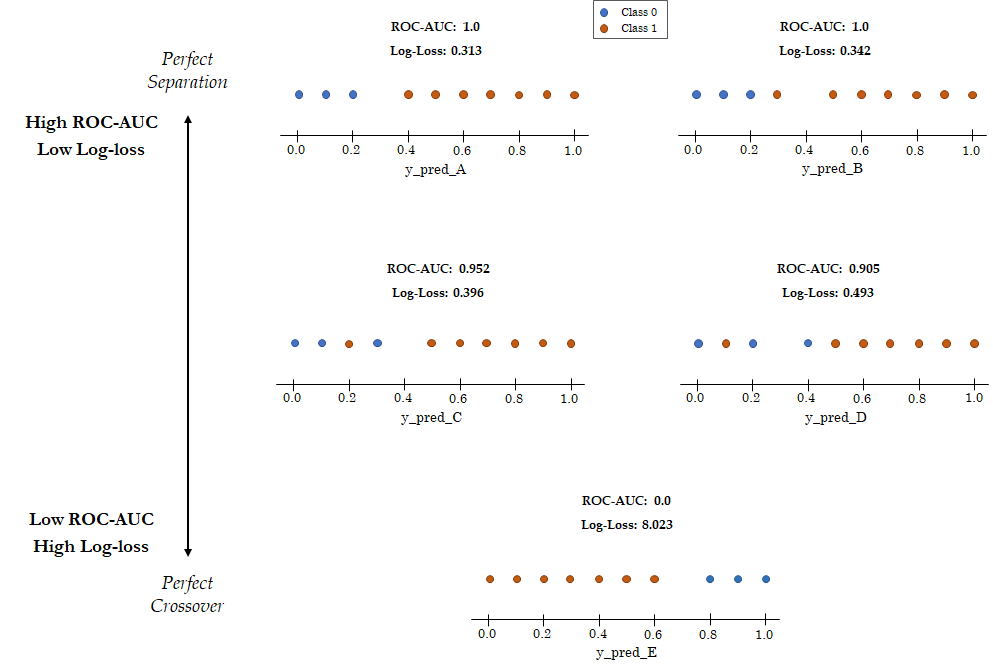


Figure 15: Relationship between ROC AUC score and Log-loss (Adapt from Dembla (2020b)).

### Reliability Diagrams

A reliability diagram (or calibration curve) is a graph tool to evaluate the reliability of classifiers. Roughly speaking, a classifier is reliable (or calibrated) when the fraction of observed positive/negative instances is consistent with its predictions. As Figure 16 shows, the axis is the mean predicted value, and the axis is the fraction of positive instances. The point of the reliability diagram is calculated in two steps: 1) The predicted values are first sorted in an ascending sequence and partitioned into equally-sized bins. 2) For each bin, a mean predicted value is calculated as the value of the point along axis, and the fraction of positives instances in the bin is the value of the point along axis. Connecting all points forms the reliability curve. As Figure 16 shows, a perfectly calibrated classifier (red dotted line) has the same value in both axes. We expect to improve the classifiers such that its reliability curve is close to the perfectly calibrated curve (Bröcker and Smith, 2007).

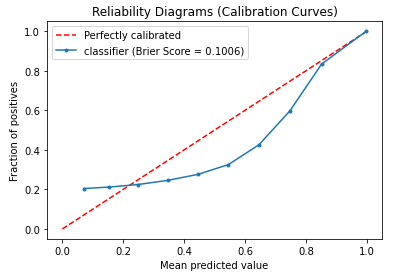


Figure 16: Reliability diagram of a classifier (Source: Own presentation).

#### Brier score

We use the brier score to calculate the mean squared error between prediction and actual values. Brier score is defined as:

|  |  |
| --- | --- |
|  | ( 2.32 ) |

Where is the number of predictions, is the predictions and is the actual values (BRIER, 1950).

### Log-loss Score

Log-loss score measures how close the predictions are to the actual values. It is defined as:

|  |  |
| --- | --- |
|  | ( 2.33 ) |
|  |  |
|  | ( 2.34 ) |
|  |  |

For a perfect classifier, its’ log-loss score is 0, which means it can correctly classify every sample. Therefore, the classifier with a lower log-loss score is better than those with a higher log-loss score. An intuitive relationship between ROC AUC score and log-loss can also be found in Figure 15.

#### Baseline log-loss score

A baseline log-loss score represents the no-skill classifier, making predictions based on the proportion of positive and negative samples. For example, there is a dataset with 90% positive and 10% negative samples. If the no-skill classifier predicts that the probability of a new sample being positive is 90%, the log-loss score is 0.325. We can also calculate all the log-loss scores for classifiers with a different constant predicted probability and connect all the points. As the figure shows, the minimum value in the plot is the point with a predicted probability of 0.1, which corresponds to the no-skill classifier. Thus, the classifier that we choose should have a lower log-loss score than the baseline log-loss score (Dembla, 2020a).

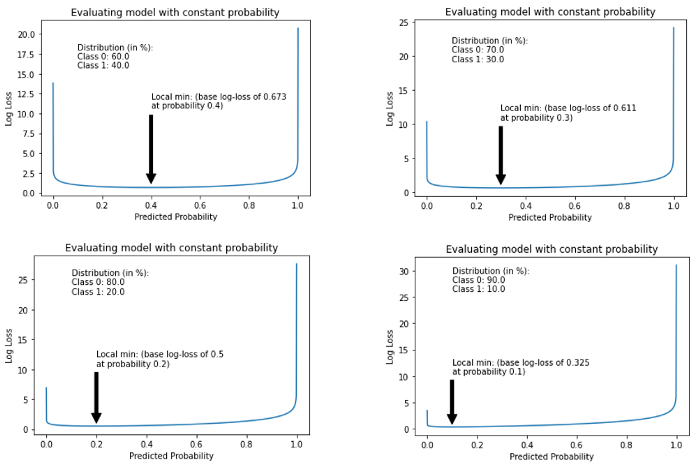


Figure 17: Log-loss of a no-skill classifier (Adapt from Dembla (2020a)).

## Data Preparation

Data preparation refers to operations such as the addition, deletion, and transformation of raw data. Since the performance of some machine learning models relies heavily on the characteristics of data, incorrect data preparation can weaken or completely break the models’ predictive performance (Kuhn and Johnson, 2016, p. 27). Data preparation can be generally summarized into the following steps (Brownlee, 2020, pp. 3–6):

**Data Cleaning.** Various reasons could cause mistakes or incorrect values in data. The goal of data cleaning is to identify the errors and correct them. On the practical level, we could use statistics tools to identify and remove outliers, remove duplicate data, mark empty values as missing or replace them with suitable values using imputation models (Yoon et al., 2018).

**Feature Selection.** For high-dimensional data, we might encounter the *curse of dimensionality (Bellman, 1966, pp. 34–37)*. One solution is to select the most relevant input features and ignore the irrelevant and redundant features.

**Data Transformation.** Generally refers to changing the distributions or the types of the raw data samples. Two common transform methods are normalization and standardization. Normalization scales the data samples between 0 and 1. Standardization scales the data samples to a normal distribution. Input data sample and output are transformed by:

|  |  |
| --- | --- |
|  | ( 2.35 ) |
|  |  |
|  | ( 2.36 ) |

The mean and standard deviation are calculated with

|  |  |
| --- | --- |
|  | ( 2.37 ) |
|  |  |
|  | ( 2.38 ) |
|  |  |

Where is the number of data samples. It is worth mentioning that the scaler (min, max, mean and standard deviation) should be fixed to avoid data leakage after transforming the training data samples (Brownlee, 2020).

**Feature Engineering.** Engineering new features based on training data may improve the model’s performance (Kuhn and Johnson, 2020, p. 14). However, the creation of new features may require experts’ knowledge. One common feature engineering method is polynomial feature transform which combines original features into a polynomial.

**Dimensionality Reduction.** Alternative to feature selection, dimensionality reduction refers to projecting the input data samples into a lower-dimensional space while retaining the original data's most meaningful properties. Common dimensionality reduction techniques are principal component analysis (Shlens, 2014) and linear discriminant analysis (FISHER, 1936).

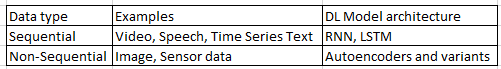
# Methodology

Anomaly detection is a fast-evolving area in which numerous novel models are proposed and applied. This chapter provides an overview of the approaches and methodology in recent studies, focusing on algorithms based on deep learning. First, Section 3.1 discusses the nature of input data and Section 3.4 points out the main challenges in anomaly detection. Next, section 3.5 presents an overview of the anomaly detection algorithms. Finally, Section 3.6 summarizes the models into two frameworks: feature extraction and feature representation.

## Nature of Input Data

The choice of deep learning architectures depends on both the task and data types, such as the recurrent neural network is more suitable for sequential data. In addition, the performance of deep learning algorithms also highly relies on the property of data. Therefore, it is crucial to understand the nature of input data before building the machine learning model. Based on the relevance in the time dimension, input data can be classified into time series or point data. Furthermore, based on the number of features, input data can also be classified into high-dimensional or low-dimensional data. In general, input data is categorized into sequential and non-sequential data. For sequential data, the order of the data sequence matters as the data points depend on the neighbors. Table 1 lists different data types and corresponding deep learning model architectures.

Table 1: Input data type and corresponding DL model architecture (Adapt from Chalapathy and Chawla (2019)).



## What Are Anomalies?

Anomaly detection is one of the critical applications in data science. It refers to the problem of identifying the patterns of data which does not satisfy expected behavior. Such patterns are also known as outliers, anomalies, and exceptions (Chandola et al., 2009). Another definition of anomalies introduced by Hawkins (1980) is that anomalies are the data samples that deviate from the majority of the data samples and plausibly produced by a different mechanism. Based on the nature and observation of anomaly data instances, Chandola et al. (2009) provided a detailed classification of anomalies: 1) point anomalies; 2) contextual anomalies; 3) collective anomalies.

### Point Anomalies

A data point is categorized as point anomalies when it deviates significantly from the rest of the data points. As Figure 18 shows, data point , and area locate far from the normal data area and . Therefore, data point , and all data points in area are identified as point anomalies. Point anomalies are the most straightforward and common type of anomalies which is also the focus of most anomaly detection research.

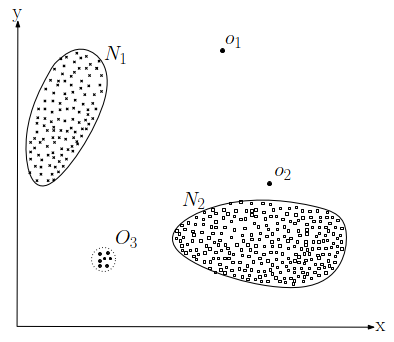


Figure 18: Anomalies and normal dataset (Source: Chandola et al. (2009)).

### Contextual Anomalies

A data point is termed as contextual anomalies or conditional anomalies when merely identified as an anomaly in a user-specified area (Song et al., 2007). In other words, a data point might be classified as contextual anomalies in the current context but could be identified as normal data in another context. For example, as Figure 19 shows, a 0 ambient temperature is recognized as abnormal at time (summer, start in June), but considered as normal at time (winter, start in December).

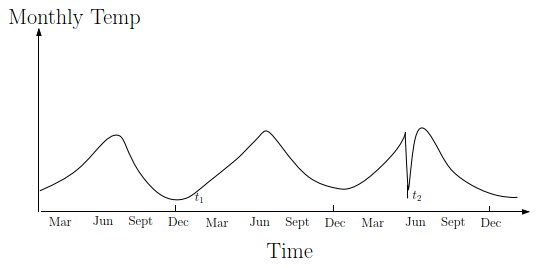


Figure 19: Contextual anomalies example (Source: Chandola et al. (2009)).

### Collective Anomalies

A collection of series data points are identified as anomalies concerning the whole dataset. However, some individual points in the collection may not be anomalies. Namely, collective anomalies emphasize the co-occurrence pattern of data points. For example, Figure 20 shows a human electrocardiogram output. The red area is identified as an anomaly since the similar value has continued for an abnormally long period. However, a single such point in the red area may not be classified as an anomaly.

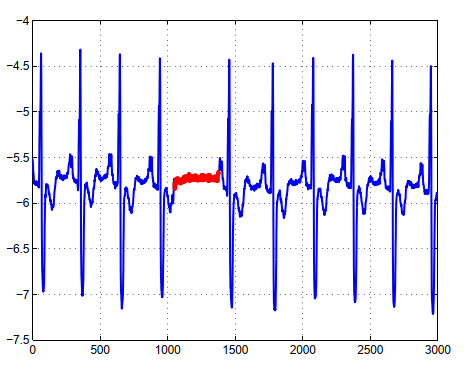


Figure 20: Collective anomalies example (Source: Chandola et al. (2009)).

## The Output of Anomaly Detection Models

The way to present the detected anomalies is critical since it involves deploying and evaluating anomaly detection methods. In general, the outputs of anomaly detection could be anomaly scores or labels.

### Anomaly score

Anomaly scores are introduced to describe the test data samples’ deviation degree from the normal operation state. For example, Hawkins et al. (2002) and Chen et al. (2017) define the anomaly score as the reconstruction error (MSE) of the autoencoder’s input. Furthermore, Schlegl et al. (2017) defined the anomaly score as a combination of reconstruction error and discriminator output of the GAN. After calculating the anomaly scores of all test data samples, anomalies are identified using a cut-off threshold or listing top data samples with high anomaly score.

### Labels

Assigning binary category labels (normal/abnormal) as the predictions of data instances is another way to present the anomaly detection result. Labels can be calculated by thresholding the probabilistic output of classifiers or the anomaly scores. Generally speaking, anomaly score provides more anomaly detection information than labels but not concise as labels.

## Challenges in Anomaly Detection

A straightforward method to detect anomalies is to define a normal region and identify the data samples that do not belong to this region as anomalies. However, this method is not feasible in the practical situation for several reasons (Chandola et al., 2009). First, there is often no clear and exact boundary between the normal and abnormal regions. Second, the current definition of the normal region might not be representative enough for detecting anomalies in the future as the anomaly pattern can be continually evolving. Third, the detailed description of anomalies could be significantly different considering various application scenarios. For example, it is normal for an adult to gain 10% weight in the summertime but abnormal to increase 10% height.

In addition to the above challenges, anomaly detection also faces the following challenges (Pang et al., 2021):

* High-dimensional data. In the age of big data, the complexity and volume of data are continuously growing. As a result, anomaly detection of time series data and multivariant data is particularly challenging. For time-series data, non-stationarity and dynamic behavior are two intractable problems. Whereas for multivariant data, high computational complexity and noise within the data affect the detecting performance significantly.
* Anomaly explanation. For most safety-critical systems, the failure or malfunction could lead to death or serious injury of people, severe property damage, and environmental damage (Knight, 2002). Therefore, it is necessary and crucial to understanding the formation mechanism of anomalies. Since most anomaly detection models work as a black-box (only the inputs and outputs are visible), it is challenging to build the relationship between the internal function parameters with physical parameters such as temperature, pressure, and volume (Zhang, 2010, p. 784). For rare but safety-critical anomaly data samples, a black-box model may predict biased results without further explaining the result. A temporary solution is to inspect anomaly detection results by human experts and manually tune the biased model. However, this solution could decrease the effectiveness of the anomaly detection models and increase the cost.

## Anomaly Detection Algorithms

### Autoencoder

An autoencoder is a feedforward artificial neural network that learns to reconstruct the input data approximately in an unsupervised way. An autoencoder has two subnetworks. Namely, an encoder network that transforms the input to a latent space , a decoder network that reconstructs the input . The loss function (also referred to as reconstruction error) of the autoencoder, in the case of mean squared error, is defined as:

|  |  |
| --- | --- |
|  | ( 3.1 ) |

The loss function measures the difference between the original input and the reconstruction . The autoencoder is trained as a usual neural network through backpropagation to minimize the loss function . As Figure 21 shows, the hidden layer has fewer neural units than the input layers. This bottleneck design is crucial for the autoencoder because it constrains the network's traversed information. Furthermore, it forces the autoencoder not to copy the input precisely and learn a compression (or meaningful representation) of the input data (Goodfellow et al., 2017, pp. 502–503).

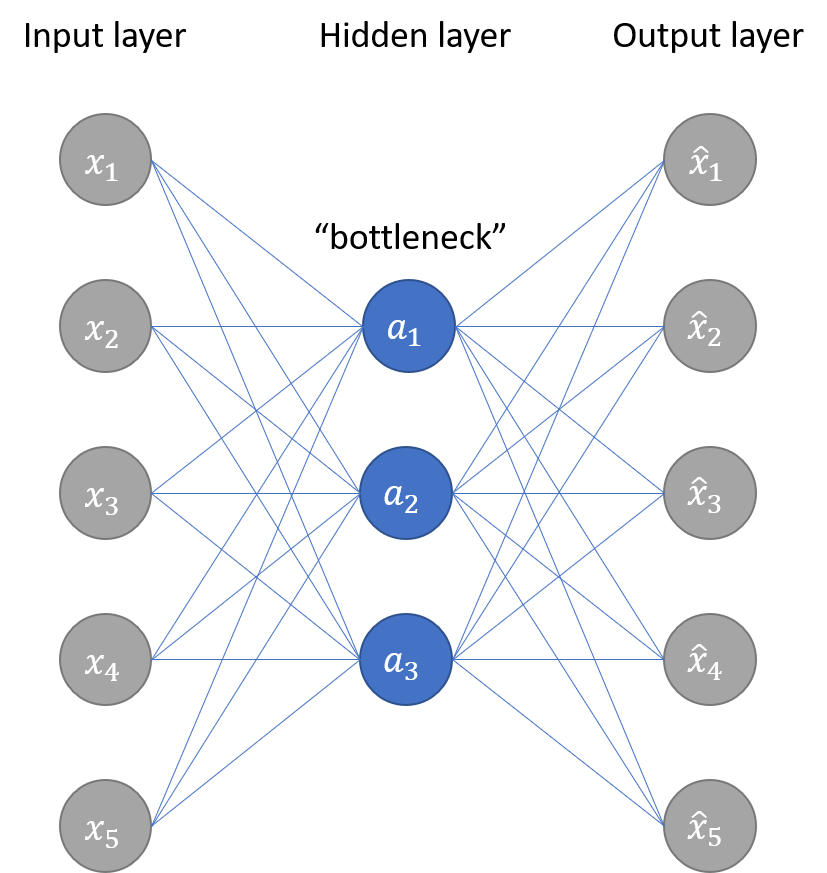


Figure 21: Autoencoder architecture (Adapt from Jordan (2018a)).

However, for deep autoencoders with deep layers and more capacity, the bottleneck design is insufficient to discourage memorization, i.e., prevent deep autoencoders from exactly memorizing the input data samples. Various techniques could be applied to the autoencoder to improve generalization properties. The first method is to introduce dropout layers in the autoencoders. The dropout layer works by probabilistically ignoring some nodes' output and improving generalization performance (Srivastava et al., 2014). The second method is to add a regularizer term in the loss function to avoid overfitting and improve the performance of autoencoders. The autoencoder is trained to minimize the new loss function to improve generalization property. The regularizer term could be the regularization methods, as in Section 2.3.2 discussed.

|  |  |
| --- | --- |
|  | ( 3.2 ) |

#### Sparse Autoencoder

Sparse autoencoder introduces sparsity restriction by uses weight or activity regularization as the regularizer term. For example, Yin and Yan (2019) and Jiang et al. (2017) deploy a sparse autoencoder for anomaly detection using the sparsity restriction. The loss function is defined as:

|  |  |
| --- | --- |
|  | ( 3.3 ) |
|  | ( 3.4 ) |
|  | ( 3.5 ) |

Where is the tuning parameter for penalty and is the number of hidden layers. is the average activation of hidden layer . is the number of input samples and is the activation of hidden layer with input . is called the sparsity parameter with a value close to zero. The value of increase as the difference between and Increase. Therefore, minimizing the loss function force to be close to and realize sparsity. In contrast, Qi et al. (2017) replace the regularizer term with the absolute value of weights in all hidden layers.

#### Denoising Autoencoder

Denoising autoencoder is first proposed by Vincent et al. (2010). It is trained with slightly corrupted input data samples. There are two underlying ideas about denoising autoencoder. The first is that the high-level representation of data samples should be robust and stable even when the input data samples are corrupted. The second idea is that autoencoders trained with corrupted data can help learn the input data samples’ helpful structure. Zhang et al. (2016) explore the performance of denoising autoencoder by adding different types of noise to the input data samples. The experiment results showed that denoising autoencoder could extract features from corrupted data effectively.

Generally speaking, autoencoders can be used for anomaly detection in two ways. The first way uses the reconstruction error of input data to calculate the anomaly score. Based on the assumption that the autoencoder can reconstruct the normal data instances better than anomaly instances when only trained with the normal data instances. The anomaly score of anomaly data instances should be larger than normal data. Then the data instances with high anomaly scores can be identified as anomalies. The second way is to use the encoder part of autoencoders to extract the features of input data. The extracted features can be directly used to calculate anomaly scores or fed into a discriminative network such as a classifier. However, this method relies on another assumption, i.e., the extracted feature can provide more discriminative information than original input data (Chalapathy and Chawla, 2019).

### Stacked Autoencoder

A stacked autoencoder (SAE) is formed by stacking the hidden layers of autoencoders. As Figure 22 shows, each autoencoder is first trained in an unsupervised way, and the trained hidden layer is extracted and stacked together as the hidden layers of the stacked autoencoder. With an additional input and output layer, the stacked autoencoder is fine-tuned in a supervised way (Liu et al., 2018). Specifically, the training process of SAE can be divided into three detailed steps:

1. Train the first autoencoder with the input data samples in an unsupervised way and acquire the hidden layer's learned features.
2. The learned features of the first autoencoder are used as input data samples for the second autoencoder . Train the second autoencoder in an unsupervised way as the first autoencoder and acquire the hidden layer's learned features. Repeat this layer-wise training procedure until all autoencoders are trained.
3. Extract the trained hidden layers of all autoencoders to build the hidden layers of the stacked autoencoder. Then, add the input and output layer (classifier) to the stacked autoencoder and train it in a supervised way. The weights of the stacked autoencoder are updated by minimizing the loss function through the backpropagation algorithms. This step is also referred to as the fine-tuning of stacked autoencoders.

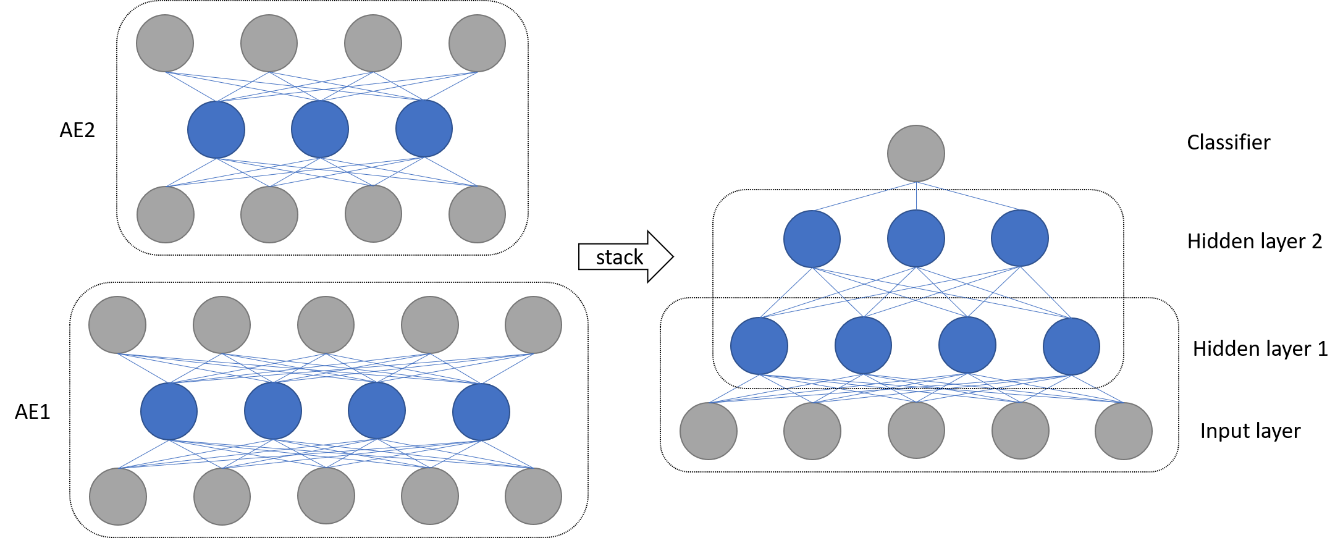


Figure 22: Structure of stacked autoencoder (Adapt from Liu et al. (2018)).

Erhan et al. (2010) discuss the potential benefit of deploying pre-trained layers in the networks and the experiment's result implies that unsupervised pre-training helps networks get a better weight initialization. It can also be viewed as a particular form of regularization, which can help minimize variance, update the parameters in an optimal direction, and improve network generalization property. In practical implements, stacked autoencoders can also be combined with the sparse or denoising autoencoders. Namely, use the loss function with a regularizer term in pre-training or fine-tuning steps of the stacked autoencoders. After the training, the stacked autoencoder can directly predict the probability being anomalies of new data samples.

### Variational Autoencoder

An autoencoder maps the input to a vector in latent space. In contrast, a variational autoencoder (VAE) maps the input to a distribution. In other words, the variational autoencoder provides a probabilistic way to transform the input data samples into latent space. As Figure 23 shows, a variational autoencoder has an encoder and a decoder . To reconstruct a data sample , the variational autoencoder first feed the data sample into encoder and encode it as a distribution in the latent space. Then a point is sampled from the distribution and reconstructed by the decoder with . The loss function of the variational autoencoder is defined as:

|  |  |
| --- | --- |
|  | ( 3.6 ) |

Where is the expectation and is the Kullback–Leibler divergence which measures the difference between two probability distributions and . For variational autoencoders, the distribution of in latent space is usually assumed to be a Gaussian distribution. The gaussian distribution provides variational autoencoders a continuous latent space, a fundamental advantage over autoencoders (Doersch, 2016). For anomaly detection, variational autoencoders adopt similar ideas as autoencoders, i.e., use the reconstruction error to calculate the anomaly score and detect anomalies by thresholding the anomaly scores.

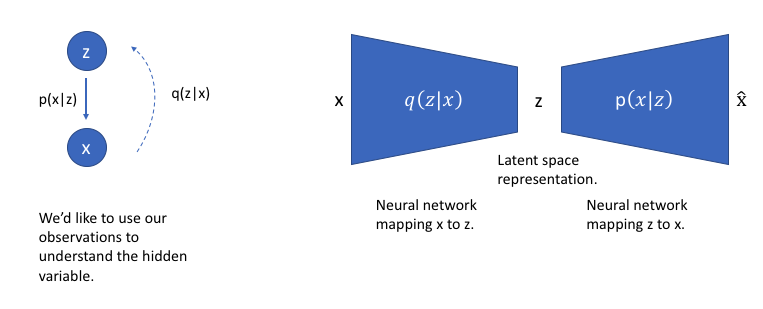


Figure 23: Structure of variational autoencoder (Source: Jordan (2018b)).

### GANs in Anomaly Detection

#### AnoGAN

Schlegl et al. (2017) first proposed an anomaly detection method based on GANs (AnoGAN) to identify the location of an anomaly in the image. AnoGAN is a deep convolutional network trained only with normal image samples using the GANs framework (Goodfellow et al., 2014). After the training, the generator has learned how to generate normal samples rather than anomaly samples. The difference between the input and reconstruction can help to detect anomalies. Radford et al. (2016) argued that the in latent space has the property of smooth transition. For example, two neighbor points , in latent space can generate similar images , in the data space. In order to find the best (to fulfil ) in latent space of input , AnoGAN uses an iterative process by backpropagating in steps. In practice, they deployed a composed loss function to measure the anomality of the input , which is defined as:

|  |  |
| --- | --- |
|  | ( 3.7 ) |
|  | ( 3.8 ) |
|  | ( 3.9 ) |

The residual loss measures the similarity between input and the generated sample , the discriminator loss helps to improve the training stability of GAN. is the weight parameter between those two losses, and is the output of the discriminator. The value of at step is defined as the anomaly score, where a small anomaly score implies that the input sample is similar to the normal samples during training. In contrast, a large anomaly score means that the input is an anomaly sample. AnoGAN is the first proposed GAN framework showing that GANs can also be used for anomaly detection. The main limitation of AnoGAN is that the calculation of anomaly score needs steps optimization, which is computationally expensive (Akcay et al., 2018).

#### EGBAD

To overcome the limitation of AnoGAN, Zenati et al. (2019) proposed Efﬁcient GAN-Based Anomaly Detection (EGBAD), which is based on BiGAN. EGBAD adopt an additional encoder as in BiGAN to map points from data space to latent space. The encoder is jointly trained with the generator, which avoids the optimization steps in AnoGAN and improves efficiency.

#### GANomaly

Akcay et al. (2018) introduced the GANomaly method, which is inspired by BiGAN (Donahue et al., 2017), AnoGAN (Schlegl et al., 2017), and EGBAD (Zenati et al., 2019). As Figure 24 shows, the network is composed of three sub-networks. The first sub-network is the generator of the GANs model. The generator deploys a *bottleneck* autoencoder network with an encoder and decoder . The generator is first fed with sample from data space and deliver it to the encoder . The encoder transforms into data in latent space, where . it is also known as the compressed expression of the autoencoder. Then the decoder reconstruct into , where . The second sub-network is the encoder of the GAN. The encoder transforms which is reconstructed by generator to a latent space point , where . The point and have the same dimensions. The third sub-network is the discriminator . The function of is to distinguish if the input pair is real or fake. GANomaly adopts a similar idea as autoencoders for anomaly detection, i.e., if the generator is trained only with normal samples, it cannot reconstruct input when the input is an abnormal sample. The objective function of GANomaly is composed of three loss functions:

|  |  |
| --- | --- |
|  | ( 3.10 ) |
|  | ( 3.11 ) |
|  | ( 3.12 ) |
|  | ( 3.13 ) |

Where , and are the weights of different loss functions. The adversarial loss is used to improve the stability of GAN training as Salimans et al. (2016) recommended. is the intermediate layer’s output of discriminator . The contextual loss helps the Generator to reconstruct the input data samples. Isola et al. (2017) use a similar loss function to help the generator generate more realistic image samples. The encoder loss is employed to minimize the distance between points and in latent space, which can help learn the feature representation in latent space. After the training, the anomaly score of the test sample is calculated by:

|  |  |
| --- | --- |
|  | ( 3.14 ) |

Where a higher anomaly score means the test point is likely to be an anomaly.

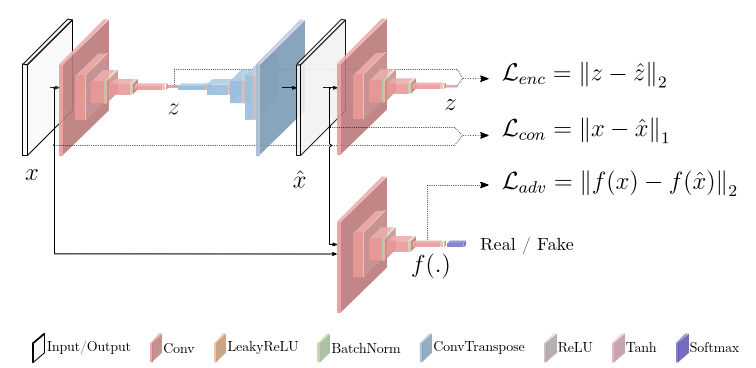


Figure 24: GANomally structure (Source: Akcay et al. (2018)).

### Long Short-Term Memory Networks

To solve the problem encountered in RNN, Hochreiter and Schmidhuber (1997) introduced Long Short-Term Memory (LSTM) network, an efficient gradient-based method using multiplicative gate units to control the error flow. Gers et al. (1999) enhance the performance of LSTM by introducing an adaptive *forget gate*, which enables LSTM to reset at an appropriate frequency. Generally, LSTM overcomes the vanishing and exploding deﬁciencies of the gradients in RNN and are commonly used for analyzing sequential data.

## Summary

Considering the model architecture, working principle, and training framework, the anomaly detection algorithms can be categorized into two main types, feature extraction anomaly detection, and feature representation anomaly detection.

### Feature Extraction Anomaly Detection

Feature extraction anomaly detection methods are consist of a feature extractor and an anomaly scorer ,

|  |  |
| --- | --- |
|  | ( 3.15 ) |
|  | ( 3.16 ) |

where is a feature extractor function with parameters , , and . In addition, can also be viewed as a function to reduce dimensionality, such as PCA. is the anomaly scorer with parameters and is the anomaly score. The feature extractor first extract features from dataset and transform it into latent space. The extracted features are then fed to the anomaly scorer to calculate the anomaly score. The feature extractor and anomaly scorer can be trained independently or jointly, which is also termed as mixed or fully deep approach in when deep learning is involved (Ruff et al., 2018). These two training methods lead to different loss functions for training. In the independently trained approach, is first pre-trained in a preceding step before feeding the extracted features into the anomaly scorer . In other words, feature extractor can be seen as a data pre-processing function for anomaly scorer , then the anomaly scorer is trained with the pre-processed data samples. For the jointly trained approach, feature extractor and anomaly scorer are trained simultaneously with the same objective function. Except for the above training procedures, a pre-trained feature extractor can also be trained (fine-tuned) again during the training of anomaly scorer as stacked autoencoders.

In practical implementation, feature extractor can deploy methods such as PCA (Gao and Hou, 2016; Onel et al., 2019), autoencoders (Erfani et al., 2016), and GANs (Schlegl et al., 2019) to extract the features. For the anomaly scorer , it can apply classifiers such as SVM (Wang et al., 2019), random forests (Yang et al., 2008), or neural network classifier (Zhang, 2000), then use the probabilistic predictions as the anomaly score. The anomaly score can also be calculated directly with the extracted features. Feature extraction anomaly detection works under the assumption that the extracted features can retain discriminative information, and there exist boundaries that can separate normal and anomaly data samples.

Feature extraction anomaly detection has one significant advantage,i.e., a large number of feature extractors and classifiers can be directly combined. However, it has two main drawbacks, i.e., performance relies heavily on accurate labels and the training data’s distribution. Accurate labels usually require a massive amount of effort from domain experts to assign manually. Furthermore, anomalies are rare events compared to normal data samples, which leads to imbalanced training data distribution between positive and negative class samples (Chalapathy and Chawla, 2019).

### Feature Representation Anomaly Detection

Feature representation anomaly detection refers to deploying the models such as autoencoders or other encoder-decoder architectures to detect anomalies with reconstruction error. As discussed in Section 3.5.1, an autoencoder must be restricted to copy the input to output approximately. Otherwise, the autoencoders cannot learn the most salient features of the data instances (Goodfellow et al., 2017, p. 503). The definition is as follows:

|  |  |
| --- | --- |
|  | ( 3.17 ) |
|  | ( 3.18 ) |
|  | ( 3.19 ) |
|  | ( 3.20 ) |

where is the encoder with parameters , is the decoder with parameters . The objective function is to find the optimal parameters pair which minimizes the reconstruction error. is the reconstruction error of new data samples using the optimized . Generally, this type of anomaly detection method assumes that the model can better reconstruct the normal data samples from latent space than anomaly data samples when it is only trained with normal data samples. Feature representation anomaly detection requires only normal training data samples and does not suffer from problems such as imbalanced data distribution. Further, the representation of meaningful features can help to understand the internal mechanism of how data samples are generated.

# Experiment and Analysis

This chapter describes the basic steps of experiments, including data preparation, model training, and result evaluation. First, Section 4.1 introduces the Tennessee Eastman process and basic steps of data preparation for experiments. Next, Section 0 builds a baseline anomaly detection model and illustrates the general evaluation steps. Finally, Section 4.4 provides an extensive evaluation of both traditional and deep learning anomaly detection algorithms, compares the performance differences, and analyzes the reasons.

## Tennessee Eastman Process

The Tennessee Eastman process is a typical industrial chemical process model and was first proposed by Downs and Vogel (1993). As a benchmark model, it can be used to evaluate the control and monitoring methods. The Tennessee Eastman process consists of five process units: a reactor where an exothermic reaction occurs, a product condenser, a vapor-liquid separator, a compressor, and a stripper, as Figure 25 shows. There are 41 measured variables, 11 manipulated variables, and 21 programmed fault types in the Tennessee Eastman process. The process faults are described in Table 2.

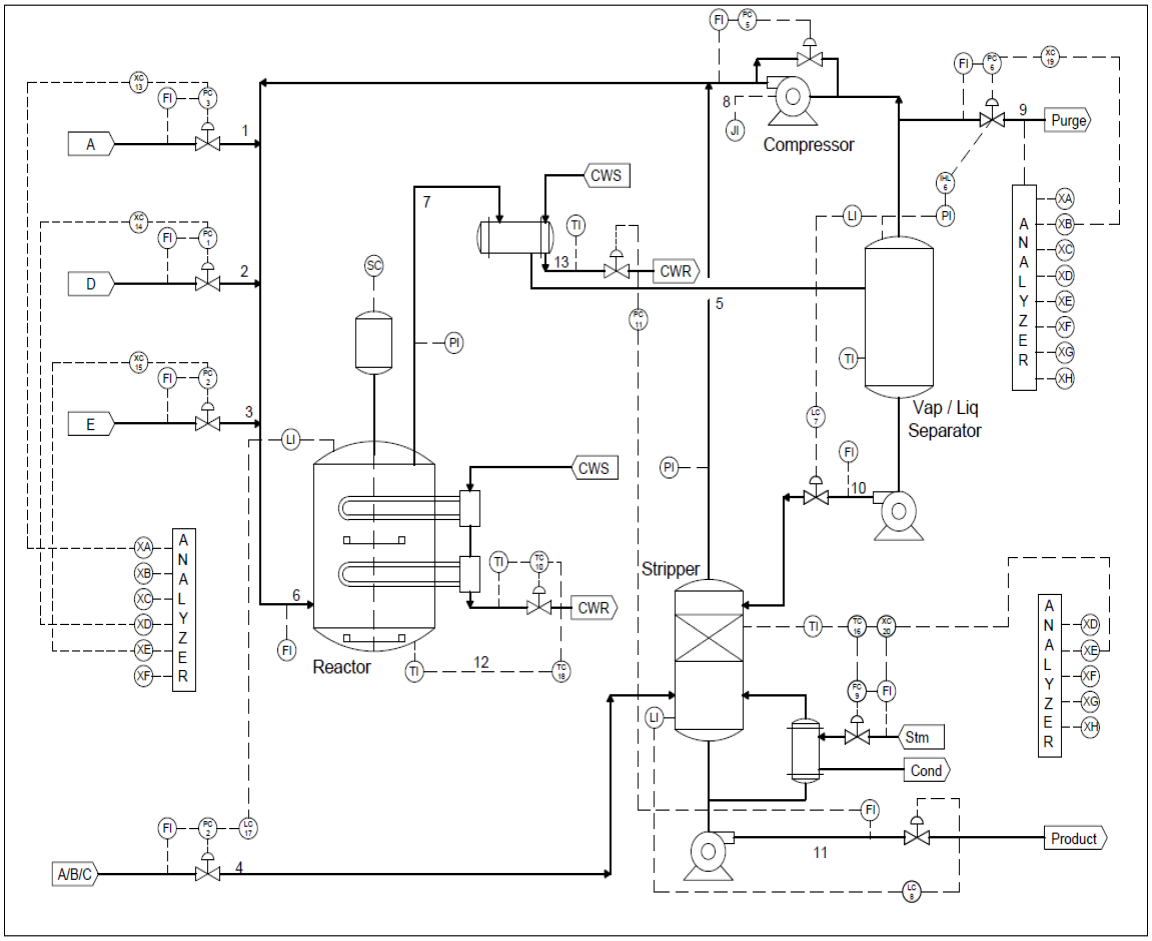
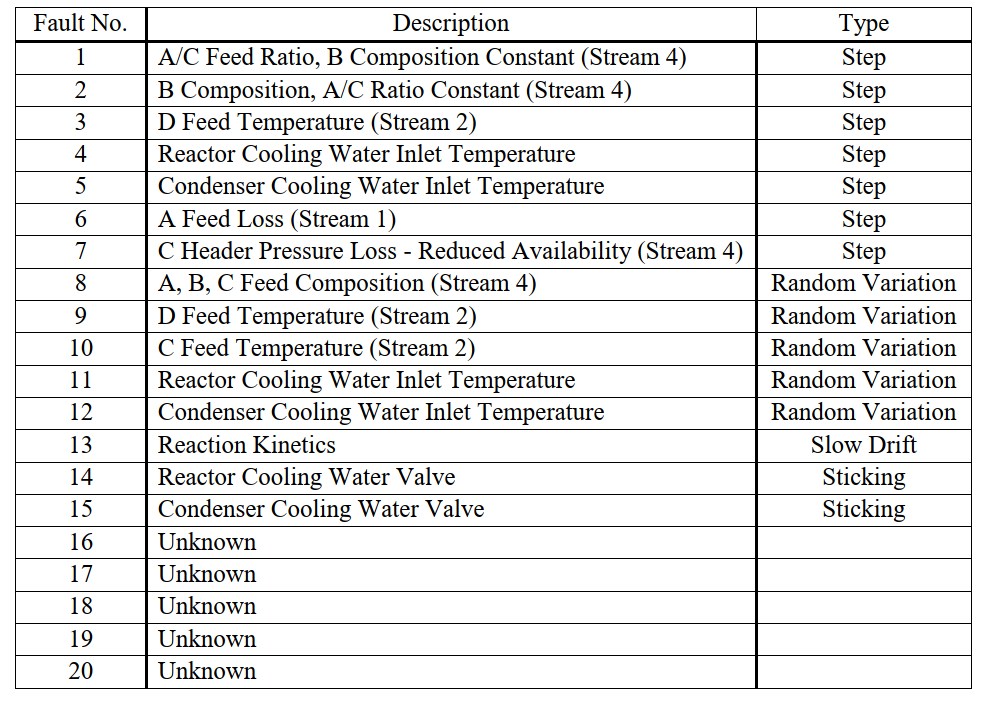


Figure 25: Tennessee Eastman process (Source: Downs and Vogel (1993)).

Table 2: TEP process faults (Source: Downs and Vogel (1993)).



### TEP Data

This thesis uses the Tennessee Eastman process simulation data (Rieth et al., 2017), which contains four datasets:

* Normal training dataset: Data generated in normal operating conditions for model training.
* Normal testing dataset: Data generated in normal operating conditions for model testing.
* Fault training dataset: Data generated in abnormal operating conditions for model training.
* Fault testing dataset: Data generated in abnormal operating conditions for model testing.

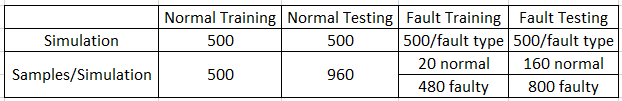
Each dataset contains 55 columns, as Table 3 shows. The first column is the fault number ranging from 0 to 20. Fault 0 stands for fault-free data generated in normal operating conditions. Fault 1-20 are 20 programmed fault types. Since fault 21 is not included in the datasets and will not be considered in the experiment. The second column ranges from 1 to 500 is the simulation index. Each simulation starts from a non-overlapping random state and generates the TEP simulation data samples. The TEP simulation data is sampled every 180 seconds from the simulator. The third column is the sample index of each simulation. The rest columns are the 52 TEP variables: 41 measured variables (, where ) and 11 manipulated variables (, where ).

Table 3: TEP datasets structure (Source: Rieth et al. (2017)).



For the normal training dataset and normal testing dataset, the sample index in each simulation ranges from 1 to 500 for a total of 25 hours. For the fault training dataset and fault testing dataset, the sample index in each simulation ranges from 1 to 960 for a total of 48 hours. For each simulation in the fault training dataset (resp. fault testing dataset), the fault is introduced at sample index 20 (resp. 160). Thus, we only use the data samples after introducing the fault. The data structures are summarized in Table 4.

Table 4: Number of samples in each TEP datasets (Source: Rieth et al. (2017)).



### TEP Data Preparation

We first check if there are errors or missing values in the TEP simulation data (Rieth et al., 2017) and find no mistakes. Data preparation steps such as feature selection, feature engineering, and dimensionality reduction are not applied because we deploy machine learning algorithms such as autoencoders to learn the meaningful features instead of creating or selecting features manually. In the data transformation step, we standardize the data samples with the formula recommended by (Fan et al., 2017):

|  |  |
| --- | --- |
|  | ( 4.1 ) |

Where is the mean value and is standard deviation, is the column index of the TEP variables, refers to the sample index. is set to 2 because some TEP variables may fall outside the range of activation functions. Both the and are calculated with the normal training data samples and used as the scaler for standardization of the other three datasets.

### TEP Data Exploration and Visualization

Before training the machine learning models, we first explore the TEP simulation data (Rieth et al., 2017) and obtain basic understandings of the normal datasets and fault datasets.

#### Statistical Measures

We calculate the basic statistical measures such as mean, standard deviation, min value, and max value. Then we compare the difference among those statistical measures regarding normal and fault datasets. There is no significant difference between normal and fault data regarding mean, min, and max values. However, the difference of standard deviation between normal and fault datasets is noticeable. As Figure 26 shows, the standard deviation of measured variables , manipulated variable , , in the fault dataset is much larger than that in the normal dataset. That means high variance and dispersion is a critical property of the fault data.

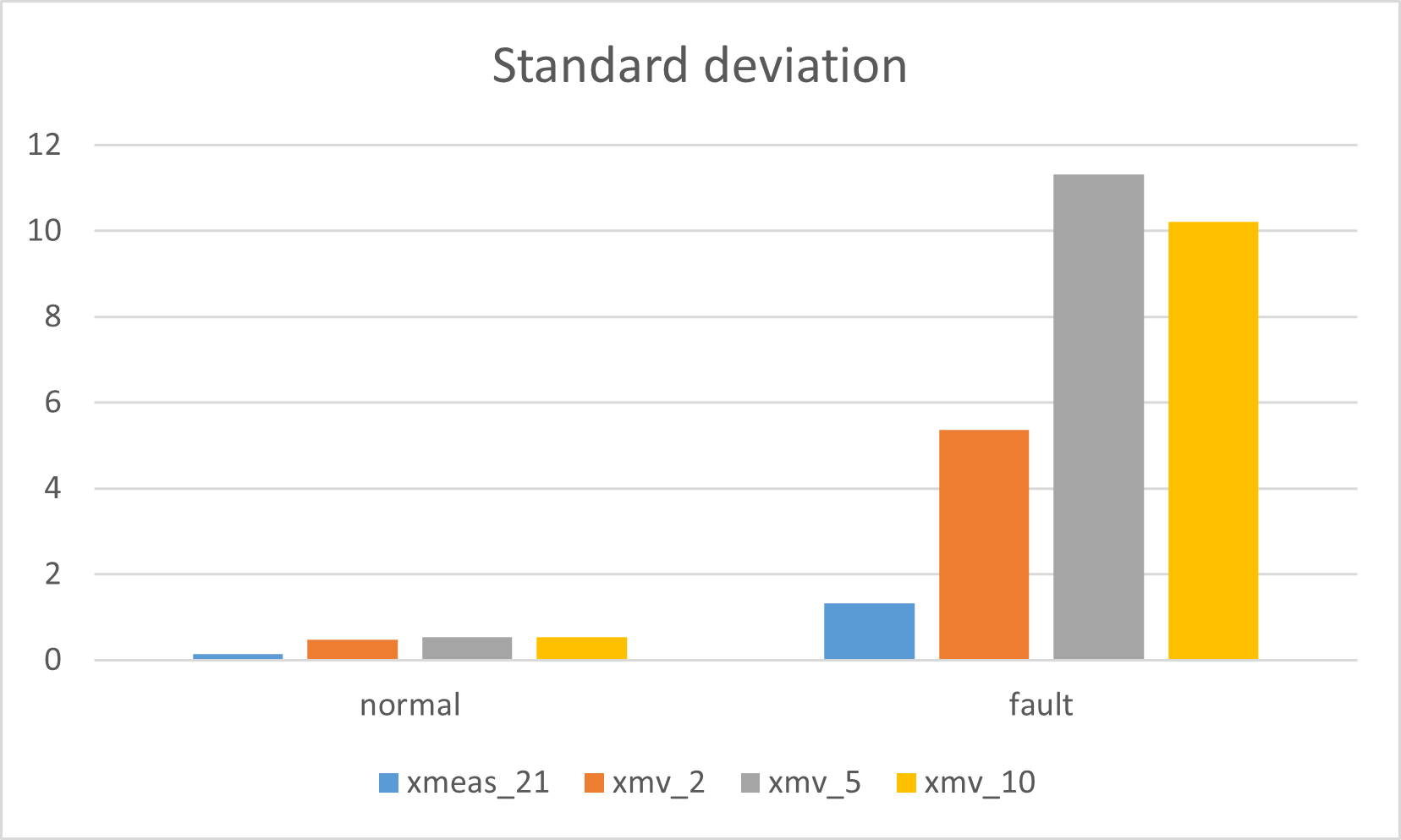


Figure 26: Standard deviation of TEP variables (Own presentation).

#### Data Distribution

We explore the data distributions of 52 TEP variables. As Figure 27 shows, the 1st measured variable follows a normal distribution. After calculation, all other 51 TEP variables also follow normal distributions.

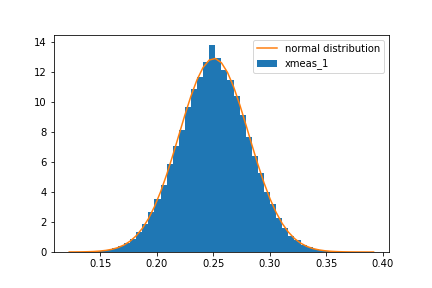


Figure 27: xmeas\_1 distribution (Own presenatation).

#### Data Correlation.

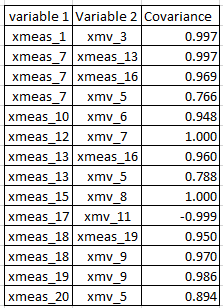
We calculate the Pearson covariance coefficient of the normal training dataset. The covariance is calculated with the formula:

|  |  |
| --- | --- |
|  | ( 4.2 ) |

where is the expectation, is the variable and is the standard deviation.

We preserve the TEP variable pairs with covariance value and create Table 5.

Table 5: Pearson covariance coefficient of the normal dataset (Own presentation).



In Table 5, the measured variable has high covariance with , and . We then plot the scatter plot of and in Figure 28. and have a positive covariance, i.e., when has a higher value at a time step, is expected to have a high value at the same time step.

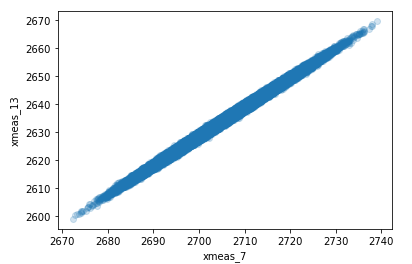


Figure 28: Scatter plot of xmeas\_7 and xmeas\_13 (Own presentation).

#### Visualization with PCA

We use PCA to extract the first two principal components of data samples of fault 1, 2, 3, as Figure 29 shows. Three types of data samples overlap, and no clear boundaries exist. One reason is that PCA is a linear method that can separate dissimilar data instances in low-dimensional representation. However, it is critical for high-dimensional and nonlinear data like TEP data to maintain a close distance between similar data samples (van der Maaten and Hinton, 2008).

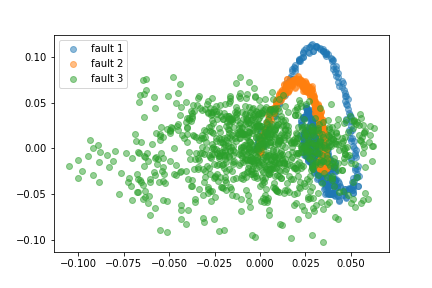


Figure 29: PCA visualization of fault 1, 2, 3 (Own presentation).

#### Visualization with t-SNE

t-SNE is a visualization for high-dimensional data capable of preserving both the global and local data structures in low-dimensional space. Unlike PCA, which separates data using the global structure, t-SNE focuses on converting the data samples into low-dimensional space using the pairwise similarities (or local structure). Furthermore, t-SNE has series of tunable parameters, “perplexity” and “steps” are two parameters that can influence the performance significantly. “perplexity” plays the role of adjusting the effective number of neighbors. In other words, it can balance the global and local structure of the data samples. We use t-SNE to plot the data samples of faults 1, 2, and 3. As Figure 30 shows, there is a clear boundary between data samples of faults 1 and 2. However, data samples of fault 3 overlap with the data samples of fault 2, which implies that fault 3 is hard to distinguish from other faults. We also observe similar results in the experiments.

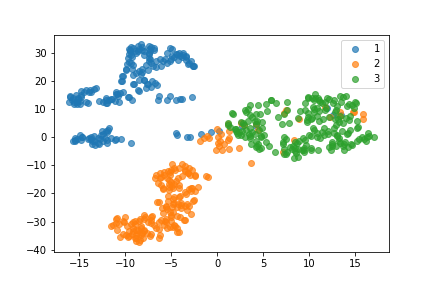


Figure 30: t-SNE visualization of fault 1, 2, 3 (Own presentation).

## Experiment Environment

The experiments are conducted on a GPU Workstation with Ubuntu 20.04 operating system. Specifics of the software and hardware are in Table 6.

Table 6: Experiment environment (Own presentation).

|  |  |
| --- | --- |
| Operation system | Ubuntu Linux 20.04 |
| Memory | 32 GB |
| Processor | AMD Ryzen 9 3900X 12 Core@3.8GHz |
| Graphic | NVidia Geforce 2080ti 12GB |
| Integrated Development Environment | Anaconda Jupyter Notebook |

## Baseline and Evaluation Metrics

A baseline can help us recognize the potential challenges, provide trivially attainable performance without costing too much time in training and debugging. We build a naïve autoencoder as the baseline and use the evaluation metrics introduced in Chapter 2.5 to illustrate the performance. Autoencoders are chosen as the baseline because they are easy to train, fast to inference, and provide remarkable anomaly detection performance (Hawkins et al., 2002).

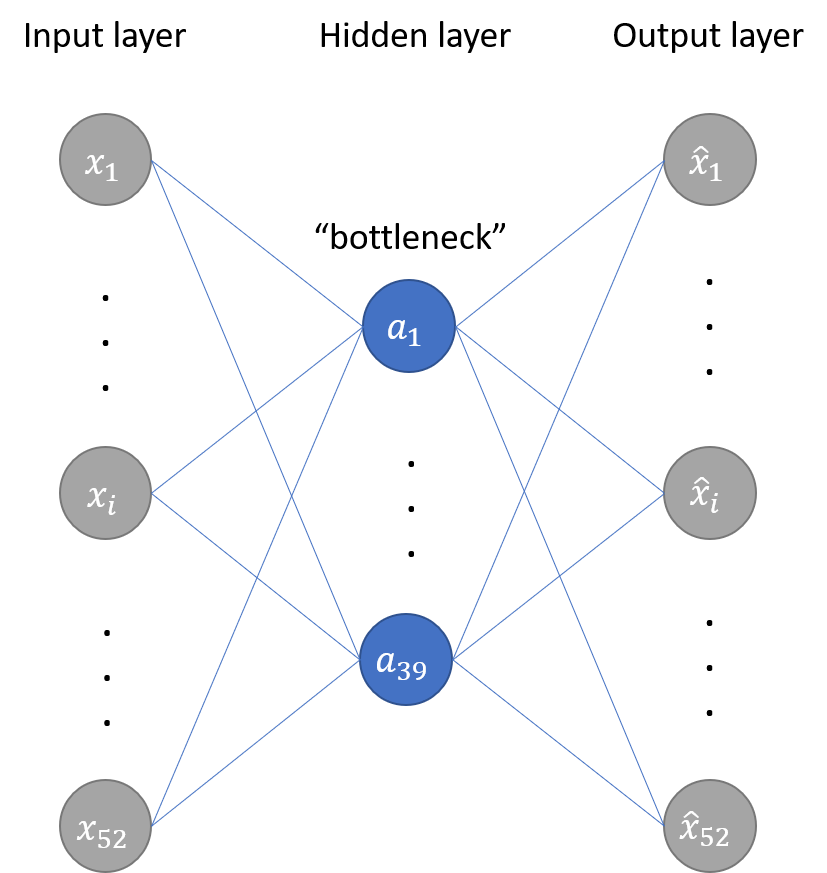
******

Figure 31: Baseline autoencoder structure (Own presentation).

The baseline is an autoencoder with one hidden layer, as Figure 31 shows. The 52 TEP variables are first fed into the input layer, then compressed by hidden layers, and finally reconstructed at the output layer. The loss function is the reconstruction error (MSE) between input and output:

|  |  |
| --- | --- |
|  | ( 4.3 ) |

Where represents the vectors of input and is the vectors of output.

Before the experiment, we first standardize all the datasets as Section 4.1.2 and train the baseline autoencoder with the normal training dataset. The training process takes about ten epochs to finish. After the training, we use the normal testing dataset and fault testing dataset to calculate the false alarm rate (FAR) and fault detection rate (FDR). A threshold value is selected to detect anomalies by thresholding the reconstruction error. Testing data samples with are classified as anomalies. is chosen such that the FAR of the normal training dataset is kept around 10%. The complete training and testing processes are repeated multiple times to mitigate the influence caused by the stochastic nature of the algorithm.

As Figure 32 and Figure 33 show, the normal testing dataset could be reconstructed moderately. By contrast, certain variables of the fault testing dataset can be barely reconstructed because the numerical range is far beyond the range of the activation function.

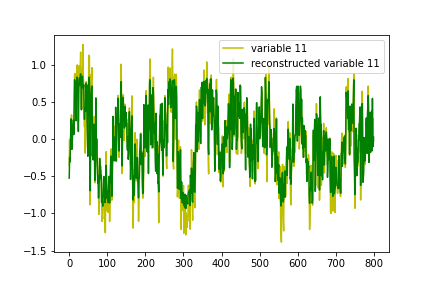


Figure 32: xmeas\_11 and reconstructed xmeas\_11 in normal dataset.

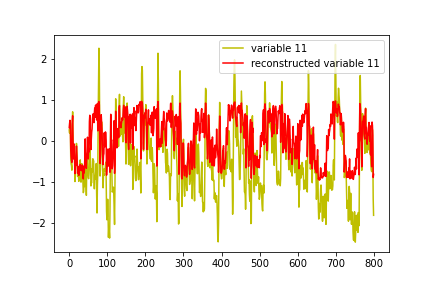
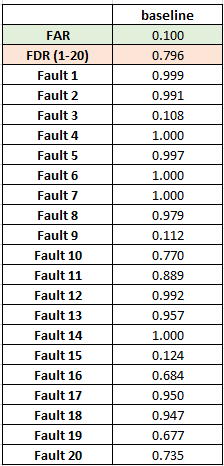


Figure 33: xmeas\_11 and reconstructed xmeas\_11 in fault dataset (fault 0).

As shown in Table 7, the FAR is 9.9% and FDR for 20 faults is 79.4%. It is worth noting that the fault detection rate for faults 3, 9, 15 is less than 13%. Zhang (2009) explained that the high missing detection is caused by the absence of observable changes in the process variables regarding mean, standard variance, or other high order variance. That is why in most literature (Heo and Lee, 2019; Yin et al., 2014), those faults are often not considered. For faults 10, 11, 16, 19, and 20, the FDR is around 70%. For the rest faults, the FDR can reach above 95%.

Table 7: Baseline autoencoder fault detection performance (Own presentation).



Next, we plot the ROC curve and calculate the ROC AUC score of the baseline autoencoder. As Figure 34 shows, the ROC AUC score of the baseline autoencoder is 0.8973. As discussed in Section 2.4.2, a higher ROC AUC score indicates that more threshold values can separate the normal and fault data samples. Therefore, the high ROC AUC score suggests that the baseline autoencoder exhibits a decent performance in separating normal and fault data samples.

Since the baseline autoencoder detects anomalies by thresholding the reconstruction error, no probabilistic prediction is available. Therefore, the reliability diagram, brier score, and log-loss are not calculated.

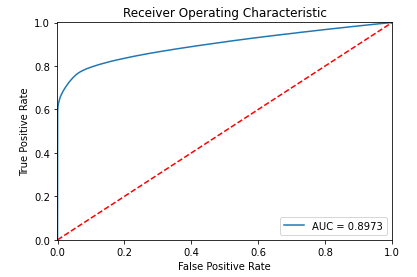


Figure 34: AUC ROC score of baseline autoencoder (Own presentation).

## Experiment Result and Analysis

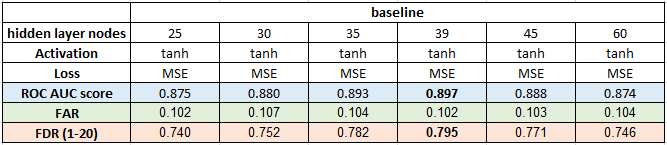
We have obtained a basic understanding of the anomalies in the Tennessee Eastman process by building the baseline autoencoder. The faults whose TEP variables with high variance and large numerical range are trivial to detect. In this section, we investigate the anomaly detection performance of series machine learning algorithms.

### Autoencoders and Regularization

In this section, we attempt to improve the performance of the baseline autoencoder by adjusting the number of nodes in the hidden layer and deploying various regularization methods. Bishop (1995a) proposed two approaches to improve the generalization property. The first approach is to adjust the number of hidden units in the neural network. For deep learning models, the number of nodes and the number of layers determine the model’s capacity. As the number of nodes and number of layers increases, the deep learning model has a higher representational capacity. In other words, it is capable of representing more complex functions (Goodfellow et al., 2017, p. 428). However, as Section 2.3.2 discussed, the model tends to overfit (resp. underfit) when the capacity is larger (resp. smaller) than optimal. The second approach is to adjust the complexity of the model using methods such as regularization techniques. The regularization techniques include weight regularization, activity regularization, weight constraints, noise regularization, and dropout.

We first use the exhaustive grid search (Pedregosa et al., 2011) to search the optimal number of nodes in the hidden layer. Then we use the random grid search (Bergstra and Bengio, 2012) to select the optimal parameters for different regularization techniques. The experiment result is shown in Table 8. We find that the number of nodes in the hidden layer has the most influence on the performance. We also find that Glorot uniform initializer (Glorot and Bengio, 2010) can stabilize the performance of the autoencoders in each training process. Activation function and loss function MSE can guarantee the best performance.

Table 8: Performance of baseline autoencoder (different hidden layer nodes) (Own presentation).



From Figure 35, a hidden layer with 39 nodes has the best performance regarding FDR and AUC ROC score

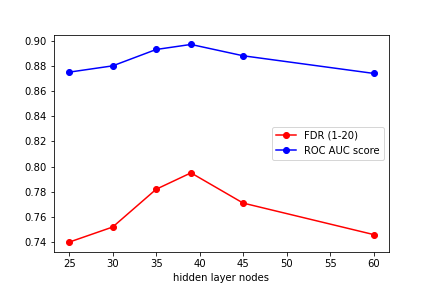
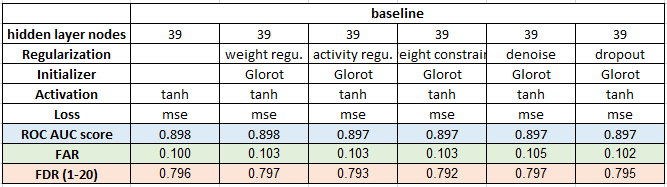


Figure 35: Baseline autoencoder FDR and AUC ROC score (different hidden layer nodes).

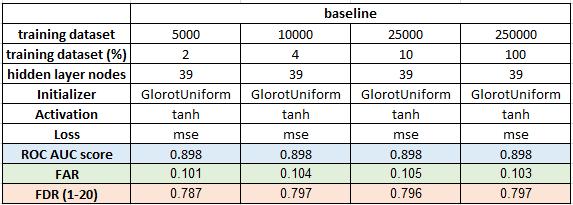
Then, we investigate the influence of different regularization techniques. The weight regularization, activity regularization, and weight constraints are applied to the hidden layers. To realize noise regularization, we add Gaussian noise to different layers in the autoencoder. The dropout layer is also similarly added to different layers of the networks. However, no noticeable performance improvement is observed after applying regularization techniques, as Table 9 shows. Significantly, the activity regularization and weight constraints degrade the performance slightly. One explanation is that the baseline autoencoder has reached the optimal capacity, and a large dataset can improve generalization better than those regularization techniques.

Table 9: Performance of autoencoder (different regularization techniques) (Own presentation).



We further explore the relationship between the number of normal training data samples and the autoencoder’s performance. Table 10 shows no performance improvement when more than 4% of the normal training data samples are used for training. No further improvement even trained with more data samples implies that the baseline autoencoder has reached the optimal capacity as discussed in Section 2.3.2.

Table 10: Performance of autoencoder (different size of the normal training dataset) (Own presentation).



### Deep Autoencoders

To further explore the performance of autoencoders, we test the autoencoders with three and five hidden layers. Two types of autoencoders structures are tested in the experiments. The first type of autoencoders has a progressively decreasing and increasing structure, as Figure 36 shows. The second type of autoencoders has an overcomplete hidden layer (Vincent et al., 2008), as Figure 37 shows. We use the random grid search (Bergstra and Bengio, 2012) to select the number of nodes in each hidden layer and use backpropagation to update the weights.

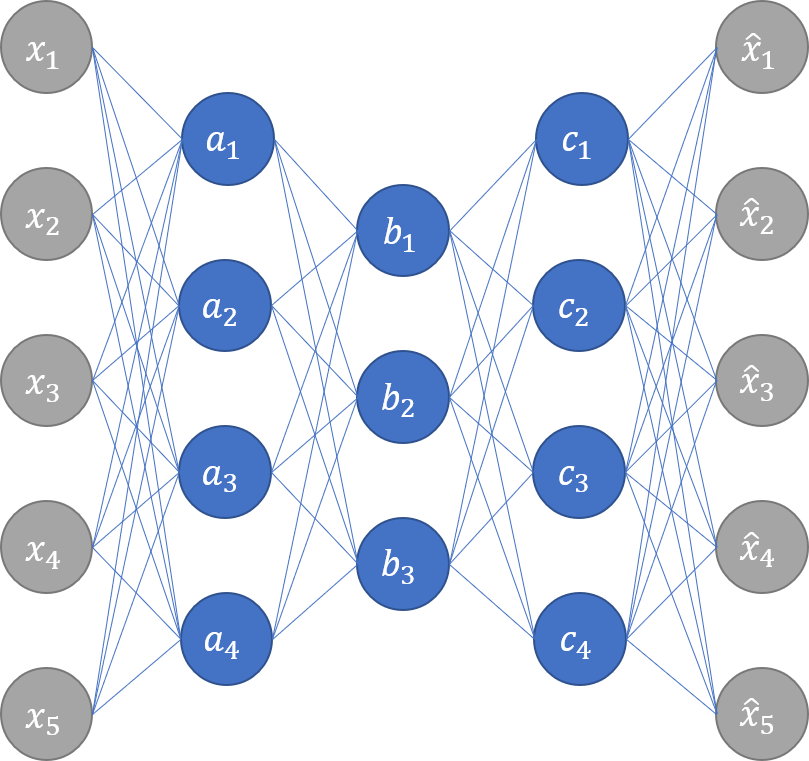


Figure 36: Autoencoder with progressively decreasing layer nodes (Own presentation).

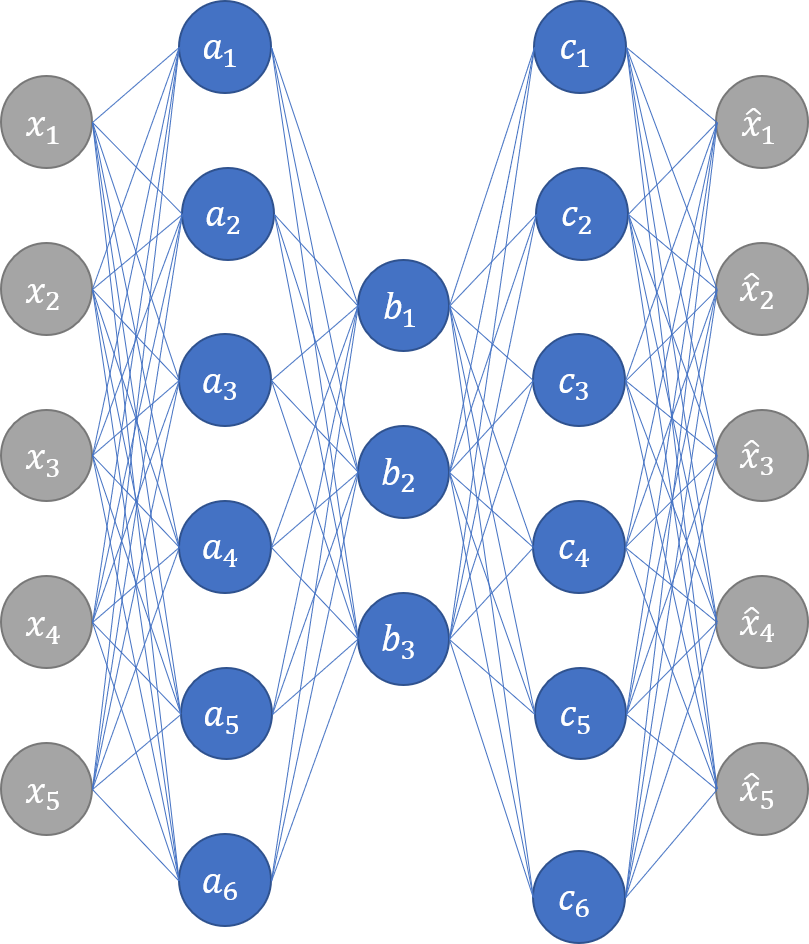
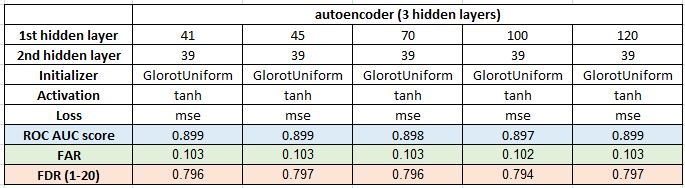


Figure 37: Autoencoder with overcomplete layer (Own presentation).

The result of the autoencoder with three hidden layers is in Table 11. Extra hidden layers do not improve the performance. In contrast, we find that the autoencoders are more prone to overfit the normal training dataset than the baseline autoencoder. The overfitting can be mitigated by deploying activity regularization in hidden layers or more strict early stopping criteria (stop training immediately when no improvement). However, the number of nodes in the middle hidden layer (bottleneck) dramatically impacts the anomaly detection performance. A middle hidden layer with 39 nodes (same as the hidden layer of the baseline autoencoder) can achieve the best anomaly detection performance. We also test a deeper autoencoder with five hidden layers. However, the performance result is similar to the autoencoder with three hidden layers, and the 39 nodes in the middle hidden layer (bottleneck) is the optimal value.

The experiment result implies that more capacity does not bring performance improvement for autoencoders; instead increases the possibility of overfitting. In addition, the number of nodes in the middle hidden layer determines the anomaly detection performance.

Table 11: Performance of autoencoder with three hidden layers (Own presentation).



### Traditional Machine Learning Classifiers

The autoencoders introduced before are trained in an unsupervised way, i.e., only normal training data samples are involved during the training. In contrast, classifiers work by building direct mapping between input (normal and fault data samples) and output (label). We have tested series of traditional machine learning classifiers and compared their performance. Random forest classifiers, decision tree classifiers, and AdaBoost classifiers need no data standardization. They can be trained directly with the raw TEP simulation data (Rieth et al., 2017). For support vector classifiers, the data is pre-processed by standardization transformation. The training data samples are created by shuffling the normal and fault training data samples. The parameters of each classifier are searched with the exhaustive grid search (Pedregosa et al., 2011). The experiment result is in Table 12. The support vector classifier has the lowest FDR and longest training time, whereas the random forest classifier has the highest FDR for 20 faults. Compared to baseline autoencoder, random forest classifiers can detect faults 3, 9, 10, 15, 16, 19 with much higher FDR. ??? wait for calculation result

Table 12: Performance of traditional machine learning classifiers (Own presentation).



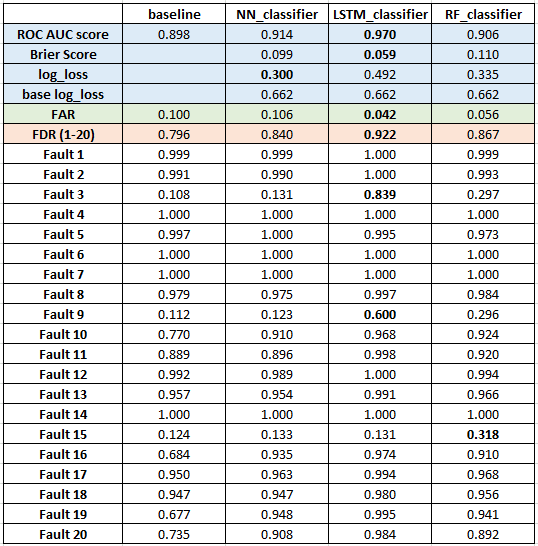
### Neural Network Classifiers

This section builds two deep learning binary classifiers and compares them with the traditional machine learning classifiers. The first classifier (NN\_classifier) is a fully connected neural network classifier. Based on the neural network classifier, the second classifier (LSTM\_classifier) deploys additional LSTM network layers. The detailed network structure can be found in Appendix. The training data is created by shuffling the normal and fault training data and standardized as Section 4.1.2. Early stopping is set to prevent overtraining. At the output layer, the activation function is Sigmoid, and the loss function is binary cross-entropy which is defined as:

|  |  |
| --- | --- |
|  | ( 4.4 ) |

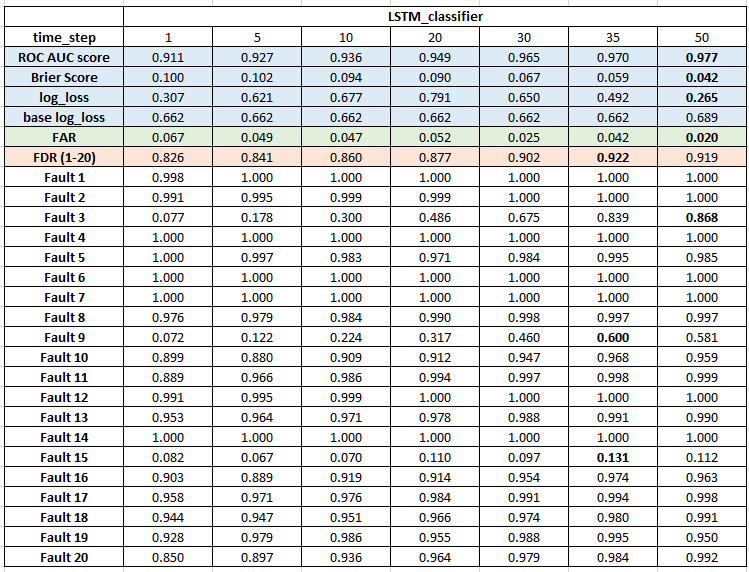
where is the number of output samples, is the expected output and is the actual output. For LSTM classifier, we choose a time\_step=35, i.e., we use the original input data samples from time to as one new input data sample for time . The experiment result is in Table 13. Faults 3, 9, 15 are still difficult to detect compared to other faults. However, the LSTM classifier has a huge advantage in detecting faults 3, 9, 20 and achieves the best FDR (92%) regarding all faults. The random forest classifier has the highest FDR for fault 15 (31%). With the additional LSTM layers, the LSTM classifier has improved the FDR for faults 3, 9, 20 significantly.

Table 13: Performance of neural network classifiers (Own presentation).



As discussed in Section 3.5.5, the LSTM network can learn the temporal correlation of data samples. To further investigate the contextual relation of data samples, we train the LSTM classifier with different time steps. The detailed result can be found in Table 14. Figure 38 shows that as the time step increases, the general performance improves. For example, for faults 3, the FDR increases significantly from 7% to 86%; for fault 9, the FDR increases from7.2% to 58.1%. In addition, Figure 39 shows that the FDR regarding all faults and ROC AUC score also increase as time steps increases. However, for time steps larger than 50, no further improvement regarding evaluation metrics is observed. In the experiments, we also find that the training and inference time increases significantly as the time step increases.

Table 14: Performance of LSTM classifiers (different time step size) (Own presentation).



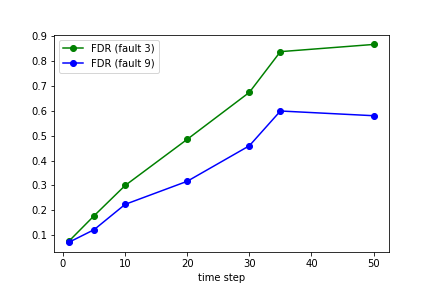


Figure 38: LSTM classifier FDR of fault 3, 9 (different time step) (Own presentation).

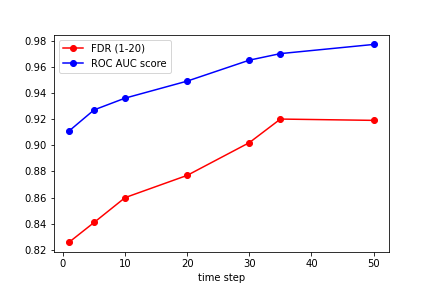


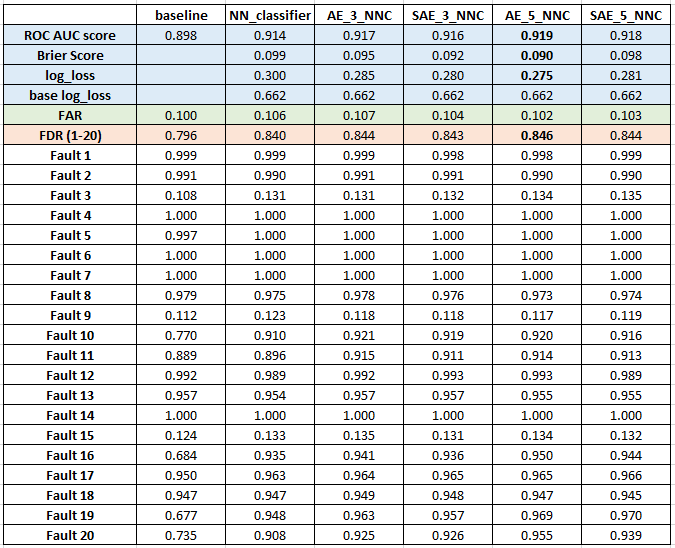
Figure 39: LSTM classifier FDR and ROC AUC score (different time step) (Own presentation).

### Autoencoders with Classifiers

In this section, we combine autoencoders and deep learning classifiers, i.e., extracting the features of TEP variables with the encoder part of autoencoders and detect anomalies based on those features. We investigate different approaches to train the combined networks. First, we train the autoencoder and classifier (AE\_3\_NNC, AE\_5\_NNC) simultaneously. Second, we used the pre-training approach introduced in Section 3.5.2 to initialize the parameters of the encoder then fine-tune the parameters of both the encoder and classifiers (SAE\_3\_NNC, SAE\_5\_NNC) in a supervised way. Third, we keep the parameters of the pre-trained encoder fixed and train the classifier.

A performance improvement is expected if the extracted features can preserve more discriminative information and help to build a clear boundary between normal and fault data samples. From the result in Table 15, compared to NN\_classifier, AE\_5\_NNC (five-layer autoencoder with neural network classifier) improve the FDR (from 0.840 to 0.846) and ROC AUC score (from 0.914 to 0.919). The result implies that pre-training and fine-tuning can improve anomaly detection.

Table 15: Performance of stacked autoencoders (Own presentation).

****

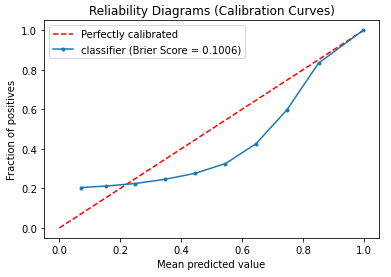
****

Figure 40: update figure

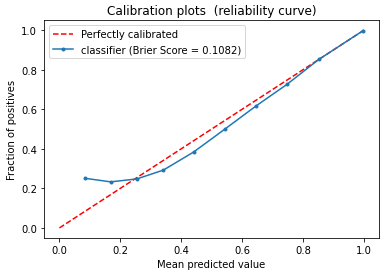


Figure 41: update figure

### Generative Adversarial Networks

We also attempt to train an autoencoder with a GANomaly-based framework. In the GANomaly-based framework, the generator is similar to an autoencoder and learns to reconstruct the input data samples; the discriminator learns to assess if the reconstructed input data samples are normal or fault data samples. Based on the assumption, if the autoencoder is only trained with normal data samples, it cannot reconstruct the fault data samples. Then, using the reconstruction error of the autoencoder and output of the discriminator, we can calculate an anomaly score and identify the anomalies. However, the limited experiments did not yield satisfactory results regarding the TEP simulation data (Rieth et al., 2017). Two possible reasons could cause the failure. The first possibility is that the generator and discriminator learn at different speeds, which breaks the equilibrium, and both parties fail to update the weights. The second reason is that the loss function is not guiding the generator towards reconstructing the input data samples. For future research, we could track both generator and discriminator's learning rate and gradients to ensure a balanced weight updating. Further, we could also try different loss functions, adjust the weight of each sub-loss function, and attempt different techniques to initialize the weights.

# Conclusion and Future Work

## Conclusion

This thesis aimed to tackle the anomaly detection of the Tennessee Eastman process using different machine learning algorithms. We first build an autoencoder as the baseline and evaluate traditional machine learning models for anomaly detection. Then we implement series of deep learning models and compare their performance with traditional machine learning models. The result showed that the deep learning model could achieve better anomaly detection performance with less training time. Further, the inference time of deep learning models is usually faster than the traditional machine learning model, which is critical in industrial monitoring.

In general, deep learning can be used to detect anomalies in two approaches. The first approach is to use the feature extractor to extract the features and calculate the anomaly score by representing the features. The second approach is to deploy the feature extractor to extract the features and train a classifier with the extracted features in a supervised way. For the first approach, we build baseline autoencoders and detect anomalies by thresholding the reconstruction error. We further explore deeper autoencoders with various regularization techniques, which is expected to improve the generalization property. However, as the experiment result showed, a deeper autoencoder does not contribute to anomaly detection; instead, the deeper autoencoders are prone to overfitting. The experiment result also implies that the number of nodes in the middle hidden layer (bottleneck) of autoencoders is critical for the anomaly detection performance. One advantage of the feature representation approach is that only non-labeled normal training data samples are needed. In a practical industrial process, normal operation data samples are easy to obtain and require no expert’s knowledge to annotate the label. For the second approach, we combine the autoencoder with neural network classifiers, i.e., extract the features with the encoder part of the autoencoder and identify anomalies with the classifiers. The experiment result shows that the extracted features can merely improve classification accuracy to a lesser degree. To further investigate if the temporal relationship between data samples can provide more discriminative information for anomaly detection, we implement the LSTM classifiers. The experiment result shows that LSTM classifiers have the best performance, especially for faults such as 3 and 9, which are hard to detect. This implies that the temporal relationship of the TEP simulation data can provide valuable patterns for anomaly detection. Finally, we attempt to implement the GAN-based anomaly detection model, which combines autoencoder and LSTM networks. However, the limited experiments do not yield satisfactory results due to imbalanced learning between generator and discriminator and improper loss function. Further experiments are needed to overcome those problems

## Future Work

Based on the experiments conducted during this research, the following recommendations for future work are proposed:

* Deep learning model optimization. In the experiments, we have applied exhaustive grid search (Pedregosa et al., 2011) and random grid search (Bergstra and Bengio, 2012) to tune the model hyperparameters. Intensive investigation of network configurations for complex models such as stacked autoencoders and LSTM classifiers could further improve the anomaly detection performance.
* Generative deep learning models. Han et al. (2020) and Dan Li et al. (2019) have shown the potential of using generative models, which combine GAN and LSTM, to exploit the spatial-temporal correlation for multivariant time series data. In addition, the generative models are trained in an unsupervised way, which requires no label but only normal data samples. One promising direction is to develop unsupervised generative models for anomaly detection in the Tennessee Eastman process.
* LSTM autoencoders. The experiments showed that the autoencoder had an advantage in detecting anomalies with the unsupervised training method. Besides, the LSTM classifiers have proved that the spatial-temporal relation between TEP variables can further improve the anomaly detection performance. Therefore, developing an LSTM autoencoder is a promising research direction (Agarwal et al., 2020).
* Feature selection and engineering. In the experiments, we use all features (52 TEP variables) of TEP simulation data (Rieth et al., 2017) as the input. Xavier and Seixas (2018) used the 11 manipulated variables to input an LSTM classifier for anomaly detection. However, in our experiments, only 11 manipulated variables as input lead to the model’s degrading performance. The main reason is that we apply a different TEP simulation dataset. For most machine learning algorithms, selecting or creating the more relevant features can significantly speed up the training and improve the discriminative capability. Therefore, studies of selecting relevant features (Bedoya et al., 2012; Chebel-Morello et al., 2016) is important research direction.
* Theoretical limits for anomaly detection performance. Since the TEP simulation data (Rieth et al., 2017) are generated in each simulation run with a random state, it is worth studying the influence on performance due to the number of simulations, the number of samples in each simulation, and the proportional of normal and fault data in training datasets. Understanding those theoretical limits can help us to design more effective deep learning models.
* Effect of noise in TEP data. The TEP simulation data (Rieth et al., 2017) is generated in the simulation software, and no errors/mistakes are found in the data pre-processing steps. However, in real industrial monitoring, noise or measurement errors caused by humans are inevitable. Therefore, building anomaly detection models resilient to the above problems is crucial to applying the anomaly detection model in actual industrial production.

# Appendix

Den Anhang auf neue Seite – ggf. neuen Abschnitt, wenn hier andere Nummerierung notwendig wird.

Bibliography

Agarwal, P., Gonzalez, J., Elkamel, A. and Budman, H. (2020) *Hierarchical Deep Recurrent Neural Network based Method for Fault Detection and Diagnosis* [Online].

Akcay, S., Atapour-Abarghouei, A. and Breckon, T. P. (2018) *GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training* [Online]. Available at https://​arxiv.org​/​pdf/​1805.06725.

Amamra, A., Robert, J.-M., Abraham, A. and Talhi, C. (2016) ‘Generative versus discriminative classifiers for android anomaly-based detection system using system calls filtering and abstraction process’, *Security and Communication Networks*, vol. 9, no. 16, pp. 3483–3495.

Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J., Fan, L., Fougner, C., Han, T., Hannun, A., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J. and Zhu, Z. (2015) *Deep Speech 2: End-to-End Speech Recognition in English and Mandarin* [Online]. Available at https://​arxiv.org​/​pdf/​1512.02595.

Bahnsen, A. C. (2017) *Building AI Applications Using Deep Learning* [Online]. Available at https://​albahnsen.com​/​2017/​06/​06/​building-​ai-​applications-​using-​deep-​learning/​.

Bedoya, C., Uribe, C. and Isaza, C. (2012) ‘Unsupervised Feature Selection Based on Fuzzy Clustering for Fault Detection of the Tennessee Eastman Process’, pp. 350–360.

Behera, R. N. and Das, K. (2017) ‘A Survey on Machine Learning: Concept, Algorithms and Applications’, *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, no. 2 [Online]. Available at https://​www.researchgate.net​/​profile/​rabi\_​behera/​publication/​316273553\_​a\_​survey\_​on\_​machine\_​learning\_​concept\_​algorithms\_​and\_​applications.

Bellman, R. (1966) ‘Dynamic programming’, *Science*, vol. 153, no. 3731, pp. 34–37.

Bengio, Y. (2012) ‘Deep Learning of Representations for Unsupervised and Transfer Learning’, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 17–36 [Online]. Available at http://​proceedings.mlr.press​/​v27/​bengio12a.

Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H. (2007) ‘Greedy layer-wise training of deep networks’, *Advances in neural information processing systems*, vol. 19, p. 153.

Bengio, Y., Simard, P. and Frasconi, P. (1994) ‘Learning long-term dependencies with gradient descent is difficult’, *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166.

Bergstra, J. and Bengio, Y. (2012) ‘Random Search for Hyper-Parameter Optimization’, *J. Mach. Learn. Res.*, vol. 13, null, pp. 281–305.

Bianchini, M. and Scarselli, F. (2014) ‘On the complexity of neural network classifiers: a comparison between shallow and deep architectures’, *IEEE transactions on neural networks and learning systems*, vol. 25, no. 8, pp. 1553–1565.

Bishop, C. M. (1995a) *Neural Networks for Pattern Recognition*, USA, Oxford University Press, Inc.

Bishop, C. M. (1995b) ‘Training with Noise is Equivalent to Tikhonov Regularization’, *Neural computation*, vol. 7, no. 1, pp. 108–116.

Bouchard, G. and Triggs, B. (2004) ‘The Tradeoff Between Generative and Discriminative Classifiers’, *IASC International Symposium on Computational Statistics (COMPSTAT)*.

Bradley, A. P. (1997) ‘The use of the area under the ROC curve in the evaluation of machine learning algorithms’, *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159 [Online]. DOI: 10.1016/S0031-3203(96)00142-2.

Breiman, L. (2001) ‘Random Forests’, *Machine Learning*, vol. 45, no. 1, pp. 5–32.

BRIER, G. W. (1950) ‘VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY’, *Monthly Weather Review*, vol. 78, no. 1, pp. 1–3.

Bröcker, J. and Smith, L. A. (2007) ‘Increasing the Reliability of Reliability Diagrams’, *Weather and Forecasting*, vol. 22, no. 3, pp. 651–661 [Online]. DOI: 10.1175/WAF993.1.

Brownlee, J. (2019) *Generative Adversarial Networks with Python: Deep Learning Generative Models for Image Synthesis and Image Translation*, Machine Learning Mastery.

Brownlee, J. (2020) *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*, Machine Learning Mastery.

Budnarain, C. (2020) *Unsupervised Deep Learning for Anomaly Detection and Explanation in Sequential Data: Unsupervised Deep Learning for Anomaly Detection and Explanation in Sequential Data* [Online]. Available at https://​tspace.library.utoronto.ca​/​handle/​1807/​101180.

Burlina, P., Joshi, N. and Wang, I.-J. (2019) ‘Where's Wally Now? Deep Generative and Discriminative Embeddings for Novelty Detection’, *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition: CVPR 2019 : 16-20 June 2019, Long Beach, California : proceedings.* Piscataway, NJ, IEEE.

Carrara, F., Amato, G., Brombin, L., Falchi, F. and Gennaro, C. (2020) *Combining GANs and AutoEncoders for Efficient Anomaly Detection* [Online]. Available at https://​arxiv.org​/​pdf/​2011.08102.

Carrasco, O. C. (2019) *Support Vector Machines for Classification* [Online]. Available at https://​towardsdatascience.com​/​support-​vector-​machines-​for-​classification-​fc7c1565e3.

Chalapathy, R. and Chawla, S. (2019) *Deep Learning for Anomaly Detection: A Survey*.

Chandola, V., Banerjee, A. and Kumar, V. (2009) ‘Anomaly detection: A survey’, *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58.

Chebel-Morello, B., Malinowski, S. and Senoussi, H. (2016) ‘Feature selection for fault detection systems: application to the Tennessee Eastman process’, *Applied Intelligence*, vol. 44, no. 1, pp. 111–122.

Chen, J., Sathe, S., Aggarwal, C. and Turaga, D. (2017) ‘Outlier Detection with Autoencoder Ensembles’, in, pp. 90–98.

Cristianini, N. and Shawe-Taylor, J. (2006) *An introduction to support vector machines: And other kernel-based learning methods*, Cambridge, New York, Cambridge University Press.

Csáji, B. C. (2001) *Approximation with artificial neural networks* [Online], Hungary. Available at https://​citeseerx.ist.psu.edu​/​viewdoc/​download​?​doi=​10.1.1.101.2647&​rep=​rep1&​type=​pdf.

Dan Li, Dacheng Chen, Lei Shi, Baihong Jin, Jonathan Goh and See-Kiong Ng (2019) *MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks* [Online].

Dembla, G. (2020a) *Intuition behind Log-loss score* [Online]. Available at https://​towardsdatascience.com​/​intuition-​behind-​log-​loss-​score-​4e0c9979680a.

Dembla, G. (2020b) *Intuition behind ROC-AUC score* [Online]. Available at https://​towardsdatascience.com​/​intuition-​behind-​roc-​auc-​score-​1456439d1f30.

Di Mattia, F., Galeone, P., Simoni, M. de and Ghelfi, E. (2019) *A Survey on GANs for Anomaly Detection*.

Ding, S. X., Zhang, P., Jeinsch, T., Ding, E. L., Engel, P. and Gui, W. (2011) ‘A survey of the application of basic data-driven and model-based methods in process monitoring and fault diagnosis’, *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 12380–12388 [Online]. DOI: 10.3182/20110828-6-IT-1002.02842.

Doersch, C. (2016) *Tutorial on Variational Autoencoders* [Online]. Available at http://​arxiv.org​/​pdf/​1606.05908v3.

Domingos, P. (2012) ‘A few useful things to know about machine learning’, *Communications of the ACM*, vol. 55, no. 10, pp. 78–87.

Donahue, J., Krähenbühl, P. and Darrell, T. (2017) *Adversarial Feature Learning*.

Downs, J. J. and Vogel, E. F. (1993) ‘A plant-wide industrial process control problem’, *Computers & Chemical Engineering*, vol. 17, no. 3, pp. 245–255 [Online]. DOI: 10.1016/0098-1354(93)80018-I.

Erfani, S. M., Rajasegarar, S., Karunasekera, S. and Leckie, C. (2016) ‘High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning’, *Pattern Recognition*, vol. 58, pp. 121–134 [Online]. DOI: 10.1016/j.patcog.2016.03.028.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P. and Bengio, S. (2010) ‘Why Does Unsupervised Pre-Training Help Deep Learning?’, *J. Mach. Learn. Res.*, vol. 11, pp. 625–660.

Fan, J., Wang, W. and Zhang, H. (2017) ‘AutoEncoder based high-dimensional data fault detection system’, *2017 IEEE 15th International Conference on Industrial Informatics (INDIN): University of Applied Science Emden/Leer, Emden, Germany, 24-26 July 2017 : proceedings.* Emden, 7/24/2017 - 7/26/2017. Piscataway, NJ, IEEE, pp. 1001–1006.

Feng, W., Guan, N., Li, Y., Zhang, X. and Luo, Z. (2017) ‘Audio visual speech recognition with multimodal recurrent neural networks’, pp. 681–688.

FISHER, R. A. (1936) ‘THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS’, *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188.

Frasconi, P., Gori, M. and Soda, G. (1992) ‘Local Feedback Multilayered Networks’, *Neural computation*, vol. 4, no. 1, pp. 120–130.

Gao, X. and Hou, J. (2016) ‘An improved SVM integrated GS-PCA fault diagnosis approach of Tennessee Eastman process’, *Neurocomputing*, vol. 174, pp. 906–911 [Online]. DOI: 10.1016/j.neucom.2015.10.018.

Gareth, J., Daniela, W., Trevor, H. and Robert, T. (2013) *An introduction to statistical learning: with applications in R*, Spinger.

Gauthier, J. (2015) ‘Conditional generative adversarial nets for convolutional face generation’.

Geiger, A., Liu, D., Alnegheimish, S., Cuesta-Infante, A. and Veeramachaneni, K. (2020) *TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks*.

Gers, F. A., Schmidhuber, J. and Cummins, F. (1999) ‘Learning to forget: Continual prediction with LSTM’.

Glasmachers, T. (2017) *Limits of End-to-End Learning* [Online]. Available at https://​arxiv.org​/​pdf/​1704.08305.

Glorot, X. and Bengio, Y. (2010) ‘Understanding the difficulty of training deep feedforward neural networks’, *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256.

Glorot, X., Bordes, A. and Bengio, Y. (2011) ‘Deep Sparse Rectifier Neural Networks’, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics.* Fort Lauderdale, FL, USA, PMLR, pp. 315–323.

Goodfellow, I., Bengio, Y. and Courville, A. (2017) *Deep Learning* [Online], Cambridge, Mass., MIT Press Ltd. Available at http://​www.deeplearningbook.org​/​.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014) *Generative Adversarial Networks*.

Han, C., Rundo, L., Murao, K., Noguchi, T., Shimahara, Y., Milacski, Z., Koshino, S., Sala, E., Nakayama, H. and Satoh, S. (2020) *MADGAN: unsupervised Medical Anomaly Detection GAN using multiple adjacent brain MRI slice reconstruction* [Online].

Hawkins, D. M. (1980) *Identification of Outliers*, Dordrecht, Springer Netherlands.

Hawkins, S., He, H., Williams, G. J. and Baxter, R. A. (2002) ‘Outlier Detection Using Replicator Neural Networks’, *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery.* Berlin, Heidelberg, Springer-Verlag, pp. 170–180.

He, K., Zhang, X., Ren, S. and Sun, J. (2016) ‘Deep Residual Learning for Image Recognition’, *Proceedings, 29th IEEE Conference on Computer Vision and Pattern Recognition: 26 June-1 July 2016, Las Vegas, Nevada.* New York, IEEE.

Heo, S. and Lee, J. H. (2019) ‘Statistical Process Monitoring of the Tennessee Eastman Process Using Parallel Autoassociative Neural Networks and a Large Dataset’, *Processes*, vol. 7, no. 7, p. 411.

Higa, K., Sato, H., Shiraishi, S., Kikuchi, K. and Iwamoto, K. (2019) ‘Anomaly Detection Combining Discriminative and Generative Models’, *2019 IEEE International Conference on Imaging Systems and Techniques (IST)*, pp. 1–6.

Hinton, G. E., Osindero, S. and Teh, Y.-W. (2006) ‘A fast learning algorithm for deep belief nets’, *Neural computation*, vol. 18, no. 7, pp. 1527–1554.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. R. (2012) *Improving neural networks by preventing co-adaptation of feature detectors* [Online]. Available at https://​arxiv.org​/​pdf/​1207.0580.

Ho, T. K. (1998) ‘The random subspace method for constructing decision forests’, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844.

Hochreiter, S. and Schmidhuber, J. (1997) ‘Long short-term memory’, *Neural computation*, vol. 9, no. 8, pp. 1735–1780.

Huang, Y. and Chen, Y. (2020) *Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies* [Online]. Available at http://​arxiv.org​/​pdf/​2006.06091v3.

Isola, P., Zhu, J.-Y., Zhou, T. and Efros, A. A. (2017) ‘Image-to-Image Translation with Conditional Adversarial Networks’, *30th IEEE Conference on Computer Vision and Pattern Recognition: CVPR 2017 : 21-26 July 2016, Honolulu, Hawaii : proceedings.* Piscataway, NJ, IEEE.

Javaid, A., Niyaz, Q., Sun, W. and Alam, M. (2016) ‘A Deep Learning Approach for Network Intrusion Detection System’, *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS),* ACM.

Jiang, L., Ge, Z. and Song, Z. (2017) ‘Semi-supervised fault classification based on dynamic Sparse Stacked auto-encoders model’, *Chemometrics and Intelligent Laboratory Systems*, vol. 168, pp. 72–83 [Online]. DOI: 10.1016/j.chemolab.2017.06.010.

Jordan, J. (2018a) *Introduction to autoencoders* [Online]. Available at https://​www.jeremyjordan.me​/​autoencoders/​.

Jordan, J. (2018b) *Variational autoencoders* [Online]. Available at https://​www.jeremyjordan.me​/​variational-​autoencoders/​.

Knight, J. C. (2002) ‘Safety Critical Systems: Challenges and Directions’, *Proceedings of the 24th International Conference on Software Engineering.* New York, NY, USA, Association for Computing Machinery, pp. 547–550.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2017) ‘ImageNet classification with deep convolutional neural networks’, *Communications of the ACM*, vol. 60, no. 6, pp. 84–90.

Krogh, A. and Hertz, J. (1992) ‘A Simple Weight Decay Can Improve Generalization’, *Advances in Neural Information Processing Systems,* Morgan-Kaufmann.

Kuhn, M. and Johnson, K. (2016) *Applied predictive modeling*, 5th edn, New York, Heidelberg, Dordrecht, London, Springer.

Kuhn, M. and Johnson, K. (2020) *Feature engineering and selection: A practical approach for predictive models*, Boca Raton, Florida, London, New York, CRC Press.

Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) ‘Gradient-based learning applied to document recognition’, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324.

Liu, G., Bao, H. and Han, B. (2018) ‘A Stacked Autoencoder-Based Deep Neural Network for Achieving Gearbox Fault Diagnosis’, *Mathematical Problems in Engineering*, vol. 2018, pp. 1–10.

Liu, X., Duh, K., Liu, L. and Gao, J. (2020) *Very Deep Transformers for Neural Machine Translation* [Online]. Available at http://​arxiv.org​/​pdf/​2008.07772v2.

Mahony, N. O., Campbell, S., Carvalho, A., Harapanahalli, S., Velasco-Hernandez, G., Krpalkova, L., Riordan, D. and Walsh, J. (2020) ‘Deep Learning vs. Traditional Computer Vision’, *in Advances in Computer Vision Proceedings of the*, vol. 943 [Online]. DOI: 10.1007/978-3-030-17795-9.

Maimon, O. Z. and Rokach, L. (2014) *Data Mining With Decision Trees: Theory And Applications (2nd Edition)*, World Scientific.

Martin-Villalba, C., Urquia, A. and Shao, G. (2018) ‘Implementations of the Tennessee Eastman Process in Modelica’, *IFAC-PapersOnLine*, vol. 51, no. 2, pp. 619–624 [Online]. DOI: 10.1016/j.ifacol.2018.03.105.

Mirza, M. and Osindero, S. (2014) *Conditional Generative Adversarial Nets*.

Montúfar, G., Pascanu, R., Cho, K. and Bengio, Y. (2014) *On the Number of Linear Regions of Deep Neural Networks* [Online]. Available at https://​arxiv.org​/​pdf/​1402.1869.

Nilsson, N. J. (1996) *Introduction to Machine learning: an Early Draft of a Proposed Textbook*, Stanford (EEUU).

Odena, A., Olah, C. and Shlens, J. (2016) *Conditional Image Synthesis With Auxiliary Classifier GANs* [Online]. Available at http://​arxiv.org​/​pdf/​1610.09585v4.

Onel, M., Kieslich, C. A. and Pistikopoulos, E. N. (2019) ‘A Nonlinear Support Vector Machine-Based Feature Selection Approach for Fault Detection and Diagnosis: Application to the Tennessee Eastman Process’, *AIChE journal. American Institute of Chemical Engineers*, vol. 65, no. 3, pp. 992–1005.

Pang, G., Shen, C., Cao, L. and van Hengel, A. den (2021) ‘Deep Learning for Anomaly Detection: A review’, *ACM Computing Surveys*, vol. 54, no. 2, pp. 1–38.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011) ‘Scikit-learn: Machine Learning in Python’, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830.

Qi, Y., Shen, C., Wang, D., Shi, J., Jiang, X. and Zhu, Z. (2017) ‘Stacked Sparse Autoencoder-Based Deep Network for Fault Diagnosis of Rotating Machinery’, *IEEE Access*, vol. 5, pp. 15066–15079.

Radford, A., Metz, L. and Chintala, S. (2016) *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*.

Rieth, C. A., Amsel, B. D., Tran, R. and Cook, M. B. (2017) *Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*.

Robinson, A. J. and Fallside, F. (1987) *The utility driven dynamic error propagation network*, University of Cambridge Department of Engineering Cambridge, MA.

Ruff, L., Vandermeulen, R., Goernitz, N., Deecke, L., Siddiqui, S. A., Binder, A., Müller, E. and Kloft, M. (2018) ‘Deep One-Class Classification’, *Proceedings of the 35th International Conference on Machine Learning,* PMLR, pp. 4393–4402.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X. (2016) *Improved Techniques for Training GANs*.

Schlegl, T., Seeböck, P., Waldstein, S. M., Langs, G. and Schmidt-Erfurth, U. (2019) ‘f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks’, *Medical Image Analysis*, vol. 54, pp. 30–44 [Online]. DOI: 10.1016/j.media.2019.01.010.

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U. and Langs, G. (2017) *Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery* [Online]. Available at https://​arxiv.org​/​pdf/​1703.05921.

Sejnowski, T. J. (2020) ‘The unreasonable effectiveness of deep learning in artificial intelligence’, *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30033–30038 [Online]. DOI: 10.1073/pnas.1907373117.

Sen, S., Datta, L. and Mitra, S. (2019) *Machine learning and IoT: A biological perspective*, Boca Raton, CRC Press, Taylor and Francis Group.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J. and Nădejde, M. (2017) *Nematus: a Toolkit for Neural Machine Translation*.

Shlens, J. (2014) *A Tutorial on Principal Component Analysis*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016) ‘Mastering the game of Go with deep neural networks and tree search’, *Nature*, vol. 529, no. 7587, pp. 484–489.

Song, X., Wu, M., Jermaine, C. and Ranka, S. (2007) ‘Conditional Anomaly Detection’, *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 5, pp. 631–645.

Spackman, K. A. (1989) ‘Signal Detection Theory: Valuable Tools for Evaluating Inductive Learning’, *Proceedings of the Sixth International Workshop on Machine Learning.* San Francisco, CA, USA, Morgan Kaufmann Publishers Inc, pp. 160–163.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’, *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958.

Ulusoy, I. and Bishop, C. M. (2006) ‘Comparison of Generative and Discriminative Techniques for Object Detection and Classification’, in *Toward Category-Level Object Recognition,* Springer, Berlin, Heidelberg, pp. 173–195.

van der Maaten, L. and Hinton, G. (2008) ‘Visualizing data using t-SNE’, *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605.

van der Maaten, L., Postma, E. and Herik, H. (2007) ‘Dimensionality Reduction: A Comparative Review’, *Journal of Machine Learning Research - JMLR*, vol. 10.

Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. (2008) ‘Extracting and Composing Robust Features with Denoising Autoencoders’, *Proceedings of the 25th International Conference on Machine Learning.* New York, NY, USA, Association for Computing Machinery, pp. 1096–1103.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A. (2010) ‘Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion’, *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408.

Wang, J., Feng, K. and Wu, J. (2019) ‘SVM-Based Deep Stacking Networks’, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 5273–5280 [Online]. DOI: 10.1609/aaai.v33i01.33015273.

Xavier, G. M. and Seixas, J. M. de (2018) ‘Fault Detection and Diagnosis in a Chemical Process using Long Short-Term Memory Recurrent Neural Network’, *IJCNN: 2018 International Joint Conference on Neural Networks : 2018 proceedings.* Piscataway, New Jersey, Institute of Electrical and Electronics Engineers.

Xie, X., Wang, C., Chen, S., Shi, G. and Zhao, Z. (2017) ‘Real-Time Illegal Parking Detection System Based on Deep Learning’, *Proceedings of the 2017 International Conference on Deep Learning Technologies.* New York, New York, USA. New York, NY, ACM.

Yang, B.-S., Di, X. and Han, T. (2008) ‘Random forests classifier for machine fault diagnosis’, *Journal of Mechanical Science and Technology*, vol. 22, no. 9, pp. 1716–1725.

Yin, J. and Yan, X. (2019) ‘Mutual Information–Dynamic Stacked Sparse Autoencoders for Fault Detection’, *Industrial & Engineering Chemistry Research*, vol. 58, no. 47, pp. 21614–21624.

Yin, S., Gao, X., Karimi, H. R. and Zhu, X. (2014) ‘Study on Support Vector Machine-Based Fault Detection in Tennessee Eastman Process’, *Abstract and Applied Analysis*, vol. 2014, pp. 1–8.

Yoon, J., Jarrett, D. and van der Schaar, M. (2019) ‘Time-series Generative Adversarial Networks’, *Advances in Neural Information Processing Systems,* Curran Associates, Inc.

Yoon, J., Jordon, J. and van der Schaar, M. (2018) *GAIN: Missing Data Imputation using Generative Adversarial Nets* [Online]. Available at https://​arxiv.org​/​pdf/​1806.02920.

Zenati, H., Foo, C. S., Lecouat, B., Manek, G. and Chandrasekhar, V. R. (2019) *Efficient GAN-Based Anomaly Detection*.

Zhang, G. P. (2000) ‘Neural networks for classification: a survey’, *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462.

Zhang, P., ed. (2010) *Advanced Industrial Control Technology*, William Andrew Publishing.

Zhang, T., Wang, W., Ye, H., Huang, D., Zhang, H. and Mingliang Li (2016) ‘Fault detection for ironmaking process based on stacked denoising autoencoders’, *2016 American Control Conference (ACC): Boston Marriott Copley Place, July 6-8, 2016, Boston, MA, USA.* [Piscataway, NJ], IEEE.

Zhang, Y. (2009) ‘Enhanced statistical analysis of nonlinear processes using KPCA, KICA and SVM’, *Chemical Engineering Science*, vol. 64, pp. 801–811.