Contents

[1. Introduction 3](#_Toc72306753)

[1.1 Motivation 3](#_Toc72306754)

[1.2 Problem Statement 5](#_Toc72306755)

[1.3 Research Questions 5](#_Toc72306756)

[1.4 Organization 6](#_Toc72306757)

[2. Background Theory 7](#_Toc72306758)

[2.1 Machine Learning Concept 7](#_Toc72306759)

[2.2 Traditional Algorithms 8](#_Toc72306760)

[2.2.1 Principal Component Analysis 8](#_Toc72306761)

[2.2.2 Support Vector Machine 9](#_Toc72306762)

[2.2.3 Decision Tree 10](#_Toc72306763)

[2.2.4 Random Forests 11](#_Toc72306764)

[2.3 Deep Learning 12](#_Toc72306765)

[2.3.1 Artificial Neural Network 12](#_Toc72306766)

[2.3.2 Neural Network Training 13](#_Toc72306767)

[2.3.3 Recurrent neural network 16](#_Toc72306768)

[2.3.4 Generative Adversarial Networks 17](#_Toc72306769)

[2.3.5 Conditional GANs and Bidirectional GANs 19](#_Toc72306770)

[2.3.6 Generative vs. Discriminative Modeling 21](#_Toc72306771)

[2.4 Evaluation Metrics 24](#_Toc72306772)

[2.4.1 Confusion Matrix 24](#_Toc72306773)

[2.4.2 Precision 24](#_Toc72306774)

[2.4.3 Recall 25](#_Toc72306775)

[2.4.4 Accuracy 25](#_Toc72306776)

[2.4.5 F1 Score 25](#_Toc72306777)

[2.4.6 Receiver Operating Characteristics and AUC 25](#_Toc72306778)

[2.4.7 Reliability Diagrams and Brier Score 28](#_Toc72306779)

[2.4.8 Log-loss Score 28](#_Toc72306780)

[3. Methodology 30](#_Toc72306781)

[3.1 Nature of Input Data 30](#_Toc72306782)

[3.2 What Are Anomalies? 30](#_Toc72306783)

[3.3 The output of Anomaly Detection Models 33](#_Toc72306784)

[3.4 Challenges in Anomaly Detection 34](#_Toc72306785)

[3.5 Anomaly Detection Models 35](#_Toc72306786)

[3.5.1 Autoencoder 35](#_Toc72306787)

[3.5.2 Stacked Autoencoder. 37](#_Toc72306788)

[3.5.3 Variational autoencoder. 39](#_Toc72306789)

[3.5.4 GAN in Anomaly detection 40](#_Toc72306790)

[3.5.5 Long Short-Term Memory Networks (LSTMs) 42](#_Toc72306791)

[3.6 Summary 43](#_Toc72306792)

[3.6.1 Feature extraction anomaly detection 43](#_Toc72306793)

[3.6.2 Feature representation anomaly detection 44](#_Toc72306794)

[4. Experiment and Analysis 47](#_Toc72306795)

[4.1 Tennessee Eastman Process Simulation Data 47](#_Toc72306796)

[4.1.1 Data Preparation 49](#_Toc72306797)

[4.1.2 TEP Data Preparation 51](#_Toc72306798)

[4.1.3 TEP Data Exploration and Visualization 51](#_Toc72306799)

[4.2 Experiment Environment 54](#_Toc72306800)

[4.3 Baseline Models and Evaluation Metrics 55](#_Toc72306801)

[4.4 Experiment Result and Analysis 58](#_Toc72306802)

[4.4.1 Autoencoders and Regularization 58](#_Toc72306803)

[4.4.2 Deep autoencoders 60](#_Toc72306804)

[4.4.3 Traditional ML Classifiers 61](#_Toc72306805)

[4.4.4 Neural Network Classifiers 62](#_Toc72306806)

[~~4.4.5~~ ~~Autoencoders with Classifiers~~ 64](#_Toc72306807)

[4.4.6 Generative Adversarial Networks 65](#_Toc72306808)

[5. Conclusion and Future Work 66](#_Toc72306809)

[5.1 Conclusion 66](#_Toc72306810)

[5.2 Future work 67](#_Toc72306811)

# 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 image classification (*Ref. ImageNet Classification with Deep Convolutional Neural Networks*), natural language processing (*Ref. Very Deep Transformers for Neural Machine Translation*) and autonomous driving (*Ref. Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies*). 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 (*Ref. Deep Learning for Anomaly Detection A Survey, 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 terms of anomaly detection, deep learning has been deployed for tasks such as surveillance video analysis (*Ref. Real-time illegal parking detection system based on deep learning*), cyber intrusion detection (*Ref. A deep learning approach for network intrusion detection system*) and medical diagnosis (*Ref. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery*). This thesis focuses on evaluating different machine learning algorithms, especially deep learning, in anomaly detection of industrial processes.

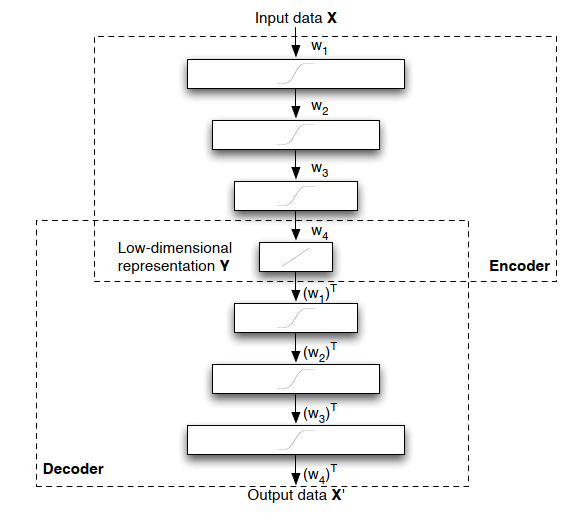
Industrial systems are becoming more complex to fulfill the requirement of productiveness, reliability, efficiency, and safety. To ensure the normal operation, more and more variables are introduced to monitor and control the system. which pose challenges for the system operator. Early detection of anomalies and emergency response plans are critical for the system’s safe and effective operation. The anomaly detection methods are generally classified into two types: data-driven and model-based (*Ref. A survey of the application of basic data-driven and model-based methods in process monitoring and fault diagnosis*). The latter usually requires expert-level knowledge and experience to build the sophisticated model, challenging to realize and economically unfeasible. Compared to the model-based method, the data-driven method such as machine learning algorithms relies on analyzing large amounts of data, which provides a practical solution for various industrial productions. 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 author name (*Ref. A PLANT-WIDE INDUSTRIAL PROCESS PROBLEM*). 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 (*Ref. Implementations of the Tennessee Eastman Process in Modelica*). 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* (*Ref. Bellman, Richard Ernest; Rand Corporation (1957). Dynamic programming*). In (*Ref. A Few Useful Things to Know about Machine Learning*), the author name gave two descriptive explanations 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 data points increase as the dimensionality increases.

To mitigate the impact of high dimensionality, methods such as feature selection (*Ref. Feature Engineering and Selection: A Practical Approach for Predictive Models, chapter 1.4, S17*) and feature extraction (*Ref. Foundations of Multidimensional and Metric Data Structures, chapter 4.6, S664*) 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 (*Ref. Feature Selection for Fault Detection Systems: Application to the Tennessee Eastman Process*). 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 categorized into linear and nonlinear approaches. A representative linear feature extraction approach is principal component analysis (*Ref. A Tutorial on Principal Component Analysis*) which extracts the low-dimensional features by representing the high-dimensional data into a linear subspace of lower dimensions. However, the principal component analysis may not adequately deal with nonlinear data like TEP data. To handle the nonlinear high-dimensional data, (*Ref. Outlier Detection Using Replicator Neural Networks, 2002*) the author name proposed autoencoder (a.k.a. replicator neural networks) is proposed. As *figure* shows, an autoencoder consists of an encoder and a decoder part. The encoder learns to represent the input data into a low-dimension space at the middle hidden layers (*bottleneck),* then the decoder maps the data back to high-dimensional space. For autoencoders, it is restricted to reconstruct the input only approximately. The encoder part of the autoencoder can be used to extract the data feature. One research point of this thesis is to explore the application of autoencoder for anomaly detection in TEP. Meanwhile, this thesis also seeks to provide a comprehensive comparison and analysis of anomaly detection methods in the TEP.



(*figure will be revised later*) (*Ref. Dimensionality Reduction: A Comparative Review*)

## Problem Statement

The main goal of this research is to evaluate the feasibility of deep learning, especially the autoencoder, in TEP's anomaly detection. Analyzing the performance of different machine learning algorithms can help enhance the understanding of anomaly detection and point the direction for further work. As the literature research implies, there is no detailed work comparing different anomaly detection methods for TEP. This thesis seeks to fill the gaps.

## Research Questions

This thesis's main contribution is to provide a comprehensive comparison between the traditional machine learning method and the deep learning method for TEP 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, and what machine learning algorithms are already applied?
* How to evaluate the performance of different algorithms regarding the TEP data?
* What are the reasons causing the performance difference?
* How to implements deep learning anomaly detection for anomaly detection?

## Organization

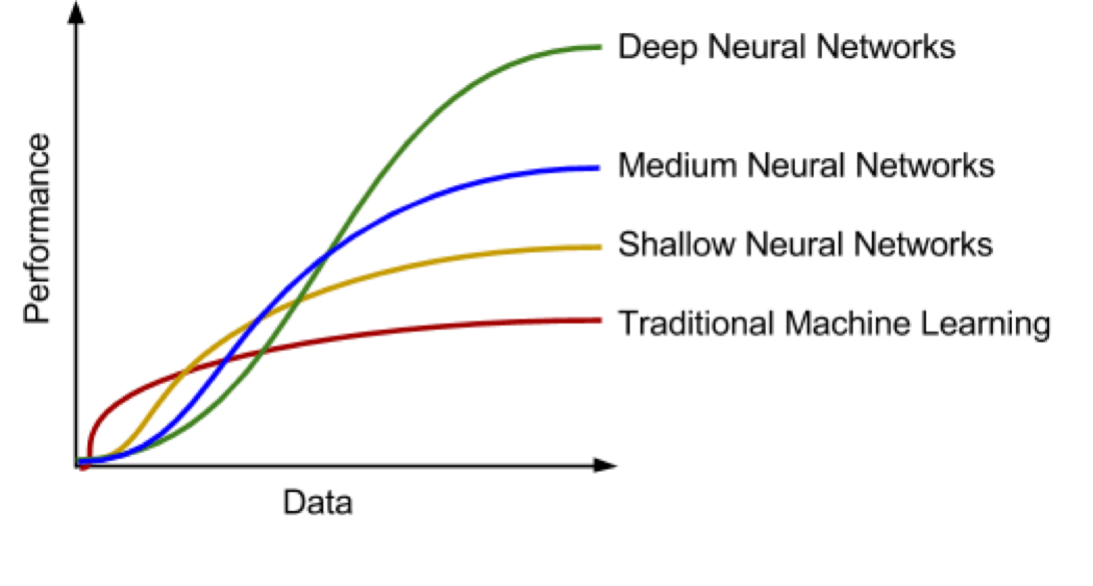
The document is structured as follows. First, Chapter 1 (*cross ref*) introduces the topic and provides an overview of the research problem. Then, Chapter 2 (*cross ref*) explains the background knowledge about machine learning algorithms and evaluation metrics. Next, Chapter 3 (*cross ref*) illustrates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

# Background Theory

This chapter illustrates machine learning concepts and provides necessary background knowledge about deep learning. First, section 2.2 (*cross ref*) introduces a series of traditional algorithms. Then, Section 2.3 (*cross ref*) provides the theoretical knowledge of deep learning, which is the basis for understanding the methodology in Chapter 3 (*cross ref*). Finally, Section 2.6 (*cross ref*) introduces the evaluation metrics to assess the performance of anomaly detection methods.

## Machine Learning Concept

A machine learns if it can achieve self-adjustment and is expected to improve the performance given more external information (*Ref. INTRODUCTION TO MACHINE LEARNING - Nils J. Nilsson, chapter 1.1, S1*). 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 massive success in pattern recognition and translation (*Ref. The Unreasonable Effectiveness of Deep Learning in Artificial Intelligence*). 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 (*fig below*); 2) deep learning can reconstruct the unknown structure and find appropriate representation based on the distribution of inputs (*Ref. Deep Learning of Representations for Unsupervised and Transfer Learning*).



(*figure will be revised later*) (*Ref. https://albahnsen.com/2017/06/06/building-ai-applications-using-deep-learning/*)

Machine learning algorithms can be grouped into different categories. For example, grouped by learning style (*Ref. A Survey on Machine Learning: Concept, Algorithms and Applications*), 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 only partially labeled. The task is to predict the label of the data and execute the clustering.
* Reinforcement learning: The algorithms are trained to choose behavior in a specific environment to obtain the most reward.

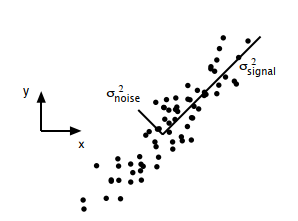
## Traditional Algorithms

The traditional machine learning algorithms in this thesis refers to the algorithms in which no artificial neural network is involved.

### Principal Component Analysis

As a dimension reduction method, the principal component analysis (PCA) 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 as the original data points.

For example, we want to transform a set of two-dimensional data points into one-dimensional data points as the figure shows. However, the two-dimensional data points contain noise, and we want to mitigate the effect of noise and extract the meaningful signal values in one dimension. To transform the two-dimensional data points, we define a new orthogonal coordinate with a 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 the 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 (≫ 1) indicates that the signal has high measurement accuracy.

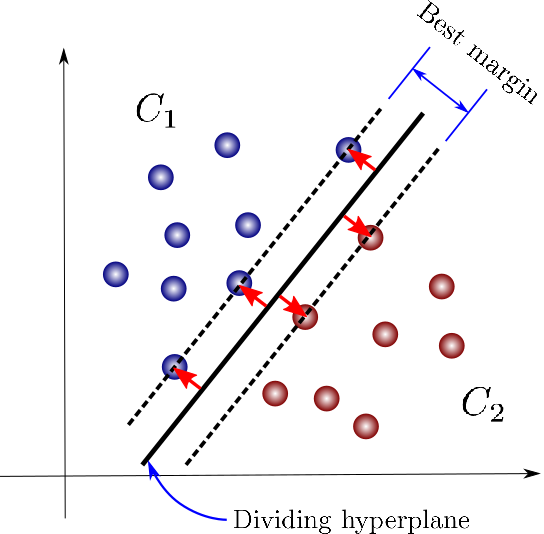


(*figure will be revised later*) (*Ref. A Tutorial on Principal Component Analysis*)

Although PCA is a non-parametric dimension reduction approach and easy to implement, it has several limitations: 1) The low-dimensional representation of data points can cause information loss and 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 (*Ref. A Tutorial on Principal Component Analysis*).

### 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 the (*figure*) shows, certain data points are chosen as the support vector, and the area between the support vectors is called margin. SVM solves the classification problem by finding a hyperplane (decision boundary) that can separate the data best and realize the maximal margin.

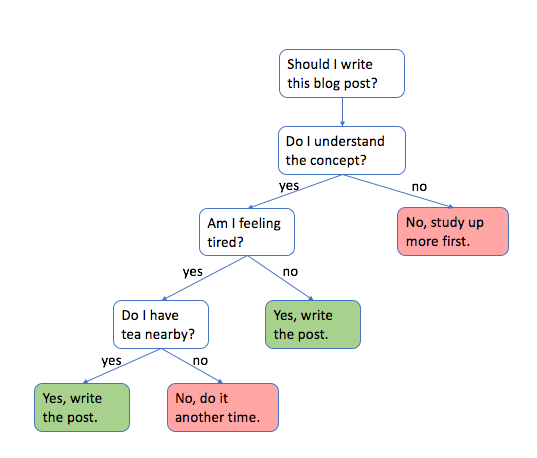


(*figure will be revised later*) (*Ref. https://towardsdatascience.com/support-vector-machines-for-classification-fc7c1565e3*)

One fact about the classification task is that mapping the data into another space may significantly improve the performance. SVM takes advantage of this fact and solves the nonlinear separable issues using the kernel, which transforms the data point into high- or infinite-dimensional space. As a classification method, SVM works effectively in high-dimensional space. However, it does not provide a probability estimate. Instead, it gives the classification result directly. Besides, the selection of kernels has a crucial impact on classification performance. Finally, in the experiments, the computation cost for SVM is much higher than other classifiers such as the random forest tree (*Ref. An introduction to support vector machines*).

### 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 (*fig*) 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 will be revised later*)

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 can lead to a different decision tree construction (*Ref. Data Mining with Decision Trees: Theory and Applications, chapter 1*). To further improve the decision tree's performance, random forests are proposed (*Ref. Random Forests Leo 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, (*Ref. The Random Subspace Method for Constructing Decision Forests*), (*Ref. Random Forests Leo Breiman 2001*) the author name 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* (*Ref. An Introduction To Statistical Learning with Applications in R, chapter 8*). 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 problem between overfitting and high accuracy by selecting a random subset of features when splitting each decision tree's sub-space. As (*fig*) shows, a random forest tree makes the classification based on the majority vote of Tree-1, Tree-2 and Tree-3. Since each tree is trained uncorrelated, the majority vote can help to provide better generalization and accuracy.



(*figure will be revised later*) (*Ref. https://community.tibco.com/wiki/random-forest-template-tibco-spotfire*)

## Deep Learning

This section introduces the background knowledge of deep learning.

### Artificial Neural Network

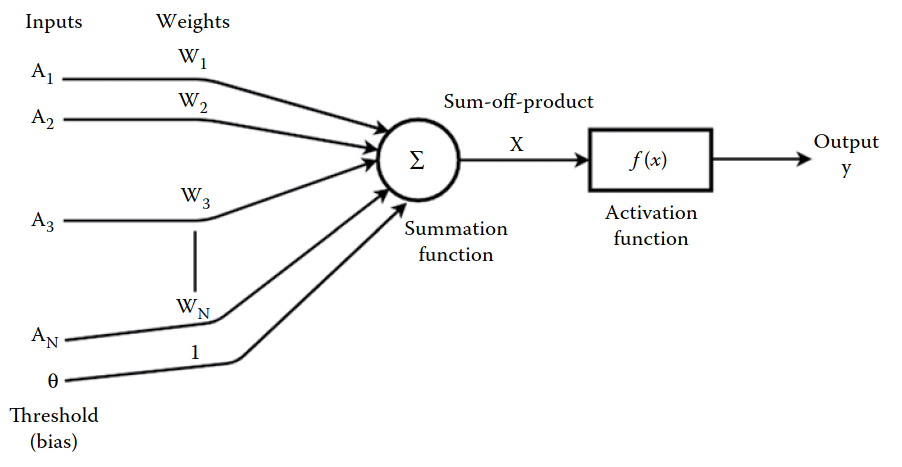
An artificial neural network is inspired by studies on biological neurons and built with connected neuron units. A neuron is as the figure shows, the output is calculated by the formula:

,

for , is the input signal, is the weights section which adjusts the signal strength and is the offset. A commonly used activation function is function ranging from -1 to 1 and is defined by:

.

Stacking and connecting multiple neuron units in different directions regarding the signal flows can build various artificial neural networks. Universal approximation theorems (*Ref. Approximation with Artificial Neural Networks*) 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 (*Ref. Limits of End-to-End Learning*). For example, in computer vision, manual feature selection or creation becomes intractable as the training samples increase. End-to-End learning can bypass such procedures and be trained with the image dataset and the image annotations (*Ref. Deep Learning vs. Traditional Computer Vision*). 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 different. Such hierarchy working principle helps deep learning learn more high-level, abstract representations and achieve better performance (*Ref. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville, chapter 1 S5*).



(*figure will be revised later*) (*Ref. Machine Learning and Iot A Biological Perspective by Shampa Sen, Leonid Datta, Sayak Mitra, chapter 1.2.2*)

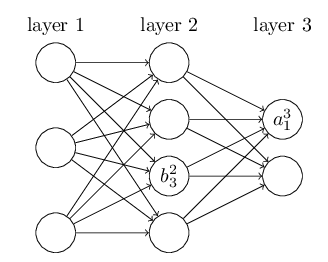
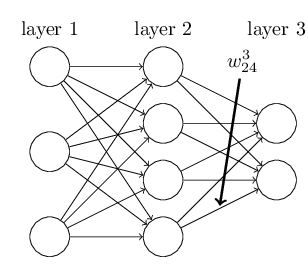
### Neural Network Training

#### Backpropagation

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

,

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 will be revised later, merge together*) (*Ref. http://neuralnetworksanddeeplearning.com/chap2.html*)

We then define the cost function of the neural network:

,

where is the input data, is the number of training data, is the expected output and is the actual output in the output layer . Mean square 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:

.

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 ,

,

,

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

,

).

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

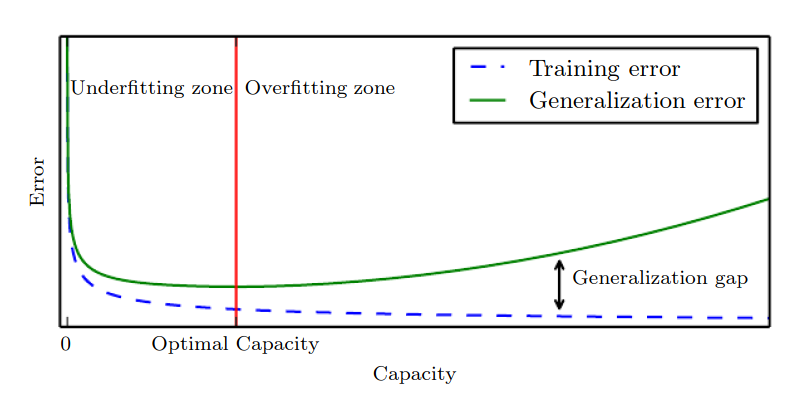
,

,

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 (*Ref. online book, Neural Networks and Deep Learning, Michael Nielsen, chapter 2, s 40*). 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 (*Ref. Neural Networks and Deep Learning - Michael Nielsen*). (*Ref. Gradient-based learning applied to document recognition*), (*Ref. Greedy Layer-Wise Training of Deep Networks*) and (*Ref. A Fast Learning Algorithm for Deep Belief Nets*) proposed several new approaches such as greedy layer-wise training and global training to solve that problem. Applying activation such as ReLU (*Ref. Deep Sparse Rectifier Neural Networks*) or residual networks (*Ref. Deep Residual Learning for Image Recognition*) can also ease the training problem. The event that promotes deep learning development is the work from (*Ref. ImageNet Classiﬁcation with Deep Convolutional Neural Networks*), which achieves a new score for the image classification task. Besides image classification, deep learning also makes significant progress in speech recognition (*Ref. Deep Speech 2- End-to-End Speech Recognition in English and Mandarin*) and natural language processing (*Ref. Nematus: a Toolkit for Neural Machine Translation*).

#### 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 fed with unobserved data (*Ref. Deep learning, chapter 5.2, S110*). Underfitting refers to when the model fails to learn 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 on the test dataset. The figure shows that 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. In addition, the training plays 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 show that too many training epochs can lead to overfitting when the model has enough capacity.



(*figure will be revised later*) (*Ref. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville (z-lib.org).pdf/, chapter 5, s115*)

??? add relation with training samples

#### Regularization Techniques

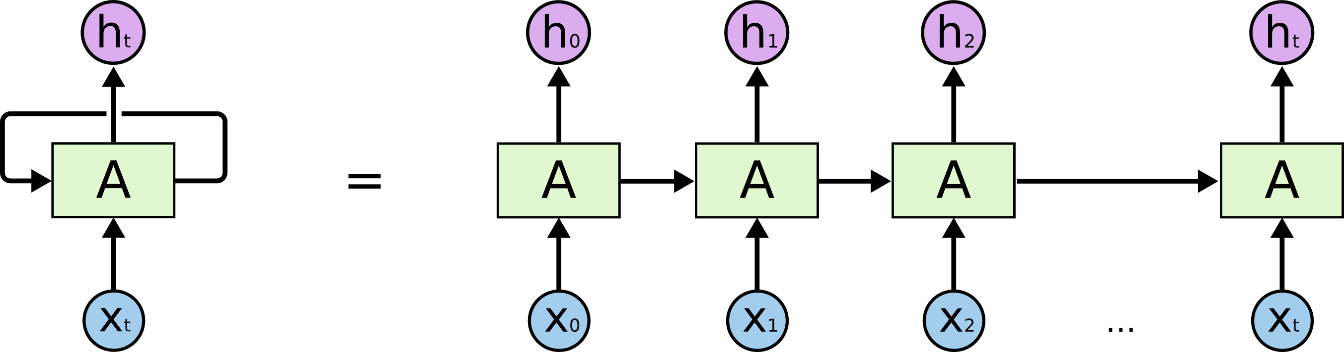
???

### 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 (*Ref. Ozonation and Biodegradation in Environmental Engineering, 2019, chapter 3.1.3*). As (*fig*) shows, RNN uses loops to preserve previous computation's state information and process the input of next time sequence. The formula for the current state in RNN is defined as:

,

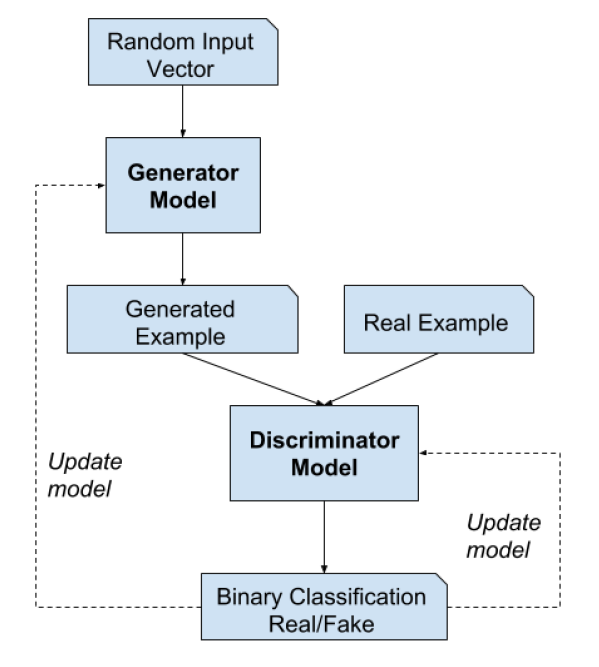
where is the activation function, is the input at time step and is the hidden vector of the RNN at time step .One observation of the RNN structure is that it should be able to build a connection between previous information and the current task. However, RNN can exhibit forgetting behavior, limiting its representation capability when dealing with long-range sequences (*Ref. Local Feedback Multi-Layered Networks*). Besides, (*Ref. Learning Long-Term Dependencies with Gradient Descent is Difficult*), (*Ref. The Utility Driven Dynamic Error Propagation Network*) have shown that RNN suffers from training problems when long-term memory is involved in the task. To mitigate the drawbacks of RNN, (*Ref. LONG SHORT-TERM MEMORY*) introduced Long Short-Term Memory



(*figure will be revised later*) (*Ref.* [*http://colah.github.io/posts/2015-08-Understanding-LSTMs/*](http://colah.github.io/posts/2015-08-Understanding-LSTMs/))

### Generative Adversarial Networks

Generative adversarial networks (GANs) proposed in (*Ref. Generative Adversarial Nets*) are neural network architectures composed of two neural networks: a generator and a discriminator. The generator is used to generate 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 (*Ref. Deep Learning*). As the *figure* 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 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 (*Ref. Generative Adversarial Networks with Python*).



(*figure will be revised later*) (*Ref. Generative Adversarial Networks with Python by Jason Brownlee (z-lib.org).pdf chapter 1.3.4, S10*)

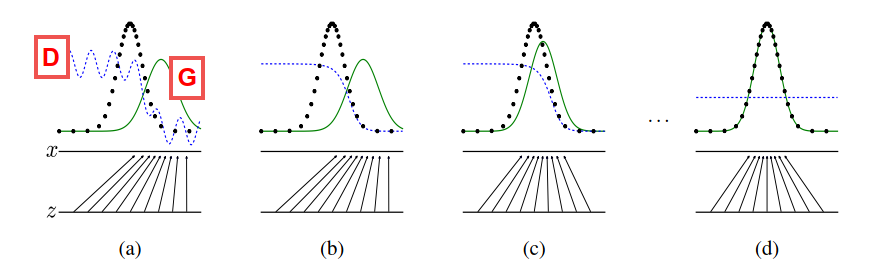
Assume both the generator and discriminator are multilayer neural networks. For generator , the random input space is defined as and the mapping to data space is defined as , where are parameters of generator . For discriminator , outputs a value representing the probability that data came from the real dataset, where are ’s parameters. During the training of GANs, discriminator is trained to maximize the ability to correctly assign labels for the data from the real dataset 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 :

,

where is the expectation.

#### 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 (*Ref*.), 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 can’t distinguish if the data distribution is from the generator or real dataset as (d) shows.



(*figure will be revised later*) (*Ref. Generative Adversarial Nets*) *Figure : discriminator (blue, dashed line), generator (green, solid line), real data distribution (black, dotted line),*

However, some practical issues remain unsolved in the GAN training process. At the early phase of training, if the learning progress of discriminator (learns to distinguish real/fake examples) is faster than the progress of 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 can’t provide sufficient gradient information for . Therefore, (*Ref. Generative Adversarial Nets*) suggest training generator to maximize rather than minimize to obtain stronger gradient information.

As a deep learning generative model, GAN has attracted attention from both practitioners and scholars. GANs avoid the intractable problem of probabilistic computation, such as standard marginalization and conditioning operations (*Ref. A Fast Learning Algorithm for Deep Belief Nets*). Besides, GANs can be applied to various scenarios such as missing data imputation (*Ref. GAIN: Missing Data Imputation using Generative Adversarial Nets*), time-series prediction (*Ref. Time-series Generative Adversarial Networks*), and anomaly detection (*Ref. TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks*).

### Conditional GANs and Bidirectional GANs

Conditional GAN and bidirectional GAN 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 from latent space and some additional information as input to the generator and discriminator . The additional information can be the class label, the style of images, and so on. Bidirectional GAN extend the GANs by adding the *inverse mapping,* which maps from data space back to latent space with an encoder (*Ref. A Survey on GANs for Anomaly Detection*).

#### Conditional GANs

The GAN framework can help train the generative models to generate images (*Ref. Unsupervised representation learning with deep convolutional generative adversarial networks*). However, there are no parameters to control the style or type of the generated images. Conditional GANs (cGANs), (*Ref. Conditional Generative Adversarial Nets*) proposed a way to conditional generating images by feeding the generator with the class label. Comparing to the original GANs (*Ref. Generative Adversarial Nets*), the objective function is:

,

where is the auxiliary information such as class labels. During the training, the class labels 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 (sampled from latent space) and the class labels . To generate a new data sample, feed the generator with input conditioned with the label information.

(*Ref. Conditional generative adversarial nets for convolutional face generation*), (*Ref. Conditional Image Synthesis with Auxiliary Classiﬁer GANs*) and (*Ref. Image-to-Image Translation with Conditional Adversarial Networks*) have demonstrated a great potential of cGANs in image synthesis. (*Ref. Conditional Image Synthesis with Auxiliary Classiﬁer GANs*) also argued that the generated image could achieve better discriminability than the models that can only generate lower resolution images.

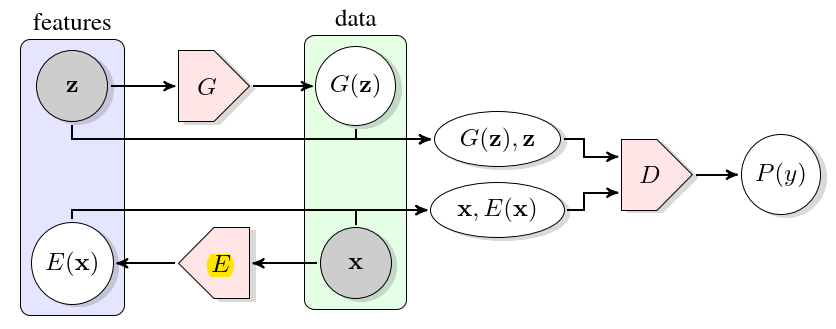
#### Bidirectional GANs

As an extension of GANs, (*Ref. Adversarial Feature Learning*) 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 (*Ref. GANomaly Semi-Supervised Anomaly Detection via Adversarial Training, 2018*) (*Ref. Combining GANs and AutoEncoders for Efﬁcient Anomaly Detection*). The objective function of BiGAN for training is as follows:

,

.

Comparing to GANs (*Ref. Generative Adversarial Nets*), BiGAN use the pairs such as or as the input for the discriminator . As *figure* shows, sampled from latent feature space is provided to the generator to generate fake sample pair to train the discriminator to identify fake sample data. On the other hand, sample pair force discriminator to identify real sample data. 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 will be revised later*) (*Ref. Adversarial Feature Learning*)

### Generative vs. Discriminative Modeling

Generative and discriminative models are two important 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 (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 .

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:

,

.

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

,

.

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

,

where the first term is the generative objective function (*Ref. The Tradeoff Between Generative and Discriminative Classifiers*).

In general, generative models attempt to estimate the joint distribution of both input and output , generating synthetic data points similar to the observed data point. Discriminative models learn to compute the mapping between input and output directly, i.e., calculate the conditional probability of given (*Ref. Machine Learning Discriminative and Generative by Tony Jebara,* *Chapter 2*). In other words, generative models focus on understanding the data points’ composition and basic characteristics of classes. In contrast, the discriminative models are designed to find the decision boundary between samples from different classes (*Ref. Generative versus discriminative classiﬁers for android anomaly-based detection system using system calls ﬁltering and abstraction process*).

In the practical application of machine learning algorithms, generative models have the following advantages (*Ref. Generative versus discriminative methods for object recognition*):

* 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 classifier.

Compared to generative models, discriminative models make predictions by mapping between input and label directly. The success of applying discriminative models in image classification (*Ref. ImageNet Classification with Deep Convolutional Neural Networks*) seems to imply that discriminative models have better performance than generative models in the classification task. Besides, a famous quote from Vladmir Vapnik “When solving a problem of interest, do not solve a more general problem as an intermediate step.” also implies that discriminative models are a preferred method. (*Ref. Where’s Wally Now? Deep Generative and Discriminative Embeddings for Novelty Detection*) 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. (*Ref. Generative versus discriminative methods for object recognition*), (*Ref. Comparison of Generative and Discriminative Techniques for Object Detection and Classiﬁcation*) showed that for object detection tasks, generative models could achieve higher accuracy while discriminative models are faster at prediction once trained and the combination of both models is expected to achieve better performance. Further, (*Ref. Anomaly Detection Combining Discriminative and Generative Models*) 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 direction for better modeling. In this section, we introduce the evaluation metrics to quantify the performance of anomaly detection algorithms.

### Confusion Matrix

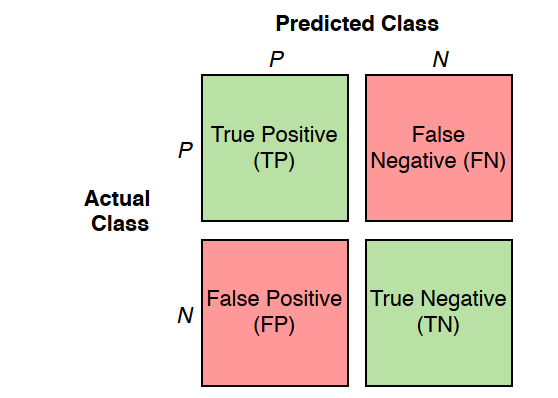
The confusion matrix is a commonly used tabular evaluation method in binary classification tasks that presents the classifier's predictions in a contingency table with four cells. To better understand it, we first define true positives, false positives, true negatives, and false negatives.

**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 will be revised later*)

One assumption made here is that the result of predicting and assessing is either positive or negative. As the *figure* 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 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:

### Accuracy

Accuracy measures the fraction of correctly predicted data points in all predictions.

Accuracy is defined:

### F1 Score

F1 score is a measure to balance both precision and recall.

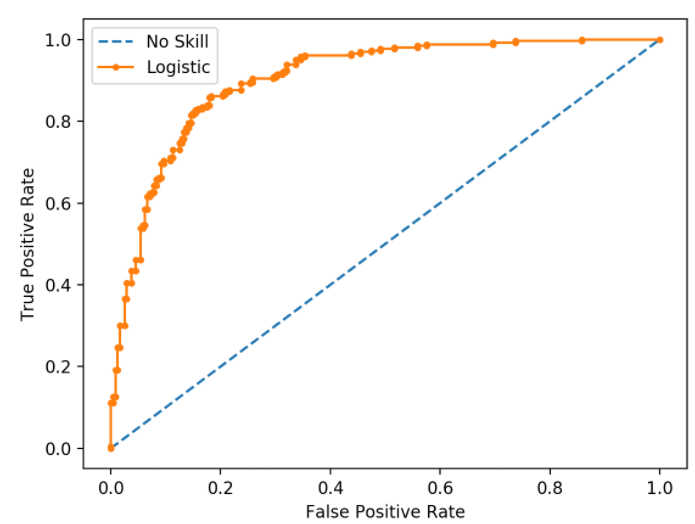
F1 score is defined:

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

### Receiver Operating Characteristics and AUC

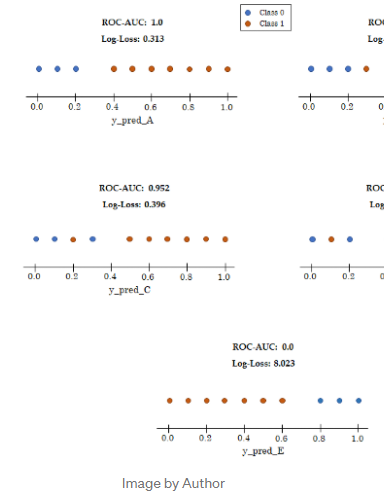
The receiver operating characteristics (ROC) curve is a graph tool for evaluating binary classifiers’ performance as a function of the cut-off threshold. The axis of the graph is (true positive rate) in, and the axis is (false positive rate). and are defined as:

For each threshold, there is a point on the graph. Connecting all the points forms the ROC curve. At the lower-left point of the graph, the classifier makes no predictions as positives (no alarms are detected). At the upper-right point , the classifier predicts the data instances as positive unconditionally (raise false alarms on all negative instances). The diagonal line (blue 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 (orange dotted line) should be higher than the diagonal line (blue dotted line). (*Ref. Signal detection theory Valuable tools for Evaluating Inductive Learning*)



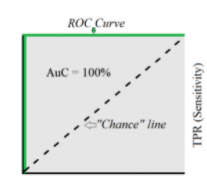
(*figure will be revised later*)

**ROC AUC.** Althougha ROC curve can provide a detailed description of the performance of classifiers under different thresholds, It is more convenient to use a single scalar value for comparison. (*Ref. The use of the area under the ROC curve in the evaluation of machine learning algorithms*) proposed 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 the figures show, there is a clear boundary (threshold) that can separate the predictions of the first classifier (with ROC AUC score=1). In contrast, for the predictions of the second classifiers (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 performance it has.



(*figure will be revised later*) (*Ref. https://towardsdatascience.com/intuition-behind-roc-auc-score-1456439d1f30*)

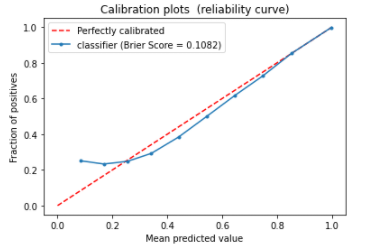
As the *figure* shows.



(*figure will be revised later*)

### Reliability Diagrams and Brier Score

A reliability diagram (also calibration curve) is a graph tool to evaluate the reliability of classifiers. Roughly speaking, a classifier is reliable (also calibrated) when the fraction of observed positive/negative instances is consistent with its’ predictions. As the figure 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 in axis, and the fraction of positives instances in the bin is the value of the point in axis. Connecting all points forms the reliability curve. As the figure shows, a perfectly calibrated classifier (red dotted line) has the same value in both axes. We expect to improve the classifier such that its’ reliability curve is close to the perfectly calibrated curve (*Ref. Increasing the Reliability of Reliability Diagrams*).



(*figure will be revised later*)

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

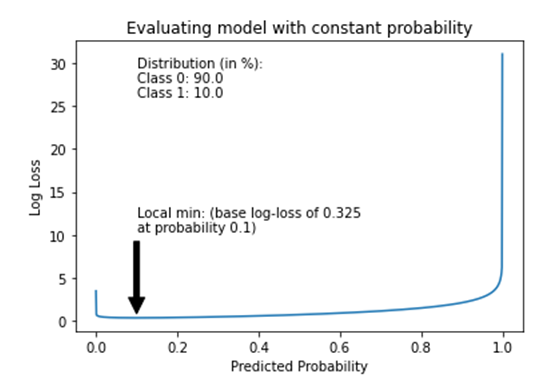
in which is the number of predictions, is the predictions and is the actual values (*Ref. VERIFICATION OF FORECASTS EXPRESSED IN TERMS OF PROBABILITY*).

### Log-loss Score

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

For a perfect classifier, its’ log-loss score is 0, which means it can correctly classify every instance. Therefore, the classifier with a lower log-loss score is better than those with a higher log-loss score.

**Baseline log-loss score**. A baseline log-loss score represents the no-skill classifier, which makes predictions based on the proportion of positive and negative instances. For example, there is a dataset with 90% positive and 10% negative instances. If the no-skill classifier predicts that the probability of a new instance 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 must have a lower log-loss score than the baseline log-loss score (*Ref. Pattern Recognition and Machine Learning. Springer, p. 209*).

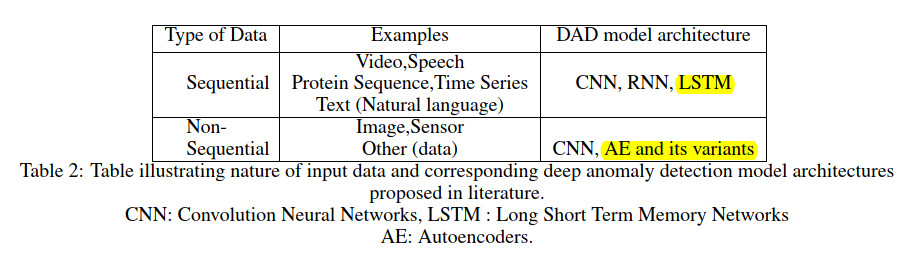


# Methodology

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 methods based on deep learning. It starts with a discussion about the nature of input data and then points out the main challenges in anomaly detection. The rest of this chapter presents an overview of the anomaly detection models and summarizes the models into two frameworks: supervised machine learning and unsupervised machine learning.

## Nature of Input Data

As discussed in chapter 2, the choice of deep learning model architecture depends on both the goal of 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-dimension or low-dimension 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. The *figure* below lists different data types and corresponding deep learning anomaly detection model architectures.

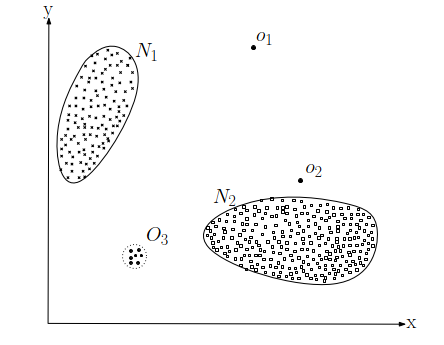


(*figure will be revised later*) (*Ref. nature of input data and corresponding deep anomaly detection model architectures*)

## What Are Anomalies?

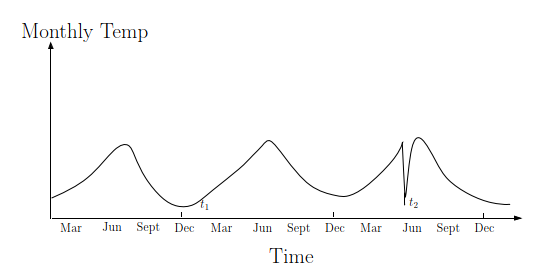
Anomaly detection is one of the most important applications in data science. It refers to the problem of identifying the patterns of data which does not satisfy expected behavior. Such patterns are also known as outliers, anomalies, and exceptions, etc (*Ref. Anomaly Detection A Survey*). Another definition of anomalies introduced in (*Ref. Identification of Outliers by D. M. Hawkins*) is that anomalies are the data that deviates so much from the majority of the data and plausibly produced by a different mechanism. Based on the nature and observation of anomaly data instances, (*Ref. Anomaly Detection A Survey*) provided a detailed classification of anomalies:

**Point Anomalies**: A data point is categorized as point anomalies when it deviates significantly from the rest of the data. As figures shows, data point , and area locate far from the normal data area and . Therefore, data point , and all data points in the 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 research.



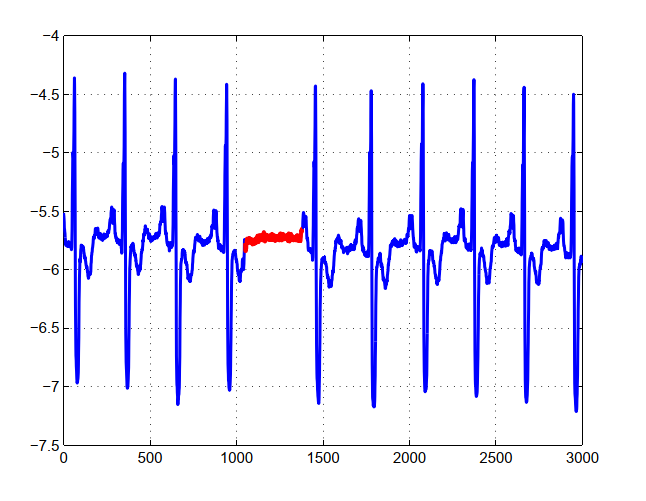
(*figure will be revised later*)

**Contextual Anomalies**: A data point is termed as contextual anomalies (also conditional anomaly (*Ref. Conditional Anomaly Detection, 2007*)) when merely identified as an anomaly in a user-specified area. 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 the figure shows, a 0 ambient temperature is recognized as abnormal at (summer, start in June), but considered as normal at the time (winter, start in December).



(*figure will be revised later*)

**Collective Anomalies**: A collection of series data is 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. The figure shows a human electrocardiogram output (Ref. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals). The red area is identified as an anomaly since the same value has continued for an abnormally long period. However, the single point in the red area are not classified as an anomaly.



(*figure will be revised later*)

## 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**. To describe the deviation degree from normal operation state of test data instances, anomaly scores are introduced. For example, in (*Ref. Outlier Detection Using Replicator Neural Networks*) (*Ref. Outlier Detection with Autoencoder Ensembles*), the anomaly score (outlyingness) is defined as the reconstruction error (mean squared error, introduced before) of autoencoder’s input. And in (*Ref. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery*), the anomaly score is defined as the combination of reconstruction error and discriminator output of the GAN. After calculating the anomaly scores of all test data instances, anomalies are identified using a cut-off threshold or listing top data instances with high anomaly scores.

**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. 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 defining a normal region and identifying the data instances that do not belong to this region as anomalies. However, this method is not feasible in the practical situation for several reasons (*Ref. Anomaly Detection A Survey*): 1) There is often no clear and exact boundary between normal and abnormal regions. 2) 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. 3) The exact description of anomalies could be significantly different considering different 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 (*Ref. Deep Learning for Anomaly Detection A Review*):

**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 (*Ref. Safety Critical Systems: Challenges and Directions*). 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 a relationship between the internal function parameters with physical parameters such as temperature, pressure, and volume (*Ref. Industrial control system simulation routines*). For some rare but safety-critical anomaly data instances, 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.

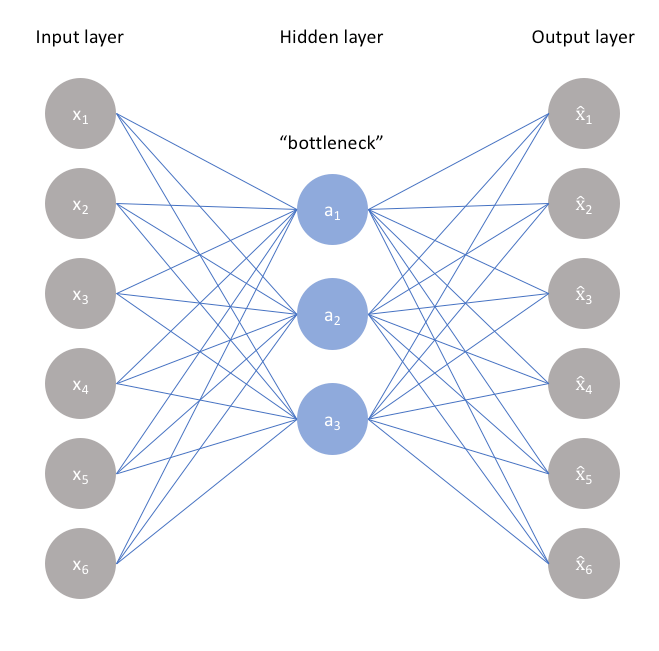
## Anomaly Detection Models

### Autoencoder

An autoencoder is a feedforward artificial neural network that learns to reconstruct the input data 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 (MSE), is defined as:

.

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 the figure shows, the hidden layer has fewer neural units than the input layers, and this bottleneck design is crucial for the autoencoder. The bottleneck constrains the amount of traversed information in the network. Furthermore, it forces the autoencoder not to copy the input precisely and learn a compression (meaningful representation) of the input data (*Ref. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville*).



(*figure will be revised later*)

However, for deep autoencoders with deep layers and more capacity, the bottleneck design is insufficient to discourage memorization. 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 (*Ref. Dropout: A Simple Way to Prevent Neural Networks from Overﬁtting, 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 chapter 2 discussed.

#### Sparse Autoencoder

Sparse autoencoder introduces sparsity restriction by uses weight or activity regularization as the regularizer term. For example, (*Ref. Mutual Information−Dynamic Stacked Sparse Autoencoders for Fault Detection.pdf*) and (*Ref. Semi-supervised fault classification based on dynamic Sparse Stacked auto-encoders model, 2017*) deploy a sparse autoencoder for anomaly detection using the sparsity restriction. The loss function is defined:

in which is the tuning parameter for penalty and is the number of hidden layers. is the average activation of hidden layer and defined as equation. 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. As can be seen from equation, 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, (*Ref. Stacked Sparse Autoencoder-Based Deep Network for Fault Diagnosis of Rotating Machinery.pdf*) replace the regularizer term with the absolute value of weights in all hidden layers.

#### Denoising Autoencoder.

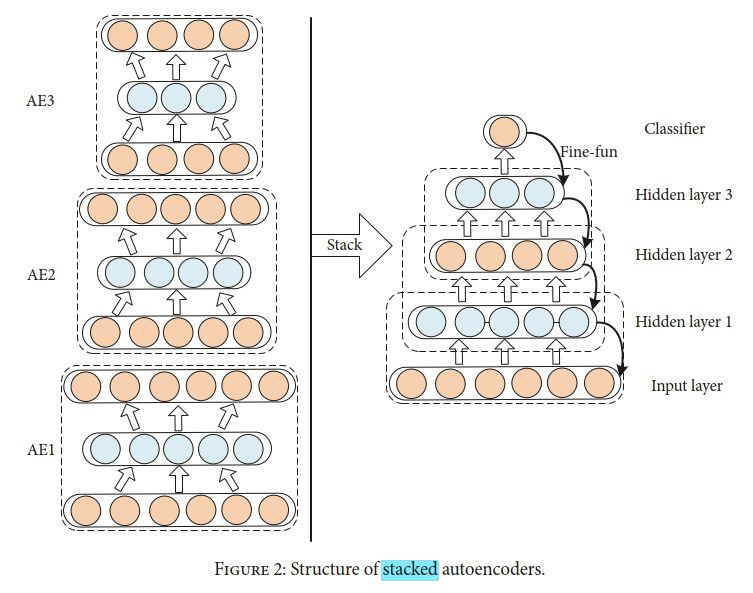
Denoising autoencoder is first proposed in (*Ref. Stacked Denoising Autoencoders Learning Useful Representations in a Deep Network with a Local Denoising Criterion, 2010.pdf*) and is trained with the slightly corrupted input data. There are two underlying ideas about denoising autoencoder. The first is that the high-level representation of data should be robust and stable even when the input data is corrupted. The second idea is that autoencoder training with corrupted data can help learn the input data's useful structure. (*Ref. Fault detection for ironmaking process based on stacked denoising autoencoders.pdf*) explore the performance of denoising autoencoder by adding different types of noise to the input. Furthermore, 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 (*Ref. Deep Learning for Anomaly Detection A Survey*).

### Stacked Autoencoder.

Stacked autoencoders (SAEs) are formed by stacking the hidden layers of autoencoders. As *figure* shows, each autoencoder is first trained in an unsupervised way, and the trained hidden layer is extracted and stacked together as the hidden layer of the stacked autoencoder. With an additional input and output layer, the stacked autoencoder is then fine-tuned in a supervised way (*Ref. A Stacked Autoencoder-Based Deep Neural Network for Achieving Gearbox Fault Diagnosis, 2018*). Specifically, the training process of SAEs can be divided into three detailed steps:

1. Train the first autoencoder with the input data 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 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 the stacked autoencoder 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 will be revised later*) (*Ref. A Stacked Autoencoder-Based Deep Neural Network for Achieving Gearbox Fault Diagnosis, 2018.pdf*)

(*Ref. Why Does Unsupervised Pre-training Help Deep Learning?*) discuss the potential benefit of deploying pre-trained layers in the networks. 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 regularizer term of sparse or denoising autoencoders in pre-training or fine tunning steps of stacked autoencoders.

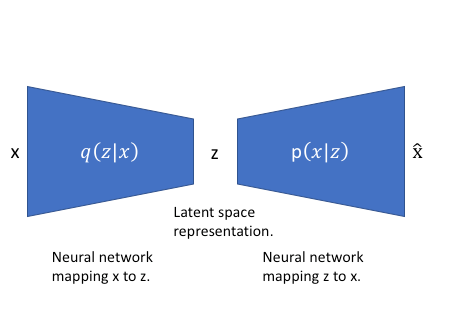
Before detecting anomalies, a stacked autoencoder first pre-train the hidden layers in an unsupervised way with both anomaly and normal data. Then the stacked autoencoder is trained (fine tunned) with the labeled data instances. After the training, the stacked autoencoder can directly predict the probability of new data instances being anomalies.

### 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 into latent space. As the *figure* shows, a variational autoencoder has an encoder and decoder . To reconstruct a data , the variational autoencoder first feed the data 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:

,

where is the expectation and is the Kullback–Leibler divergence which measures the difference between two probability distributions and . One fundamental advantage of variational autoencoder over autoencoder is the continuous latent space because the distribution of in latent space is assumed to be a Gaussian distribution (*Ref. Tutorial on Variational Autoencoders*). For anomaly detection, variational autoencoder uses similar ideeas as autoencoders, i.e., use the reconstruction error to calculate the anomaly score and detect anomalies based on it.

\

(*figure will be revised later*) (*Ref. Anomaly Detection With Conditional Variational Autoencoders*)

### GAN in Anomaly detection

#### AnoGAN

(*Ref. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery*) first proposed an anomaly detection method based on GAN (AnoGAN) to identify the location of an anomaly in the image. AnoGAN is a deep convolutional network trained with only normal image samples using the GAN framework (*Ref. Generative Adversarial Nets*). After the training, the generator has learned to generate normal samples rather than anomaly samples. The difference between the input and reconstruction can help to detect anomalies. As (*Ref. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*) discussed, the in latent space has the property of smooth transition. In other words, two neighbor point , in latent space can generate similar images , in data space. In order to find the best (to fulfil ) in latent space of an 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 , where

,

,

The residual loss measures the similarity between input and the generated sample , and the discriminator loss helps to improve the training stability of GAN. is the weight parameter between those two losses. The value of at step is defined as the anomaly score, where a small anomaly score implies that the input is similar to the normal samples during training, whereas a large anomaly score means that the input is an anomaly sample. AnoGAN is the first proposed GAN framework showing that GAN can also be used for anomaly detection. The main limitation of AnoGAN is the calculation of anomaly score needs steps optimization, which is computational expensive (*Ref. GANomaly Semi-Supervised Anomaly Detection via Adversarial Training*).

#### EGBAD

To overcome the limitation of AnoGAN, (*Ref. Efficient GAN-Based Anomaly Detection*) 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 as in AnoGAN and improves efficiency.

#### GANomaly

(*Ref. GANomaly Semi-Supervised Anomaly Detection via Adversarial Training*) introduced the GANomaly method which is inspired by BiGAN (*Ref. Adversarial Feature Learning*), AnoGAN (*Ref. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery*) and EGBAD (*Ref. Efficient GAN-Based Anomaly Detection*). As figures shows, the network is composed of three sub-networks. The first sub-network is the generator of the GAN model. The generator deploys a *bottleneck* autoencoder network with an encoder and decoder . The generator is first fed with 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 using the autoencoder 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:

,

,

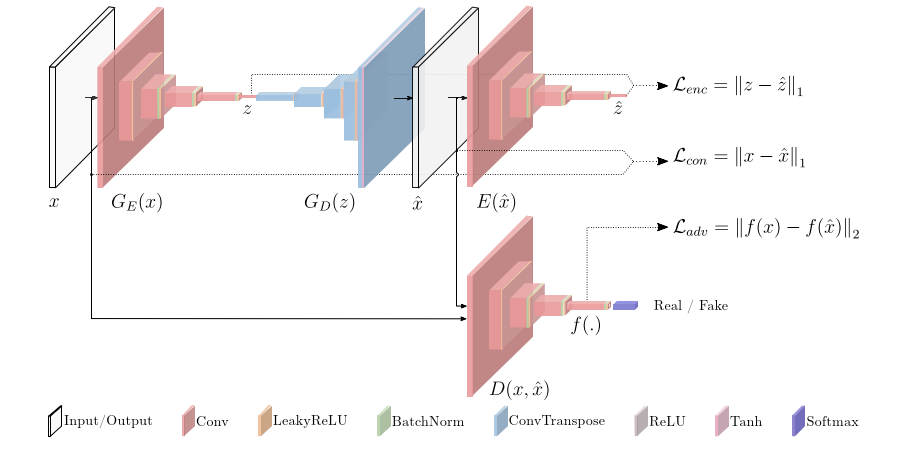
,

,

where , and are the weights of different loss functions. The adversarial loss is used to improve the stability of GAN training as (*Ref. Improved techniques for training gans*) recommended. is the intermediate layer’s output of discriminator . The contextual loss helps the Generator to reconstruct the input data as (*Ref. GANomaly Semi-Supervised Anomaly Detection via Adversarial Training*) discussed. (*Ref. Image-to-Image Translation with Conditional Adversarial Networks*) also use a similar loss function to help the generator generate more realistic data. 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

,

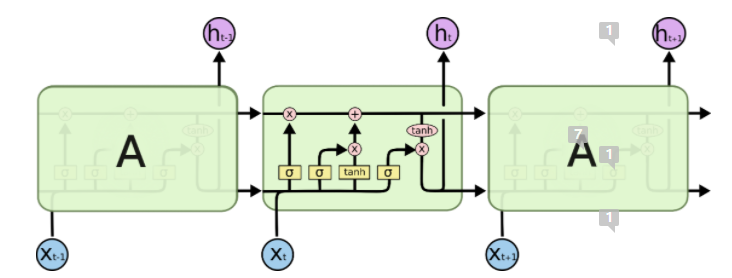
where a higher anomaly score means the test point is likely to be an anomaly.



(*figure will be revised later*) (*Ref. GANomaly Semi-Supervised Anomaly Detection via Adversarial Training*)

### Long Short-Term Memory Networks (LSTMs)

To solve the problem encountered in RNN, (*Ref. LONG SHORT-TERM MEMORY, 1997*) introduced Long Short-Term Memory network, an efficient gradient-based method using multiplicative gate units to control the error flow. (*Ref. Learning to forget: continual prediction with LSTM*) enhance the performance of LSTM by introducing an adaptive *forget gate*, which enables LSTM to reset at an appropriate frequency. The LSTM network is composed of the memory cell and every memory cell has weights and gates as the figure shows.



(*figure will be revised later*)

## Summary

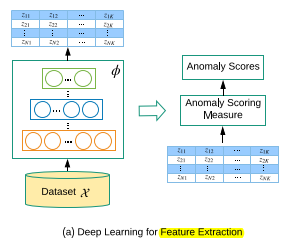
Considering the model architecture, working principle and training framework, the anomaly detection methods are categorized into two main types, feature extraction anomaly detection and feature representation anomaly detection.

### Feature extraction anomaly detection

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

,

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. As figure shows, the feature extractor first extract features from dataset and transform it into latent space. Then the extracted features are fed to the anomaly scorer to calculate the anomaly scorer . The feature extractor and anomaly scorer can be trained independently or jointly, which is also termed as mixed or fully deep approach in (*Ref. Deep One-Class Classification*) when deep learning is involved. 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 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. For the jointly trained approach, feature extractor and anomaly scorer are trained simultaneously with the same objective function. Except for the above training procedure, a pre-trained feature extractor can also be trained (fine-tuned) again during the training of anomaly scorer as introduced in stacked autoencoders.



(*figure will be revised later*)

In practical implementation, feature extractor can deploy methods such as PCA (*Ref. An improved SVM integrated GS-PCA fault diagnosis approach of Tennessee Eastman process*) (*Ref. Nonlinear dynamic process monitoring based on dynamic kernel PCA*), autoencoders (*Ref. High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning*) and GAN (*Ref. f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks*) to extract the features. For the anomaly scorer , it can apply classifiers such as SVM (*Ref. SVM-based Deep Stacking Networks*), random forest tree (*Ref. Random forests classifier for machine fault diagnosis*), or neural network classifier (*Ref. Neural Networks for Classification A Survey, 2000*) and 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 some boundaries can separate normal and anomaly data instances.

Feature extraction anomaly detection has one significant advantage,i.e., a large number of deep learning models and off-the-shelf classifiers can be directly combined. However, it has two main drawbacks, i.e., performance relies heavily on accurate labels and a balanced distribution of training data if the anomaly scorer involves the classifier. Accurate labels usually require a massive amount of effort from domain experts to assign manually. Furthermore, anomalies are rare events compared to normal data instances, leading to imbalanced training data distribution between positive and negative class instances (*Ref. Deep Learning for Anomaly Detection A Survey*).

### Feature representation anomaly detection

Feature representation anomaly detection refers to deploying the models such as autoencoder or other models with encoder-decoder architectures to detect anomalies with reconstruction error. As discussed in chapter 2, 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 (*Ref. Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville*). The definition is as follows:

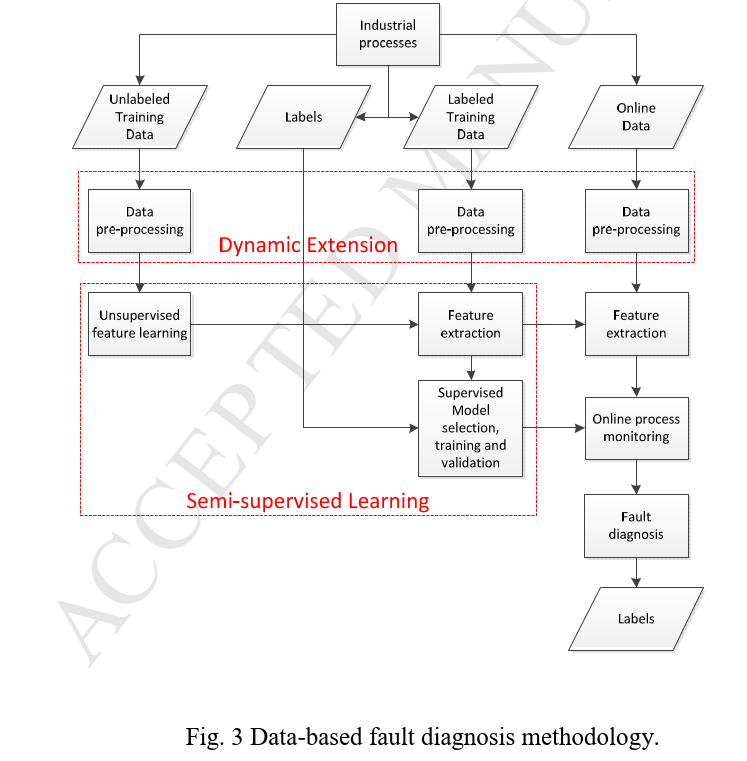
,

,

,

,

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 (here use mean squared error). is the reconstruction error of new data instances using the optimized . Generally, this type of anomaly detection method assumes that the model can better reconstruct the normal data instances from latent space than anomaly data instances when it is only trained with normal data instances.



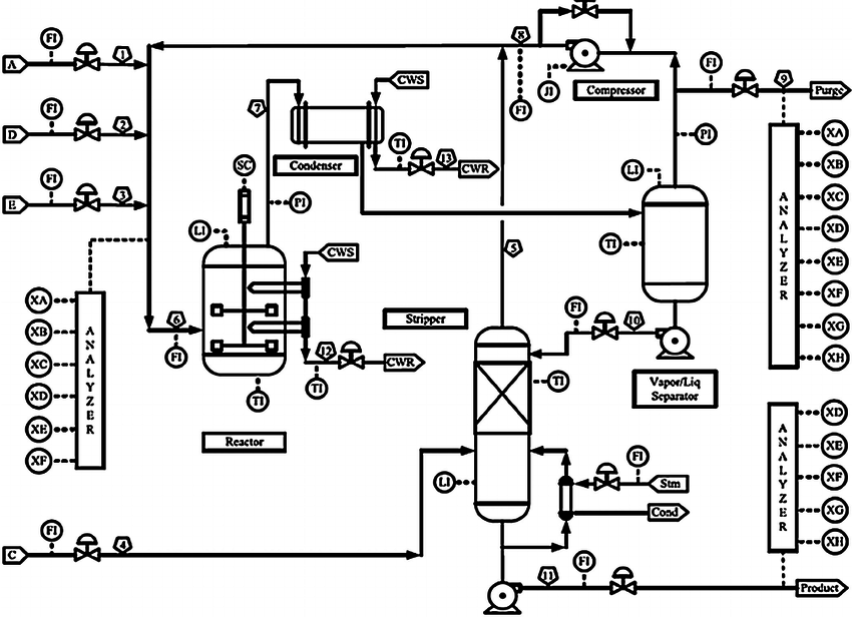
(*figure will be revised later*) (*Ref. Semi-supervised fault classification based on dynamic Sparse Stacked auto-encoders model, 2017.pdf*)

# Experiment and Analysis

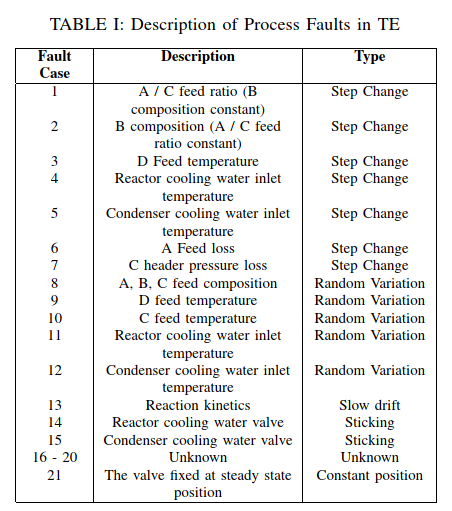
This chapter describes the basic steps of experiments includes data preparation, model training and evaluation. First, Section 4.1 (cross ref) introduces the Tennessee Eastman process simulation data in experiments and the basic steps for data preparation. Next, Section 4.3 builds a baseline anomaly detection model and illustrates the general evaluation steps. Finally, Section 4.4 provides an extensive evaluation of both traditional and deep learning anomaly detection methods, compares the performance differences, and analyzes the reasons.

## Tennessee Eastman Process Simulation Data

The Tennessee Eastman process is a typical industrial chemical process model and was first proposed by (author) (*Ref. A PLANT-WIDE INDUSTRIAL PROCESS PROBLEM*). 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 (*Ref. A PLANT-WIDE INDUSTRIAL PROCESS PROBLEM*) as the figure 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 the following table in detail.



(*figure will be revised later*) (*Ref. A PLANT-WIDE INDUSTRIAL PROCESS PROBLEM*)

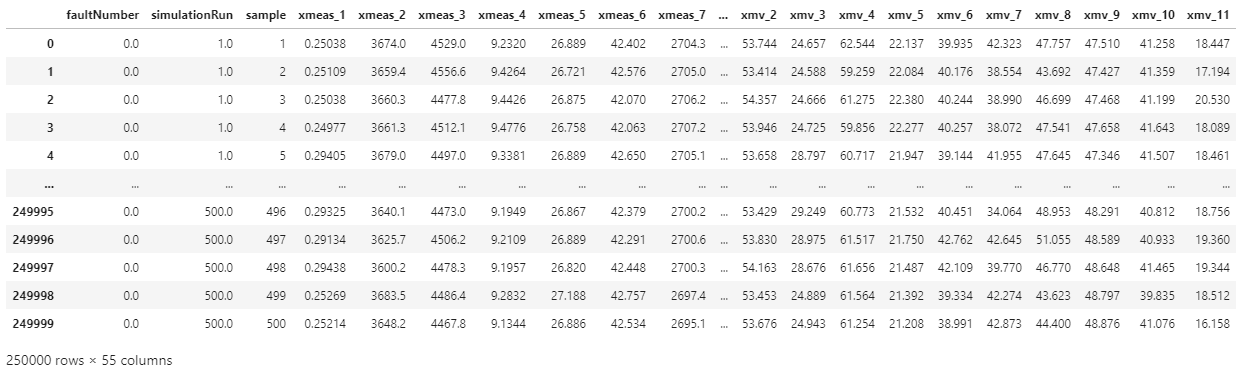


(*figure will be revised later*) (*Ref. Comparison of Deep Neural Network Architectures for Fault Detection in Tennessee Eastman Process.pdf*)

This thesis uses the Tennessee Eastman Process (TEP) simulation data (*Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*) as the training and testing data in the experiments. The simulation data contains four datasets:

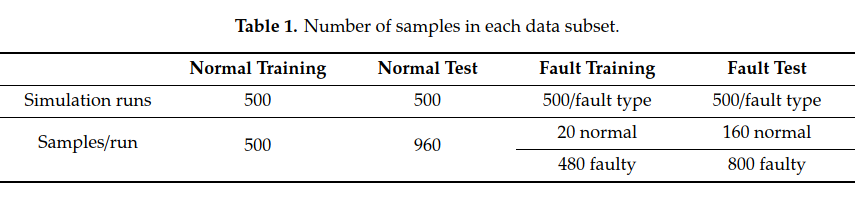
* fault\_free\_training.RData. Data generated in normal operating conditions for model training.
* fault\_free\_testing.RData. Data generated in normal operating conditions for model testing.
* faulty\_training.RData. Data generated in abnormal operating conditions for model training.
* faulty\_testing.RData. Data generated in abnormal operating conditions for model testing.

Each dataset contains 55 columns as the figure shows. The first column is the fault number, which ranges from 0 to 20. 0 stands for fault-free data, whereas 1-20 are the indexes for 20 types of fault. Note that fault 21 is not included in this dataset and thus not 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, then generates the TEP simulation data (52 TEP variables). The TEP simulation data is sampled every 180 seconds from the simulator. The third column is the sample index in each simulation. The 52 TEP variables are sampled every 180 seconds. The rest columns are the process variables: 41 measured variables (, where ) and 11 manipulated variables (, where ).



(*figure will be revised later*) (fault\_free\_training.RData)

For fault\_free\_training.RData and faulty\_training.RData, the sample index in each simulation ranges from 1 to 500 for a total of 25 hours. For fault\_free\_testing.RData and faulty\_testing.RData, the sample index in each simulation ranges from 1 to 960 for a total of 48 hours. For faulty\_training.RData (resp. faulty\_testing.RData), the fault is introduced at sample index 20 (resp. 160). Thus, only the data after introducing the fault is used in the experiments. The data structures are summarized in the *table*.



(*figure will be revised later*) (*Ref. Statistical Process Monitoring of the Tennessee Eastman Process Using Parallel Autoassociative Neural Networks and a Large Dataset, 2019.pdf*)

### 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 (*Ref. Applied Predictive Modeling by Max Kuhn, Kjell Johnson, chapter 3, s27*). Data preparation can be generally summarized into the following steps (*Ref. Data Preparation for Machine Learning by Jason Brownle, chapter 1, s 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 (*Ref. GAIN: Missing Data Imputation using Generative Adversarial Nets*).

#### Feature Selection

For high-dimensional data, we might encounter the *curse of dimensionality* (*Ref. Bellman, Richard Ernest; Rand Corporation (1957). Dynamic programming*). One solution is to select the most relevant input features and ignore the irrelevant and redundant features.

#### Data Transforms

Generally refers to changing the distributions or the types of the raw data. Two common transform methods are normalization transform and standardization transform. Normalization transform scales the data between 0 and 1. Standardization transform scales the data to a normal distribution. A value and is transformed by:

,

.

The mean and standard deviation are calculated with

,

,

where is the number of samples. It is worth mentioning that the scaler (min, max, mean and standard deviation) should be fixed after transforming the training data to prevent data leakage (*Ref. Data Preparation for Machine Learning by Jason Brownlee (z-lib.org).pdf, chapter 4.2, s27*).

#### Feature Engineering

Engineering new features based on training data may improve the model’s performance (*Ref. Feature Engineering and Selection A Practical Approach for Predictive Models by Max Kuhn, Kjell Johnson (z-lib.org).chapter 1.4*). However, the creation of new features may require experts’ knowledge.

#### Dimensionality Reduction

Alternative to feature selection, dimensionality reduction refers to projecting the input data into a lower-dimensional space while retaining the original data's most meaningful properties. Common dimensionality reduction techniques are principal component analysis (*Ref. A Tutorial on Principal Component Analysis*) and linear discriminant analysis (LDA) (*Ref. The Use of Multiple Measurements in Taxonomic Problems.*).

### TEP Data Preparation

There are no errors nor missing values found in the TEP simulation data (*Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*). Data preparation steps such as feature selection, feature engineering, and dimensionality reduction are not applied because we first use an autoencoder as the baseline model to learn the useful features of the TEP simulation data rather than manually engineer features. In data transformation, the data is transformed using standardization transform with the formula as recommended in (*Ref. AutoEncoder based High-Dimensional Data Fault Detection Detection System*):

,

where and is the mean value and standard deviation of the TEP variables in th column, refers to the sample index. is set to 2 as (*Ref. AutoEncoder based High-Dimensional Data Fault Detection System*) recommended because some TEP variables fall outside the range of activation functions, such as (range: -1 to 1) or sigmoid (range: -1 to 1). Both the and are first calculated based on fault\_free\_training.RData, then fixed 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 (*Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*) and obtain some basic understandings of the normal (fault-free data) datasets and abnormal (faulty data) datasets.

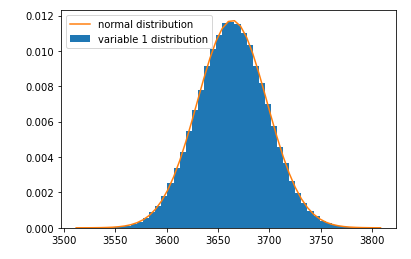
#### Statistical Measures

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

(*figure will be revised later*)

#### Data Distribution

We explore the data distributions of 52 TEP variables. As the plot shows, the measured variable follows a normal distribution. Besides, all other 51 TEP variables also follow normal distributions.



(*figure will be revised later,*) (*2nd variable measurement distribution of fault\_free\_training.RData*)

#### Data Correlation.

We calculate the covariance matrix between different TEP variables. The covariance is calculated with the formula:

,

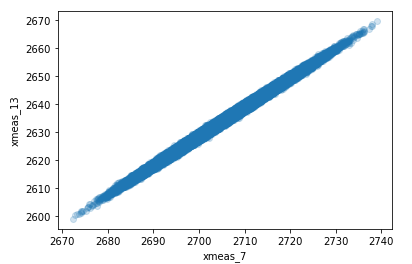
where is the expectation, is the variable and is the variable.

We preserve the variable pairs with covariance value and create the following table.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **xmeas\_13** | **xmeas\_16** | **xmv\_3** | **xmv\_5** | **xmv\_6** | **xmv\_7** | **xmv\_8** | **xmv\_9** | **xmv\_11** |
| **xmeas\_1** | -0.35 | -0.26 | 1.00 | -0.47 | 0.15 | 0.03 | 0.05 | -0.06 | 0.05 |
| **xmeas\_7** | 1.00 | 0.98 | -0.34 | 0.84 | -0.52 | -0.10 | -0.03 | -0.18 | 0.07 |
| **xmeas\_10** | -0.37 | -0.42 | 0.06 | -0.20 | 0.94 | 0.03 | 0.01 | 0.40 | 0.02 |
| **xmeas\_12** | -0.10 | -0.12 | 0.03 | -0.04 | 0.05 | 1.00 | 0.04 | 0.10 | 0.00 |
| **xmeas\_13** |  | 0.97 | -0.35 | 0.86 | -0.51 | -0.10 | -0.03 | -0.15 | 0.07 |
| **xmeas\_15** |  | -0.03 | 0.05 | -0.04 | 0.00 | 0.04 | 1.00 | -0.01 | 0.03 |
| **xmeas\_17** |  |  | -0.03 | -0.06 | 0.00 | 0.00 | -0.03 | 0.00 | -1.00 |
| **xmeas\_18** |  |  | -0.07 | 0.10 | 0.39 | 0.09 | -0.02 | 0.98 | -0.02 |
| **xmeas\_19** |  |  | -0.09 | 0.27 | 0.28 | 0.09 | -0.01 | 0.99 | -0.02 |
| **xmeas\_20** |  |  | -0.45 | 0.92 | -0.25 | -0.02 | -0.03 | 0.45 | 0.01 |

(*figure will be revised later,*)

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



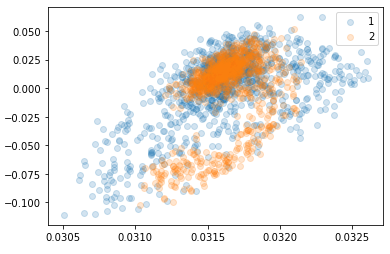
(*figure will be revised later,*) (add mean and std in the figure)

#### ~~Visualization with PCA~~

~~We use PCA to extract the first two principal components of abnormal data with fault type 1 and 2 as the figure shows. First, two types of abnormal data instances overlap, and no clear boundary can separate them. Secondly,~~

~~(Ref. https://plotly.com/python/pca-visualization/)~~

~~One reason is that PCA is a linear method that can better separate dissimilar data instances in low-dimensional representation. However, for high-dimensional and nonlinear data like TEP data, it is more important to maintain the close distance between similar data instances (~~*~~Ref. Visualizing Data using t-SNE~~*~~).~~

~~~~

~~(~~*~~figure will be revised later~~*~~)(PCA visualization of data for fault 1 and fault 2)~~

#### ~~Visualization with t-SNE~~

~~t-SNE is a visualization for high-dimensional data which is capable of preserving both the global and local structure of data in low-dimentsional space. Unlike PCA which separates data using the global structure, t-SNE focus on covert the data into low-dimentsional space using the pairwise similarities (local structure). Besides, t-SNE has series of tunnable parameters, “perplexity” and “steps” are two parameters which can influence the performance greatly. “perplexity” is plays the role of adjust the effective number of neighbors, in other words, it can balance the global and local structure of the data.~~

(*Ref. Visualizing data using t-SNE*).

~~show normal data, faulty data distribution~~

**~~Visualization with variational autoencoder~~**~~. ???~~

**~~Visualization with LDA~~**~~. ???~~

## Experiment Environment

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

|  |  |
| --- | --- |
| 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 |

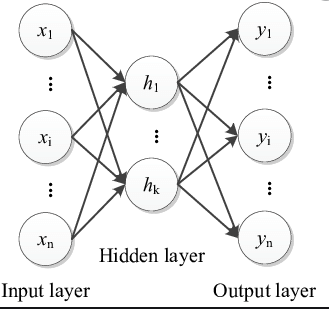
(*figure will be revised later,*)

## Baseline Models and Evaluation Metrics

As author name (*Ref. Ameisen, Emmanuel. “Always Start with a Stupid Model, No Exceptions.” Medium, Insight Fellows Program, (2018), Link*) remarked, “A baseline will take you less than 1/10th of the time, and could provide up to 90% of the results.” A baseline model can help us recognize the potential challenge, provide decent performance without wasting too much time training and debugging. We build a naïve autoencoder as the baseline model, which has only one hidden layer. Then we use the baseline autoencoder as an example to illustrate the evaluation metrics introduced in Chapter 3 (*cross ref*). We choose autoencoder as the baseline model because autoencoders are trivial to train and often provide remarkable anomaly detection performance (*Ref. Outlier Detection Using Replicator Neural Networks, 2002*). The baseline autoencoder is a feedforward 3-layers neural network with one hidden layer, as *figures* shows. The baseline autoencoder is trained only with the standardized normal training dataset (fault\_free\_training.RData) and learns to reconstruct 52 TEP variables during the test phase. The complete training and testing are repeated several times to prevent the influence caused by the stochastic nature of the algorithm. The loss function uses the mean square error to calculate the reconstruction error:

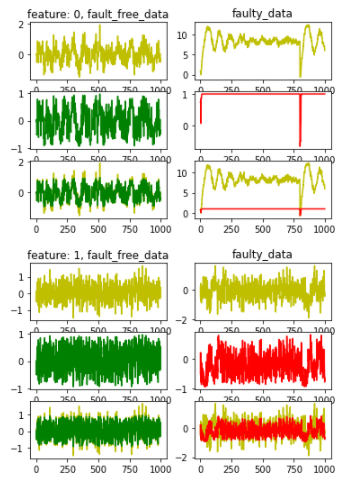
,

where is the input data and is the reconstructed data.

******

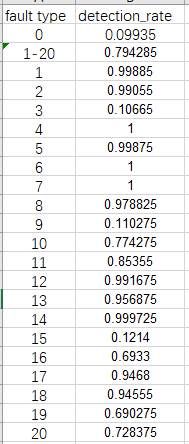
(*figure will be revised later*) (add layer node number, detail structure )

During the testing phase, normal testing dataset (fault\_free\_testing.RData) and abnormal testing dataset (faulty\_testing.RData) are used to calculate the false alarm rate(false positive rate) and detection rate (true positive rate). A threshold is chosen to detect anomalies by thresholding the reconstruction error. Fault free data (resp. faulty data) with is classified as false positive (resp. true positive). value is chosen such that the false alarm of the normal training dataset (fault\_free\_training.RData) is kept around 10%, and an early stopping is set to prevent overfitting. The training takes about ten epochs to finish. Then we use the trained baseline autoencoder to reconstruct the testing dataset (fault\_free\_testing.RData and faulty\_testing.RData). As the figure shows, the normal testing dataset could be reconstructed reasonably well. By contrast, some variables of the abnormal testing dataset can be barely reconstructed because those variables fall out of the activation range (: from -1 to 1).



(*figure will be revised later*) (add labels, fault type, reconstruction of features, simulation index, attach complete figure)

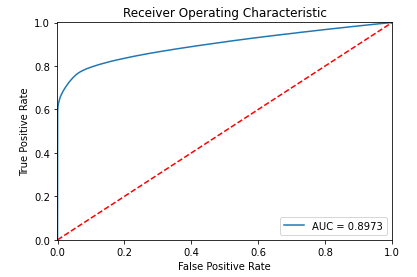
The table lists the detection rate of 20 faults. the false alarm (fault\_free\_testing.RData) is 9.9% and the total detection rate (faulty\_testing.RData) is 79.4%. It is worth noting that the detection rate for faulty 3, 9, 15 is less than 13%. (*Ref. Enhanced statistical analysis of nonlinear processes using KPCA, KICA and SVM, 2009.pdf*) explained that one reason for high missing detection is the absence of observable change in the process variables regarding mean and standard variance. We also observed similar effects in the experiments. Therefore, in most literature (*Ref. Statistical Process Monitoring of the Tennessee Eastman Process Using Parallel Autoassociative Neural Networks and a Large Dataset, 2019.pdf*), (*Ref. Study on Support Vector Machine-Based Fault Detection in Tennessee Eastman Process.pdf*), those faults are often not considered in the testing. For faults 10, 11, 16, 19, and 20, the detection is around 70%. For the rest faults, the detection rate can reach about 95%.



(*figure will be revised later*) (detection rate of 20 fault types) (number accuracy revise)

Then we plot the ROC and calculate the ROC AUC score of the baseline autoencoder. As the figure shows, the ROC AUC score of the baseline autoencoder is 0.8973. As discussed in Chapter 2 (*cross ref*), a higher ROC AUC score indicates that more threshold values can separate the normal and abnormal data. Therefore, the high ROC AUC score suggests that the baseline autoencoder exhibits a decent performance in separating anomaly and normal data instances.

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



(*figure will be revised later*) (ROC curve)

## Experiment Result and Analysis

After building the baseline autoencoder, we have obtained a basic understanding of the anomalies in TEP simulation data (*Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*). The faults whose TEP variables have high variance and a large range of values are trivial to detect. In this section, we investigate the performance of different anomaly detection methods.

### Autoencoders and Regularization

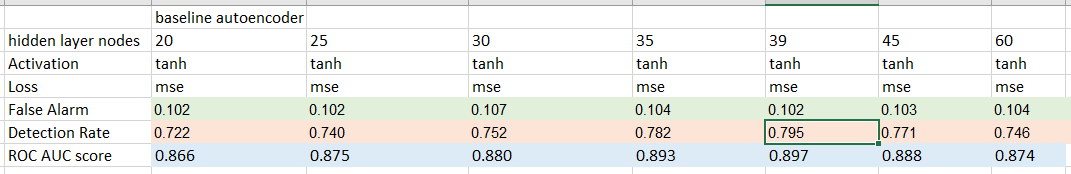
As the number of nodes and layers increases, the deep learning model has a higher representational capacity. In other words, it is capable of representing more complex functions (*Ref. Deep learning, chapter 11.4.1*). However, as the Section 2.3.2 (*cross ref*) discussed, the model tends to overfit when the capacity is too large and underfitting when the capacity is too small.

(*Ref. Neural Networks for Pattern Recognition by Christopher M. Bishop (z-lib.org).pdf, chapter 9, s332*) advised two approaches to improve the generalization property. The first approach is to adjust the number of hidden units in the neural network. The second approach is to adjust the complexity of the model using the regularization technique. For deep learning models, the number of nodes and the number of layers determine the capacity. Moreover, the regularization can be realized by applying weight regularization, activity regularization, weight constraint, noise regularization, and dropout (*assumes introduced before*).

In this section, we attempt to improve the performance of the baseline autoencoder by adjusting the number of nodes in the hidden layer, deploying various regularization methods, changing the loss function and activation functions.

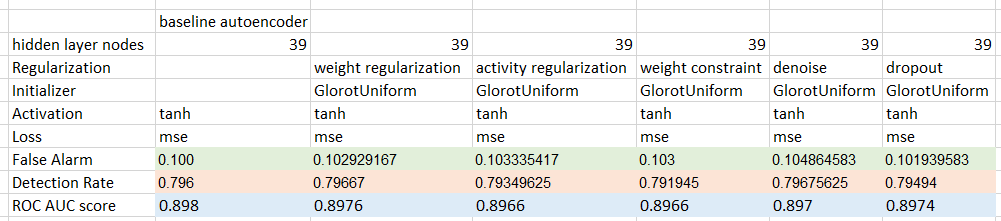
We first use the exhaustive grid search (*Ref.* [*https://scikit-learn.org/stable/modules/grid\_search.html*](https://scikit-learn.org/stable/modules/grid_search.html)) to select the optimal number of nodes in the hidden layers, weight initializer, activation function and loss function. Then we use the random grid search (*Ref. Random Search for Hyper-Parameter Optimization*) to select the parameters for different regularization methods.

The experiment result is shown in the following table. The number of nodes in the hidden layer has the most influence on the performance and 39 nodes. We also find that GlorotUniform initializer (*Ref. Understanding the difficulty of training deep feedforward neural networks*) can stabilize the performance of the autoencoders in each training process, activation function and loss function MSE can guarantee the best performance.



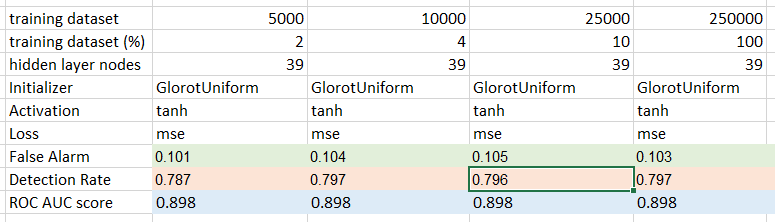
(*figure will be revised later*) (baseline autoencoder with hidden layer nodes)

We then investigate the influence of regularization methods. The weight regularization, activity regularization, and weight constraint are applied to the hidden layers. To realize noise regularization, we add different levels of Gaussian noise to different layers in the autoencoder. The dropout layer is also introduced to the networks in a similar way. However, no noticeable performance improvement is observed after applying regularization techniques. One explanation is that the baseline autoencoder has reached its’ optimal capacity. With the optimal capacity, a large enough dataset can achieve similar regularization effects as the regularization techniques.



(*figure will be revised later*) (baseline autoencoder with different hyperparameters)

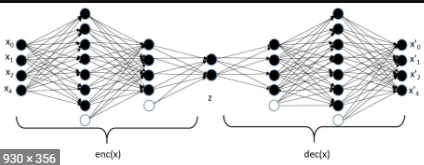
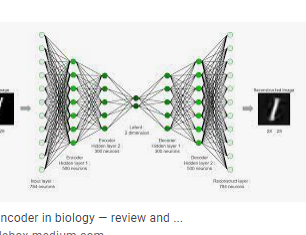
We further explore the relationship between the size of the training dataset and the autoencoder’s performance. The figure shows that no performance improvement is observed when more than 4% of the normal training data points are used for training.



(*figure will be revised later*) (baseline autoencoder with different training dataset size)

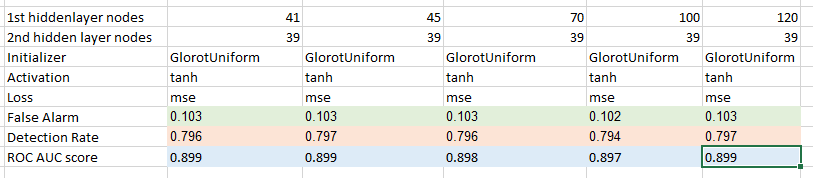
### Deep autoencoders

To further explore the performance of autoencoder in anomaly detection, we test the autoencoders with three and five hidden layers. Generally, we test two types of deep autoencoders structures in the experiments. The first type of deep autoencoders has a progressively decreasing structure until the middle hidden layer (bottleneck) as the left part in the figure shows. The second type of autoencoders has an overcomplete first or second hidden layer (*Ref. Extracting and composing robust features with denoising autoencoders.pdf*) as the right part in the figure shows. We use the random grid search (*Ref. Random Search for Hyper-Parameter Optimization*) to select the number of nodes in each hidden layer and use backpropagation to update the weights.



(*figure will be revised later*) (image autoencoder with extending and shrink structure, 3 hidden layer and 5 hidden layers)

The result is as the following table, the deep autoencoders do not improve performance with more hidden layers. In contrast, we found that the deep autoencoders are more prone to overfit the training dataset than the baseline autoencoder. The overfitting can be prevented by deploying activity regularization in all hidden layers or more strict early stopping criteria (stop training as soon as no improvement in loss function). We further test the decoder with linear activation as author name (*Ref. Semi-supervised fault classification based on dynamic Sparse Stacked auto-encoders model, 2017.pdf*) suggested and no performance difference is observed. More importantly, the number of nodes in the middle hidden layer dramatically impacts the anomaly detection performance. A middle hidden layer with 39 nodes (same as the hidden layer of the baseline autoencoder) can ensure the best detection performance. The experiment result implies that a more complex autoencoder structure does not bring performance improvement instead increases the possibility of overfitting. In contrast, the number of nodes in the middle hidden layer determines the anomaly detection performance.



(*figure will be revised later*) (deep autoencoder AE5 performance result )

### Traditional ML Classifiers

The autoencoders introduced before are trained in an unsupervised way, i.e., non-labeled training data is used during the training. Classifiers work by building direct mapping between input (normal and abnormal data) and output (label). We test series of traditional machine learning classifiers to show their performance. Random forest classifiers, decision tree classifiers and AdaBoost classifiiers need no data pre-processing, it can be trained directly with the TEP simulation data ((*Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*)). For support vector classifiers, the data is pre-processed by standardization transformation. Besides, the training data is created by shuffling the normal and abnormal training data. The experiment result is as table shows. The support vector classifier has the worst performance and longest training time. Random forest classifier has slightly better performance than neural network classifier. The parameters of the random forest classifier are selected with Exhaustive Grid Search (*Ref. https://scikit-learn.org/stable/modules/grid\_search.html*). The neural network classifier is defined as the figure. The neural network classifier has three layers, one input layer, one hidden layer, and one output layer. The detailed parameter can be found in the table.



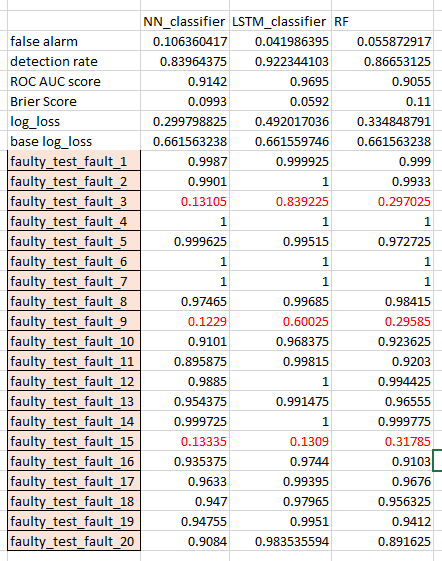
(*figure will be revised later*)

### Neural Network Classifiers

In this section, we build two deep learning binary classifiers and compare them with random forest classifiers. The first classifier (NN\_classifier) is a fully connected neural network classifier. Based on the NN\_classifer, the second classifier (LSTM\_classifier) deploys additional LSTM network layers. The detailed network structure can be found in the Appendix (cross ref.). The training data is created by shuffling the normal and abnormal training data and standardized with the formula in Section 4.1.2 (cross ref.). Early stopping is set to prevent overtraining. At the output layer, the activation function is Sigmoid, and the loss function is binary crossentropy which is defined:

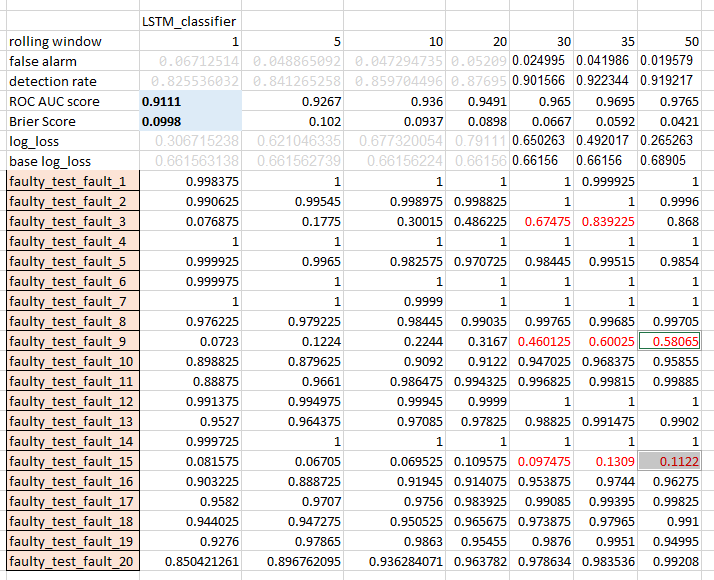
,

where is the number of output values, is the expected output value and is the actual output value. For LSTM\_classifier, we use the time\_step=35 (also rolling window size) to pre-process the input data, i.e., we use the original input data points from time to as the new input data points for time . The experiment result is as the following table. LSTM\_classifier has the lowest false alarm rate, while NN\_classifier has the highest false alarm rate. LSTM\_classifier has the best detection rate (92%) regarding all faults, and NN\_classifier can only achieve an 83% detection rate. Faults 3, 9, 15 are still difficult to detect. However, LSTM\_classifier can improve the detection performance for faults 3 (resp. 9) to 83% (resp. 60%), whereas random forest classifiers have an advantage in detecting fault 15 with a 31% detection rate.



(*figure will be revised later*) (neural network classifiers performance)

As discussed in Section 3.5.5 (cross ref. ), the LSTM network can learn the temporal relation of data points. To further investigate the contextual relation of data points, we train the LSTM\_classifier with different time\_steps. From the result, it can be seen, as the time\_step increases, the general performance is improved. For faults 3, 9, the detection rates increase significantly as the time\_step increases. However, the training and inference time also increases as the time\_step increases.



(*figure will be revised later*) (LSTM classifiers with different rolling window sizes) (need to add more data)

### ~~Autoencoders with Classifiers~~

~~In this section, we conbine autoencoder and classifiers, i.e., extracting the features of TEP variables with the encoder and classify those features with classifiers. We investigate different approaches to train the combined networks. First, we train the encoder and classifier simulitaneouly. Second, we used the pre-training (introduced in Section 3.5.2: stacked autoencoder) approach to initialize the parameters of encoder then fine tune the encoder and classifiers. Third, we keep the parameter of pre-trained encoder unchanged and only train the classifier.~~

~~A performance improvement is expect if the extracted features can preserve discriminative information and help to build clear boundary between normal and abnormal data points.~~

~~(NN\_classifier structure add to appendix)~~

### Generative Adversarial Networks

We also attempt to train an autocoder with a GANomaly-based framework. In the BiGAN-based framework, the generator is an autoencoder and learn to reconstruct the input data points, the discriminator learns to assess if the reconstructed input data points is normal or abnormal. Based on the assumption, the if the autoencoder is only trained with normal data points, it can’t reconstruct the abnormal data points. Finally, based on the reconstruction error and output of discriminator, we can identify the anomalies. However, the limited experiments didn’t yield satisfactory results regarding the TEP simulation data. There are two possible reasons that cause the failure: 1) The generator and discriminator are learning at different speeds which breaks the equilibrium and finally one side fails to learn; 2) the loss function are not guiding the autoencoder towards reconstrucing the input data points. For future research, we could track the learning rate and gradients of both generator and discriminator to ensure a balanced weight updating; try different loss functions and adjust the weight of different sub-loss functions; use different technique to initialize the weights.

# Conclusion and Future Work

## Conclusion

This thesis aimed to tackle the anomaly detection of the Tennessee Eastman process by using different machine learning models. We first evaluate some traditional machine learning models for anomaly detection. Then we implement series of deep learning models based on autoencoders and compare their performance with traditional machine learning models. The performance evaluation showed that the deep learning model could achieve better performance with less training time. Further, the inference time of deep learning models is generally faster than the traditional machine learning model, which is critical in industrial monitoring.

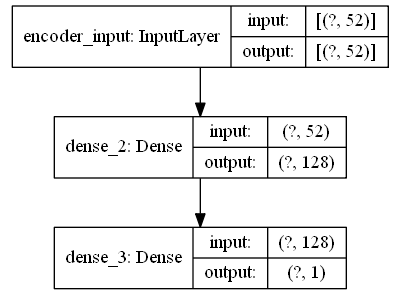
In general, deep learning can be applied used to detect anomalies in two ways. The first way is to use the autoencoder to reconstruct the input data and detect anomalies by thresholding the reconstruction error. As the experiments showed, an autoencoder with one hidden layer can achieve reasonably good performance, and only non-labeled normal training data is need. The second way is to use the encoder part of the autoencoder to extract the input features and train a classifier with the extracted features in a supervised way. The experiments showed that extracted features could help the classifier improve performance. Moreover, the deep learning classifiers which deploy LSTM networks can achieve the best performance. This implies that the temporal relationship of the TEP simulation data can provide extra anomaly patterns.

## Future work

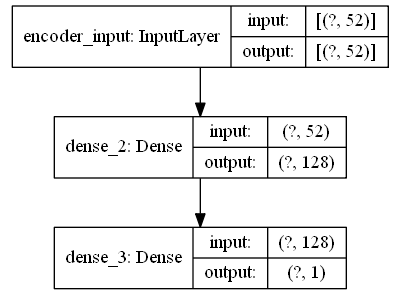
Based on the experiments conducted during this research, the following recommendations for future work are proposed:

* Deep learning model optimization. In the experiments, we have applied Exhaustive Grid Search (*Ref.* [*https://scikit-learn.org/stable/modules/grid\_search.html*](https://scikit-learn.org/stable/modules/grid_search.html)) and random grid search (*Ref. Random Search for Hyper-Parameter Optimization*) to tune the hyperparameters. However, there is still improvement room for hyperparameters, especially for the stacked autoencoders and LSTM classifiers.
* Generative deep learning models. (*Ref. MADGAN: unsupervised Medical Anomaly Detection GAN using multiple adjacent brain MRI slice reconstruction*) and (*Ref. MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks*) has shown the potential of using generative models combining 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 data but only normal data. One promising direction is to develop unsupervised generative models which are specialized for the Tennessee Eastman process.
* Feature selection and engineering. In the experiments, we use all features of TEP simulation data *Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*) as the input. In (*Ref. Fault Detection and Diagnosis in a Chemical Process using Long Short-Term Memory Recurrent Neural Network*) author name used the 11 manipulated variables as the input. However, in our experiments, training with only 11 manipulated variables lead to degrading performance. The main reason is that we use a different TEP simulation dataset. In view of the working mechanism of deep learning models, selecting or creating the more relevant features can significantly speed up the training, inference process and improve the discriminative capability. Studies of selecting relevant features such as (*Ref. Feature selection for fault detection systems : application to the Tennessee Eastman process*) and (*Ref. Unsupervised Feature Selection Based on Fuzzy Clustering for Fault Detection of the Tennessee Eastman Process*) are important research direction.
* 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 as (*Ref. Hierarchical Deep Recurrent Neural Network based Method for Fault Detection and Diagnosis*) is a promising research direction.
* Theoretical limits for anomaly detection performance. Since the TEP simulation data (*Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*) are generated in each simulation run with random state, it is worth studying the influence on performance due to the number of simulations, the number of samples in each simulation, proportional of normal and abnormal data in training data. Understanding those theoretical limits can help us to design more effective deep learning models.
* Effect of noise in TEP data. The TEP simulation data (*Ref. Additional Tennessee Eastman Process Simulation Data for Anomaly Detection Evaluation*) is generated in the simulation software, and no errors/mistakes are found in the data pre-processing steps. However, in real industrial monitoring, noise and measurement errors caused by humans are very common. Therefore, building anomaly detection models that are resilient to the above problems is crucial to applying the anomaly detection model in real industrial production.

**Appendix**



(*figure will be revised later*) (NN\_classifier structure, put in appendix)



(*figure will be revised later*) (LSTM\_classifier structure, rolling window, put in appendix)

……….